PISCO: A smart simulator to deploy energy saving methods in microservices based networks

Hernan Humberto Alvarez Valera, Marc Dalmau, Philippe Roose, Jorge Larracoechea, Christina Herzog

PISCO: A smart simulator to deploy energy saving methods in microservices based networks

Hernan Humberto Alvarez Valera*, Marc Dalmau†, Philippe Roose‡,¶Jorge Larracoechea and §Christina Herzog
*†‡ ¶ E2S UPPA, University of Pau
64600 Anglet / FRANCE
Email: *hhavalera@univ-pau.fr, †Marc.Dalmau@iutbayonne.univ-pau.fr, ‡Philippe.Roose@iutbayonne.univ-pau.fr,
¶jorge-andres.larracoechea@etud.univ-pau.fr, §herzog@efficit.com
§*EFFICIT SAS* - Mauzac, France

***Abstract*—Nowadays, many researchers work to identify microservices-based application deployments and scheduling solutions to save energy without decreasing functional QoS. In this work, we present PISCO: A simulator that allows facing this challenge in a simple and efficient way, enabling its users to focus uniquely on microservices deployment/scheduling algorithms and its hardware/software repercussions (load vs. energy consumption) without worrying about low-level network configurations or operating system issues. PISCO is able to deploy and schedule (move, duplicate, start/stop) microservices and their dependencies on various devices with software and hardware heterogeneity (CPU, bandwidth, RAM, Battery, etc.), taking into account various scheduling heuristics algorithms: centralized vs non-centralized. To do this, PISCO allows deploying custom network topologies based on client-server schemes or p2p distributions, where devices can (dis)appear, turn on/off obeying random circumstances or user strategies.**

Finally, the simulator performs relevant operations such as QoS definition, resource monitoring, calculation of energy saved and consumption tracking (at device and network level). We tested some ideas based on our previous work "Kaligreen" to demonstrate the effectiveness of PISCO.

***Index Terms*—microservices, middleware, energy, consumption, CPU, network, hard disk, prototype, simulator**

I. Introduction

Currently, many companies and scientists use microservices because they allow architectural advantages such as acceleration of deployment cycles, modularity, improvement of maintainability, high availability, scalability, etc [1]. For example, Docker and Kubernetes [2] manage microservices on the cloud or even on grid environments, while Kalimucho [3] does the same on user device level.

Then, to manage them correctly and to achieve a desired QoS (i.e. performance and response time), it is necessary to face several issues such as, load balancing, scalability, etc. Some centralized approaches, such as the Netflix Conductor [4] , help addressing them by managing microservices' connection aspects; whereas decentralized approaches like Kalimucho [3], address these issues by allowing the development of in-device algorithms for each node to perform microservices movement or duplication operations. Then, knowing that both (and also hybrid) approaches have advantages and disadvantages; we find it useful to have a tool that allows working with these two approaches. Kalimucho middleware [3] enables each device to manage microservices and their connections, allowing to deploy both types of approaches. It deploys and operates (start, stop, move and duplicate) software components from one device to another, regardless of whether the device belongs to cloud entities or user terminals. Then, as Kalimucho is able to measure on each device CPU activity, current bandwidth and RAM availability, it is a good middleware candidate for energy savings purposes. For this reason, in some of our previous works [5] [6] we proposed a middleware called "Kaligreen" strongly inspired by Kalimucho. Kaligreen proposes that each device in the network has a supervisory entity which: (1) constantly monitors the device energy situation (i.e. battery, energy consumption, etc.) and the load of its hardware components; (2) classifies microservices according to their type and features [6] and (3) executes its own scheduling algorithm based on smart negotiations [5].

Then, while we did a POC to prove that moving/duplicating services in a distributed architecture can save energy,we did not study enough the heterogeneity of devices or the capabilities that each hardware component offers. Nor did we consistently consider the QoS of applications when microservices are scheduled at different types and levels of network topologies. For these reasons, we propose PISCO: A tool that allows evaluating different heuristics of microservice deployment/scheduling and its repercussions taking into account: (1) the capabilities and particular characteristics of each hardware component, (2) the simple configuration, through a GUI or an API, of a centralized (i.e. client server), decentralized (i.e. p2p architectures [7]) or hybrid network configuration and (3) the definition and evaluation of QoS, energy metrics and quantity of operations quantity (movements, duplications, deployments and deletions) when a scheduling algorithm is executed. Thereby, PISCO is able to virtualize an environment controlled by any middleware that deploys and schedules microservices, allowing researchers to create any type of scheduling algorithm and measure its consequences in terms of energy and application efficiency.

This article presents the policies and the architecture of the simulator in Section II and in section III, some examples and results are shown.

II. PISCO SIMULATOR

PISCO is a desktop and portable simulator, able to manage (deploy/schedule) any microservice based application in any network architecture (non-centralized, centralized and hybrid). Then, in our simulator context, an application is defined as a directed graph whose nodes are microservices and edges the connections between them. Furthermore, every application is deployed on a graph of connected devices, where the nodes could be devices with heterogeneous characteristics or abstract entities such as clusters or cloudlets. Device graph edges are the network connections that exist between each node (ethernet, wireless, 4g, or bluetooth), which allow to find the transfer rate at which the microservices can send or receive data to each other. Then, to understand the interaction of both graphs, it is necessary to describe their nodes and edges in detail, which are the entities of our simulator:

A. Simulator entities

1) Device: Processing entity identifiable by a unique ID. The simulator by default supports desktop computers, laptops and smartphones; however, users are also able to configure their own type of device. Each one of these has different components capabilities in terms of CPU frequency (presence or absence of PCPG and DVFS are also included), RAM size, hard drive speed, network transfer rate and battery (optional), depending on which type they are or what custom configuration was performed. Each device is capable of executing microservices, providing them a quantity of resources according to their own needs and those of other competitors microservices (like proportional share scheduling; however we are currently implementing other philosophies as round robin or a simplified version of CFS [8], which uses a red and black tree to organize processes). In addition, each device has a supervisor service running on it [5]. It allows (1) tracking the status of each of its components in terms of load/capacity/energy consumption and (2) executing a distributed scheduling algorithm (i.e. move or duplicate a microservice to another known or reachable device, considering negotiation criteria, load balancing, number of hops, etc.)

2) Microservice: Functional entity identifiable by a unique ID, which has a defined and precise function (i.e. a code that represents a function and that allows to be duplicated by the simulator as explained in the next section II-B). When deployed on a device, the microservice claims for a certain amount of resources in terms of CPU, network, RAM and disk. PISCO supports by default 3 types of microservices: graphical user interface microservices, calculation microservices and data management microservices. The only differences between them are their default resources consumption, the amount of data they send/receive, their size and the restriction of being moved or not from one device to another. For example, a UI microservice may not be moved (since the user experience would be affected); while a computing microservice may be duplicated or moved to another nearby device. A data management microservice can be moved, but may have a strong impact on bandwidth consumption. As with devices, a user has the ability to create its own type of microservices.

3) Connection: PISCO defines and manages two kinds of connections: physical connections that exist between devices and logical/functional connections between microservices. A physical connection logically supports many microservices connections and enables to track the current and maximum possible transfer rate (i.e. between 2 devices: the maximum transmission capacity of the device with the network interface with less transmission capacity). Then, this allows understanding: (1) the general state (i.e. at energy and load level) of a given device network interface, even if it belongs to abstract entities such as clusters, cloudlets, etc. and (2) the set of microservice connections status (i.e. expected transmission rate vs. real) that use a physical connection. Furthermore, connections between microservices logically store the set of physical connections they use to communicate. That is, it is possible to know, through the supervisor microservice, the path of devices a microservice connection uses to work, as well as the current and expected transfer rate.

4) Application: A directed graph of microservices identifiable by an ID. Each microservice (node) can have connections (edges) as dependencies relations with several other microservices which run on different devices on the network.

5) Operations center: Centralized entity which stores all references to connections and existing devices of a deployed scenario. In this way, any type of middleware can be configured and any centralized scheduling algorithm can be applied.

6) Abstract entities: Resources providers such as cloudlets and clusters. Both can be deployed and connected in the same way as devices, establishing the amount of resources to offer and manage. Therefore, our simulator is compatible with both, small networks of user devices and cloud networking.

B. Simulator Operations

PISCO is able to apply different types of heuristics for microservices (re)deployment and scheduling through a network. For this, the following operations have been implemented:

1) Device deployment: As previously mentioned, a user can deploy/declare default PISCO devices (i.e. desktop computer, laptop and smartphone); however, he can also deploy "custom" devices, specifying their capabilities in terms of (1) CPU frequency in Ghz, (2)RAM capacity in MB, (3) network capacity in Mb/s, (4) hard drive in MB/s and (5) battery specification. Furthermore, each time a device is deployed, the user must specify an x, y position on a Cartesian plane of metric units. This possibility allows studying deployment techniques based on distance heuristics or techniques to improve routes.

2) Device suppression: PISCO allows the suppression of devices (1) Manually, by the user's decision and (2) automatically, according to the deployed scenario. A user can activate a "disappear" option for each device, causing the simulator to delete the device after a random, established or battery related time; simulating unexpected connection loss.

3) Microservice deployment: PISCO allows simulating the execution of microservices on devices. This means that once deployed, a microservice uses a quantity of resources for a determined time or indefinitely [6]. This time can be set by: (1) the PISCO UI or (2) automatically by an operation center function. Furthermore, the user must also specify the microservice size in terms of disk usage and serialization (i.e. the amount of data that would be sent over the network if the microservice is moved or duplicated). This attribute allows analyzing the cost of operations in terms of efficiency and energy. For example, a user can configure the deployment of a microservice (setting up resources consumption: CPU (ghz), RAM (MB), network (MB/s), HDD (MB/s) and size(MB)) to start running after X minute after simulation start.

4) Microservices suppression: Similar to deployment, a microservice can be terminated/killed by using a simulator UI button, or according to a defined amount of time (also set using the UI). The microservice will simply stop using the device's resources and will no longer be available for any item.

5) Microservices migration: PISCO can move microservices between devices. This can be done from the operations center or from a specific device, which knows its connected peers and is able to execute decentralized algorithms.

When a microservice is moved, firstly it releases the resources of its current device. Then, PISCO simulates the moving process by using the involved network connection for a period of time based on the microservice size. Finally, it starts to compete for the resources of the new device. If a microservice is moved and becomes unreachable for one of its dependencies, the user will be able to specify search mechanisms for a duplicate instance or establish special search and path optimization mechanisms to solve the problem.

6) Microservices duplication: PISCO knows each microservice's function. Thus, it is also capable of duplicating microservices when, for example, the scheduling algorithm decides that a microservice is very requested and its overload produces excessive energy consumption. Thus, a user or an algorithm can specify where to duplicate this microservice: (1) on the same device as the original microservice, or (2) on another connected or reachable device.

7) Microservices and devices Start/Stop: PISCO allows in a scheduled or manual way to (re)start/stop devices or microservices. Both also operate as deployment and suppression of devices and microservices. The only difference is that both elements do not disappear permanently. This allows generating scenarios based on energy saving techniques such as putting into sleep mode unused devices [9].

8) QoS definition: PISCO allows defining any QoS heuristics, either at microservice or application level. The simulator offers a default heuristics based on the relation between requested resources and obtained resources.

9) QoS analysis: Our simulator can evaluate the QoS of applications and microservices at a specific rate defined by the user when a scheduling algorithm is being executed. Operations such as graphical PLOT, data storage and linear/polynomial regression are supported.

10) Energy consumption parameters definition: For PISCO, power consumption is based on the usage level of each component of each device. Then, the user can specify the formula of his choice for each device to relate the CPU frequency used, and the current transfer rates of the hard disk and the network (RAM consumption is independent of its load), with the power consumption expressed in Watts. This dynamism allows the user to display specific parameters of each component model, normally specified in the datasheets. For example, by default, PISCO specifies the CPU power measurement [10] in terms of its capacitance C, voltage V and frequency F.

$$P = CV^2F \tag{1}$$

Then, the user must specify C, V and F to measure the current CPU energy consumption. Note that to measure the CPU consumption of a microservice M, F must be equal to $M's$ frequency, assuming that M is the only current process in the device.

11) Energy consumption analysis: PISCO also allows measuring the energy consumption in a time range. This analysis can be done for each device or for the entire scenario (all devices). Operations such as graphical PLOT, data storage and linear/polynomial regression are supported.

12) Centralized/non-centralized scheduling algorithm Start/Stop: When PISCO executes a scheduling algorithm, it automatically displays metrics necessary for performance analysis: (1) Run time, (2) number of operations performed, (3) data transmitted by movements/duplications on device or across the network, (4) energy used for movements/duplications on device or across the network and (5) load per device and global load.

13) Save/Load: Deployments and execution stages.

III. Results

In order to test the effectiveness of PISCO, we implemented a "naive" approache for energy savings. The objective here, is to validate that we can deploy any type of energy saving scheduling approach and then, perform its analysis in terms of QoS.

A. Approach 1 : Decentralized

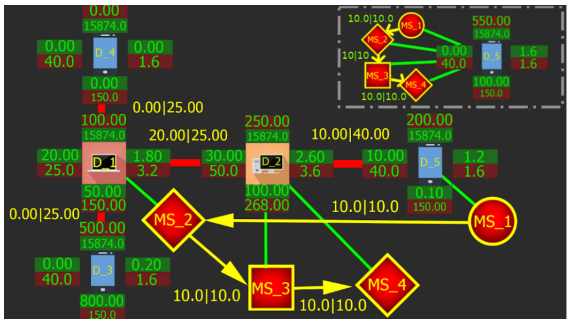

Fig. 1. Sample decentralized-mixed network

The network of figure 1, is managed by PISCO according to equations described in section II-B for energy management/evaluation and for QoS calculation. We deployed 5

connected devices with capacities shown in the outline of each one in terms of used/available network (left: MB/s), RAM(top: MB), CPU(right: Ghz) and hard drive rate(down:MB/s). Initially (Top right of figure 1) D_5 is running an application composed of 4 differents (i.e. GUI and "control" CPU intensive for this example) microservices which have requirements in terms of [CPU-Ghz,RAM-MB,network-MB/s,HDD-MB/s, size-MB]: (1) GUI: MS_1 [1.2,200,0,0.1,50], (2)Control MS_2 [1.8,100,0,50,50], (3)Control MS_3 [0.8,150,0,50,50], (4)Control MS_4 [1.8,100,0,50,50]. Then, every 1 to 5 seconds each supervisor checks energy consumption of its device against an arbitrarily threshold. If exceeded, the device will try to move the heaviest microservice to the freest neighbor.

At runtime, since this approach has no end and is quite naive, it makes the microservices MS_2, MS_3 and MS_4 oscillate between devices D_1, D_2, D_3 and D_4. However PISCO is able to store the best iteration. It found that the best deployment is the one represented in figure 1, where the microservices that saturated the CPU of the smartphone D_5 are running now on the laptop D_1 and the PC D_2. Moreover in figure 2, we note an initial low power consumption since only D_5 resources are used. Then, it has increased considerably, because now the network cards of D_1, D_2 and D_5 are saturated and the D_1 and D_2 CPU, RAM and transfer rate have increased their load as well (note that: (1)microservices now use the network to communicate to each other and (2) in this particular case, since no direct connections between D_5 and D_1 exist, microservices 1 and 2 use D_2 as a connection path, saturating its network card). The prototype interprets this deployment as an ideal one, because it obtains highest QoS (i.e. 100 %) and lowest possible consumption (i.e. 97,8 watts). Furthermore, to obtain this solution, PISCO ran the algorithm for 43.8 seconds, performing 148 microservices movements between all devices, which means 7400MB transmitted and 0.0036 kWh consumed.

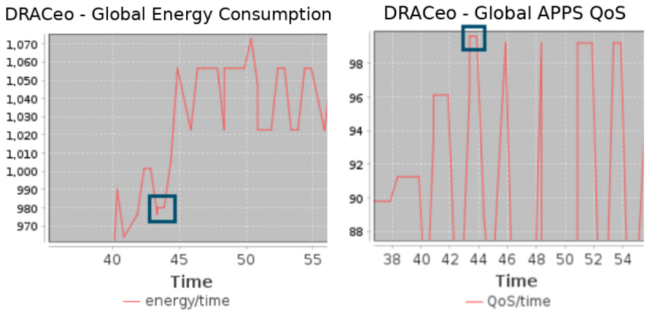

Fig. 2. Results of non-centralized algorithm evaluation

PISCO is also capable of showing metrics for each device. For instance, D_4 was the device least involved in the execution of the algorithm, performing only 16 operations which means 800MB transmitted and an energy use of 0.000016kWh.

We can conclude for example that, if we apply naive distributed scheduling algorithms like this, we can get some interesting deployment proposals in which there is an inverse relation between QoS and energy consumption. However, it lacks efficiency and predictability in terms of the amount of time and resources invested to reach a solution.

IV. Conclusions and future Works

In this work, we have presented our simulator called PISCO. It is capable of deploying and managing any type of network and heterogeneous devices to run distributed applications based on services or microservices. PISCO implements functions of (un)deploying dynamically devices and deploying, deleting, moving and duplicating microservices to allow performing centralized and non-centralized planning algorithms. At run time, the simulator is capable of monitoring several variables that allow understanding the efficiency of the technique deployed: Execution time, amount of energy spent, current quality of service and amount of data transmitted are some good examples. It is important to say that in order to find the value of QoS and energy consumption, the simulator allows users (defaults approaches are provided) to define their own heuristics or formulas. The objective of PISCO is to allow testing dynamic deployment scheduling algorithms that redeploy microservices in order to save energy while conserving a certain QoS. Thereby, PISCO will help developers and researchers find the best deployment and best distribution behavior in any network of heterogeneous devices.

Actually, we are improving the scalability of PISCO, taking into account discrete and deterministic approaches as well as real deployments of microservices and devices. On the other hand, we are updating the heuristics to determine the power consumption in some hardware components with special features(CPU turbo boost, DVFS, etc).

V. Demo video URL

https://youtu.be/ihb4qrex5Is

References

[1] A. W. Services, "Implementing microservices on aws," vol. 2019, 2019.

[2] DOCKER, "Debug your app, not your environment," https://www.docker.com/, 2020.

[3] K. Da, M. Dalmau, and P. Roose, "Kalimucho: Middleware for mobile applications," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, ser. SAC '14, 2014.

[4] NETFLIX, "Netflix conductor: A microservices orchestrator," https://netflixtechblog.com/netflix-conductor-a-microservices-orchestrator-2e8d4771bf40, 2016.

[5] H. H. lvarez Valera, P. Roose, M. Dalmau, C. Herzog, and K. Respicio, "Kaligreen: A distributed scheduler for energy saving," *Procedia Computer Science*, 2018.

[6] H. H. lvarez Valera, M. Dalmau, P. Roose, and C. Herzog, "The architecture of kaligreen v2: A middleware aware of hardware opportunities to save energy," in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, 2019.

[7] E. Bongers and J. Pouwelse, "A survey of p2p multidimensional indexing structures," 2015.

[8] IBM, "Inside the linux 2.6 completely fair scheduler," https://developer.ibm.com/technologies/linux/tutorials/l-completely-fair-scheduler/, 2018.

[9] N. M. Azmy, I. A. El-Maddah, and H. K. Mohamed, "Adaptive power panel of cloud computing controlling cloud power consumption," in *Proceedings of the 2Nd Africa and Middle East Conference on Software Engineering*, 2016.

[10] Intel, "Enhanced intel speedstep technology for the intel pentium m processor," 2004.