Elastic Allocation of Docker Containers in Cloud Environments

Matteo Nardelli

Department of Civil Engineering and Computer Science Engineering
University of Rome Tor Vergata, Italy
nardelli@ing.uniroma2.it

Abstract Docker containers wrap up a piece of software together with everything it needs for the execution and enable to easily run it on any machine. For their execution in the Cloud, we need to identify an elastic set of virtual machines that can accommodate those containers, while considering the diversity of their requirements. In this paper, we briefly describe our formulation of the Elastic provisioning of Virtual machines for Container Deployment (EVCD), which takes explicitly into account the heterogeneity of container requirements and virtual machine resources. Afterwards, we evaluate the EVCD formulation with the aim of demonstrating its flexibility in optimizing multiple QoS metrics.

Keywords: Container, Cloud computing, Resource allocation, QoS

1 Introduction

Docker[1] containers enable to package an application together with all of its dependencies and then run it smoothly in other environments. They exploit the operating system level virtualization, therefore multiple containers can co-exist and run in isolation on the same machine, thus improving resource utilization. Differently from virtual machines, containers are lightweight [4], because they bundle only the application dependencies while reusing the underlying operative system. 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 named *container deployment problem* (or container allocation problem). 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 solutions consider container features while determining their allocation (e.g., [3,6,9,11]) and the most of them are characterized by different assumptions and optimization goals. Aside from the work presented in [6], there is no general formulation of the container deployment problem in the Cloud.

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

O. Kopp, J. Lenhard, C. Pautasso (Eds.): 9th ZEUS Workshop, ZEUS 2017, Lugano, Switzerland, 13-14 February 2017, published at http://ceur-ws.org/

In this paper, we investigate the problem of deploying containers in the Cloud. Specifically, we evaluate EVCD [8], a general formulation of the container deployment problem that determines the optimal allocation on virtual machines, acquired on-demand, while considering the QoS attributes of containers and virtual machines. Besides computing the initial deployment and the following runtime adaptation, EVCD provides a benchmark against which other deployment algorithms can be compared. With respect to our previous work [8], in this paper we show how EVCD can optimize different user-oriented QoS metrics, such as the deployment time and cost.

This paper is organized as follows. We review related work in Sect. 2; in Sect. 3 we describe the system model and the problem under investigation, before formulating EVCD as an Integer Linear Programming problem in Sect. 4. We evaluate the flexibility of EVCD in Sect. 5 and conclude in Sect. 6.

2 Related Work

In literature a limited number of works provides a formal definition of the allocation problem for containers; however, due to NP-hardness of the problem, many heuristics have been proposed. As regards the modeling, 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. [6]. Their model accounts for container replication as well as their allocation, while considering several QoS attributes. We do not explicitly consider container replication, however we postpone to future work the extension of EVCD for considering it.

The existing heuristics aim at optimizing a diversity of utility functions, namely fairness, load balancing, network traffic, or energy consumption. The fairness of resource allocation is considered in [5,10]. In [5], 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 in a Pareto-optimal manner. Then, Wang et al. [10] further generalize the notion of DRF to work with multiple heterogeneous servers. Considering a topology of communicating containers, Zhao et al. [11] 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 [3,9]; these works propose a greedy placement scheme that allocates containers on the most energy efficient machines first. All these works focus on system-oriented metrics, whereas we consider user-oriented metrics, such as the deployment time and cost. However, EVCD provides a general framework for solving the deployment problem that can be easily extended to incorporate also those kind of metrics. Proprietary solutions that support container allocation (e.g., Amazon ECS[2], Google Container Engine[3]) and open-source alternatives (e.g., Kubernetes[4],

[2] https://aws.amazon.com/ecs/
[3] https://cloud.google.com/
[4] http://kubernetes.io/

Docker Swarm[5]) usually bundle simple scheduling heuristics and also enable the execution of custom policies. Among the most common heuristics, we can find the round-robin policy, which tries to evenly use resources, and well-known heuristics that solve the bin packing problem, namely greedy best-fit and first-fit. Specifically, Docker Swarm, the official Docker component that allocates containers on a centralized pool of resources, includes a bin packing policy and a strategy that balances the number of containers among computing nodes. In our previous work [8], we used EVCD to compare some of these heuristics (i.e., round-robin, greedy first-fit) in terms of achievable QoS performance.

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.

Software Container Model. A Docker container is an instance of an *image*, which represents a container snapshot and contains all the data needed for its execution. For efficiency reasons, an image is structured as a series of layers (e.g., libraries, custom files), where each one can be downloaded independently from the others. We define the set of containers as S. A container $s \in S$ is characterized by the following QoS attributes: C_s, the number of required CPUs; D_s^{sc}, the startup time; M_s, the amount of required memory; and I_s, the set of image layers needed for its instantiation. We assume $I_s \subseteq I$, where I is the set of all the available containers images. Each image layer $i \in I$ is characterized by the size of data l_i composing the layer.

Virtual Machine Model. A virtual machine (VM) hosts and executes containers with respect to its capabilities. Being a Cloud resource, a VM can be acquired and released as needed and paid for the amount of, albeit partially, consumed Billing Time Units (BTU). Finally, although in theory unlimited, we assume the number of leasable virtual machines to be limited in a certain time period. Let V be the set of all VMs, including the active (leased) ones and the leasable ones. A virtual machine $v \in V$ has the following QoS attributes: C_v, the amount of available CPUs; M_v, the available memory capacity; DR_v, the download data rate of v; D_v^{sv}, the startup time; P_v, the cost per BTU; and I_v, with $I_v \subseteq I$, the set of image layers available in v without downloading them from an external repository.

Container Deployment Problem. To solve the deployment problem, we need to determine a mapping between the set of containers S and the set of virtual machines V in a such 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. We model the container deployment with binary variables $x_{s,v}$, $s \in S$, $v \in V$: $x_{s,v} = 1$ if s is deployed on v and $x_{s,v} = 0$ otherwise. The variables z_v denote whether $v \in V$ is active and hosts

[5] https://www.docker.com/products/docker-swarm

at least one container. Relying on the deployment configuration determined at time $t - \tau$, we also define the following auxiliary binary variables: a_v, which indicates whether $v \in V$, turned off in $t - \tau$, has to be activated in t; $a_{s,v}$, which indicates whether s is deployed on a newly activated virtual machine v (i.e., with $a_v = 1$); and $\delta_{s,v}$, which indicates whether, in $t - \tau$, $s \in S$ was not allocated or was allocated on $u \neq v$, with $u \in V$.

4 Elastic Provisioning Model

In this section we present our formulation of EVCD (acronym of Elastic provisioning of Virtual machines for Container Deployment) as an Integer Linear Programming (ILP) model. Due to space limitations, we do not report the full formulation of EVCD, which can be found in [8]. EVCD is solved periodically, every τ unit of time. We model as QoS metrics the deployment time $D(\boldsymbol{x}, \boldsymbol{z})$ and cost $C(\boldsymbol{x}, \boldsymbol{z})$. The former represents the time needed to deploy every container in S, whereas the latter represents the monetary cost of the virtual machines leased for executing the containers. Since the relative importance of these metrics depends on the utilization scenario, EVCD provides a general formulation that can be aimed at optimizing specific QoS attributes. Therefore, the objective function of EVCD is the minimization of $F(\boldsymbol{x}, \boldsymbol{z})$, which is defined as a weighted sum of the normalized QoS attributes:

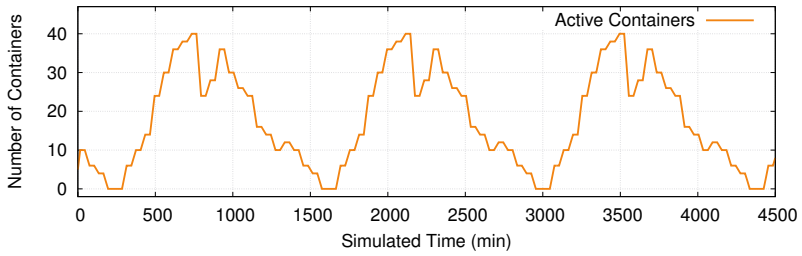
$$F(\boldsymbol{x}, \boldsymbol{z}) = w_d \frac{D(\boldsymbol{x}, \boldsymbol{z}) - D_{\min}}{D_{\max} - D_{\min}} + w_c \frac{C(\boldsymbol{z}) - C_{\min}}{C_{\max} - C_{\min}} \tag{1}$$

where w_d, w_c, with $w_d,\ w_c \geq 0$, $w_d + w_c = 1$, are weights for 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 these normalization factors can be computed by solving other optimization problems or can be approximated relying on previous experiments or on statistical analysis of the runtime execution. The overall deployment time $D(\boldsymbol{x}, \boldsymbol{z})$ accounts for the time needed to spawn new VMs, retrieve container images, and finally start the containers. It can be formally defined as:

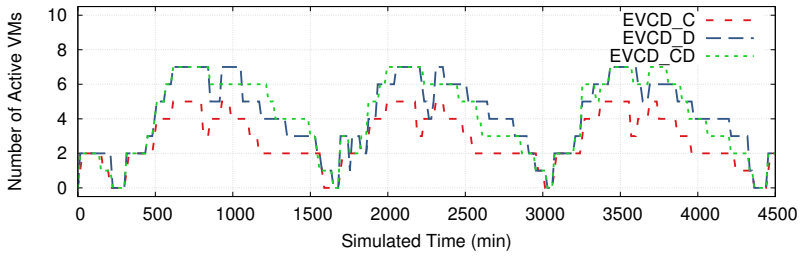
$$D(\boldsymbol{x}, \boldsymbol{z}) = \sum_{s \in S} \sum_{v \in V} \left(D_v^{sv} a_{s,v} + \sum_{i \in I_s \setminus I_v} \frac{l_i}{DR_v} x_{s,v} + D_s^{sc} \delta_{s,v} \right) \tag{2}$$

where D_v^{sv} is the startup time of a new VM, considered only if a new one is needed, $\sum_i \frac{l_i}{DR_v}$ represents the time needed to download the images I_s not yet on v, and D_s^{sc} is the startup time of s, if s was not already running on v. The cost $C(\boldsymbol{x}, \boldsymbol{z})$ includes the leasing of the new VMs and the renewing the expired ones which are still needed. Relying on the activation vector $\boldsymbol{z}$ and on the auxiliary variables a_v, we have that:

$$C(\boldsymbol{z}) = \sum_{v \in V} P_v a_v + \sum_{v \in V^{exp}} P_v z_v \tag{3}$$



(a) Number of active containers



(b) Number of active virtual machines

Figure 1: Elastic provisioning of virtual machines when EVCD receives a varying demand for resource allocation by containers.

where the first term on the right end side accounts for the cost of newly acquired VMs, and the second one accounts for renewing the expired leasings, if needed (i.e., if the related VM is still used). The set $V^{exp} \subseteq V$ includes the VMs whose leasing is going to expire between the current time t and the next execution of EVCD in $t + \tau$.

The full optimization in [8] includes constraints that limit the number of containers deployable on a VM with respect to the available resources as well as the formal definition of the auxiliary variables.

5 Experimental Results

To evaluate the EVCD model, we simulate its execution in a system that receives containers and allocates them on VMs. The aim of this experiment is to show the flexibility of EVCD, which can elastically acquire and release VMs to host and execute software containers while optimizing the deployment time of containers, the cost of leased VMs, or a combination thereof.

We solve EVCD with CPLEX©[6], the state-of-the-art solver for ILP problems, on a machine with 4 CPUs and 16 GB RAM. We simulate the execution of EVCD

[6] http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

Table 1: QoS metrics and VMs acquired under different configurations of EVCD

Average values of the QoS metrics of interest

QoS metric	**EVCD_C**	**EVCD_D**	**EVCD_CD**
Cost of leased VMs per hour	3.8	4.9	4.1
Deployment time per container	24.0 s	13.6 s	14.8 s

Relative number of used virtual machines

VM type	**EVCD_C**	**EVCD_D**	**EVCD_CD**
type A	23.4%	44.8%	36.8%
type B	76.6%	55.2%	63.2%

every $\tau = 15$ minutes. Each container requires unitary resources (i.e., CPU, memory) and has a lifespan of $L = 30$ minutes. A container depends on a set of image layers, 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 layer has a size l_i uniformly defined in $[200, 800]$ MB. The startup time D_s^{sc} of $s \in S$ ranges uniformly in $[8.5, 11.5]$ s. Two type of VMs are available in V: *type A* with 4 CPUs, 16 GB of RAM, and $P_v = 1$; and *type B* with 8 CPUs, 32 GB of RAM, and $P_v = 1.7$. Similarly to the most popular commercial solutions, the BTU is 60 minutes. The startup time D_v^{sv} of $v \in V$ ranges uniformly in $[85, 115]$ s, in accordance with [7]. During the experiment, the number of containers requiring resources fluctuates between 0 and 20 during the timespan of a (simulated) day, and this pattern is repeated for the following two days. Figure 1a shows the number of active containers during the whole experiment; being $L \geq \tau$, the number of active container can be greater than 20.

We evaluate the effects on the containers deployment of three different configurations of EVCD, namely EVCD_C, EVCD_D, and EVCD_CD. **EVCD_C** deploys containers by minimizing, as QoS metric, the cost $C(\boldsymbol{x}, \boldsymbol{z})$, i.e., it solves EVCD with F parametrized with weights $w_c = 1$ and $w_d = 0$. **EVCD_D** minimizes the deployment time $D(\boldsymbol{x}, \boldsymbol{z})$, by solving EVCD with $w_d = 1$ and $w_c = 0$. Finally, **EVCD_CD** minimizes both the QoS metrics, by solving EVCD with $w_d = w_c = 0.5$, and normalization terms $D_{\min} = 8.5$ s, $D_{\max} = 152.1$ s, $C_{\min} = 0$, and $C_{\max} = 10.1$. These values result from preliminary experiments. Figure 1b shows the number of active VMs during the experiment, whereas Table 1 reports the average values of the considered QoS metrics and the relative number of used VMs per type. From Fig. 1b we can see that, although every configuration of EVCD acquires and releases VMs to satisfy the incoming load, each of them follows a different strategy. Differently from the other configurations, EVCD_C tries to use as few VMs as possible and, if needed, prefers *type B* VMs (76.6% of the time, see Table 1), because of their economy of scale. However, this leads to the highest deployment time, which is higher then the optimal one of about 76% (see Table 1). EVCD_D neglects the differences in terms of price between VMs of *type A* and *type B*, which are used almost equally. This strategy has the highest cost (29% higher than the optimal one), however it obtains the lowest

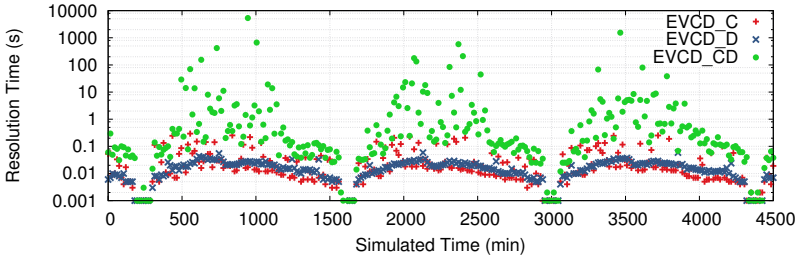


Figure 2: Resolution time of EVCD.

possible deployment time, because it allocates containers on VMs that already host some of the needed image layers, thus reducing the container startup time. EVCD_CD finds a trade-off between the previous strategies, as can be seen from Table 1. It optimizes both the QoS metrics, achieving a deployment time and cost about 9% and 8% higher than the optimal values, respectively. Moreover, this strategy shows that, by selecting the weights of F according to the user preferences, EVCD can be aimed at optimizing different QoS metrics of interest. Observe that any other combination of weights w_c, w_d lead to configurations that lie in between EVCD_C and EVCD_D. Determining the best trade-off between deployment time and cost depends on the relative importance of these QoS metrics on the utilization scenario (e.g., different user preferences).

On EVCD Resolution Time. We now discuss about the *resolution time* of EVCD and its relationship with the optimization goals. We consider as resolution time the time needed to compute the exact solution of the ILP problem. During this experiment, the maximum number of managed virtual machines (both active and leasable) in V is 19, whereas the maximum number of active containers in S is 40. Figure 2 reports the resolution time of EVCD for the different configurations (i.e., EVCD_C, EVCD_D, and EVCD_CD) during the experiment. In general, we observe that the resolution time is influenced by the size of the problem as well as by the optimization function F resulting from the different set of weights. Optimizing a single QoS attribute, i.e., solving EVCD_C or EVCD_D, is less computationally demanding and has better performances; as can be seen from Fig. 2, the resolution time of EVCD_C and EVCD_D is always below 1 s. Conversely, solving a multi-objective optimization problem is harder and produces higher resolution times with greater fluctuations with respect to the previous configurations. The complexity of CPLEX does not easily reveal the motivations behind the number and amplitude of these fluctuations. During the whole experiment, the 95th percentile of the resolution time for EVCD_C and EVCD_D is 189 ms and 33 ms, respectively, whereas EVCD_CD has a 95th percentile of about 24.6 s, i.e., two order of magnitude higher.

It can be demonstrated that the deployment problem is NP-hard (see [2]), therefore EVCD does not scale well as the problem instance increases in size. Nevertheless, by determining the optimal deployment of containers over an elastic

set of VMs and evaluating the subsequent runtime reconfigurations, EVCD provides a benchmark for evaluating heuristics, for developing new ones, and for identifying the most suitable ones with respect to specific optimization objectives.

6 Conclusion

In this paper we have described and evaluated EVCD, a formulation of the elastic provisioning of virtual machines for container deployment. EVCD is a general and flexible model that can be conveniently configured to optimize different QoS metrics. Aside computing the initial allocation, EVCD can adapt the containers deployment at runtime, if needed. The experimental evaluation has shown the flexibility of the proposed model, which has optimized the deployment time of containers, the monetary cost for their execution, and a combination thereof.
As future work, we plan to extend the formulation of EVCD to include other QoS attributes (e.g., network traffic, resource utilization) and the runtime replication of containers. Moreover, we plan to develop efficient heuristics to deal with large instances of the container deployment problem for the initial deployment and to support runtime reconfigurations.

References

1. Abdelbaky, M., Diaz-Montes, J., Parashar, M., et al.: Docker containers across multiple clouds and data centers. In: Proc. of IEEE/ACM UCC 2015 (2015)
2. Cardellini, V., Grassi, V., Lo Presti, F., Nardelli, M.: Optimal operator placement for distributed stream processing applications. In: Proc. of ACM DEBS '16 (2016)
3. Dong, Z., Zhuang, W., Rojas-Cessa, R.: Energy-aware scheduling schemes for cloud data centers on google trace data. In: Proc. of IEEE OnlineGreenComm 2014 (2014)
4. Felter, W., Ferreira, A., Rajamony, R., et al.: An updated performance comparison of virtual machines and linux containers. In: Proc. of IEEE ISPASS 2015 (2015)
5. Ghodsi, A., Zaharia, M., Hindman, B., et al.: Dominant resource fairness: Fair allocation of multiple resource types. In: NSDI. vol. 11 (2011)
6. Hoenisch, P., Weber, I., Schulte, S., et al.: Four-fold auto-scaling on a contemporary deployment platform using docker containers. In: Proc. of ICSOC 2015 (2015)
7. Mao, M., Humphrey, M.: A performance study on the vm startup time in the cloud. In: Proc. of IEEE CLOUD 2012 (2012)
8. Nardelli, M., Hochreiner, C., Schulte, S.: Elastic provisioning of virtual machines for container deployment. In: Proc. of ACPROSS 2017, colocated with ACM/SPEC ICPE '17 (2017)
9. Piraghaj, S.F., Dastjerdi, A.V., Calheiros, R.N., et al.: A framework and algorithm for energy efficient container consolidation in cloud data centers. In: Proc. of IEEE DSDIS 2015 (2015)
10. Wang, W., Li, B., Liang, B.: Dominant resource fairness in cloud computing systems with heterogeneous servers. In: Proc. of IEEE INFOCOM 2014 (2014)
11. Zhao, Z., Mandagere, N., Alatorre, G., et al.: Toward locality-aware scheduling for containerized cloud services. In: Proc. of IEEE Big Data 2015 (2015)