Resources management for controlling dynamic loads in clouds environments. The Wolphin project experience

Ahmed Amamou, Martin Camey, Christophe Cérin, Jonathan Rivalan, Julien Sopena

Resources management for controlling dynamic loads in clouds environments. The Wolphin project experience.

Ahmed Amamou (Member IEEE)
Gandi SAS - France

Martin Camey
Objectif Libre - France

Christophe Cérin (Member IEEE)
Université Sorbonne Paris Nord - France - christophe.cerin@univ-paris13.fr

Jonathan Rivalan
Alter Way - France

Julien Sopena
Sorbonne Université, LIP6 - France

***Abstract*—This article presents the Wolphin joint R&D effort between French labs (LIP6, LIPN) and companies (Alter Way, Objectif Libre, Gandi) to tackle the management and scheduling optimization problems in an unknown context, yet critical due to their exponential usage, of containers based infrastructures. The project ran from Jan 2017 to Feb 2019. First, we analyze the differential factors from the legacy VM (Virtual Machine) virtualization and scheduling principles to their containers counterpart. Second, this article showcases research and industrial strategies to optimize resource management for elastic loads, with offline and online demands and model-based approaches. Last, we provide with insights through an Open Source prototype implementation and we synthesis the research papers we published during the project duration. Finally, the paper concludes with some perspectives for the future and discusses lessons learned from the joint effort. Overall, the paper is reporting the Wolphin experience and results, a project managed by industrialists. It serves as a crossroad between academic researchers and software professionals in cloud technologies.**

CURRENT trends in software development and delivery are aiming for shorter iterations between the software engineers teams and the system administrators ones, both in packaging and deploying new versions of the applications, pushed for a technological change at the virtualization level. Strategies of cloud providers are also evolving into the direction of serving any types of workload. Alibaba for instance, serves both online services (aka long running applications) and batch workloads colocated in every machine in the cluster. Over the past year, in Alibaba clusters, the scale of colocation of

online services and batch workloads have greatly increased, resulting in the improvement of the overall resource utilization.

Virtualization through virtual machines [1], found limitations within its core design. Vertical elasticity (extending the resources of a VM on one server) is limited by the physical layer (optimized in regards of a provision planning) and by the ability of the OS installed in the VM to acknowledge, at runtime, this addition. In a similar manner, the time to achieve the horizontal scalability (adding more VMs instances on the hypervisor) is strongly tied to the storage speed, as well as the VM image size, ranging from hundreds of MB to few GB.

The vertical elasticity and horizontal scalability limitations outline a lack of resource management dynamicity in the VM technology, amplified by the need to package a full Operating System (OS) along with dedicated system libs in the deployment workflow. The difficulty is addressed through a mandatory complex image build phase, whose duration limits update iterations, opening a risk for possible security flaws.

LXC (Linux containers) and their modern higher level implementations [2], such as Docker, Kubernetes, Mesos, provide a broad new approach in leveraging these aspects. Based on the host kernel, limiting the virtualization layer to cgroup isolation, the Docker engine provides a layered file-system enabling the declarative description of an application stack with Docker images. Services executed through the Docker engine are known as containers. The Docker layered feature enables fast provisioning for an application, since the building time is limited to either downloading the images and bind them, or only binding them if they already exist on the host system.

Modern container systems [2] rely also on key components, among them the components for storing data. Union file systems (UnionFS), are file systems that operate by creating layers, making them very lightweight and fast.

On the application architecture side, description of system libs dependencies through a YML declaration, has the advantage to provide an immutable description for the application dependencies, limiting the risk for undesired system updates, possible source for a shift in internal system APIs that could result in undesired behaviors or bugs. Such mechanism facilitates the spawn of a new application instance on any infrastructure node.

Consequently, the container technology of the Wolphin project is based on Docker. Docker also offers horizontal elasticity through native load balancing, and also vertical elasticity thanks to system-based process management and service discovery.

The contributions of this paper address the weakness of the current containers systems and challenges for a better consideration of dynamic loads. Starting with the taxonomy of container-based cluster orchestration systems as introduced by Rodriguez and Buyya in [3], the Wolphin project contributes at the following levels:

- Cluster manager: we propose a novel view of requests scheduling based on multi-criteria approaches (exact and approximated) and also based on qualitative and quantitative criteria, hence a contribution for resource and QoS requirements for requests;
- Resource monitor: we propose a new mechanism for controlling the memory usage hence a contribution at the level of resource requirements in the kernel. We also propose some integrated metrology approaches.

The organization of the paper is as follows. In Section *Problem and constraints* we introduce the elasticity problem we are faced to, and some methodological elements. Section *Proposals of Wolphin members in detail* sketches the proposals of all the Wolphin project members. Section *Summary and comments on the contributions* is dedicated to a deep comment of quantifiable results. Section *Conclusion* draws perspectives.

Problem and constraints

Elasticity

Containers approaches induce counterparts [4,5] that could put applications or infrastructures at risk, without a proper architecture taking into account the various defunct scenarios, dynamic elasticity of resources consumption being the most notable. Although the idea of a service, consuming the exact needed resources and scaling them up or down graciously, is elegant, the

downside, from a scheduling point of view, is its complexity.

By default, the Docker engine accepts any new micro service provision request within its host. In this situation there is no warranty that the host may offer sufficient resources to fulfill every micro services needs when they will reach a peak of activity or will flip to a regular scenario. In extreme situations, the lack of warranty not only implies a degradation of services quality or uptime, but may also result in an out of memory exception (OOME), hence a fault.

Moreover, available strategies within the scheduling Docker ecosystem components are limited to only three strategies: random, which will place the service on any of the available nodes ; spread, that will try to optimize the resources of the less used nodes by placing containers equally on the cluster; and finally Binpack that uses an optimization algorithm in order to pack as many containers as possible on each node, reducing fragmentation and therefore using fewer servers. None of the strategies solve the dynamicity problem.

Optimization One of the problems we faced within the Wolphin project, was to actually offer optimizations to the placement problem, mainly build up on unknown elements or uncertainties for the resources usage [6].

First of all, the inability to finely define needed resources for a given container, is based on the observations that, at the placement time, the requests number, the sizes and the resulting computational needs are not known to the user submitting the request. The classical approach is an empirical one, where the user, most likely a system administrator, defines resources from a prior experience with a similar service or type of service. Another approach, is to actually limit resources to their minimum and gradually increase them as the container will not boot up, until it meets its requirements.

Both approaches have inherent downsides. First, they reduce the domain customization by applying opinional parameters based on a human expertise; second, they could end up in reserving more resources than needed for the service. Third, the approach may actually limit the container efficiency by not providing sufficient resources to serve a medium level of services.

In summary, limiting resource settings of containers by configuration is counter productive in most cases, and does not take advantage of their inherent elasticity characteristics; conversely, not providing limits, even in a known application context, will indubitably result in a risk for infrastructure failures.

Methodological Elements and Industrial domain specific

As a service provider, whose expertise is in providing its customers with resilient web platforms based on Open Source solutions, Alter Way provided a specific use case for the Wolphin project. How to achieve a 99% availability at the lowest possible cost, meaning that the fewest possible resources were to be consumed, both by optimizing containers resources consumption and provide consolidation by limiting, at best, their fragmentation on the infrastructure.

On the other hand, academics participants in the R&D effort, specifically researchers at LIPN, had a very different use case, as they able to find the best possible placement in order to compute scientific workloads in the shortest time. They developed a custom scheduling framework for the Grid'5000 computing infrastructure mutualised between scientific entities.

On top of these two different objectives (availability versus computational time), other parameters were to be taken into account in the Wolphin project. First, the different SLAs that Alter Way could propose to its customers or more likely infrastructure options, could also require to add constraints to the scheduling efforts, such as disk type and available bandwidth. Second, the valorisation map used as a billing driver should be fixed in advance, as it was the case and still is in most hosting providers. The billing can also be a combination between fixed and dynamic, assessing the customer with both a minimum guaranteed support or quality of service, and scalable options to ensure quality of experience to its customers.

Proposals of Wolphin members in detail

This section introduces significant contributions obtained both by academic actors and industrialists actors. This section is a general discus-

sipon and it also considers cooperative work done between the actors. Then, in a dedicated section we comment the quantifiable results.

Academic proposals

Scheduling and Allocation Framework for Containers (SAFC) [8, 9, 10, 11] is a tool to experiment with our ideas for scheduling containers. This tool, originated at the LIPN laboratory, allows emulation, not simulation, in creating LXC containers, on demand. The starting point of our scheduling mechanism is on relying on the assumption that the user is not an expert in the choice of the number of resources to allocate to his job/container. For her/him, the SAFC system computes the exact number of resources, dynamically. The user specifies a class of service (Premium, Advanced and Best effort) for qualitative and quantitative criteria, and the SAFC decides for her/him, for instance, to allocate between 8 and 12 cores. Notice that our scheduling principle is based on a range a values and no more on a fixed quantity of resource as with AWS, Alibaba and many other clouds. Our technique allows to relax the constraints on the underlying optimization problem which is, given a set of user requests, to find an allocation such that the number of resources is minimized and such that the user satisfaction is maximized.

The idea is that the system could properly answer the question ”how many resources are required to satisfy a request?” and better than any user. This is one of the motivations of the work, the other being the ”navigation” between the compromises that are offered to the user. We may ask if the system could do better than an experienced user. We argue that in such a multi-criteria context a human will never be capable of deciding in an optimal way.

The LIP6 laboratory focused on the dynamic allocation of memory resources, with an emphasis on the memory sizing of the containers. In effect, Linux uses free memory as a disk cache, which means that it does not always need to access the (hard) drive, to fetch some data. It is therefore crucial to be able to correctly size his cache so that the performance of the applications is not degraded. We are, at first, interested in a metric already present in the Linux kernel, the ”refault distance” that could help determine whether a container would benefit from being made larger. If the idea to use this metric is taken by many authors, we have highlighted that it has too much imprecision to help the sizing of containers.

In consequence, we developed an algorithm to decide the precise amount of additional memory the container needs. After theoretically proving bounds on its accuracy, we experimentally validated our new metric on specific applications, as well as on the benchmarks of the literature, namely Sysbench [12] and Perfkit [13]. We then developed, via the sysfs file system, a textual interface allowing the Linux kernel to expose users to the computed metric values over multiple temporal windows (default: 1min, 5mins, 15mins).

Industrialists proposals

Alter Way and Gandi contributed at the metrology level and implemented a set of components that can be used to collect metrics and data from Docker-based clusters. The innovation addressed of this item is the possibility of collecting service data (containers) running during very short amounts of time (on the order of a few milliseconds), thereby escaping the conventional mechanisms of information collection.

Post-mortem metrology Alter Way proposed to leverage this limiting factor through a ”post-mortem” implementation, meaning the extraction of the data relating to a service, not over a defined time range (the frequency of collection) but at its time destruction by the system. Alter Way’s work on this issue has resulted in an implementation in the original ContainerD project[1] The problem addressed by their implementation is linked to the heart of the Wolphin project: the precise measurement of resources consumed, in order to report information to the customer and charge them.

The current implementation integrates with the collection mechanisms already present in ContainerD. This is meant to collect data from Linux cgroups and then expose them to the next inspection of the agent. This implementation has the advantage to keep the formalism of the data already exposed in Containerd, which avoids hav-

[1] https://github.com/containerd/containerd/pull/1586

ing to make a specific implementation to process them.

Data management The data storage issues related to the metrology deliverable were addressed in order to avoid design errors as well as to facilitate the integration of components from the various teams involved. We quickly identified the main difficulties, and concerns are the interoperability with current monitoring solutions, their performance in compliance with a specific type of query, latency, elasticity, the maturity and the community support. In order to get confidence into the state-of-the-art systems, we evaluated the InfluxDB, RRDTool, Elasticsearch, Graphite, Prometheus, DalmatinerDB, OpenTSDB and RiakTS solutions. The comparison of these solutions is available online[2].

The current architecture uses a Postgres (Timescale) extension that provides optimizations for time series. Timescale authors also provide two open source projects for using Timescale as a Prometheus external data storage[3]. The possible pipeline for Wolphin are now as follows (where the arrows depicting the flux of information between the components):

1) Telegraf (collector) $\leftarrow$ Prometheus $\leftrightarrow$ p_prom_adapter $\leftrightarrow$ Timescale
2) Telegraf $\rightarrow$ Timescale
3) Telegraf $\rightarrow$ InfluxDB $\rightarrow$ ETL $\rightarrow$ Timescale (non recommended)

Resource monitor In a container based architecture, the optimization of every entity's boundaries is primordial. To address this issue, we developed Autorange, a Docker feature that discover and apply limits (CPU%, RAM) to a given container. Implemented directly into the collector module of the docker daemon, the feature consists of a patch of docker components.

The approach is as follows. From the metrics we collect, time series are generated in regards of some key values, including highest, lowest rate of change. These values are then used as weights to generate our predictions, which are added to a prediction array. Time collection, as well as the number of time series taken into account are tied to the evolution of the consumption. Finally, we use our predicted values to generate optimal limits to be applied.

The efficiency of the solution primarily depends on the workload type. In most cases involving single core tasks, benchmarks have shown the the algorithm did not cause any drop in performance. The difference between lowest and highest latency was lightly impacted, but the average remained stable with limits. The CPU intensive tasks were impacted the most, but remained still very close to the performance without limitations.

Billing The upgrade of the CloudKitty solution (OpenStack official component) to a container context has been investigated through the evolution of the collection engine, the storage backend, and the authorization layer. In short, the Objective Libre company, within the framework of the Wolphin project, has provided a rating solution and chargeback for container metrics for the purpose of billing. Valuation reports are generated from the various measurements from the resource consumption of the containers.

It was decided that the approach selected by Objectif Libre would be based on the OpenStack chargeback and rating component, namely Cloudkitty which is a free software (Apache v2 license), and originally created by Objective Libre. Several features desired for the Wolphin project, such as the valuation of a consumption or exporting reports, were already present in Cloudkitty.

Additional developments were accomplished to extend the range of metrics that can be manipulated Cloudkitty in order to switch from an OpenStacks centric model to a more generic one. This new model has been tested according to various metrics, used in the project and in live and post-mortem contexts.

The collecting part of Cloudkitty was also extended and redesigned to embrace a large scope of data storage engine used/tested at some point of the Wolphin Project: Prometheus, Postgres and KairosDB.

Summary and comments on the contributions

Academic contributions on containers scheduling have been published in [8, 9, 10,

[2] https://spreadshare.co/spreadsheet/open-source-time-series-db-comparison-2

[3] https://github.com/timescale/prometheus-postgresql-adapter and https://github.com/timescale/pg_prometheus

11]. In [10] we presented a new scheduling and resource management allocation system based on an economic model. The goal of the proposed system is to address the problems of companies that manage a private infrastructure of machines, and would like to optimize the scheduling of several requests submitted online by users. The economic model used in this paper is based only on two SLAs (Service Level Agreements) classes (a qualitative one and a quantitative one). The novelty of the system is that it allocates dynamically, for each selected request, a set of computing cores according to the user economic model.

In [8] we introduced a variant of the problem of resource allocation. The two qualitative classes represent the satisfaction time criterion, and the reputation criterion. Moreover, the two quantitative classes represent the criterion over the number of resources that must be allocated to execute a container and the redundancy (number of replicas) criterion. The novelty of this work is also based on the possibility to adapt, dynamically, the scheduling and the resources allocation of containers according to the different qualitative and quantitative SLA classes.

The difference between the papers [8] and [10] is that in [8] the framework is proposed in the context of containers unlike the system proposed in [10] which is designed to schedule requests. In [8] the economic model is proposed with more SLAs classes than in [10].

In [11] we described in details how to give the possibility to specify ranges on the resources demand for containers, instead of specifying a fixed amount of resource. The novelty of the paper is in relaxing strict constraints on the requested number of resources and on letting the system to adjust the amount of resources according to the execution context, such as the peaks of activities. Moreover, the number of resources allocated for each container is comprised between hard and soft lower/upper bounds of cores. The hard lower/upper bounds of cores are set statically according to the selected service in the quantitative SLA class and the configuration of the cloud nodes. The soft lower/upper bounds of cores which are taken ”inside” the hard bounds, are set dynamically according to the global load of cloud nodes. The goal of the bounds on cores is to improve the quality of scheduling and to be compliant with the different SLA classes.

The difference between [11] and [8] is that in [11] the number of allocated cores is set according to a hard and soft lower/upper bounds unlike the framework proposed in [8] for which the number of cores is set according to hard lower/upper bounds only. In [11], the framework is experiment with LXC containers. However, in [8], the framework is experiment with Docker containers. Under the experimental conditions of [11] we get a degradation of the execution time of around 3% but a use of 10% less cores.

In [9] we introduced new multi-objectives scheduling strategies. The novelty of this paper is to introduce personalized multi-objectives scheduling strategies adapted for Cloud Computing environments. The principle of the proposed strategies consists to select a node which has a good compromise between multi-objectives criteria to execute a container. The proposed scheduling strategies are based on PROMETHEE and Kung multi-objectives decision algorithms in order to place containers. The implementation inside Docker Swarmkit and experiments show the potential of our approach under different scenarios such as burst mode for submissions. Using PROMETHEE and Kung strategies we demonstrated that the load of work in each node is greater than with the native spread strategy of Swarmkit. This means that the strategies enforce a better use of nodes to keep them more loaded.

Open Source software contributions are available at URL https://gitlab.com/wolphin. A showcase site presenting the functionalities of the project is also available at the URL http://wolphin.gitlab.io/. It refers to the project source code, documentation and allows to request for access to the platform. These sites expose the different parts of the work that we have abovementioned in this paper. For instance, readers will find the contributions according to the Telegraf, Fluend, Prometheus and Timescale tools for collecting ”log events”. Readers will also find the Cloudkitty modifications for the Wolphin project in two branches, namely docker-cloudkitty and cloudkitty. Least, but not last, you will find the source codes of the Wolphin frontend, the Wolphin CLI (Command Line Interface) and a simple Wolphin administration tool (Wolphin.admin)

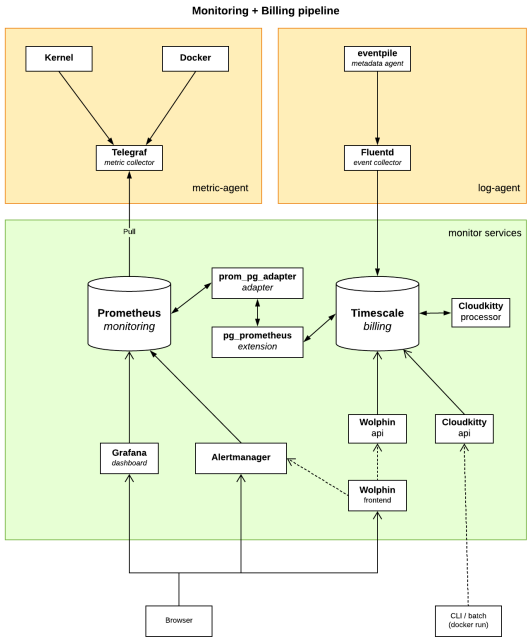

Figure 1. Architecture of the Wolphin system

which is a Docker image that could implement a GUI (Graphical User Interface) to manage Wolphin services in the future.

The architecture of the system that has been built is depicted on Figure 1. Since the Wolphin project focuses on the Docker technology, this architecture is fully implemented on top of this technology.

The modifications that Wolphin brings to Docker ”native” are as follows (see Figure 1) and they consider the monitoring of events and the billing pipeline. Each Docker node runs now two agents located on the top of Figure 1:

- The metric agent collects both host and container metrics with Telegraf, and makes them available to Prometheus through HTTP.

- The event collector exports Docker events (creation/removal of container and networks) to a Fluend instance, which can optionally collect host and container logs as well.

On the monitor node, located in the bottom of Figure 1, the metrics are retrieved by Prometheus, and a subset of them is relayed to the Timescale RDBMS using the pg_adapter and pg_prometheus extension. Once there, Cloudkitty applies the pricing rules and makes them available in aggregated form to the Wolphin API. The event logs are used to add the metadata which could not be gathered by Prometheus (notably, the network attach/detach events). The Wolphin frontend synthesizes all the information from the rated data frames and its metadata, and includes the outstanding alerts from the Alert manager component which complements Prometheus.

Conclusion

In this article we summarized the problems and solutions implemented within the Wolphin project which aimed to control dynamic loads in clouds [6]. The project ran from Jan 2017 to Feb 2019 and it has already been evaluated positively by the funders. The project has resulted in many software developments and we refer the reader to the project URLs as mentioned above. The outcome is also with academic publications; see the bibliography. There are many prospects and we have already identified some leads.

In a particular container platform, it may be necessary to terminate applications that behave erratically. A new container can then be created on a different host, and a memory leak can be avoided without requiring costly debugging. To this end, developers or system operators can apply maximum limits to the resources granted to each container or service, but they must measure and calculate themselves the desired limits for each service. In this general context, the Wolphin project has yet started to develop a system to automatically predict the optimal limits for memory and CPU utilization. The calculated limits can then be sent to the system metric collector. A better approximation allows the Scheduler container to better plan the placement on each host, in order to maximize the number of containers for each server. The result of this ongoing work was yet proposed in the form of a set of patches to the following projects and many works are still to be done:

1) Docker CLI: https://github.com/docker/cli/pull/1664
2) Swarmkit: https://github.com/docker/swarmkit/pull/2818
3) Moby/docker engine: https://github.com/moby/moby/pull/38706

Indeed, for recognition of any Open Source software development work, the community is

sensitive to contributions in the form of pull request, in new branches, or in well-established projects. It is a mark of success. In the academic sector, recognition is not done by producing lines of code in large projects, but by lines of text that are peer-reviewed. We hope that our article will bring together the different points of view related to the management of dynamic loads in clouds environment, and for a better mutual understanding between the industry and academic sectors. And for that issue, we adopted a style close to an industry/academic report or, much more simply, a project report.

Acknowledgment

The authors wish to thank the funders and main facilitators, Banque Publique d'Investissement (BPI), Région Ile-de-France and our respective universities (Paris 13 and Sorbonne Université). We also express are sincere thanks to all the project members, among them Valentin Daviot, Tarek Menouer, Étienne Leclerc, Frédéric Roupin, Damien Carver, Maxime Bittan, Marco Mariani, Danilo Cerovic, Claire Gayan, Christophe Sauthier.

Least but not last, we thank Jean-Luc Gaudiot from the university of California at Irvine for his valuable comments, notably those accomplished during his stay at Paris 13 university as an invited professor. Some experimental work has been conducted on the Grid'5000 testbed, and we are grateful to the team for their help accessing the machines. Grid'5000 is supported by a scientific interest group (GIS) hosted by INRIA and including CNRS, RENATER and several Universities as well as other organizations.

REFERENCES

1. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the art of virtualization. In Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03). ACM, New York, NY, USA, 164-177. DOI: https://doi.org/10.1145/945445.945462
2. Pahl, Claus & Brogi, Antonio & Soldani, Jacopo & Jamshidi, Pooyan. (2017). Cloud Container Technologies: a State-of-the-Art Review. IEEE Transactions on Cloud Computing. PP. 1-1. 10.1109/TCC.2017.2702586.
3. Maria Alejandra Rodriguez, Rajkumar Buyya: Container-based cluster orchestration systems: A taxonomy and future directions. Softw., Pract. Exper. 49(5): 698-719 (2019)
4. Herbst, Nikolas; Samuel Kounev; Ralf Reussner (2013). Elasticity in Cloud Computing: What It Is, and What It Is Not. Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013), San Jose, CA, June 2428.
5. Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar. Specifying, Monitoring, and Controlling Elasticity of Cloud Services, Proceedings of the 11th International Conference on Service Oriented Computing. Berlin, Germany, 25 December 2013. doi=10.1007/978-3-642-45005-1_31
6. W. Lin, S. Xu, J. Li, L. Xu, and Z. Peng. Design and theoretical analysis of virtual machine placement algorithm based on peak workload characteristics. Soft Comput., 21(5):1301-1314, Mar. 2017.
7. The NIST Definition of Cloud Computing. Peter Mell and Timothy Grance, NIST Special Publication 800-145 (September 2011). National Institute of Standards and Technology, U.S. Department of Commerce. Publication available online at https://csrc.nist.gov/publications/detail/sp/800-145/final
8. Tarek Menouer, Christophe Cérin, Walid Saad, Xuanhua Shi: A Resource Allocation Framework with Qualitative and Quantitative SLA Classes. Euro-Par Workshops 2018: 69-81
9. Tarek Menouer, Christophe Cérin, Étienne Leclercq: New Multi-objectives Scheduling Strategies in Docker SwarmKit. ICA3PP(3) 2018: 103-117
10. Tarek Menouer, Christophe Cérin: Scheduling and Resource Management Allocation System Combined with an Economic Model. ISPA/IUCC 2017: 807-813
11. Tarek Menouer, Christophe Crin, Congfeng Jiang, Jonathan Rivalan: SAFC: Scheduling and Allocation Framework for Containers in a Cloud Environment. International Conference on High Performance Computing & Simulation (HPCS 2019), Jula 15-19, Dublin, Ireland
12. Kopytov A., SysBench manual, MySQL AB, 2012, source: https://github.com/akopytov/sysbench ; manual: https://www.yumpu.com/en/document/read/17130663/-sysbench-manualpdf
13. Perfkit Benchmarker. Project home page: https://github.com/GoogleCloudPlatform/PerfKitBenchmarker

Ahmed Amamou (Member IEEE) is head of research engineer at GANDI SAS. He received a Ph.D. degree in network and computer science from the University of Pierre and Marie Curie in 2013, Paris, France. His research interests are Cloud computing, virtualization technologies and machine learning. E-mail: ahmed@gandi.net

Martin Camey is a cloud consultant who is working with Objectif Libre. Holder of a master in computer science he joined Objectif Libre as a python software developer for Cloudkitty, the rating solution that is developed by the company under the OpenStack's umbrella. E-mail: martin.camey@objectif-libre.com

Christophe Cérin (Member IEEE) has been a professor of computer science at the Université Sorbonne Paris Nord (former name Université Paris 13), France since 2005. His research focuses on High Performance Computing, including Grid and Cloud Computing and he develops middleware, algorithms, tools and methods for distributed systems. E-mail: christophe.cerin@univ-paris13.fr

Jonathan Rivalan is currently a R&D manager with Alter Way, Jonathan pursues, with his team, technological enhancement in innovative fields such as optimization for micro-services based cloud computing, and workflow automation through machine learning. E-mail: jonathan.rivalan@alterway.fr

Julien Sopena obtained his PhD in 2008 at the University Pierre et Marie Curie (UPMC). His research interests include operating system, resource management, distributed algorithms and sat solver. E-mail: julien.sopena@lip6.fr