Elastic Provisioning of Virtual Machines for Container Deployment

Matteo Nardelli
University of Rome Tor Vergata
Rome, Italy
nardelli@ing.uniroma2.it

Christoph Hochreiner
Distributed Systems Group
TU Wien, Vienna, Austria
c.hochreiner@infosys.tuwien.ac.at

Stefan Schulte
Distributed Systems Group
TU Wien, Vienna, Austria
s.schulte@infosys.tuwien.ac.at

ABSTRACT

Docker containers enable to package an application together with all its dependencies and easily run it in any environment. Thanks to their ease of use and portability, containers are gaining an increasing interest and promise to change the way how Cloud platforms are designed and managed. For their execution in the Cloud, we need to solve the container deployment problem, which deals with the identification of an elastic set of computing machines that can host and execute those containers, while considering the diversity of their requirements.

In this paper, we provide a general formulation of the Elastic provisioning of Virtual machines for Container Deployment (for short, EVCD) as an Integer Linear Programming problem, which takes explicitly into account the heterogeneity of container requirements and virtual machine resources. Besides optimizing multiple QoS metrics, EVCD can reallocate containers at runtime, when a QoS improvement can be achieved. Using the proposed formulation as benchmark, we evaluate two well-known heuristics, i.e., greedy first-fit and round-robin, that are usually adopted for solving the container deployment problem.

Keywords

Container, Cloud computing, Resource allocation, QoS

1. INTRODUCTION

Software containers enable to package an application together with all of its dependencies and then run it smoothly in any environment. Moreover, by exploiting operating system level virtualization, multiple containers can co-exist and run in isolation on the same machine, thus improving resource utilization. Differently from virtual machines that exploit hardware virtualization, containers are lightweight, because they only bundle the application dependencies while reusing the underlying operating system. As such, containers introduce a small overhead during application execution [5, 6]. Software containers became popular after the launch of the Docker[1] open source project in 2013, which boosted their adoption. Indeed, Docker augments the benefits of containers with an easy to use management API and a packaging format, enabling rapid adoption among developers as well as users [13].

For the execution, a container needs to be deployed on a hosting machine, which provides computing and memory resources. While doing this, we still want to exploit the Cloud computing principles, which promote the elastic usage of on-demand resources. This problem is known as the *container deployment problem* (or container allocation problem). As discussed in [3, 15], despite the great interest in this technology, the container deployment problem represents a topic only partially explored. Indeed, if the deployment of a single container can be done easily, deploying lots of them, belonging to multiple applications with different requirements, can be complicated and might lead to resource shortage or resource under-utilization. In the literature only few allocation solutions consider containers features while determining their allocation (e.g., [4, 11, 16, 20]). Moreover, most of them are heuristics, characterized by different assumptions and optimization goals, which makes them difficult to analyze and compare. Aside from the work presented in [11], there is no general formulation of the container deployment problem in the Cloud.

In this paper, we propose Elastic provisioning of Virtual machines for Containers Deployment (for short, *EVCD*), a general formulation of the container deployment problem for Cloud environments, which takes into account the heterogeneity of containers requirements and computing resources. EVCD determines the container deployment on virtual machines, which can be acquired and released on-demand, while optimizing Quality of Service (QoS) metrics. At runtime, EVCD evaluates the container deployment and, if an improvement of the optimized QoS metrics can be achieved, plans a reconfiguration. Moreover, EVCD provides a benchmark against which other container deployment heuristics can be compared.

The remainder of this paper is organized as follows. In Section 2 we review related work, in Section 3 we describe the system model and the problem under investigation, and in Section 4 we formulate EVCD as an Integer Linear Programming (ILP) problem. Then, in Section 5, we validate the proposed solution and, relying on EVCD as benchmark, we compare two deployment heuristics in a simulated runtime scenario. Finally, we conclude in Section 6.

ICPE '17 Companion, April 22-26, 2017, L'Aquila, Italy
© 2017 ACM. ISBN 978-1-4503-4899-7/17/04...$15.00
DOI: http://dx.doi.org/10.1145/3053600.3053602

[1] https://www.docker.com/

2. RELATED WORK

The advent of Docker containers [13] has created an increasing interest among users and companies, because of their ability to accelerate the development and simplify the deployment of applications. Nowadays several companies extensively rely on this technology both for internal usage and for providing their services (e.g., [2, 7, 14, 17]). Although containers can run on physical machines, their execution on virtual machines enables to exploit the elastic usage of on-demand resources, promoted by the Cloud computing principles. As surveyed by Peinl et al. [15], a huge ecosystem of tools revolves around Docker containers, which help to monitor, manage, orchestrate, and allocate containers, albeit often on statically defined clusters or on private Clouds. To date, most Cloud service providers have extended their offerings to provide containers as a service (CaaS), usually by automating the container deployment on the offered IaaS. Examples are Amazon ECS[2], Azure Container Service[3], Google Container Engine[4], and IBM Bluemix[5]. As discussed in [3, 15], despite the great interest in this technology, there are still many challenges to be solved, and in this paper we focus on the container deployment problem.

In literature, only few works specifically deal with the allocation problem for containers and, due to NP-hardness of the problem, most of them proposes heuristics. In [1] the authors propose a constraint programming model that, differently from our approach, finds a feasible (but not optimal) deployment solution. The work most closely related to ours has been presented by Hoenisch et al. [11]. Their solution accounts for container replication as well as their elastic allocation, while considering several QoS attributes. We postpone to future work the extension of EVCD for considering the container replication problem.

The existing heuristics aim at optimizing a diversity of utility functions, like fairness, load balancing, network traffic, or energy consumption. The fairness of resource allocation is considered in [8, 18]. In [8], Ghodsi et al. propose the Dominant Resource Fairness (DRF) policy, which works in a system containing different resource types (i.e., CPU, memory) and assigns them to containers, pursuing a Pareto-optimal configuration. DRF is nowadays the default allocation policy of Mesos [9]. Wang et al. [18] further generalize the notion of DRF to work with multiple heterogeneous servers. Considering a topology of communicating containers, Zhao et al. [20] propose a policy that minimizes the traffic exchanged using the network, while balancing the load among virtual machines. The minimization of energy consumption is considered in [4, 16]; these works propose a greedy placement scheme that deploys containers on the most energy efficient machines first. All these works focus on system-oriented metrics, whereas we consider user-oriented metrics, such as deployment time and cost. However, EVCD provides a general framework for solving the deployment problem that can be easily extended to incorporate also these system-oriented metrics.

Proprietary solutions that support container allocation (e.g., Amazon ECS, Borg [17], Google Container Engine) as well as open-source alternatives (e.g., Kubernetes[6], Mesos [9], Marathon[7], Docker Swarm[8], Openstack[9]) usually include simple placement heuristics and enable the utilization of custom policies. Among the most common heuristics, we can find the round-robin policy, which tries to evenly use resources, and the well-known heuristics that solve the bin packing problem, namely greedy best-fit and first-fit. These approaches usually do not consider the specific characteristics of containers (e.g., the reusability of containers images) and therefore can lead to a sub-optimal utilization of the available resources. It worth mentioning the approach used by Kubernetes: first, containers are grouped in user-defined *pods*, and then pods are deployed on the best candidate node. The latter is determined by applying a set of filters and priority functions that can be also defined by the user.

3. SYSTEM MODEL AND PROBLEM STATEMENT

Devising an optimal container deployment strongly depends on the assumptions made about the domain it will be applied to. In this section, we provide a formal description of the domain entities: containers and virtual machines.

3.1 Software Container Model

A software container wraps up a piece of software together with everything it needs for the execution (i.e., runtime, libraries, code), and enables to easily run it on any machine, whether bare metal or virtual machine. Container technology exploits operating system level virtualization to realize flexibility and portability of software [13]. Following the Docker model, a container is an instance of a container image (or simply *image*), which represents a container snapshot and contains all the data needed for its execution. To improve reusability and efficiently use memory, an image is structured as a series of layers (e.g., base Linux image, libraries, custom files), where each one can be downloaded or updated independently from the others. When a software container has to be instantiated on a host, the host needs to download each layer from an external repository, before starting the container.

We define the set of containers as S. A container $s \in S$ is characterized by the following QoS attributes: C_s, the amount of required CPU shares on the hosting machine; D_s^{sc}, the startup time of s; M_s, the amount of memory (RAM) required for its execution; and I_s, the set of image layers needed for its instantiation. Observe that, for Docker, the CPU share represents the container relative weight with respect to all other containers running on the same host while assigning the CPU cycles. To simplify the assignment problem, we consider C_s as the number of CPU required by s. Moreover, we assume $I_s \subseteq I$, where I is the set of all container image layers that can be instantiated. Each image layer $i \in I$ is characterized by the size l_i of data composing the layer.

3.2 Virtual Machine Model

Computing resources accommodate and execute containers with respect to their capabilities. We consider these re-

[2]https://aws.amazon.com/ecs/
[3]https://azure.microsoft.com/services/container-service/
[4]https://cloud.google.com/
[5]https://www.ibm.com/cloud-computing/bluemix/
[6]http://kubernetes.io/
[7]https://mesosphere.github.io/marathon/resources.html
[8]https://www.docker.com/products/docker-swarm
[9]http://www.openstack.org/

sources to be virtual machines belonging to private, public, and hybrid Clouds. Furthermore, we assume that all the virtual machines reside in Internet and can exchange data with no (practical) bandwidth limitations. Following the principle of Cloud computing, a virtual machine can be acquired and released as needed. In practice, a virtual machine can be leased for a minimum period of time, known as Billing Time Unit (BTU); the overall cost of a virtual machine depends on the number of, albeit partially, BTUs consumed. Moreover, although in theory unlimited, we assume that the number of virtual machines which can be leased in a certain time period is limited.

Let V be the set of all virtual machines, including the active (leased) ones and those turned off but leasable. A virtual machine $v \in V$ has the following QoS attributes: C_v, the amount of available computing resources, i.e., CPUs; DR_v, its download data rate; M_v, the available memory capacity; P_v, its cost per BTU; and I_v, with $I_v \subseteq I$, the set of image layers already available in v without the need of downloading them from an external repository. Note that the proposed model can easily include physical machines, which represent, from a modeling point of view, a special case of v, where the meaning of P_v is defined accordingly.

3.3 Container Deployment Problem

The container deployment problem requires to determine a suitable mapping between the set of containers S and the set of virtual machines V in a way that all constraints are fulfilled. We investigate the initial deployment as well as its adaptation at runtime, therefore we solve EVCD periodically, every τ unit of time. Due to the generality of the proposed model, we need also to consider that a container cannot be usually placed on every machine in V, because of security or management motivations (e.g., the container requires a specific resource, such as a sensing device, or the virtual machine belongs to a private Cloud). This observation allows us to consider for each container $s \in S$ a subset of candidate virtual machines $V^s \subseteq V$ where it can be deployed.

We can conveniently model the container deployment with binary variables $x_{s,v}$, $s \in S$, $v \in V^s$: $x_{s,v} = 1$ if container s is deployed on virtual machine v and $x_{s,v} = 0$ otherwise. A correct instantiation must deploy a container on one and only one virtual machine; this condition can be guaranteed requiring that $\sum_v x_{s,v} = 1$, with $v \in V^s$, $s \in S$. We also consider the variables z_v, $v \in V$, which denote whether the virtual machine is active and hosts at least one container. By definition, we have $z_v = \vee_{s\in S, v\in V} x_{s,v}$. For short, in the following we denote by $\boldsymbol{x}$ and $\boldsymbol{z}$ the deployment vector for containers and the activation vector for virtual machines, respectively, where $\boldsymbol{x} = \langle x_{s,v} \rangle$, $\forall s \in S$, $\forall v \in V^s$ and $\boldsymbol{z} = \langle z_v \rangle$, $\forall v \in V$.

Since we solve EVCD periodically, it is convenient to define a set of binary variables that keep track of the previous deployment configuration, i.e., the one determined at time $t - \tau$. We define $x^{\tau}_{s,v}$, which indicate whether $s \in S$ was deployed on $v \in V^s$ in $t-\tau$, and z^{τ}_v, which indicate whether $v \in V$ was active in $t - \tau$. Leveraging on the previous configuration, we also define some auxiliary variables, namely a_v, $a_{s,v}$, and $\delta_{s,v}$. We use the binary variables a_v to indicate whether the virtual machine $v \in V$, turned off in $t - \tau$, has to be activated in t. To indicate that a container has to be deployed on a newly activated virtual machine v, i.e., with $a_v = 1$, we use the binary variables $a_{s,v}$. The binary variables $\delta_{s,v}$ indicate whether a container $s \in S$ has a different allocation with respect to the configuration defined in $t - \tau$: $\delta_{s,v} = 1$, if in $t-\tau$ the container s was either not allocated or allocated on $u \in V^s$, with $u \neq v$. Observe that $x^{\tau}_{s,v}$ and $\delta_{s,v}$ enable to model the runtime re-allocation of containers on virtual machines and can be used to model migration protocols that, e.g., relocate the container in-memory state [10].

4. ELASTIC PROVISIONING MODEL

In this section, exploiting tools provided by optimization theory, we propose a model for the EVCD problem that can be adjusted to satisfy different optimization functions. Since the latter depend on non-functional attributes, we first derive the expression for the different QoS metrics of interest and then present the EVCD formulation.

4.1 QoS Metrics

The considered QoS metrics are the deployment time of containers and the cost of virtual machines leased for their execution.

Deployment Time.

We define the deployment time of containers $D(\cdot)$ as the time needed to deploy every container in S. This term accounts for the time needed to spawn new virtual machines, retrieve container images, and finally start the containers. Given a placement vector $\boldsymbol{x}$ (and resulting $\boldsymbol{z}$), we have:

$$D(\boldsymbol{x}, \boldsymbol{z}) = \sum_{s\in S} \sum_{v\in V} D_{s,v}(\boldsymbol{x}, \boldsymbol{z}) \tag{1}$$

where $D_{s,v}(\boldsymbol{x}, \boldsymbol{z})$ denotes the time needed to deploy the container s on the virtual machine v and it is defined as:

$$D_{s,v}(\boldsymbol{x}, \boldsymbol{z}) = D^{sv}_v a_{s,v} + \sum_{i\in I_s \setminus I_v} \frac{l_i}{DR_v} x_{s,v} + D^{sc}_s \delta_{s,v} \tag{2}$$

where D^{sv}_v is the time needed to start a new virtual machine, considered only if a new virtual machine is needed, $\sum_i \frac{l_i}{DR_v}$ represents the time needed to download the container image layers not yet on the virtual machine v, and D^{sc}_s is the startup time of s, if s was not already running on v.

Cost.

We define the deployment cost $C(\cdot)$ as the monetary cost needed to instantiate and execute all the containers. This term considers that leasing a virtual machine v for a BTU imposes a cost of P_v and that renewing a leasing when the BTU ends exposes a new cost of P_v. Relying on the activation vector of virtual machines $\boldsymbol{z}$ and on the auxiliary variables a_v, $v \in V$, which indicate whether the virtual machine v, not previously active, has to be activated, we have:

$$C(\boldsymbol{z}) = C^n(\boldsymbol{z}) + C^r(\boldsymbol{z}) \tag{3}$$

where

$$C^n(\boldsymbol{z}) = \sum_{v\in V} P_v a_v \tag{4}$$

$$C^r(\boldsymbol{z}) = \sum_{v\in V^{exp}} P_v z_v \tag{5}$$

denote respectively the cost of the newly acquired virtual machines and the cost of the virtual machines in V^{exp} that have to be renewed. The set $V^{exp} \subseteq V$ includes the virtual machines whose leasing is going to expire between the current time t and the next execution of EVCD in $t + \tau$.

4.2 EVCD Formulation

We formulate EVCD as an ILP model which is solved periodically, every τ unit of time. In each round of execution, EVCD determines an optimal mapping between the set of containers and the available resources, i.e., virtual machines. While determining a suitable mapping, EVCD considers an objective function that, depending on the utilization scenario, could be aimed to optimize specific QoS attributes. These different optimization goals could be possibly conflicting, thus leading to a multi-objective optimization problem, which can be transformed into a single objective problem using the Simple Additive Weighting (SAW) technique [19]. According to SAW, we define the objective function $F(\boldsymbol{x}, \boldsymbol{z})$ as a weighted sum of the normalized QoS metrics (i.e., deployment time and cost) to be minimized, as follows:

$$F(\boldsymbol{x}, \boldsymbol{z}) = w_d \frac{D(\boldsymbol{x}, \boldsymbol{z}) - D_{\min}}{D_{\max} - D_{\min}} + w_c \frac{C(\boldsymbol{z}) - C_{\min}}{C_{\max} - C_{\min}} \qquad (6)$$

where w_d, $w_c \geq 0$, with $w_d + w_c = 1$, weigh the different QoS attributes, and $D_{\max}$ ($D_{\min}$) and $C_{\max}$ ($C_{\min}$) denote, respectively, the maximum (minimum) value for the overall expected deployment time and cost. Observe that after normalization, each metric ranges in the interval $[0, 1]$, where the value 0 corresponds to the best possible case and 1 to the worst case.

The EVCD problem can be formulated as follows:

$$\min_{\boldsymbol{x}, \boldsymbol{z}} F(\boldsymbol{x}, \boldsymbol{z})$$

subject to:

$$\sum_{v \in V} x_{s,v} = 1 \qquad \forall s \in S \qquad (7)$$

$$\sum_{s \in S} C_s x_{s,v} \leq C_v \qquad \forall v \in V \qquad (8)$$

$$\sum_{s \in S} M_s x_{s,v} \leq M_v \qquad \forall v \in V \qquad (9)$$

$$a_v \geq z_v - z_v^{\tau} \qquad \forall v \in V \qquad (10)$$

$$a_v \leq z_v \qquad \forall v \in V \qquad (11)$$

$$a_{s,v} \geq a_v + x_{s,v} - 1 \qquad \forall s \in S, \forall v \in V^s \qquad (12)$$

$$a_{s,v} \leq a_v + x_{s,v} \qquad \forall s \in S, \forall v \in V^s \qquad (13)$$

$$\delta_{s,v} \geq x_{s,v} - x_{s,v}^{\tau} \qquad \forall s \in S, \forall v \in V^s \qquad (14)$$

$$\delta_{s,v} \leq \frac{(1 - x_{s,v}^{\tau}) + x_{s,v}}{2} \qquad \forall s \in S, \forall v \in V^s \qquad (15)$$

$$z_v \geq \frac{\sum_{s \in S} x_{s,v}}{M} \qquad \forall v \in V \qquad (16)$$

$$z_v \leq \sum_{s \in S} x_{s,v} \qquad \forall v \in V \qquad (17)$$

$$a_v \in \{0, 1\} \qquad \forall v \in V \qquad (18)$$

$$a_{s,v} \in \{0, 1\} \qquad \forall s \in S, \forall v \in V^s \qquad (19)$$

$$\delta_{s,v} \in \{0, 1\} \qquad \forall s \in S, \forall v \in V^s \qquad (20)$$

$$x_{s,v} \in \{0, 1\} \qquad \forall s \in S, \forall v \in V^s \qquad (21)$$

$$z_v \in \{0, 1\} \qquad \forall v \in V \qquad (22)$$

In the problem formulation, Equation (7) guarantees that each container $s \in S$ is placed on one and only one virtual machine $v \in V^s$. Constraints (8) and (9) limit the placement of containers on a virtual machine $v \in V$ according to its available resources. Equations (10) and (11) model whether a virtual machine needs to be activated to host at least a container. The auxiliary variables $a_{s,v}$ are defined by Equations (12) and (13), which indicate whether a container has to be placed on a newly instantiated virtual machine. Equations (14) and (15) model whether a container s has a different allocation with respect to the configuration defined in $x_{s,v}^{\tau}$. Finally, constraints (16)–(17) are the activation constraints for the variables z_u, with M large constants.

Table 1: Parameters of the experimental setup

Container

Parameter	Value	Parameter	Value
C_s	1	M_s	1 MB
D_s^{sc}	$\mathcal{U}(8.5, 11.5)$ s	$\lvert I_s \rvert$	$\mathcal{U}(2, 4)$
l_i	$\mathcal{U}(200, 800)$ MB	L	30 mins

Virtual Machine

Parameter	Value	Parameter	Value
C_v	4	M_v	16 MB
D_v^{sv}	$\mathcal{U}(85, 115)$ s	DR_v	1 Gbps
P_v	1	I_v^{max}	10
BTU	60 mins		

5. EXPERIMENTAL RESULTS

To evaluate EVCD and other deployment heuristics, we simulate their execution in a system that receives containers and allocates them on virtual machines. We first present the experimental setup in Section 5.1 and then, in Section 5.2, we use EVCD as baseline against which two deployment heuristics are compared.

5.1 Experimental Setup

The EVCD model allows to elastically acquire and release virtual machines to host and execute software containers, while optimizing different QoS attributes. We consider as QoS metrics the deployment time of containers, defined in Equations (2), and cost of leased virtual machines, defined in (3). We solve the ILP problem using CPLEX© (version 12.6.3) on a machine with 4 CPUs and 16 GB RAM.

In the experiments, we simulate the execution of EVCD every $\tau = 15$ units of time (i.e., minutes) and we assume that EVCD manages at time t the requests for allocation by containers, received between $t - \tau$ and t. Each container requires the same amount of resources (i.e., CPU, memory) and has a lifespan of $L = 30$ minutes. A container depends on a set of images, whose cardinality is uniformly chosen between 2 and 4; this set includes 70% of existing images (if any) and 30% of new images. Each image has a size l_i uniformly defined in $[200, 800]$ MB. To have a fair comparison with the heuristics, introduced later, we consider homogeneous virtual machines, i.e., every virtual machine $v \in V$ is characterized by the same QoS attributes (i.e., CPU, memory, cost). Similarly to the most popular commercial solutions, we define a BTU of 60 minutes. We define the startup time D_v^{sv} of a virtual machine according to the results presented in [12]. Finally, we assume that a virtual machine can cache only a limited number of image layers, thus each virtual machine preserves only the I_v^{max} most recently used ones. Table 1 summarizes the parameters used during the experiments. As regards the objective function of EVCD, we

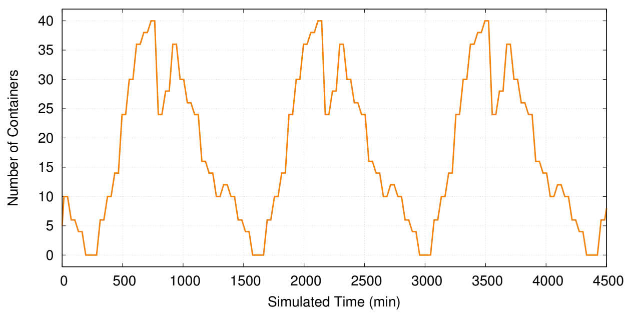


Figure 1: Number of containers during the experiment

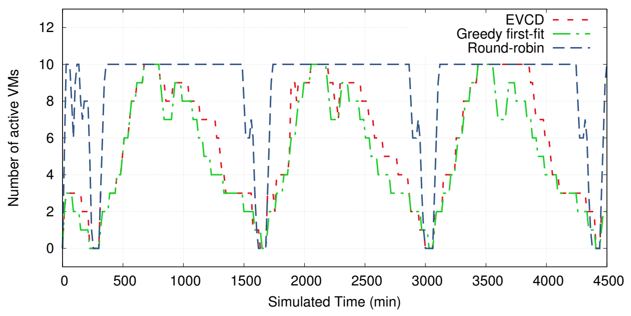


Figure 2: Active virtual machines

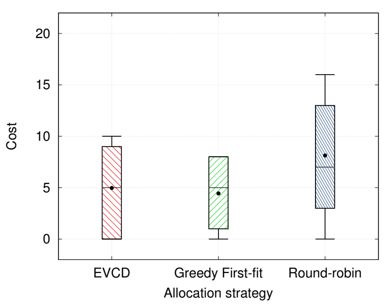
are interested in a contextual minimization of the QoS metrics, therefore we define $w_d = w_c = 1$, $D_{\max} = C_{\max} = 1$, and $D_{\min} = C_{\min} = 0$. During the experiment, the number of containers that EVCD has to deploy fluctuates between 0 and 20 during the timespan of a (simulated) day, and this pattern is repeated for the following two days. Figure 1 shows the number of active containers during the whole experiment. Note that, since a container has a lifespan greater than the timespan between two consecutive executions of EVCD, i.e., $L \geq \tau$, the number of active container can be greater than 20.

To validate the proposed solution and use it as a benchmark tool, we have developed two approaches inspired by the well-known meta-heuristics: first-fit and round-robin. The first heuristic, *greedy first-fit*, can elastically acquire and release virtual machines. As soon as a container asks for resources, this strategy allocates it on the first active virtual machine that has enough available resources; if no suitable virtual machine can be found, a new one is leased from the Cloud resource provider. Observe that, if the resource provider proposes machines with different QoS attributes, differently from EVCD, this heuristic cannot easily detect the most suitable one for deploying the managed containers. The second heuristic, *round-robin*, works on a fixed pool of virtual machines and assigns containers to each of them in equal portions and in circular order. We define a pool of 10 virtual machines; this number, which results from preliminary experiments, guarantees enough resource availability for handling the incoming load.

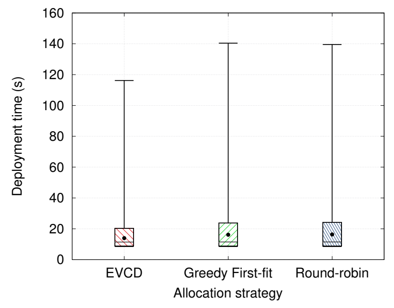
5.2 Comparison Between Allocation Strategies

We evaluate how different strategies can accommodate a varying incoming load of containers that require computing resources. Using EVCD as benchmark, we evaluate the deployment strategies pursued by the greedy first-fit and round-robin heuristics. Figure 2 presents how the allocation strategies acquire and release virtual machines to satisfy the incoming load. The round-robin heuristic cannot perform an elastic provisioning of virtual machines and uses all the ones available in the statically defined pool. We can observe that the number of active virtual machines decreases only when the number of containers that have to be executed is lower than 10 (see Figure 1). This happens because the round-robin heuristic evenly places container on every virtual machine within the pool of resource. Conversely, EVCD and greedy first-fit use a new virtual machine only when the active ones have not enough resources to host a new container. As such, these approaches can dynamically acquire and release virtual machines, following the incoming load fluctuations. EVCD and greedy first-fit have a similar behavior, albeit the former seems to be slightly more conservative in releasing virtual machines. Indeed, EVCD tries to consolidate less the containers and prefers to use virtual machines which already include the needed container images. The performance of these allocation strategies, expressed in terms of QoS metrics, are summarized in Figure 3 by leveraging on a boxplot, which represents their distribution through the minimum value, the 5th percentile, 50th percentile, 95th percentile, and the maximum value; the average value is also represented using a full dot. The elastic utilization of virtual machines by EVCD and greedy first-fit leads to a reduction of the execution costs (see Figure 3a); on average these strategies reduce respectively the cost of about 40% and 45% with respect to round-robin. Figures 3b and 3c show the deployment time per container. We can observe that only a very limited number of containers experience high deployment times, i.e., between 118 s and 140 s, due to the startup time of new virtual machines. The 95th percentile of the deployment time for all the allocation strategies is always below 25 s, therefore Figure 3c focuses only on a limited range of the deployment time. EVCD determines the allocation of resources which produces the fastest deployment time for software containers, whereas the first-fit heuristic behaves similarly to the round-robin one. This happens because EVCD allocates containers minimizing the time needed to retrieve their images from an external repository, if not locally available, as defined in Equation (2). On the contrary, the other heuristics neglect this information. On average, the deployment times per container obtained by greedy first-fit and round-robin are, respectively, 16% and 17% higher than the optimal one by EVCD.

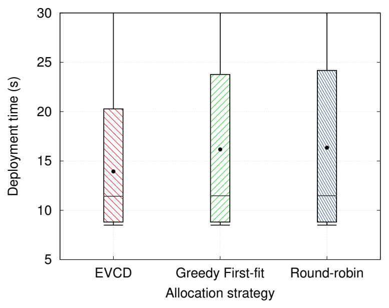
EVCD elastically acquires and releases virtual machines for determining the optimal deployment of containers, which jointly minimizes the deployment time and cost. In the presented experiment, EVCD has shown that the round-robin heuristic produces the highest deployment time and, working on a static pool of resources, is the most expensive solution for running containers. Greedy first-fit considerably reduces costs, which are on average 11% lower than the optimal value, but, by neglecting the features of containers, it produces on average a deployment time 40% higher than the optimal one. Since these heuristics are not aware of the features of containers and virtual machines, they cannot determine an allocation that optimizes their QoS attributes.



(a) Cost of leased virtual machines



(b) Deployment time per container



(c) Detail on the deployment time per container: timespan between 5 and 30 seconds

Figure 3: Comparison of EVCD against baseline heuristics: impact on QoS metrics

6. CONCLUSION

In this paper we have presented a formulation of the elastic provisioning of virtual machines for container deployment. EVCD is a general and flexible formulation that can be conveniently configured to optimize different QoS metrics, and we have considered the deployment time and cost for executing containers in the Cloud. Besides computing the container allocation, EVCD can be used as a benchmark framework against which to compare other allocation strategies. Therefore, we have evaluated and highlighted the drawbacks of two well-known heuristics that are commonly used for defining the container deployment.

As future work, we plan to extend the proposed formulation of EVCD to model other QoS attributes (e.g., availability, network traffic, privacy), network communication among containers, and their runtime replication. Moreover, we plan to develop efficient heuristics to deal with large instances of the container deployment problem.

7. REFERENCES

[1] M. Abdelbaky, J. Diaz-Montes, M. Parashar, et al. Docker containers across multiple clouds and data centers. In *Proc. of IEEE/ACM UCC 2015*, pages 368–371, 2015.

[2] B. Burns, B. Grant, D. Oppenheimer, et al. Borg, omega, and kubernetes. *Commun. ACM*, 59(5):50–57, 2016.

[3] E. Casalicchio. Autonomic orchestration of containers: Problem definition and research challenges. In *Proc. of InfQ '16 (in conj. with VALUETOOLS '16)*, 2016.

[4] Z. Dong, W. Zhuang, and R. Rojas-Cessa. Energy-aware scheduling schemes for cloud data centers on google trace data. In *Proc. of IEEE OnlineGreenComm 2014*, pages 1–6, 2014.

[5] R. Dua, A. R. Raja, and D. Kakadia. Virtualization vs containerization to support paas. In *Proc. of IEEE IC2E 2014*, pages 610–614, 2014.

[6] W. Felter, A. Ferreira, R. Rajamony, et al. An updated performance comparison of virtual machines and linux containers. In *Proc. of IEEE ISPASS 2015*, pages 171–172, 2015.

[7] W. Gerlach, W. Tang, K. Keegan, et al. Skyport: Container-based execution environment management for multi-cloud scientific workflows. In *Proc. of IEEE DataCloud '14*, pages 25–32. IEEE, 2014.

[8] A. Ghodsi, M. Zaharia, B. Hindman, et al. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, volume 11, pages 24–24, 2011.

[9] B. Hindman, A. Konwinski, M. Zaharia, et al. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.

[10] P. Hoenisch, C. Hochreiner, D. Schuller, et al. Cost-efficient scheduling of elastic processes in hybrid clouds. In *Proc. of IEEE CLOUD 2015*, pages 17–24, 2015.

[11] P. Hoenisch, I. Weber, S. Schulte, et al. Four-fold auto-scaling on a contemporary deployment platform using docker containers. In *Proc. of ICSOC 2015*, pages 316–323, 2015.

[12] M. Mao and M. Humphrey. A performance study on the vm startup time in the cloud. In *Proc. of IEEE CLOUD 2012*, pages 423–430, 2012.

[13] D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), 2014.

[14] D.-T. Nguyen, C. H. Yong, X.-Q. Pham, et al. An index scheme for similarity search on cloud computing using mapreduce over docker container. In *Proc. of ACM IMCOM '16*, pages 60:1–60:6. ACM, 2016.

[15] R. Peinl, F. Holzschuher, and F. Pfitzer. Docker cluster management for the cloud - survey results and own solution. *Journal of Grid Computing*, 14(2):265–282, 2016.

[16] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, et al. A framework and algorithm for energy efficient container consolidation in cloud data centers. In *Proc. of IEEE DSDIS 2015*, pages 368–375, 2015.

[17] A. Verma, L. Pedrosa, M. Korupolu, et al. Large-scale cluster management at google with borg. In *Proc. of EuroSys '15*, pages 18:1–18:17. ACM, 2015.

[18] W. Wang, B. Li, and B. Liang. Dominant resource fairness in cloud computing systems with heterogeneous servers. In *Proc. of IEEE INFOCOM 2014*, pages 583–591, 2014.

[19] K. P. Yoon and C.-L. Hwang. *Multiple Attribute Decision Making: an Introduction.* Sage Pubns, 1995.

[20] Z. Zhao, N. Mandagere, G. Alatorre, et al. Toward locality-aware scheduling for containerized cloud services. In *Proc. of IEEE Big Data 2015*, pages 263–270, 2015.