HPCLatAm 2013, pp. 37-48 (full paper)
Session: Evolutionary Computation & Scheduling
C. Garcia Garino and M. Printista (Eds.)
Mendoza, Argentina, July 29-30, 2013

List scheduling heuristics for virtual machine mapping in cloud systems

Sergio Nesmachnow[1], Santiago Iturriaga[1], Bernabé Dorronsoro[2], El-Ghazali Talbi[2], and Pascal Bouvry[3]

[1] Universidad de la República, Uruguay
[2] LIFL, University of Lille, France
[3] University of Luxembourg, Luxembourg

Abstract This article introduces the formulation of the Virtual Machine Planning Problem in cloud computing systems. It deals with the efficient allocation of a set of virtual machine requests from customers into the available pre-booked resources the broker has in a number of cloud providers, maximizing the broker profit. Eight list scheduling heuristics are proposed to solve the problem, by taking into account different criteria for mapping request to available virtual machines. The experimental evaluation analyzes the profit, makespan, and flowtime results of the proposed methods over a set of 400 problem instances that account for realistic workloads and scenarios using real data from cloud providers.

1 Introduction

Cloud computing [1,4] is, undoubtedly, one of the main existing computing paradigms nowadays. In the last years, it raised the interest of both academic and industrial worlds thanks to their interesting properties, such as elasticity, flexibility, or computational power, among many others. Cloud computing provides a stack composed of different cloud service provisioning models: Infrastructure as a Service (IaaS), dealing with resources as servers, storage, or networks; Platform as a Service (PaaS), which provide an operating system as well as a set of tools and services to the user; or Software as a Service (SaaS), that allows providers to grant customers with access to licensed software.

Many different public and private clouds are arising in the last years [13]. They all have distinct features, and it makes difficult for users to find the best choice among all the existing offers. The figure of *cloud broker* [5] arises as an intermediary entity between cloud providers and users to help the latter ones in that process. There are different services that cloud brokers can provide, from simply finding the best deals among a set of clouds for the user requirements to defining the best possible design to deploy the user's application in the cloud [5].

One of the salient features of cloud computing is the elasticity and the seemingly infinity of resources they provide. This is achieved by using virtual machines (VMs), that can be dynamically allocated and deallocated to the resources according to the demand and availability, and the possibility of consolidating a number of VMs into the same physical server.

However, it may happen that the available resources are not enough for the current demand at a given peak time, especially in the case of private clouds. In these cases, the *cloud bursting* technique [10] is used to get on-demand VMs (typically) from the public cloud, therefore extending the computing capacity of the facility in a transparent way to the user.

In this paper, we focus on a recent business model in which the broker sublets on demand cloud resources to his customers at low prices. The broker owns a set of reserved VMs with different features, and probably from distinct cloud providers. These VMs are offered to the customers as on-demand resources at cheaper prices than those the customer would get if directly contacting a cloud provider. This business model is feasible and profitable thanks to the large difference in price between reserved and on-demand VMs in the cloud [14]. Additionally, in the case the broker does not have enough VMs to cope with the customers' requests without violating the contracted service level agreement (SLA), he will buy on-demand VMs to the cloud to satisfy the demand (as the cloud bursting technique does in hybrid clouds). In this case, the profit of the broker will be reduced. From now, we will refer the reserved VMs of the broker as *reserved instances* (RI) to differentiate from the VMs the customers demand.

The considered model does not only benefit customers, but also cloud providers, who have more information (thanks to the reserved instances by the broker) to be able to forecast the resources that will be needed in the future.

The problem of efficiently allocating the customers VMs requests into the available RI arises for the broker. All VMs should be allocated into RIs that are offering at least the same performance requested by the customer, and some quality of service (QoS) levels must be achieved by the solution. This is a resource allocation problem with additional constraints making it more complex. Underutilization of the available reserved instances must be avoided, as well as the overbooking, which might force the broker reserving on-demand VMs to the cloud provider in order to offer the promised service, despite the money loss. The resource allocation problem itself is NP-hard [15].

The main contributions of this work are: i) a formulation for the optimization problem of allocating VM requests on a set of reserved instances to maximize the broker profit is developed, ii) eight list scheduling heuristics to efficiently solve the problem are evaluated, and iii) novel benchmark instances were generated for the problem, which are publicly available.

The paper is structured as follows. Next section presents the formulation of the optimization problem tackled in this work. Section 3 offers a review of the main existing works about related brokering proposals for cloud computing. The eight list scheduling heuristics proposed in this work to tackle the virtual machine mapping problem are described in Section 4, just before presenting the experimental evaluation over a set of realistic workloads and scenarios using real data from actual cloud providers in Section 5. Finally, the conclusions and main lines for future work are formulated in Section 6.

2 The Virtual Machine Mapping Problem

We consider that the broker owns a number of pre-booked VMs (that we call *reserved instances*—RI) on one or more cloud providers, and that it receives a set of VM requests from its customers, all of them having specific hardware demands and execution deadline. The Virtual Machine Mapping Problem (VMMP) is to assign all VM requests to the available RIs such that the broker's profit is maximized. In case some user requests cannot be handled with the available RIs, the broker must book on-demand VMs in the cloud for them to keep a high QoS, with the resultant profit reduction. The VMMP is formalized next. Given:

- A set of virtual machine requests $VM = \{v_1, \ldots, v_n\}$, each one to be executed for a given time $T(v_i)$, as it is normally considered in cloud systems.
- Each VM has specific hardware demands, including processor speed $P(v_i)$, memory $M(v_i)$, storage $S(v_i)$, and number of cores $nc(v_i)$.
- Virtual machine requests arrive in batches (i.e., hourly, diary). Arrival times A_i follow a stochastic homogeneous Poisson process with parameter λ.
- The execution of any VM must start before its deadline $D(v_i)$.
- A set of cloud resource instances pre-booked by the broker $B = \{b_1, \ldots, b_m\}$, $m \ll n$, with specific features including processor speed $P(b_j)$, memory $M(b_j)$, and storage $S(b_j)$, according to a predefined list of instance types.
- A cost function C for pre-booked cloud resource instances, and a cost function COD for on-demand instances, with $C(b_j) \ll COD(b_j)$. Both cost functions are given in an hourly basis.
- A pricing function $p(b_j)$ that defines the price the broker charges to the customers per hour for the RI b_j. In order to attract customers, the broker should charge for a VM b_j a lower cost than the on-demand pricing for that VM, i.e., $p(b_j) < COD(b_j)$. Moreover, if the cheapest offered RI that can allocate a request v_i by the user, for instance b_k, is not available, the broker can assign the request to another RI of higher capacity, but charging the same amount as for b_k. This will decrease the profit, but it will prevent the broker from loosing customers, who at the same time will benefit from the better performance offered.

The VMMP consists in finding a mapping function $f : VM \to B$ for the VM requests in the available RIs that maximizes the broker profit, according to the following optimization problem ($ST(v_i)$ is the starting time of v_i, based on f):

$$
\begin{array}{ll}
\max & \displaystyle\sum_{j=1}^{j=m} \left(\sum_{i:f(v_i)=b_j} (p(BF(v_i)) - C(b_j)) \times T(v_i) \right) + \\
& \displaystyle\sum_{h:ST(v_h)>D(v_h)} (p(BF(v_h)) - COD(BF(v_h))) \times T(v_h) \\
\text{subject to} & M(v_i) \leq M(b_j)\ P(v_i) \leq P(b_j) \\
& S(v_i) \leq S(b_j),\ nc(v_i) \leq nc(b_j) \\
\text{where} & \text{the } BF(v_k) \text{ function gives the less expensive instance} \\
& \text{capable of executing the request } v_k
\end{array}
\tag{1}
$$

In the previous problem model, deadlines are considered as hard constraints. In case the broker cannot accommodate the VM request to start execution by the specified deadline, it must either use a larger RI offering more resources than those requested (but at the cost of the one requested by the user) or buy an on-demand instance to fulfill the request. Both solutions obviously accounting for a negative impact in the total cost of the schedule.

The first summation in the objective function accounts for the total profit of the broker thanks to the RIs booked by the customers. The second summation accounts for the additional and the second is the cost that suppose avoiding the violation of the deadline constraints.

Data transmission for VM requests are not considered for the objective function: the model assumes that the broker directly transfers the data transmission costs to the user, thus no economic profit is gained from data transmission.

3 Literature review

Cloud brokering [5] typically deals with the problems of finding the cloud providers whose offer better suits the customer needs (both technically and in terms of cost) [2,16], or providing the customer with the best possible way to deploy his/her application in the cloud [8].

The literature review indicates that there are a number of works proposing methods for scheduling applications in private resources using cloud bursting technique [3]. These works enhance the local schedulers with the capability of using VMs from the public cloud when additional resources are required. This is a similar concept to the one addressed in this paper, since when all RIs are used and a request cannot start before its deadline, then the broker will buy an on-demand instance from the cloud to execute it. However, we do not address the resource provisioning problem, since the broker is an intermediate entity which always works with VMs (either reserved or on-demand) from the public cloud.

Closer to the problem we consider in this article, Wu et al. [17] proposed a mechanism to encourage customers to provide realistic likelihood that they will purchase a given resource, at the reward of price reductions. This mechanism allows the provider to efficiently forecast the required resources, minimizing this way the underutilization and/or overbooking of the available resources, and it will benefit the customer too, who will have the service at a low price. This mechanism was adopted in [14] for the case of a cloud broker subletting reserved VMs to his customers. Then, the broker will use the information given by the customers to decide whether to invest in buying more resources or not, and what kind of resources should be bought. This technique is shown to provide up to 44% increase in the profit of the broker.

In this paper, we investigate how the broker can optimally manage his VMs for obtaining the optimum profit while providing acceptable QoS, by allowing the use of on-demand instances to satisfy the needs of users that cannot be satisfied with the current resources, despite the money loss.

4 List scheduling heuristics for the VMPP

This section introduces the class of list scheduling heuristics and describes the new scheduling heuristics proposed for the VMPP.

4.1 List scheduling heuristics

The class of *list scheduling* heuristics [7] comprises many deterministic scheduling methods that work by assigning priorities to tasks based on a particular criterion. After that, the list of tasks is sorted in decreasing priority and each task is assigned to a processor, regarding the task priority and the processor availability.

Since the pioneering work by Ibarra and Kim [6], where the first list scheduling algorithms were introduced, many list scheduling techniques have been proposed to provide easy methods for tasks-to-machines scheduling. This class of methods has also often been employed in hybrid algorithms, with the objective of improving the search of metaheuristic approaches applied to solve scheduling problems in heterogeneous computing systems [11,12].

4.2 List scheduling heuristics for the VMPP

In this work, eight list scheduling heuristics are introduced to provide different approaches to solve the VMPP, by using diverse criteria for assigning priorities to VM requests. The new heuristics take into account the VMMP objective of maximizing the profit, but also reducing *makespan* metric (defined as the difference between the start and finish time of a given batch of tasks), since obtaining balanced schedules implies a better utilization of the available resources.

The new list scheduling heuristics include:

- **Best Fit Resource (BFR)**. This heuristic assigns each VM request to its best fit RI, disregarding the deadline values. The best fit is defined as a RI with the same number of cores than requested, and the closest amount of memory to the requested value. The approach used in this heuristic intends to take advantage of assigning the requests to those RIs that fit the most, making room for most restrictive requests to be executed in larger RIs.
- **Earliest Finish Time (EFT)**. This heuristic gives priority to those VM requests that can be finished the soonest. The availability of each suitable booked instance for the request is considered to compute the finish time. The main idea behind this heuristic is to take advantage of executing the fastest (i.e., shortest) requests to increase the availability of RIs.
- **Lower Gap First (EGF)**. This heuristics prioritizes those VM requests with tightest deadlines, taking into account the arrival time. The tightness is evaluated by the gap metric, defined as the difference between the deadline and the arrival time of each VM request. Once the tightest request is found, it is assigned to the booked instance with the earliest availability. This heuristic tries to avoid penalizations due to buying on-demand instances by attending first those requests with the smaller room to execute before its deadline.

– **Shortest Task First (STF)**. It gives priority to VMs with shorter execution times, following the idea of the STF method for makespan minimization. The heuristic searches for the shortest unattended VM request, and then it is assigned to the lowest-cost RI that satisfies its hardware requests.
– **Earliest Deadline First (EDF)**. This heuristic gives priority to those VM requests with the earliest deadlines (without taking into account the arrival time), and assigns each request to the suitable booked instance with the earliest availability. The main idea behind this heuristic is to take advantage of fast execution of restrictive requests, to avoid the penalization of buying on-demand instances due to deadline violations.
– **Cheapest Instance (CI)**. This heuristic sorts the VM requests by arrival time and selects the cheapest RI that allows the execution of each request, applying a First Come First Served strategy. This heuristic is intended to reduce the average waiting time of VM requests on the cloud system.
– **MaxProfit (MaxP)**. It is a greedy profit-oriented list scheduling heuristic that uses the contribution of each request to the global profit objective function used in the VMPP formulation. After that, the request that contributes the most to the global profit is assigned to execute in the cheapest reserved instance that fulfill the VM hardware requirements.
– **Shortest Request to Cheapest Instance (SRCI)**. This heuristic works by sorting the VM requests by duration and selecting the cheapest instance that allows the execution of each request, in a similar way than the well-known Shortest Job to the Fastest Resource (SJFR) heuristic for makespan optimization. As SJFR, the SRCI heuristic is intended to maximize the broker profit, as well as to minimize the response time perceived by the user of the cloud infrastructure: since shortest requests are assigned to execute first, they will finish earlier and users with short computing time demands will find that their requests are completed very fast.

5 Experimental analysis

This section presents the experimental evaluation of the proposed list scheduling heuristics over a realistic set of VMMP intances.

5.1 Problem instances

In this work, we developed a set of VMMP instances by following a specific methodology and using real data gathered from public reports, webpages, and nowadays real cloud infrastructures.

The problem instances are defined by a *workload file* and a *scenario file*, with the information about VM requests and RIs, respectively. Each workload defines the requirements for a batch of VM requests, including: memory and storage needed, processor speed required, and number of cores requested. Each scenario file describes the features for the set of RI already pre-booked by the

broker, including: available memory, available storage, processor speed, number of cores, and the cost (both pre-booked and on-demand) and pricing values.

A total number of 400 problem instances are solved in the experimental analysis, by combining workloads and scenarios with diverse characteristics. We consider batches of 50, 100, 200, and 400 VM requests with different durations (ranging from 10 to 200 time units), arriving according to a Poisson process. The considered scenarios built a pre-booked cloud computing infrastructure with 10, 20, 30, and 50 RIs for the broker, by combining VMs from both Amazon and Azure cloud computing services, according to the details presented in Table 1 (price and costs are in US dollars, all configuration and prices are from May, 2013). The VMs selected account for different configurations, including small and average machines (#1 to #3), large machines (#4 and #6), and instances with large memory, CPU and/or storage (#5, #7 and #8).

Regarding the pricing function, we consider in this work that it is 20% cheaper than the cost on demand price (i.e. $p(b_j) = 0.8 * COD(b_j)$). This is a reasonable value for attracting users to the service, while obtaining reasonable profit values.

The VMMP instances generated are available for downloading from `http://www.fing.edu.uy/inco/grupos/cecal/hpc/VMMP`.

Table 1. VMs considered to built the broker pre-booked cloud infrastructure

#	VM id	provider	memory	storage	proc.	nc	price	C	COD
1	m1.small	Amazon	1.7 GB	160 GB	1.0 GHz	1	0.048	0.027	0.06
2	m1.medium	Amazon	3.75 GB	410 GB	2.0 GHz	2	0.096	0.054	0.12
3	A2.medium	Azure	3.5 GB	489 GB	1.6 GHz	2	0.096	0.09	0.12
4	m1.large	Amazon	7.5 GB	850 GB	2.0 GHz	4	0.192	0.108	0.24
5	m2.xlarge	Amazon	17.1 GB	420 GB	3.25 GHz	2	0.192	0.136	0.24
6	A3.large	Azure	7.0 GB	999 GB	1.6 GHz	4	0.328	0.18	0.41
7	c1.xlarge	Amazon	7.0 GB	1690 GB	2.5 GHz	8	0.384	0.316	0.48
8	A4.xlarge	Azure	14.0 GB	2039 GB	1.6 GHz	8	0.464	0.36	0.58

5.2 Development and execution platform

The proposed heuristics were implemented in C, using stdlib and GNU gcc. The experiments were performed on a Xeon E5430, 2.66 GHz, 8GB RAM, and CentOS Linux 5.2 from Cluster FING (`http://www.fing.edu.uy/cluster`).

5.3 Experimental results

We present in this section the results obtained during our experimentation, summarized by instance size. The complete tables of results for each one of the 400 VMMP instances solved are available at `http://www.fing.edu.uy/inco/grupos/cecal/hpc/VMMP`.

Table 2 reports the *average relative ranking* (over all problem instances for every batch dimension) for each heuristic according to the profit objective, defined by Eq. 1. This rank was computed using the non-parametric Friedman statistical test. In the table, the rank of a given heuristic means its position (in average) when sorting the eight heuristics regarding the profit value. The last rows of the table report the χ^2 and the p-values for the Friedman test in each case. We test if the p-values are less than 10^{-3}, in order to state that there exists enough evidence to conclude that the results distributions differ in their true medium profit values for the studied heuristics, with 99% confidence.

Table 2. Mean ranks computed using the Friedman statistical test for the studied heuristics, regarding the profit objective results

		batch dimension (n)				**overall**
		50	**100**	**200**	**400**	
heuristic	**BFR**	5.35	6.34	6.50	7.59	6.44
	EFT	6.61	5.81	5.25	4.26	5.48
	LGF	6.76	6.44	5.82	5.47	6.12
	STF	2.99	3.36	3.77	4.20	3.58
	EDF	6.41	6.19	6.03	5.33	5.99
	CI	3.17	3.56	3.61	4.21	3.64
	SRCI	1.71	1.16	1.10	1.41	1.34
	MaxP	3.02	3.14	3.92	3.54	3.41
	χ^2	474.08	384.74	364.45	365.20	1497.30
	p-value	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$	$< 10^{-3}$

Table 2 shows that SRCI is consistently the heuristic providing the highest benefit to the broker: it got the lowest average rank for every batch dimension. We can see that, aside SRCI, the heuristics that target the profit optimization (namely, CI and MaxP) are generally outperformed by STF in all problem sizes. Both SRCI and STF assign first the shortest tasks, which arises as an appropriate policy. Finally, EDF, EFT, LGF, and BFR performed the worst in all cases.

The profit results are reported in Table 3. The *GAP* metric is the relative difference between the profit computed using each heuristic and the best result for each problem instance. Row *#1* indicates how many times the corresponding heuristic performed the best (i.e., the number one) regarding the profit value.

Some heuristics report negative profit values for several of the tested instances. Particularly, this happens for those instances with a high ratio between the VM requests and the available RIs, for which the available RIs are not enough to process all requests. In such cases, the broker is obliged to book a high number of on-demand machines, which are more expensive than the price charged to the customer, in order to avoid deadline violations and keep offering high QoS and so having high reputation. This scenario is expected to occur only during sporadic peak periods, and the money loss is compensated by avoiding the loss of reputation produced by rejecting the customer request. In order to reduce such

Table 3. Profit results for the proposed heuristics

n	metric	heuristic							
		BFR	**EFT**	**LGF**	**STF**	**EDF**	**CI**	**SRCI**	**MaxP**
50	*avg. profit*	27.33	18.50	17.50	37.10	18.57	37.00	**40.67**	36.92
	best profit	47.47	46.21	46.92	**56.75**	47.19	**56.75**	**56.75**	**56.75**
	avg. GAP	35.4%	57.1%	59.5%	9.4%	54.8%	9.5%	**1.0%**	9.6%
	#1	5	0	0	30	1	16	**79**	29
100	*avg. profit*	27.45	28.92	25.14	50.19	25.19	49.67	**63.94**	50.62
	best profit	70.70	70.38	67.75	87.45	64.99	83.35	**92.94**	84.75
	avg. GAP	69.9%	66.3%	72.0%	21.0%	73.2%	21.6%	**1.2%**	19.4%
	#1	0	0	0	1	0	6	**87**	7
200	*avg. profit*	19.96	30.21	26.75	45.94	25.41	46.41	**68.36**	45.13
	best profit	78.25	101.02	100.39	104.62	104.45	98.94	**130.03**	106.37
	avg. GAP	280.0%	184.6%	261.9%	44.1%	298.9%	36.7%	**3.3%**	50.6%
	#1	0	1	0	0	0	5	**92**	2
400	*avg. profit*	-59.22	-8.63	-22.41	-6.63	-19.30	-7.15	**37.42**	-5.07
	best profit	96.97	183.40	174.86	129.55	193.94	115.02	**204.05**	110.44
	avg. GAP	239.7%	106.1%	154.8%	87.0%	154.5%	92.1%	**0.9%**	84.8%
	#1	0	0	0	0	4	0	**75**	21

situations to a minimum, the broker should have good forecasting methods to predict the VM requests in future months to anticipate and reserve more nodes, if considered to be profitable. This issue is studied in detail in Table 5.

In order to further analyze the planning computed by each list scheduling heuristic, we also evaluate the *makespan* and the *flowtime* for each schedule. The makespan is a metric used to evaluate the resource utilization; it is defined as the time spent from the moment when the first task in the batch begins execution to the moment when the last task in the batch is completed. Lower values of the makespan metric means that the RIs will be able to attend more batches in a given period of time. The flowtime evaluates the sum of the tasks finishing times, and it is an important metric from the user point-of-view, since it reflects the response time of a computational system for a set of submitted tasks [9].

The average makespan and flowtime results are reported in Table 4. If we focus on makespan, LGF clearly stands out as the best heuristic, providing the best average, *GAP*, and *#1* values for all instances but the largest ones, for which it is outperformed by BFR, STF, CI, and SRCI in terms of average solution. EDF is the heuristic providing the best found result for all instance classes, however, the most accurate one is LGF, as it can be concluded from its value for *#1*. Regarding flowtime, EFT is the most efficient heuristic for all sets of instances except the largest one, for which CI is the best one.

We proceed now to analyze the performance of the heuristics according to the number of deadline violations in the solutions. This is an important issue because, as it was previously discussed, the broker will need to book on-demand resources in order to satisfy these violated requests. The broker will pay for these on-demand VMs more than the price charged to the customer, implying money loss. Table 5 shows the results obtained for all the considered heuristics, according

Table 4. Makespan and flowtime results for the proposed heuristics

n	metric	makespan								flowtime							
		BFR	**EFT**	**LGF**	**STF**	**EDF**	**CI**	**SRCI**	**MaxP**	**BFR**	**EFT**	**LGF**	**STF**	**EDF**	**CI**	**SRCI**	**MaxP**
50	*avg.*	99.3	71.12	**67.1**	94.1	69.4	94.0	80.9	92.6	1481.5	**1157.8**	1320.0	1462.2	1324.8	1660.8	1248.7	1619.4
	best	55.3	32.6	32.1	67.1	**30.2**	57.1	38.7	56.2	936.8	681.9	**673.9**	997.1	681.7	1105.3	718.6	1060.9
	GAP	77.4%	12.4%	**4.9%**	66.7%	7.7%	66.0%	36.1%	63.6%	39.1%	**2.2%**	14.4%	36.2%	14.8%	55.2%	12.5%	51.1%
	#1	4	16	**50**	6	42	6	7	2	14	**68**	9	1	0	0	9	0
100	*avg.*	107.1	87.1	**84.5**	106.9	86.5	105.3	102.0	103.0	3116.5	**2846.3**	3212.5	3184.4	3283.5	3287.5	2958.2	3371.8
	best	90.2	46.2	44.9	88.0	**42.5**	90.2	63.6	91.3	2531.1	**1705.0**	1846.4	2573.1	1862.7	2808.5	1919.1	2948.8
	GAP	46.0%	10.0%	**6.0%**	46.3%	8.6%	43.9%	35.5%	40.7%	22.0%	**6.6%**	20.4%	24.5%	22.5%	29.8%	13.0%	32.8%
	#1	4	16	**32**	3	26	7	5	8	15	**57**	0	4	0	18	3	3
200	*avg.*	101.3	91.2	**86.0**	103.2	87.6	102.6	101.9	102.1	5400.9	**5363.8**	6322.1	5525.4	6341.6	5758.9	5425.0	5544.8
	best	91.1	52.0	45.3	93.9	**43.1**	91.3	71.7	90.8	4694.0	**3197.7**	3984.8	4726.4	3928.0	4497.7	3738.0	4297.0
	GAP.	30.8%	11.1%	**3.8%**	33.4%	5.7%	32.1%	29.5%	31.2%	19.9%	**15.2%**	35.9%	22.5%	36.2%	28.6%	18.5%	23.9%
	#1	15	8	**31**	8	23	9	2	5	12	**43**	0	6	0	8	2	29
400	*avg.*	106.5	109.7	109.0	**106.2**	110.7	107.2	109.1	110.5	10210.5	12602.8	13107.4	10604.4	13292.8	**9724.0**	11632.1	9806.9
	best	96.5	95.7	96.5	95.7	**92.7**	95.5	96.9	98.0	8456.2	9847.1	7513.3	8521.1	7546.6	6955.9	8963.9	**6947.1**
	GAP	5.5%	8.6%	7.9%	**5.2%**	9.6%	6.1%	8.0%	9.5%	10.2%	35.7%	37.7%	14.3%	39.4%	**3.7%**	25.2%	4.6%
	#1	19	6	8	**23**	16	16	9	4	24	0	2	2	3	**36**	0	33

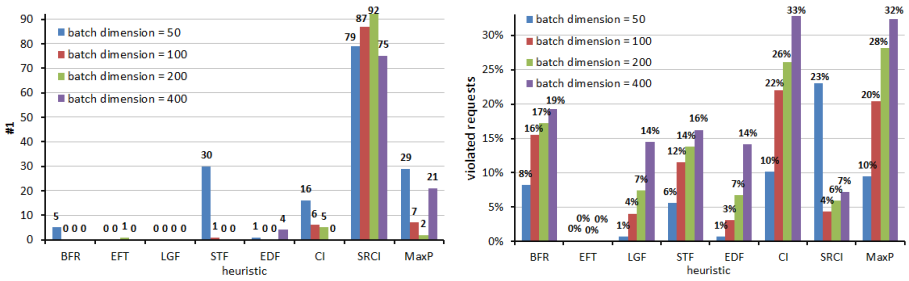
the average number of VM request violated (*avg.*), to the number of times every heuristic was finding the solution with less violated deadlines (*#1*), and the number of times they found solutions without any deadline violation (*zero*). EFT stands out as the best compared scheduler according to these metrics. It is only outperformed by EDF in the number of times a solution without deadline violations is found for the largest instances. For medium and large problem instances, SRCI—the best heuristic regarding the profit objective—has acceptable values of deadline violations.

Table 5. Deadline violations for the proposed heuristics

n	metric	heuristic							
		BFR	**EFT**	**LGF**	**STF**	**EDF**	**CI**	**SRCI**	**MaxP**
50	*avg.*	8.2%	**0.2%**	0.8%	5.6%	0.7%	10.1%	23.0%	9.5%
	#1	17	**91**	71	17	87	8	53	11
	zero	17	**78**	69	17	81	8	52	11
100	*avg.*	15.5%	**0.1%**	4.1%	11.5%	3.1%	22.0%	4.4%	20.4%
	#1	0	**91**	52	0	61	0	17	0
	zero	0	**48**	**48**	0	51	0	15	0
200	*avg.*	17.2%	**0.0%**	7.4%	13.8%	6.7%	26.1%	5.9%	28.2%
	#1	0	**98**	44	0	47	0	7	0
	zero	0	**45**	44	0	**45**	0	7	0
400	*avg.*	19.3%	**0.1%**	14.5%	16.3%	14.1%	32.9%	7.2%	32.4%
	#1	0	**83**	14	0	18	0	1	0
	zero	0	0	14	0	**15**	0	0	0

Figures 1(a) and 1(b) graphically summarizes the main comparative results for the proposed heuristics. In particular, they represent the number of times they are the best heuristic and the percentage of violated requests, respectively. The values in the plots were rounded for the sake of clarity. In the plots, it can be clearly seen how SRCI outperforms all the other heuristics according to profit,

while it also provides accurate results for the number of violated deadlines. In this case, EFT clearly stands out as the best algorithm. At this point, we would like to emphasize that a constraint violation does not imply a decrease on the QoS provided, because the broker will book an on-demand resource to avoid that, and the extra cost for such resource is considered in the total profit value.



(a) Number of times the heuristic is best (b) Number of violated constraints

Figure 1. Comparative results for the proposed heuristics

Summarizing, the experimental analysis indicates that the SRCI heuristic is the most appropriate choice for the broker: it maximizes the profit, clearly outperforming the others, and is among the best according to the number of violated constraints.

6 Conclusions and future work

This article has introduced the VMMP, a relevant problem when planning resource utilization in cloud infrastructures. We focus on a novel cloud brokering business in which the broker sublets reserved resources to his customers in an on-demand basis, and at cheaper prices than those offered by cloud providers. The model is profitable thanks to the large difference in price between on-demand and reserved VMs in the cloud. Customer requests have an associated deadline, and if the broker does not have enough resources to process the requests within their deadlines, he will need to book on-demand VMs in the cloud, with the consequent money loss.

Eight different fast and accurate heuristics were proposed to solve the problem. They focus on different aspects, such as VM cost or QoS. The schedulers are deeply analyzed and evaluated over a large set of (publicly available) benchmark problem instances that account for realistic workloads and scenarios using real data from cloud providers. The main conclusion of the experimental analysis is that the *Shortest Request to Cheapest Instance (SRCI)* heuristic provides the best results, clearly outperforming all the other ones accounting for the broker's profit, and being among the best ones in terms of QoS of the provided solutions.

The main lines for future work include to further studying the VMMP and designing more accurate heuristics and multi-objective metaheuristics to search for trade-off solutions to the problem. Finally, the development of prediction techniques to efficiently manage the amount of reserved resources to cope with the expected demand at minimum cost is a further issue to be considered.

Acknowledgements

B. Dorronsoro is supported by the NRF, Luxembourg, AFR contract no 4017742.

References

1. Buyya, R., Broberg, J., Goscinski, A.: Cloud Computing: Principles and Paradigms. Wiley (2011)
2. Buyya, R., Ranjan, R., Calheiros, R.: Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In: Int. Conf. on Algorithms and Architectures for Parallel Processing. pp. 13–31 (2010)
3. Calheiros, R., Buyya, R.: Cost-effective provisioning and scheduling of deadline-constrained applications in hybrid clouds. In: Proc. of 13^{th} Int. Conf. on Web Information System Engineering. pp. 28–30 (2012)
4. Foster, I., Zhao, Y., Lu, S.: Cloud computing and grid computing 360-degree compared. In: Grid Computing Environments Workshop. pp. 1–10 (2008)
5. Grozev, N., Buyya, R.: Inter-cloud architectures and application brokering: Taxonomy and survey. Software: Practice and Experience pp. 1–22, online first (2012)
6. Ibarra, O., Kim, C.: Heuristic algorithms for scheduling independent tasks on non-identical processors. Journal of the ACM 24(2), 280–289 (1977)
7. Kwok, Y., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys 31(4), 406–471 (1999)
8. Lampe, U.: Optimizing the distribution of software services in infrastructure clouds. In: IEEE World Congress on Services. pp. 69–72 (2011)
9. Leung, J., Kelly, L., Anderson, J.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press (2004)
10. Mattess, M., Vecchiola, C., Garg, S., Buyya, R.: Cloud Computing: Methodology, Systems, and Applications, chap. Cloud Bursting: Managing Peak Loads by Leasing Public Cloud Services, pp. 343–368. CRC Press (2011)
11. Nesmachnow, S., Cancela, H., Alba, E.: Heterogeneous computing scheduling with evolutionary algorithms. Soft Computing 15(4), 685–701 (2010)
12. Nesmachnow, S., Cancela, H., Alba, E.: A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. Applied Soft Computing 12(2), 626–639 (2012)
13. Rimal, B.P., Choi, E., Lumb, I.: A taxonomy and survey of cloud computing systems. In: 5^{th} Int. Joint Conf. on INC, IMS and IDC. pp. 44–51 (2009)
14. Rogers, O., Cliff, D.: A financial brokerage model for cloud computing. Journal of Cloud Computing: Advances, Systems and Applications 1(2), 1–12 (2012)
15. Tang, C., Steinder, M., Spreitzer, M., Pacifici, G.: A scalable application placement controller for enterprise data centers. In: World Wide Web Conf. pp. 331–340 (2007)
16. Tordsson, J., Montero, R.S., Moreno-Vozmediano, R., Llorente, I.M.: Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Future Generation Computer Systems 28(2) (2012)
17. Wu, F., Zhang, L., Huberman, B.: Truth-telling reservations. Algorithmica 52(1), 65–79 (2008)