Docker Containers Across Multiple Clouds and Data Centers

Moustafa AbdelBaky
Javier Diaz-Montes
and Manish Parashar
Rutgers Discovery Informatics Institute
Piscataway, New Jersey
Email: moustafa.a, javier.diaz, parashar@rutgers.edu

Merve Unuvar
and Malgorzata Steinder
IBM T. J. Watson Research Center
Yorktown Heights, New York
Email: munuvar, steinder@us.ibm.com

***Abstract*—Emerging lightweight cloud technologies, such as Docker containers, are gaining wide traction in IT due to the fact that they allow users to deploy applications in any environment faster and more efficiently than using virtual machines. However, current Docker-based container deployment solutions are aimed at managing containers in a single-site, which limits their capabilities. As more users look to adopt Docker containers in dynamic, heterogenous environments, the ability to deploy and effectively manage containers across multiple clouds and data centers becomes of utmost importance. In this paper, we propose a prototype framework, called C-Ports, that enables the deployment and management of Docker containers across multiple hybrid clouds and traditional clusters while taking into consideration user and resource provider objectives and constraints. The framework leverages a constraint-programming model for resource selection and uses CometCloud to allocate/deallocate resources as well as to deploy containers on top of these resources. Our prototype has been effectively used to deploy and manage containers in a dynamic federation composed of five clouds and two clusters.**

I. Introduction

Docker containers aim at solving a long-standing issue in the software development and deployment world, portability. With them, users can easily create customized environments perfectly tailored to fit the needs of their software products. More importantly they can deploy exact replicas of their environments in any infrastructure (e.g., cloud, cluster, virtualized, bare metal, etc.) much faster and more efficiently than using virtual machines. As a result of their ease-of-use and performance enhancements, these containers are being widely adopted in industry, academia, and other scientific communities.

Current research with Docker containers is mostly focused on deployment at a single site/cloud level (e.g., Kubernetes [1], Mesos [10] and Marathon [2], Amazon's EC2 Container Service [3], and IBM Bluemix [4]). These solutions are limited in that they cannot accommodate for cloud-bursting (scale out when on-premise resources are busy or unavailable), provide resilience (against zone or datacenter outages or provide disaster recovery), ensure privacy (keep private data in house, other services outside), or allow for *in-situ* data analyses (minimizing data transfers and keeping container services near data sources). In addition, relying on a single provider can lead to "provider lock-in," which prevents users from taking advantage of a competitive pricing market. Most of these issues can be solved by allowing users to deploy Docker containers across different clouds and data centers. Currently, Google is working on the Ubernetes project – an extension to Kubernetes that would enable the deployment of containers across K8S clusters [5]. However, although the concept is interesting and takes advantage of Kubernetes' power, it is only thought to work with K8S clusters and is currently still at the proposal stage. Thus, as more users look to adopt this technology, having the ability to manage containers across multiple distributed and infrastructure will become a key issue.

The use of multiple sites for container deployment brings about scenarios and challenges that have not been previously explored. Docker containers show usage trends, life-cycle management, behaviors, and performance requirements that differ from virtual machines. We identified three major areas of concern associated with container deployment across federated distributed infrastructure. These can be categorized as follows: i) resource discovery and coordination, ii) container scheduling and placement, and iii) dynamic adaptation.

Resource discovery and coordination: Selecting which physical resources to use can be challenging since the selection process can be based on various inputs from multiple actors interacting with the system. For example, users might want to select resources based on cost, performance, or privacy concerns, whereas resource providers might want to provide resources based on availability or utilization. The increasing fluctuations in resource availability results in a need to control which resources can be used over time. Additionally, once a set of resources is selected, one must be able to connect them together or at least expose them to a resource scheduler in a uniform matter.

Container scheduling and placement: The main challenges associated with container scheduling in a federated environment arise from selecting the proper affinity when placing container clusters (i.e., should a cluster of containers that provides a single service be placed close together or far apart, assuming close offers good performance, but far offers greater resilience?). Additionally, how do we support workload and/or container migration? Finally, the life-cycle of some containers can be very short (compared to virtual machines), thus there is

more emphasis on the fast placement of containers as opposed to an optimum placement scheme.

Dynamic adaptation and optimization: As resources are becoming more dynamic and workloads vary over time, it is essential to be able to dynamically and autonomically adapt the entire federation to meet the requirements of users and resource providers as well as those of the varying workloads.

In this paper, we propose C-Ports, a framework that enables transparent deployment and migration of Docker containers across multiple clouds and hybrid infrastructure. In the rest of this paper, we will present the conceptual architecture of C-Ports while highlighting the challenges associated with running containers in a multi-cloud/multi-data center environment and how our proposed framework addresses these challenges.

II. Methodology

Federated computing is a well-known technique that can be used to aggregate distributed resources in order to satisfy the needs of large-scale science and engineering problems when their required capacities and/or capabilities exceed those of a single resource. However, unlike science and engineering applications, which try to aggregate as many resources as possible, deploying containers in a business environment has other considerations, such as high availability, and thus poses different challenges (coordination, scheduling, and adaptation). In our research approach, we aim to address these challenges by separating the resource selection from the container placement. We have developed a constraint-programming model [16] that can be used for dynamic resource discovery and selection. This allows both users and resource providers to have better control over the selection process and define fine-grained workload objectives and requirements, which are considered by the system when selecting resources. For example, some of the constraints that we have been able to accommodate include availability, capacity, utilization, cost, performance, security, or power consumption. We have also included the ability to easily add or remove new/existing constraints without modifying the rest of the system. Once resources that satisfy all constraints at a given time are selected, they are exposed to the scheduler which can then make the decision on where containers should be deployed. The entire process is continuously repeated to allow for dynamic adaptation.

We chose a constraint programming (CP) model as opposed to linear [18] or integer [19] programming (LP, IP) models due to the requirement for fast container deployment. Of the three methods, CP emphasizes speed while LP emphasizes an optimum solution, which often takes longer to compute. In addition to these three approaches, optimizing resource selection can be based on multi-criteria algorithms [15], [7], [12], [14], or using advanced reservation mechanisms [17]. However, it is not our goal to find an optimum solution but rather a fast one. Hence, we do not use any CP optimization techniques; instead, we use CP to filter out resources that do not satisfy the constraints imposed by users or resource providers. Using CP in this manner, there is no objective function to maximize or minimize.

III. System Architecture

C-Ports is built on top of the open source project CometCloud. CometCloud [8], [11] is a software designed and developed at the Rutgers Discovery Informatics Institute that enables software-defined federation of cyberinfrastructure. The overall architecture of CometCloud is shown in Figure 1. Central to the CometCloud architecture, and the key mechanism used to coordinate different aspects of federation and application execution, is the Comet coordination space [13]. Specifically, Comet has two types of spaces. First, there is a single federation management space for creating the actual federation and orchestrating different resources. This space is responsible for exchanging operational messages used to discover resources, announce changes at a site, route users' requests to appropriate sites, and initiate negotiations to create ad-hoc execution spaces. Second, there are multiple shared execution spaces that are created on demand to satisfy the computational needs of applications. A federation agent creates and controls each shared execution space and coordinates the resources that execute a particular set of tasks on that site. Agents can act as a master of the execution or delegate this duty to a dedicated master (M) when more complex workflows are executed. Additionally, agents deploy workers to perform the actual execution of tasks. These workers can be in a trusted network, be part of the shared execution space and store data, or they can be part of external resources, and therefore in a non-trusted network. The first type of worker is called a Secure Worker (W) and can pull tasks directly from the space. The second type is called an isolated worker (IW) and cannot directly interact with the shared space. Instead, isolated workers depend on a proxy (P), and a request handler (R), to obtain tasks from the space.

The overall architecture of C-Ports is shown in Figure 2. The constraint programming solver is responsible for enforcing constraints and selecting resources that meet these constraints. The scheduler is responsible for deploying the current workload of containers on a set or subset of available resources based on different optimization objectives. We can use any scheduler for the actual placement of containers. The solver is exposed using a RESTful [9] web service to allow for easy integration with different schedulers. The scheduler communicates with CometCloud using the CometCloud Web Service [6] which provides a second RESTful web service for the operation of the federation. The CometCloud Federation Execution Engine [6] is responsible for managing different CometCloud agents. The Container Deployment Enactor is composed of CometCloud workers that can deploy containers in different environments. Workers can deploy containers directly on any allocated resource or interact with the local scheduler to submit container jobs.

IV. Use Case Scenarios

In order to demonstrate the effectiveness of this approach, we have developed two use cases that demonstrate the capabilities enabled by C-Ports.

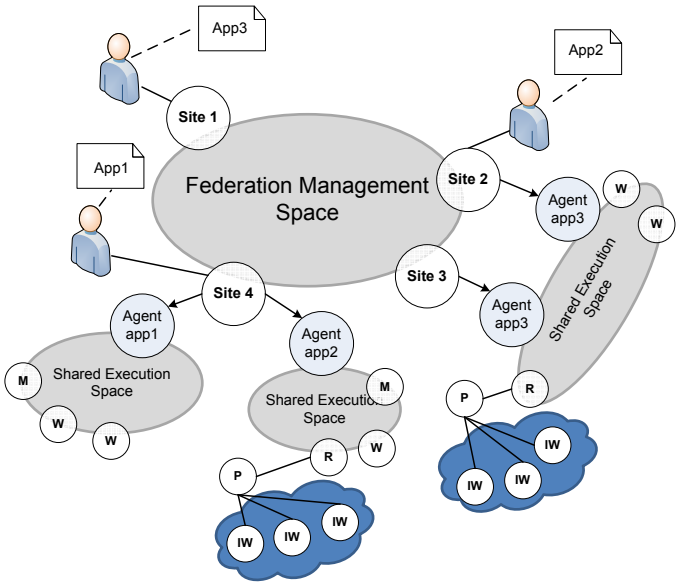

Fig. 1. Architecture of the CometCloud federation model [8] Here, (M) denotes a master, (W) is a secure worker, (IW) is an isolated worker, (P) is a proxy, and (R) is a request handler. Arrows represent deployment of a computational site, while lines show communication.

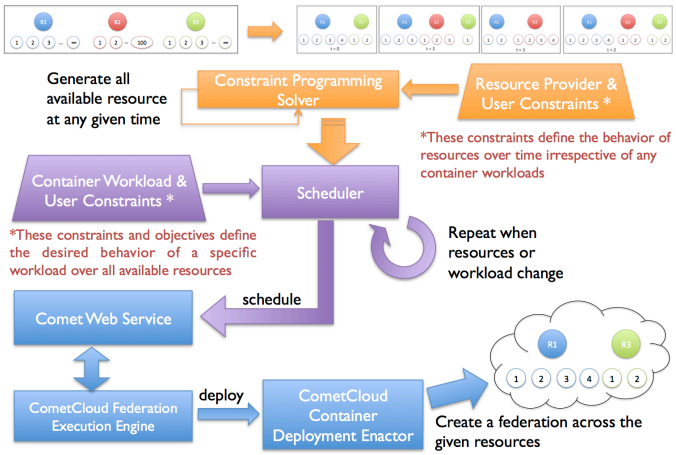

Fig. 2. Overall architecture of C-Ports.

A. Cloud-bursting

The first use case applies to a cloud-bursting scenario, wherein a user would like to deploy containers on a local cluster as long as its utilization is below a certain threshold (e.g., 80%) and burst to a cloud (e.g., IBM Bluemix) when that threshold is exceeded. The constraint given to the solver will be `(utilization <80)`. The actual utilization of the local cluster is continuously monitored and supplied to the solver, and the utilization of a cloud is assumed to be infinite. Therefore, as long as the threshold of the local cluster is less than 80%, the solver will always return the available resources to use as both the local cluster and Bluemix. Given this information, the scheduler (optimizing for cost and considering the local cluster cost to be zero) will pick the local cluster to deploy containers. The scheduler notifies the CometCloud Federation Engine and the engine starts an agent on the local cluster and deploys containers there. As soon as the utilization of the system exceeds 80%, the solver will return the set of available resources to include Bluemix only. The scheduler will then notify the federation engine to stop using the local cluster and to use Bluemix instead. The engine will notify the local cluster agent to stop accepting new containers until its utilization goes below the threshold. The engine will also start an agent on Bluemix and start deploying containers there. Thus, any containers that have not yet been deployed will be moved to Bluemix as long as the utilization of local resources is above the established threshold.

B. Running across clouds

In the second use case, we describe the scenario where the user would like to run across multiple clouds and data centers. The selection criteria for clouds is based on cost per hour, which can be static (e.g., reserved price) or dynamic (e.g., spot price), whereas the selection criteria for data centers is based on a power consumption threshold (e.g., 60%). The objective of the scheduler in this case is to maximize throughput (i.e., deploying as many containers as possible). The constraints to the solver will be `(cost <0.1 and power <60)`. The actual cost of different clouds and the power consumption rate for the data centers are continuously monitored and also provided to the solver. The solver constantly evaluates the constraints against the properties of the available systems and reports to the scheduler which resources satisfy the given constraints. These constraints can be altered at anytime during execution (e.g., the cost can be changed to `cost <0.2`). Similar to the first use case, the scheduler notifies the federation agent to start/stop agents on the proper resources and the containers are deployed accordingly. In this scenario, we can potentially use multiple clouds at the same time and we could move containers around to ensure that we adapt to the requirements and needs over time.

V. Conclusion

In this paper, we have presented our approach for managing and deploying containers in a federated environment. We have an initial implementation, called C-Ports, which has been demonstrated to effectively deploy containers in a dynamic federation composed of up to seven heterogeneous and distributed resources, including five clouds (IBM Bluemix, Amazon AWS, Google Cloud Engine, Chameleon Cloud, and FutureSystems) and two clusters (one at IBM and another at Rutgers University). With C-Ports, users can build container images for their applications one time and then rapidly deploy the image at any time using constraints that can change from one deployment to the next. The aggregation of resources is not limited to those listed above, i.e., it can be expanded to any cloud, data center, cluster, or alternative resource classes

within or across cloud providers. Further, C-Ports is not tied to a specific container scheduler. As a result, it can work with any local container scheduler, such as Kubernetes or Bluemix, or directly deploy containers on the given resource/cloud, thereby increasing its portability and flexibility. C-Ports is currently being evaluated as part of an ongoing performance analyses, which we plan to release as soon as the data becomes available. In future work, we plan to expand the C-Ports prototype to support container migration or traffic and data rerouting. In addition, we are building a heuristic-based model to compare the performance of our CP model, and to better evaluate our approach.

ACKNOWLEDGMENT

The research presented in this work is supported in part by National Science Foundation (NSF) via grants numbers ACI 1339036, ACI 1310283, CNS 1305375, and DMS 1228203, and by IBM Faculty awards. The research at Rutgers was conducted as part of the Rutgers Discovery Informatics Institute (RDI2). M. AbdelBaky would like to thank IBM T. J. Watson for their support through the EPIC internship program.

REFERENCES

[1] http://kubernetes.io.
[2] https://mesosphere.github.io/marathon/docs/native-docker.html.
[3] https://aws.amazon.com/ecs/.
[4] http://www.ibm.com/cloud-computing/bluemix/.
[5] http://tinyurl.com/ubernetesv2.
[6] M. AbdelBaky, J. Diaz-Montes, M. Zou, and M. Parashar. A framework for realizing software-defined federations for scientific workflows. In *Proceedings of the 2nd International Workshop on Software-Defined Ecosystems*, pages 7–14. ACM, 2015.
[7] A. Amato and S. Venticinque. Multi-objective decision support for brokering of cloud sla. In *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pages 1241–1246. IEEE, 2013.
[8] J. Diaz-Montes, M. AbdelBaky, M. Zou, and M. Parashar. Cometcloud: Enabling software-defined federations for end-to-end application workflows. *IEEE Internet Computing*, 19(1):69–73, 2015.
[9] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002.
[10] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.
[11] H. Kim and M. Parashar. Cometcloud: An autonomic cloud engine. *Cloud Computing: Principles and Paradigms*, pages 275–297, 2011.
[12] K. Kurowski, J. Nabrzyski, and J. Pukacki. User preference driven multiobjective resource management in grid environments. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 114–121. IEEE, 2001.
[13] Z. Li and M. Parashar. Comet: A scalable coordination space for decentralized distributed environments. In *Intl. Workshop on Hot Topics in Peer-to-Peer Systems*, 2005.
[14] N. Ranaldo and E. Zimeo. A framework for qos-based resource brokering in grid computing. In *Emerging Web Services Technology, Volume II*, pages 159–170. Springer, 2008.
[15] I. Rodero, J. Corbalán, R. M. Badia, and J. Labarta. enanos grid resource broker. In *Advances in Grid Computing-EGC 2005*, pages 111–121. Springer, 2005.
[16] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier, 2006.
[17] C. Vecchiola, X. Chu, and R. Buyya. Aneka: a software platform for .net-based cloud computing. *High Speed and Large Scale Scientific Computing*, 18:267–295, 2009.
[18] L. A. Wolsey. *Integer programming*, volume 42. Wiley New York, 1998.
[19] H.-J. Zimmermann. Fuzzy programming and linear programming with several objective functions. *Fuzzy sets and systems*, 1(1):45–55, 1978.