# DockerSim: Full-stack Simulation of Container-based Software-as-a-Service (SaaS) Cloud Deployments and Environments

Zahra Nikdel, Bing Gao, and Stephen W. Neville
Department of Electrical and Computer Engineering
University of Victoria
Victoria, BC, Canada

***Abstract*—Container-based Software-as-a-Service (SaaS) systems are rapidly emerging as a dominate cloud deployment paradigm, as supported via Docker, etc. This has opened new research avenues as lighter-weight containers replace heavier-weight virtual machines (VMs) as atomic cloud deployment units. Advancing such research requires sufficiently rich, container-aware cloud simulation frameworks so as to explore issues such as optimal cloud orchestration and elastic service strategies, achieving assured application-layer quality of service (QoS), etc. This work presents DockerSim, an extension and augmentation of the iCanCloud OMNet++ cloud simulator to incorporate: i) a full container deployment and behavioral layer, ii) full packet-level network and protocol behaviors, iii) full multi-layer OS process scheduling behaviors, and iv) a generic queuing network approach to modeling application-layer SaaS deployments. To our knowledge DockerSim is the first cloud simulation platform fully incorporating capabilities (i)-(iv), as required to rigorous explore a wide range of timing sensitive container-based computing cloud and SaaS-deployment issues.**

***Index Terms*—Cloud Computing, Simulation, Docker-style container simulation, Container-as-a-Service(CaaS), Full-stack cloud simulation, Software-as-a-Service (SaaS)**

## I. Introduction

It has long be recognized that cloud computing research advances can be accelerated through the availability of sufficiently rich and scalable cloud computing simulation platforms, such as via CloudSim [1], iCanCloud [2], etc. As arose previously in network engineering, simulation-based research is an attractive paradigm in cloud computing, in part, as the scale and complexity of modern commercial cloud platforms (e.g., Amazon EC2, Microsoft Azure, etc.) render it very difficult to achieve the scientific method's control and repeatability tenets within actual cloud environments. Moreover, confounding variable effects become difficult to control and isolate for. Simulation-based methods offer a cost-effective means of exacting closer per-experiment control and, when combined with Monte Carlo simulation methods, offer paths to explore performance envelopes and sensitivities in manners that are largely cost-prohibitive in commercial cloud settings.

Simulation, of course, cannot replace real-world testing but, when properly done, it can provide cost- and time-effective paths by which promising methods and approaches can be identified and characterized ahead of engaging in higher cost and richer fidelity full-scale in situ testing. By definition, simulations abstract out real-world fidelity. Hence, where, when, and why fidelity is lost must be clearly understood and wherever possible sufficient maintained so as to ensure simulation-based results and insights maintain real-world relevance.

Importantly, simulation-based cloud research's end-goals need to be clearly differentiated from the the use of commercial clouds as platforms by which to pursue research (i.e., as supported via commercial cloud platform research allocations, national-level computing platforms, etc.). In general, the latter traditionally expressly forbids research activities that have significant potentials to adversely impact a platform's other users. Hence, actively evaluating new and proposed cloud resource consolidation, orchestration, and elastic service approaches generally fall into the domain of disallowed research activities. Moreover, commercial cloud platforms are structured and designed to support commercial software deployments and not the scientific method's per-experiment control and repeatability tenets. As such commercial cloud platforms introduce additional sources of process variation that, from a formal research perspectives, are very difficult to impossible to control for via standard cloud deployment APIs. Commercial cloud platforms also tend to be of scales and complexities that exclude analytical tractability. Overall, this leaves cloud simulation as the clearly viable path by which to explore and vet proposed approaches so as to identify promising candidates for subsequent higher fidelity real-world and at-scale evaluations.

Clearly, a cloud simulator's value in developing real-world useful research insights hinges on its supported level of detail and fidelity. Within existing cloud simulators (e.g., CloudSim, iCanCloud, etc.), a common choice has been made to trade-off network- and packet-level fidelity in favor of increased simulator performance. To our knowledge, no available cloud simulator currently supports or appropriately models: i) the increasingly commonplace container-based cloud deployment regimes, ii) full packet- and protocol-level cloud network behaviors, iii) the realities of in-cloud multi-layer OS process scheduling behaviors, while iv) providing generic support for modeling application-layer behaviors, such as those of SaaS system deployments.

978-1-5386-0700-8/17/$31.00 ©2017 IEEE

In our view, supporting the (i)-(iv) details is critical to obtaining useful insights regrading cloud computing's timing sensitive aspects, such as developing optimal cloud orchestrations, resource consolidation, and elastic service solutions, as well as developing deeper formal understandings as when and why QoS guarantees can (or cannot) be met. In part, our view is that on-going advances in multi-core systems and high performance computing (HPC) now allow (i)-(iv) to be incorporated without incurring what has previously been considered to be overly onerous simulator performance penalties.

This work presents DockerSim, our extension and augmentation of the existing iCanCloud OMNet++ based cloud simulator so as to incorporate: i) a Docker simulation and behavioral layer, ii) full packet, routing, and network protocol behaviors, iii) full host kernel, VM, and container OS-level scheduling behaviors, and iv) a general queuing network (QN) approach to modeling SaaS application layer behaviors. More specifically, (iv) involves each simulated Docker container hosting a SaaS application-layer queue (or QN) which is then interconnected via fully simulated real-world equivalent TCP/IP networking processes to other simulated Docker container hosted queues (or QNs). On composite, these inter-communicating container-hosted queues then form and properly simulate a container-deployed SaaS system's full application-layer information flows. DockerSim is intended to be wrapped in our previously developed statistically rigorous Monte Carlo-based testing harness so as to exploit HPC to enable statistically rigorous exploration of cloud-deployed SaaS system performance envelopes and cloud management strategies.

The remainder of this work is structured as follows. Section II introduces the related work in the area of cloud simulation. Section III presents the details of DockerSim architecture and design choices. Section IV then details the initial results obtained from DockerSim for a basic SaaS simulation. Section V then discusses DockerSim's current limitations and the planned extensions to address them. Section VI then concludes the work.

## II. Related Work

The significant research benefits of cloud simulation platform has lead to several developed simulators each with their own tragetted objectives. The most widely used of these are iCanCloud [2], CloudSim [1], MDCSim [3], and GreenCloud [4], with other existing simulators tending to be variations of these core frameworks. These core simulators tend to focus, to varying degrees, on the details of virtualization techniques and application-layer behaviors. The details required to simulate modern container-based cloud deployments are generally lacking. ContainerCloudSim [5], an off-shoot of CloudSim, which in turn was an extension of GridSim [6], does provide a degree of container support.

More specifically, ContainerCloudSim augments CloudSim's core with an added layer to provide the ability to simulate the emerging Container-as-a-Service (CaaS) paradigm. But, a known CloudSim limitation is that the simulated cloud's underlying network behaviors are abstracted out and replaced by statically configurable and in-experiment prescribed network delay parameters. Network layer behavioral dynamics and protocol behaviors are, therefore, largely abstracted out of CloudSim and ContainerCloudSim. As a result, for example, the performance costs associated with container (or VM) migrations, a non-trivial factor in cloud orchestration and resource consolidation methods, can only be roughly estimated as opposed to directly simulated. Such details become important and potentially critical in seeking insights into timing sensitive and dynamic cloud behaviors such as exploring when, where, and how QoS statistical predictability is gained (or lost) within SaaS system cloud deployment regimes and elastic services. More directly, DockerSim and ContainerCloudSim seek to address different issues in cloud-based CaaS environments. ContainerCloudSim's focus centers on the critical problem of estimating and exploring data center power consumption issues and reduction strategies, whereas DockerSim targets statistically rigorous assessments of the performance envelopes of cloud deployed systems and the trade-offs that exist between SaaS system deployment costs and their statistical QoS performance envelopes.

ContainerCloudSim extended CloudSim to incorporate the modeling of containerized cloud environments while also adding support for incorporating cloud data center resource utilization traces. Currently, ContainerCloudSim employs the workload models of PlanetLab [7] to provide estimated containers usage data. Known differences though exist between PlanetLab observed performance measurements and those of actual commercial cloud platforms, in part as a result of PlanetLab's custom VM and network overlay approach. DockerSim's approach is to instead model the container resource usages characteristics via the in-container run-time application-layer processes it hosts. DockerSim also exposes OMNet++'s user interface capabilities to provide a graphical interface by which experiments can be step-up, executed, and visualized, capabilities not currently supported in ContainerCloudSim.

Unlike DockerSim, ContainerCloudSim does not seek to incorporate the (i)-(iv) capabilities discussed above, thereby, placing DockerSim in a better position to support research into timing sensitive cloud computing issues whereas ContainerCloudSim is better positioned to explore energy consumption issues. DockerSim, via its OMNet++ underpinnings, does not require code level changes to enable explorations of (or into) new cloud architectures, resource provisioning strategies, etc. Instead, these leverage OMNet++'s existing well-vetted capabilities. iCanCloud [2] is one of the most popular open-source cloud simulation platforms. It provides good trade-offs between flexibility, accuracy, performance, scalability, and ease of use. iCanCloud has been shown to be capable of modeling and simulating large environments up to thousands of active nodes. iCanCloud gains these advantages via its use of SIMCAN [8] which, in turn, leverages OMNeT++'s in-place network simulation capabilities. This allow iCanCloud to

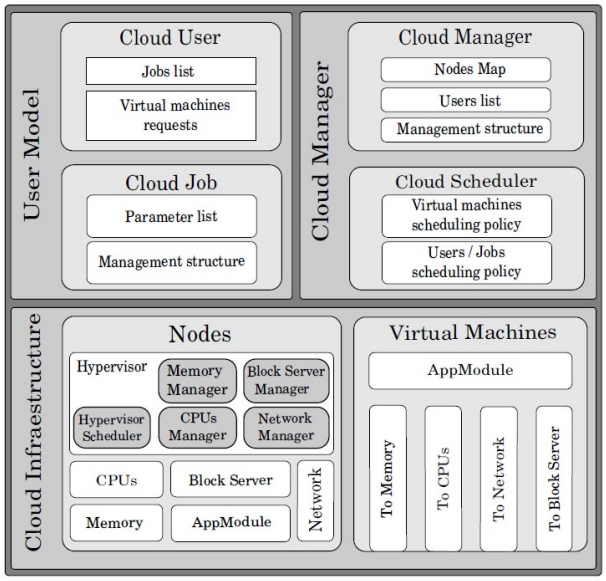


Fig. 1. The modular structure of iCanCloud [10].

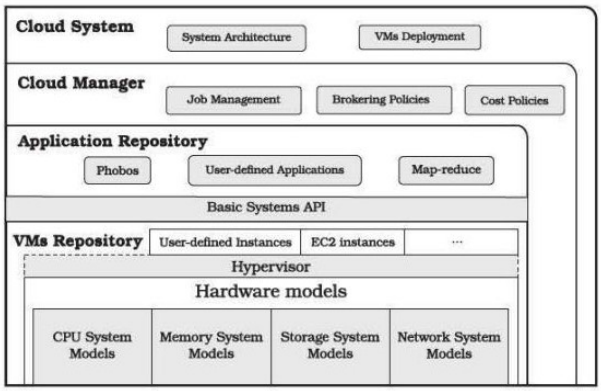


Fig. 2. iCanCloud modular and layered architecture [2].

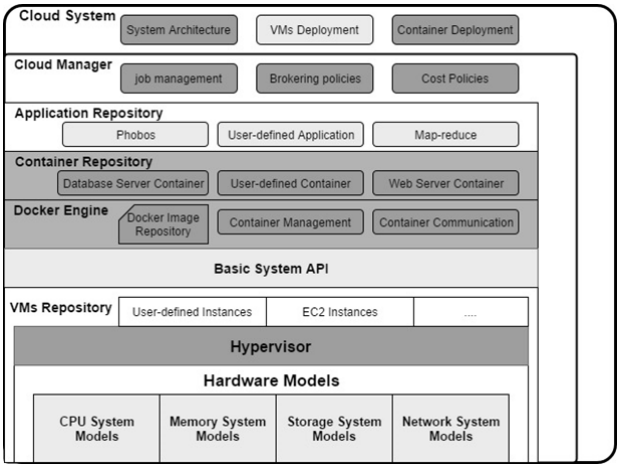


Fig. 3. Introduced DockerSim modular extensions and augmentations to the iCanCloud architecture.

model network communication between machines using OMNet++'s INet full-protocol simulation framework. OMNet++ also supports parallel simulation enabling increased scalability, where parallelism is currently not supported in many alternate simulation platforms (e.g., CloudSim [1], GreenCloud [4], and CloudSched [9]). Hence, iCanCloud provides the appropriate bridging off place for DockerSim.

iCanCloud is built on a modular architecture in which the three main code-level modules are grouped by functionality into the: user model, cloud infrastructure model, and cloud manager model [10], as shown in Figure 1. DockerSim extends and augments iCanCloud with the (i)-(iv) capabilities so as to include the modeling and simulation of Docker containers along with richer support for simulating OS-level process scheduling characteristics, while also enriching iCanCloud's network-level behavioral support.

MDCSim [3] is another major cloud simulator supporting the in-depth analysis of multi-tier data centers. Unlike the other simulators, it is a fully commercial cloud simulation toolbox designed for analyzing performance and power consumption issues in large/complex computing environments and distributed systems. It focuses on enabling the evaluation of various resource management strategies so as to optimize application-layer performance while minimizing power consumption. The topology of the data center is supplied as a directed graph by the MDCSim network package, but unlike iCanCloud based approaches, MDCSim does not seek to incorporate detailed networking and TCP/IP protocol behaviors. GreenCloud [4] is an open-source power management cloud simulator built on top of the widely used ns-2 network simulation platform [4]. Ns-2 though is known to suffer from performance constraints which, in part, is fostering the transition towards ns-3. GreenCloud does not seek to provide support for container-base SaaS deployments or OS-level process scheduling behaviors. DockerSim was developed to fill the unaddressed gap for a cloud simulator that fully supported the (i)-(iv) capabilities as required to enable statistically rigorous research into the wide variety of cloud computing's timing sensitive application-layer SaaS deployment issues.

## III. DockerSim Architecture

DockerSim follows iCanCloud's layered and modular architecture [2], as is illustrated in Figure 2. DockerSim introduces additional layers, such as the Docker Engine layer, while modifying other existing iCanCloud layers so as to support the delivery of the services required to support the introduced Docker containers. The dark gray modules of Figure 3 show more directly the extensions and augmentions DockerSim provides to iCanCloud's original architecture.

Docker Containers are distinct from standard applications running inside of VMs in several ways, including:

- Containers need to be isolated from each other.
- Each Container may have its own name-space.
- Containers usually run a single service, which may be compose of multiple processes.

- Containers share (and compete for) resources (CPU/memory/storage/network) on the host virtual machine.
- Containers co-hosted within a VM communicate via the Docker engine (docker0 port).
- Containers hosted on separate VMs communicate through TCP/IP protocols.
- Docker Containers possess their own network interfaces and IP tables.

To support Docker Container simulation, DockerSim incorporates a developed Docker Engine module, which rests top of iCanCloud's original Basic System API layer. This Docker Engine module is responsible for supporting the operating environment for simulating container-deployed SaaS applications by providing the interfaces and underlying support for simulating the building and running of containers. SaaS application-level components are modeled as queues (or queuing networks (QNs)) running inside isolated containers. These containers are modeled so as to share the same OS kernel when containers are co-hosted on the same physical cloud server. Intercommunication between a simulated SaaS system's application components is then performed via OMNet++'s standard network communication layers, e.g., via a fully simulated cloud computing facility's server, rack, top-of-rack switching, hierarchical networking layer(s), and employed network protocol behaviors, as per how such communications would unfold within an actual cloud platform.

The most important difference between standard applications and jobs running on VMs, which DockerSim also supports, and applications and jobs running within containers is that container-based applications entail higher levels of isolation such that only one service runs per-container. For example, a simulated web application may involve several different inter-acting containers located on different VMs which themselves may be on different physical hosts (or servers) which may even be on different racks. DockerSim incorporates a Docker Container Repository such that collections of predefined Containers can be easily added to and instantiated via the DockerSim developed API.

Each container is modeled as having its own job queue, ensuring real-world like inter-container isolation. Containers within the same VM can communicate via the Docker Engine module's supported message passing constructs. To better model SaaS systems (or general distributed systems) the container model is in the process of being extended so as to support full standard distributed system QN models, which can be easily supported via OMNet++'s own underlying capabilities. SaaS systems will then be model in terms of an interconnection of intercommunicating queues (i.e., a QN), where under DockerSim each queue will be "deployed" within a simulated container and their intercommunications will occur via DockerSim's fully modeled cloud networking layer, which in turn utilizes OMNet++'s full TCP/IP protocol, packet transport, and routing/switching simulation support. This provides DockSim with a generic method by which to model SaaS system application-layer information flows while preserving explorations into their full timing behaviors while abstracting out the complexities of a SaaS system's code level details.

Incorporating code-level details generally requires transitioning from simulation into full fidelity emulation, which is not DockerSim's intent. A choice has been made to preserve packet transmissions and OS-level process scheduling details while relying on traditional QN approaches to model SaaS system's application-layer behaviors. It should be noted that these application-layer QNs are fully simulated, as opposed to addressed analytically via Markov methods. Hence, DockerSim can incorporate self-similar, heavy-tailed, and long-range dependent behaviors that can be untenable to explore analytically.

DockerSim supports the modeling of servers within racks, VMs within servers, and containers within servers and/or VMs, with container-to-container, container-to-VM, and VM-to-VM inter-communications fully supported across all possible deployment regimes and combinations. This allows DockerSim to provide a useful platform by which to explore general cloud orchestration and resource consolidation strategies. Moreover, via OMNet++'s underlying services, changes can be quickly and easily made to the nature and characteristics of the DockerSim simulated cloud platform, enabling comparison evaluations and research into effective and efficient cloud architectures for supporting specific flavors of SaaS systems (or distributed systems in general). Exploiting such cloud architecture comparison capabilities though innately requires substantial HPC resources, particularly if statistically rigorous research outcomes and insights are sought. We are in the process of integrating DockerSim into our previously developed HPC framework for supporting statistically rigorous simulation-based network and software engineering research via Monte Carlo methods so as to support this and other computationally intensive DockerSim research directions.

DockerSim also incorporates a Cloud Manager upper layer module responsible for managing all incoming jobs, and container and VM instances. As DockerSim in further extended, this module will incorporate cloud costing and brokering details so as to allow for research into SaaS QoS versus costing concerns. DockerSim's top layer consists of a Cloud System module containing the full definitions of the hypervisors, VMs, and Containers. The full class diagram of DockerSim simulation platform is shown in Figure 4 and consists of: the classes representing a cloud environments hardware components, the modules responsible for managing these so as to form a VM-based cloud system, and the modules and class responsible for overlying container-based operations into the VM-based architecture.

Clearly, it would be possible to construct a container-focused commercial cloud so as to support containers directly, thereby removing the extra level of abstraction the VM layer introduces. DockerSim has been structured to support the general case whereby containers and VMs may co-exist within a clouds servers. Hence, DockerSim supports the full scope of deployment regimes, namely: i) applications within VMs, ii) applications within containers within VMs, iii) applications

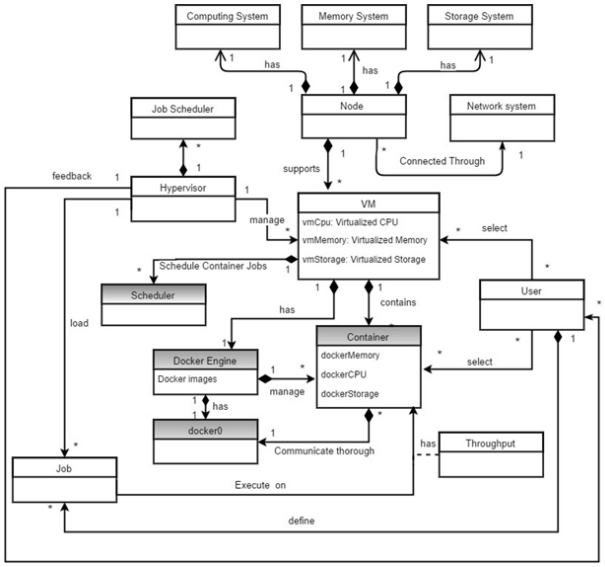


Fig. 4. Full DockerSim class diagram.

within server-hosted containers, and (iv) applications spanning any combination of (i)-(iii).

DockerSim's core extensions and augmentations to iCanCloud can be summarized as follows:

- Introduction of the Docker Engine module to address container-based deployments, inclusive of their underlying OS scheduling issues while supporting full-stack network communications.
- Restructuring iCanCloud's existing VM support so as to incorporate richer OS scheduling and full-stack network communications concerns.
- Re-engineering iCanCloud's existing inter-server and inter-rack VM-to-VM communications so as to incorporate container-to-container and container-to-VM TCP/IP communications while ensuring all communications involve fully modeled packet payloads, routing and switching operations, and protocol behaviors (i.e., as as to ensure correct information transport timing characteristics).
- Introduction of a full network-interconnected application-layer queuing network model so as to model general SaaS system and distributed system cloud deployments[1].

DockerSim has also been structured to support the dynamic scaling of the VM and container resources, so as to support research into elastic services. Such dynamical resource adjustments may be in response to a simulated SaaS system's incoming workload dynamics or closed-loop feedback approaches based on a simulated SaaS system's observed QoS performance. Clearly, a sufficient degree of statistical QoS predictability is required to support the latter, given that behavioral predictability is a foundational tenet of formal control theory. A core motivation for DockerSim's developed was the need to develop a cloud simulation framework that was

[1]At the time of publication, this feature is actively under development and on-going testing.

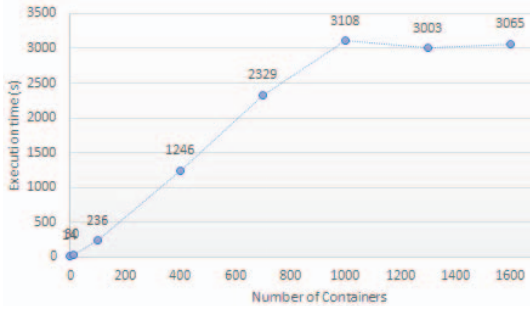


Fig. 5. DockerSim wall time required to simulate Scenario 1 versus the number of simulated containers.

sufficiently rich to explore more general software engineering research concerns into how, when, where, and why QoS statistical predictability may be gained (or lost) within cloud computing environments and SaaS deployment regimes.

DockerSim's QN modeling approach for simulating application-layer SaaS behaviors provides the additional advantage that standard workload modeling tools and approaches can be applied [11]. For example, the Tsung workload generation tool [12] can be used to produce statistically rich incoming workloads into the DockerSim simulated SaaS system development (e.g., workloads incorporating heavy-tailed, self-similar, and long range dependent behaviors). Hence, this allows DockerSim to become a useful framework by which to explore open questions, such as raised in [13] as to whether their predictive QoS modeling approach extends to bursty traffic scenarios. This approach also enables real-world collected SaaS traffic traces to be directly replayable back into DockerSim simulated SaaS systems, thereby, allowing explorations into SaaS system architectural tunings and potential bottlenecks.

## IV. SaaS Simulation Results

In order to obtain initial results of DockerSim's capabilities with respect to container-based SaaS cloud deployments, we conducted a set of experiments to simulated the behaviors of a typical web server servicing 1 hour's worth of incoming web requests. Within DockerSim, the web server is modeled according to its incoming HTTP request size, its requests per hour rate, and the processing time required per request measured in the millions of instruction (MI) executions (i.e., so as to accommodate different simulated server capabilities). Each web server is simulated as being deployed within its own Docker Container. All experiments involve DockerSim running on a Intel i5 core computer with 8 GBytes of internal memory. For Experiment Scenario 1, the container-deployed web server is configure to have an input size of 1,246 KBytes, with a 15,000 request per hour rate, with each request requiring 10,000 MI to service. Figure 5 shows the actual wall clock time required for DockerSIm to simulate Scenario 1 for varying numbers of simulated executing containers.

Figure 5 highlights a linear relationship between the DockerSim's simulated container-deployed web server count and

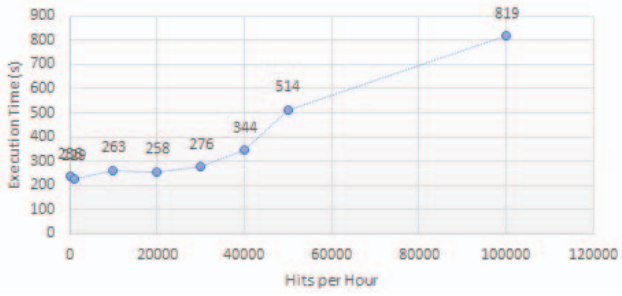


Fig. 6. DockerSim wall time for Scenario 2 execution versus simulated incoming web request (hits) per hour.

the wall time required for the simulation to run, up to the maximum of 1,000 containers that were tested for Scenario 1. By comparison, Experiment Scenario 2 involves the simulation of 100 container deployed web servers, but where their incoming workload rates are now varied between 0 and 100,000 requests per hour, with each request retaining Scenario 1's configuration of 1,246 KBytes per request and 10,000 MI service level. Figure 6 shows DockerSim's wall time simulation cost versus Scenario 2, where all 100 containers are exposed to identical service rates. In both Scenarios 1 and 2, DockerSim's ability to provide reasonable performance levels, as facilitated by OMNet++'s underlying efficiency, is apparent. Scenario 1's worst case DockerSim performance result entails 1 second of simulated time requiring just under 1 second of wall time to simulate, but only once the level of 1,000 active containers is reached. In Scenario 2, with only 100 containers, the peak tested incoming request rates of 10,000 request per hour can be simulated by DockerSim with only 25 milliseconds of wall time being required for each simulated second. DockerSim's performance will clearly degrade with richer SaaS system simulations, but DockerSim's ability to easily achieve a sub 1:1 wall time to simulated time ratio bodes well for DockerSim's ability to efficiently support larger-scale more complex simulation scenarios.

## V. Limitations and Planned Extensions

DockerSim is fully implemented and has undergone detailed functional testing relative to its support for: VMs, containers, full stack full-packet cloud communications, and full supports for OS scheduling issues at each of the relevant abstraction layers. Work remains regarding the need to: i) more fully test at-scale so as to determine DockerSim's upper performance limits (i.e., how many racks, servers, VMs, containers, etc.), ii) finalize the integration and testing of the application-layer queuing network modeling, iii) identify and address underlying issues and bugs as they are exposed, and iv) to incorporate OmNet++'s parallelization support into DockerSim. DockerSim will be released as an open-source OMNet++ project once our internal testing and vetting concludes.

## VI. Conclusions

This work introduced the DockerSim, an OMNet++ based extension and augmentation of iCanCloud so as to support container-based cloud computing research. DockerSim is designed to provide: i) a full container deployment and behavioral layer, ii) full packet-level network and protocol behaviors, iii) full multi-layer OS process scheduling behaviors, and iv) a generic queuing network approach to modeling application-layer SaaS deployment behaviors. DockerSim's is tailored to better support for timing sensitive areas of cloud research. This work has demonstrated that (i)-(iv) can now be incorporated without incurring prohibitive simulation performance degradations. Further augmentations to DockerSim will incorporate parrallelization and HPC to provide a full rich simulation-based platform for supporting statistically rigorous research into open container-based cloud computing research concerns.

## References

[1] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[2] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, "icancloud: A flexible and scalable cloud infrastructure simulator," *Journal of Grid Computing*, vol. 10, no. 1, pp. 185–209, 2012.

[3] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. R. Das, "Mdcsim: A multi-tier data center simulation, platform," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–9.

[4] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. U. Khan, "Greencloud: a packet-level simulator of energy-aware cloud computing data centers," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE, 2010, pp. 1–5.

[5] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "Containercloudsim: An environment for modeling and simulation of containers in cloud data centers," *Software: Practice and Experience*, vol. 47, no. 4, pp. 505–521, 2017.

[6] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and computation: practice and experience*, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[7] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[8] A. Núñez, J. Fernández, J. D. Garcia, L. Prada, and J. Carretero, "Simcan: a simulator framework for computer architectures and storage networks," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, p. 73.

[9] W. Tian, Y. Zhao, M. Xu, Y. Zhong, and X. Sun, "A toolkit for modeling and simulation of real-time virtual machine allocation in a cloud data center," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 1, pp. 153–161, 2015.

[10] G. G. Castane, A. Nunez, and J. Carretero, "icancloud: A brief architecture overview," in *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*. IEEE, 2012, pp. 853–854.

[11] D. G. Feitelson, *Workload modeling for computer systems performance evaluation*. Cambridge University Press, 2015.

[12] N. Niclausse, "Tsung documentation release 1.6.0," Tech. Rep., July 20 2015, [Accessed: July 20, 2017]. [Online]. Available: http://tsung.erlang-projects.org/user_manual.pdf

[13] H. Khazaei, J. Misic, and V. B. Misic, "Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems," *IEEE Transactions on parallel and distributed systems*, vol. 23, no. 5, pp. 936–943, 2012.