Carretero J., Vasile-Cabezas M., Sosa V. (2019) Adaptive Application Deployment of Priority Services in Virtual Environments. In: Montella R., Ciaramella A., Fortino G., Guerrieri A., Liotta A. (eds) Internet and Distributed Computing Systems. IDCS 2019. Lecture Notes in Computer Science, vol 11874. Springer, Cham.

DOI: 10.1007/978-3-030-34914-1_5

Adaptive Application Deployment of Priority Services in Virtual Environments

Jesus Carretero(✉), Mario Vasile-Cabezas, and Victor Sosa

Computer Science and Engineering Department,
University Carlos III of Madrid, Madrid, Spain
jesus.carretero@uc3m.es

Abstract. This paper introduces an adaptive application deployment service for virtualized environments (named DECIDE). This service facilitates the definition of customized cluster/cloud environment and the adaptive integration of scheduling policies for testing and deploying containerized applications. The service-based design of DECIDE and the use of a virtualized environment makes it possible to easily change the cluster/cloud configuration and its scheduling policy. It provides a differentiated service for application deployment based on priorities, according to user requirements. A prototype of this service was implemented using Apache MESOS and Docker. As a proof of concept, a federated application for electronic identification (eIDAS) was deployed using the DECIDE approach, which allows users to evaluate different deployment scenarios and scheduling policies providing useful information for decision making. Experiments were carried out to validate service functionality and the feasibility for testing and deploying applications that require different scheduling policies.

Keywords: Application deployment · Resource management · Application scheduling

1 Introduction

Virtualization technologies provide a way to share resources and to create portable, scalable and elastic applications. Virtual machines facilitate the dynamic building of clusters or clouds. Containers are a lightweight virtualization technique that has demonstrated to be a scalable and high-performance alternative to virtual machines for a more portable and faster deploying of applications. Apache Mesos is a resource manager that provides a two-level scheduling mechanism. Mesos slaves (agents) report to master the amount of free resources they can provide and master decides how many resources to offer to the framework, i.e., the system that manages and executes applications. The decisions about which application should receive the next resource offer is based on the Dominant Resource Fairness (DRF) algorithm, which is a solution to the problem of fair resource allocation in a system sharing different resources (cpu,

memory, storage, etc). However, applications like the federated electronic identification system presented in require a different treatment, where resource fairness is not the main concern. Mesos allows customizing its scheduling module, what motivated us to implement DECIDE. The main contributions of this paper are: (a) design and implementation of a web-based application deployment service (DECIDE) that facilitates the creation of customized cluster or cloud scenarios; (b) implementation of an adaptive deployment service based on user requirements; and (c) prototype of a federated electronic identification system that demonstrates the feasibility of using the DECIDE service and its benefits.

This paper is structured as follows, Sect. presents related work, Sect. introduces the DECIDE architecture and its adaptive deployment algorithm. Sectiondescribes our use case. Section depicts experiments and obtained results and Sect. presents conclusions.

2 Related Work

Nowadays, new platforms for resource sharing in virtualized cluster or clouds, such as Mesos, DC/OS (Mesosphere), Kubernetes and Swarm have arisen. Scheduling is a fundamental part that determines quality of service aspects such as response time, availability, service continuity, etc. Mesos is a platform for sharing commodity clusters between multiple diverse cluster computing frameworks, such as Hadoop and MPI. Mesos manages resources, but delegates control over scheduling to the frameworks using a decentralized scheduling model. Mesos resource allocation is called Hierarchical DRF and it is based on online domi-nant resource fairness (DRF), which provides fairness with dynamic resource partitioning harmonizing the execution of heterogeneous workloads (in terms of resource demand) by maximizing the sharing of resources. The major problem with this policy is that a mistake in the allocation of mandatory resources could leave a high number of tasks pending for an indeterministic duration, which is not possible in prioritized system with some quality of service. DC/OS is a distributed system, cluster manager, container platform, and an operating system . It includes two built-in task schedulers, Marathon (applications) and Metronomo (jobs), and two container runtimes (Docker and Mesos). Docker is a Linux based lightweight container that allows different applications to run isolated from each other but safely share the machine's resources. Docker provides a good basis to run composite applications in the cloud. Therefore, a number of tools emerged that claim to solve the problem. The paper classifies the solutions to deploy a container management solution for multiple hosts and presents they own suite. One possible solution to provide complex environment is to manage dockers using Mesos, as shown in. Kubernetes is an open-source platform created by Google for container deployment operations. It wraps one or more containers into a higher-level structure called a pod, wherein resources are shared. Its scheduler ensures that pods are only placed on nodes that have sufficient free resources. Swarm [9] is an open-source container platform that is

the native Docker clustering engine. Swarm turns a pool of Docker hosts into a virtual, single host. Swarm has a single scheduling strategy, called spread, which attempts to schedule a task based on an assessment of the resources available on cluster nodes. Most of the frameworks look for fairness in resource allocation [8], which can be contradictory with a strict priority scheduling policy. Our proposal faces this problem, that we have not seen solved in the literature.

3 Adaptive Deployment of Priority Services in Virtual Environments

This section presents the architecture and algorithm of our proposal of an adaptive application deployment service for virtual environments named DECIDE.

3.1 Architecture and Tools

We have developed a prototype of DECIDE as an upper layer of a Mesos cluster with a virtualized environment supported by Docker. However, our service-based architecture, shown in Fig. 1, allows to use a different cluster environment, for example Kubernetes [1]. The components of DECIDE are: (1) Web-based manager to configure Mesos clusters and to instantiate frameworks in the virtual environment; (2) adaptive deployment service with different scheduling policies which is integrated as framework of Mesos; and (3) application manager to instantiate containers (applications) according to the deployment policies. The most important tools used are: (a) Apache Mesos to manage cluster resources; (b) Dockers as a container platform for deploying applications; (c) Apache Zoo Keeper to provide fault tolerance; (d) HTML5+CSS3+JS (front-end) and Go language (back-end) to develop the deployment services manager and the client simulator.

The 6 main steps (see Fig. 1) to deploy applications are: (1) execution of the web-based interface for user to select the cluster/cloud nodes that comprise the virtualized environment; (2) reading the configuration file that will be sent to MESOS for setting the node roles (MESOS master, agent and Apache Zookeeper); (3) selection of an available scheduling policy; (4) selection of the applications configuration to be sent into MESOS; (5) reading of the previous application configuration profiles to download the container images into the cluster/cloud; and (6) scheduler execution until the whole workload is over.

3.2 Algorithm

The adaptive deployment service includes a generic framework implemented as a decoupled framework of Mesos (Mesos scheduler). The scheduling algorithm determines the order in which applications will be deployed in the containarized infrastructure (Mesos + Docker) according to the scheduling policy configured. The generic framework can be updated to modify the scheduling policy according to the applications requirements. The current prototype of DECIDE includes the

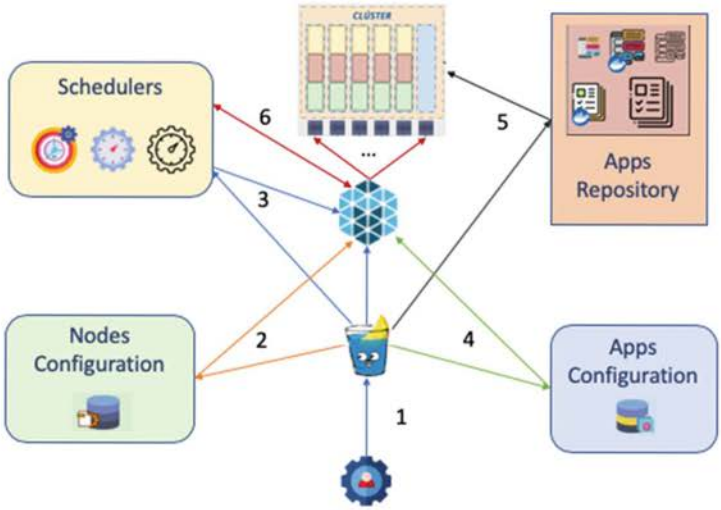

Fig. 1. General architecture of the adaptive deployment service

following scheduling policies: Preemptive Priority and FCFS (First Come First Served). In the current prototype, load balancing strategies were associated to scheduling policies to distribute the workload into the resources offered by Mesos.

For FCFS, Round-Robin load balancing is used and for preemptive priority, resource usage information will be the main indicator, giving always resources first to high priority applications pending.

Algorithm 1 shows the general steps carried out by our deployment service. The external loop in the algorithm verifies that the configured scheduler is a registered Mesos framework. The scheduler waits for the arrival of new application deployment requests or for a signal from Mesos to manage applications. The function $getNextTaskWithHighestPriority()$ returns the next task to be executed depending on the scheduler registered. If it is FCFS, it will return the task in the head of the scheduler queue. If it is using priorities, it will return the task in the head of the highest priority queue that is not empty. This function has an adaptive behavior depending on the scheduling algorithm configured. The internal loop creates a queue of tasks (containerized applications) that will be deployed considering the number of instances required. The deployment service will carry out the load balancing strategy associated to the scheduling policy using the *loadBalancer: RegisterServer* function. The service-based design of DECIDE allows to integrate a component to provide elasticity to the deployed applications. The current prototype of DECIDE does not include this component. However, our use case application implements a manager service that carries out this function. In this way, if all existing servers are saturated, a new server is instantiated, up to the maximum number defined in the application manager is reached. Initially a minimum number of instances defined in the framework are created.

Algorithm 1. Scheduling algorithm

```
   Input: Enqueued applications (tasks)
   Output: successful enqueued or deployed applications
 1 while SchedulerRegistered do
 2 |   t = waitForaTask() OR availableTaskMesosSignal();
 3 |   enqueueNewTask(tasks,t);
 4 |   nextTask = getNextTaskWithHighestPriority();
 5 |   i = 0;
 6 |   var scheduledTasks = []MesosTasks;
 7 |   while len(Mesos.offeredNodes) < nextTask.Instances AND i <
   |   nextTask.Instances do
 8 |   |   loadBalancer.RegisterServer(nextTask);
 9 |   |   scheduledTasks [i] =
   |   |   createTask(Mesos.offeredNodes[i],nextTask.Command);
10 |   |   i++;
11 |   end
12 |   deployTasks(scheduledTasks);
13 |   nextTask.Instances = nextTask.Instances - i;
14 |   if nextTask.Instances > 0 then
15 |   |   enqueueTask(tasks,nextTask);
16 |   end
17 end
```

After the internal loop, tasks are deployed. If there are still pending tasks in the request because there were not enough available nodes, they will be enqueued again to be scheduled in the next round. Thus, we can avoid discarding tasks due to a temporary lack of resources.

4 Use Case: European Identity Federation Initiative

The Regulation (EU) 910/2014 [12], the so-called eIDAS Regulation, which is in force in all Member States since the end of 2018, ensures that people and businesses can use their own national electronic identification schemes (eIDs) to access public services in other EU countries where eIDs are available. To implement this system, a federation consisting of a network of eIDAS nodes has been deployed, one per member state, which has the role of Identity Provider for the national electronic identification scheme (eID) from any other country. All Service Providers participating in the network must be subscribed to the eIDAS Node of their country. This is a very ambitious approach, since it enables crossborder authentication of Member States citizens without the need to unify the authentication method (eID Scheme) of the member states participating in the Identity Federation. A detailed description of a previous evaluation of the eIDAS system is shown in [2]. The eIDAS nodes apply FCFS scheduling policy to serve client requests, as it assumes that all requests will have the same priority,

which may be not truth in the near future. This is why we have decided to evaluate the eIDAS service applying also a priority-based scheduling as use case.

5 Evaluation

This section presents the evaluation of our deployment service using the eIDAS network use case.

5.1 Experiments Description

To test our proposal, eIDAS Manager, that uses the scheduler to send allocate resources and execute eIDAS nodes in Mesos. We have defined a test scenario as described in Fig. 2. All clients are created in a single node and they send requests to our eIDAS Manager, which uses the deployment framework to allocate and execute eIDAS servers in Mesos.

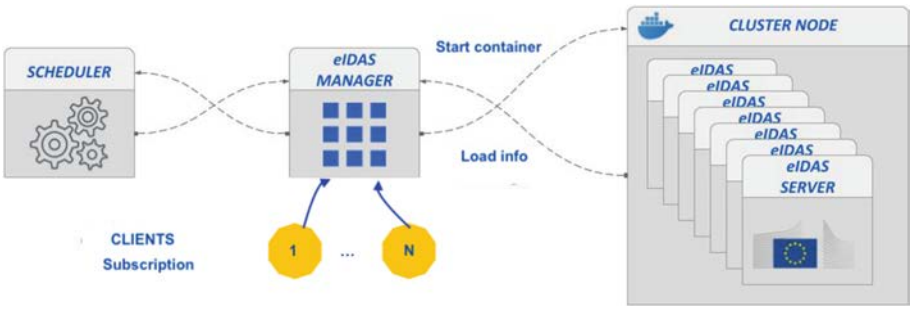

Fig. 2. Architectural view of clients and servers in the experimental scenario

The service has been deployed in the cluster defined in Table 1. The system has been implemented using Dockers on Linux OS. Several dockers are started at the beginning of the system depending on the configuration parameters. The number of dockers evolves with the load following the scheduling policy. Load is distributed among the cluster nodes. Each client requesting for an electronic identification (eID) starts a protocol including 12 messages between the different components of the eIDAS network. Because of consistency and security restrictions, each client request must be managed by the instance of the eIDAS node accepting it, but, in case of preemption, the client can be stopped between two messages. For these experiments, a maximum of 64 simultaneous servers can be instantiated, a parameter that we have defined as a limitation in the MESOS resources. In the case of the scheduling with priorities, we segment users into three groups of priorities, **high, medium, low**. The allocation of priority follows a uniform distribution, so that they subscribe to the server destinations with the same probability.

Table 1. Features of the experimental environment

Feature	Compute 11	Compute 7
Application	eIDAS servers	eIDAS Manager and clients
Nodes	5	1
RAM	128 GB	128 GB
CPU	Xeon E5-2603 v4 12 cores	XeonE7-4807 24 cores
Network	Ethernet 10 Gbps	Ethernet 10 Gbps
HDD	1 TB	1 TB

Two scenarios have been tested (see Table 2), both processing 3750 eID client requests. Every eIDAS client is associated to an eIDAS server that serves its requests. In scenario A, FCFS policy is applied and the eIDAS manager associates a client to an eIDAS server using a Round Robin approach to distribute the load across the servers. In scenario B, priority scheduling is applied and system usage information is used to dynamically allocate server resources. The goal is to keep the waiting time as constant as possible, to enhance interactivity and avoid timeouts in the eID protocol, while optimizing the usage of resources and favouring the execution of high priority requests. To measure the results of the experiments, we have chosen the following metrics: CPU and memory usage; Average time and typical deviation for the identification process; Average time and typical deviation for the waiting time of a ready ID request.

Table 2. Evaluation experiments executed.

Scenario	Scheduler	eIDAS servers	Clients
A	FCFS and Round Robin	up to 64	3750
B	Priorities and node usage	up to 64	3750

5.2 Experiments Results

Scenario A. The scheduler (MESOS framework) has a single queue using FCFS policy. The eIDAS Manager allocates the requests and instantiates resources using Mesos taking the first eID requests in the queue (FCFS) and applying a Round-Robin load balancing policy for all requests, as quality of service is not enforced by the scheduler. Figure 3 shows the CPU and memory usage in every cluster node when executing 3750 clients. eIDAS servers were deployed in Compute 11-2, 11-4, 11-5, 11-6 and 11-7 nodes, while eIDAS manager and clients were deployed in Compute 7-2 node. As may be seen, the distribution of load is mostly uniform across the server nodes, even if two are more loaded. Table 3 shows the time metrics during the experiment. The total execution time for the

test is 280 s. In this scenario, eID clients had a low waiting time with a reasonable execution time (Aver. Id time). The maximum number of simultaneous clients processed by the 64 eIDAS servers, before discarding requests, was around 4000. The worst-case execution time (WCET) of a client eID request was 25.17 s for any request.

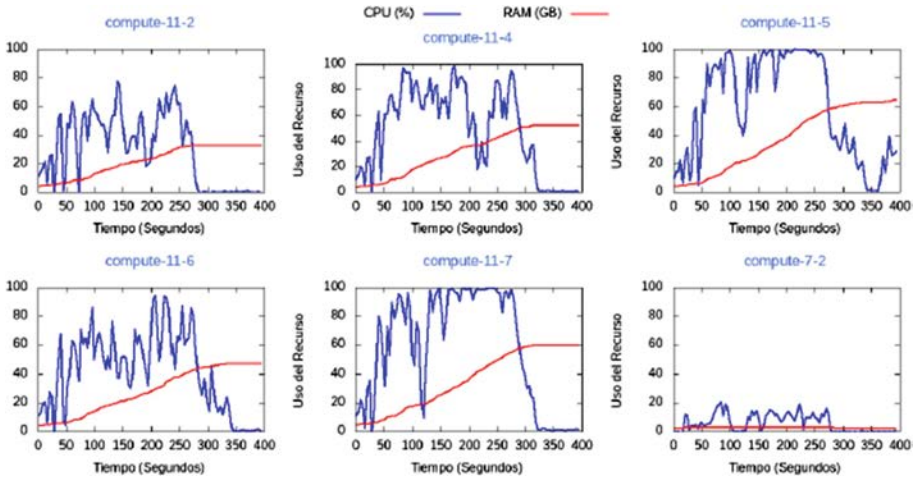

Fig. 3. CPU and memory usage with scenario A

Table 3. Scenario A. Time metrics with FCFS (all time in sec.)

Priority	Aver. wait time	Typical deviation	Aver. Id time	Typical deviation
FCFS	3.21	1.75	18.07	2,14

Scenario B. In this scenario, the scheduler uses priority policy based on one queue per priority (low, medium, high). The eIDAS manager takes the first eID requests in the highest priority queue available and uses resource information provided by Mesos to start dynamically the servers. Figure 4 shows the CPU and memory usage in every node to execute 3750 clients. As may be seen, load distribution changes in the nodes due to the distribution per priority. Again, the node running the eIDAS manager and clients has a negligible load, being able to receive many more client requests. The manager could send the 3750 client requests to the Mesos frameworks, even though the CPU load on some nodes was very high. Memory was not overloaded in any node. The total execution time for the test was 350 s. Table 4 shows how the waiting time is low for high and medium priority clients, as desired, but it is higher for low priority requests. With this policy, the worst-case execution time (WCET) of a high-priority client

eID request was 13.73 s. Preemption effect can be appreciated in the execution time, which is higher for lower priority clients, because they are removed from the system when high priority clients arrives and there are not free resources. Obviously, this strict priority policy could lead to starvation of low priority clients, but we are assuming that this fact comes with the policy. To avoid this problem, aging could be applied to clients in low priority queues.

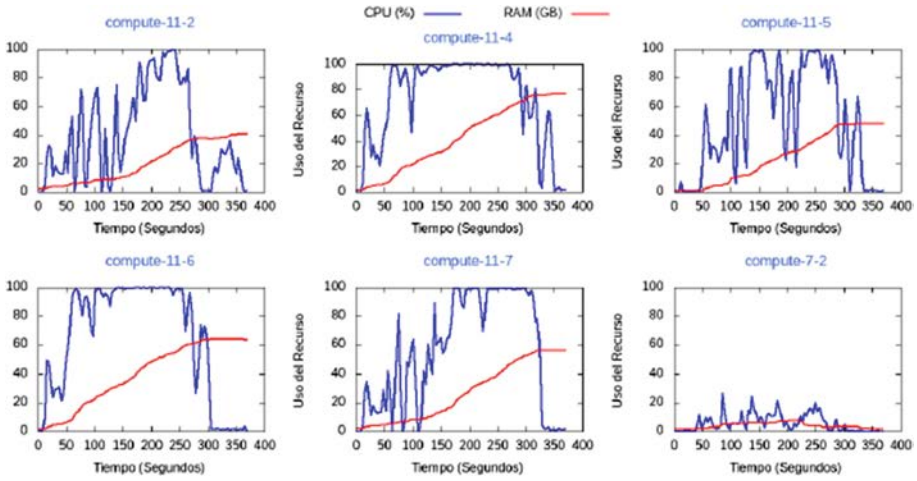

Fig. 4. CPU and memory usage with scenario A

Table 4. Scenario B. Time metrics per priority queue (all time in sec.)

Priority	Aver. wait time	Typical deviation	Aver. Id time	Typical deviation
High	2.83	1.03	8.00	2.87
Medium	3.31	1.72	13.11	2.87
Low	55.15	1.40	20.43	1.2

6 Conclusion

We introduced DECIDE, an adaptive application deployment service for virtualized environments, which provides: (a) a tool to facilitate the setup of customized cluster/cloud environments and the deployment of applications in resource sharing platforms as Mesos; (b) a scheduling and deployment framework for a differentiated services according to user requirements; and (c) a tool for testing different scheduling policies to evaluate their convenience for applications. Setting up a cluster and a framework dedicated to run a distributed application can be a tedious and time consuming work. Mesos is a solution for sharing resources

of a cluster where different frameworks can be executed. However, Mesos applies a fairness scheduling policy without considering special needs of applications that require a prioritized access. However, our use case eIDAS might require a different scheduling policy where fairness is not the main requirement. DECIDE is a solution for this type of limitations, providing a polymorphic framework on Mesos to deploy applications using a scheduling policy dynamically selected by users. Experiments demonstrated the DECIDE functionality and its feasibility of use. Future research directions will be to enhance low-priority requests response times and extending the solution to other applications and policies.

Acknowledgments. This work was partially funded by the Spanish Ministry of Economy, Industry and Competitiveness under the grant TIN2016-79637-P "Towards Unification of HPC and Big Data Paradigms".

References

1. Bernstein, D.: Containers and cloud: from LXC to docker to kubernetes. IEEE Cloud Comput. **1**(3), 81–84 (2014)
2. Carretero, J., Izquierdo-Moreno, G., Vasile-Cabezas, M., Garcia-Blas, J.: Federated identity architecture of the european eID system. IEEE Access **6**, 75302–75326 (2018)
3. Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., Stoica, I.: Dominant resource fairness: fair allocation of multiple resource types. In: NSDI, vol. 11, pp. 24–24 (2011)
4. Google: Kubernetes cluster configuration. https://kubernetes.io/docs/user-journeys/users/application-developer/foundational/
5. Greenberg, D.: Building Applications on Mesos: Leveraging Resilient, Scalable, and Distributed Systems. O'Reilly Media, Inc., Newton (2015)
6. Hindman, B., et al.: Mesos: a platform for fine-grained resource sharing in the data center. In: NSDI, vol. 11, p. 22 (2011)
7. Kakadia, D.: Apache Mesos Essentials. Packt Publishing Ltd., Birmingham (2015)
8. Kesidis, G., Shan, Y., Jain, A., Urgaonkar, B., Khamse-Ashari, J., Lambadaris, I.: Scheduling distributed resources in heterogeneous private clouds. In: 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 102–108. IEEE (2018)
9. Naik, N.: Building a virtual system of systems using Docker Swarm in multiple clouds. In: 2016 IEEE International Symposium on Systems Engineering (ISSE), pp. 1–3. IEEE (2016)
10. Peinl, R., Holzschuher, F., Pfitzer, F.: Docker cluster management for the cloud - survey results and own solution. J. Grid Comput. **14**(2), 265–282 (2016). https://doi.org/10.1007/s10723-016-9366-y
11. Puetm, A.: Mesosphere: DC/OS distributed cloud operating system, June 2019. https://dcos.io/
12. EU Regulation: No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC (eIDAS Regulation). European Union, pp. 44–59 (2014)

13. Saha, P., Govindaraju, M., Marru, S., Pierce, M.: Integrating apache airavata with docker, marathon, and mesos. Concurr. Comput.: Pract. Exp. **28**(7), 1952–1959 (2016)
14. Soltesz, S., Potzl, H., Fiuczynski, M., Bavier, A., Peterson, L.: Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. SIGOPS Oper. Syst. Rev. **43**(3), 275–287 (2007)
15. Turnbull, J.: The Docker Book: Containerization is the New Virtualization (2014)