Evaluating Distributed Systems and Applications through Accurate Models and Simulations

Marc Frincu, Bogdan Irimie, Teodora Selea, Adrian Spataru, Anca Vulpe

Abstract Evaluating the performance of distributed applications can be performed by in situ deployment on real-life platforms. However, this technique requires effort in terms of time allocated to configure both application and platform, execution time of tests, and analysis of results. Alternatively, users can evaluate their applications by running them on simulators on multiple scenarios. This provides a fast and reliable method for testing the application and platform on which it is executed. However, the accuracy of the results depend on the cross-layer models used by the simulators. In this chapter we investigate some of the existing models for representing both applications and the underlying distributed platform and infrastructure. We focus our presentation on the popular SimGrid simulator. We emphasize some best practices and conclude with few control questions and problems.

1 Introduction

A **distributed system** (DS) is a collection of *entities* (e.g., process or device) which *communicate* through a communication medium (e.g., wired or wireless network) appearing to end users as a single coherent system. The entities are characterized

Marc Frincu
West University of Timisoara Romania, e-mail: marc.frincu@e-uvt.ro – contact author

Bogdan Irimie
West University of Timisoara Romania e-mail: bogdan.irimie90@e-uvt.ro

Teodora Selea
e-Austria Research Institute Timisoara Romania e-mail: teodora.selea93@e-uvt.ro

Adrian Spataru
e-Austria Research Institute Timisoara Romania e-mail: florin.spataru92@e-uvt.ro

Anca Vulpe
West University of Timisoara Romania e-mail: anca.vulpe94@e-uvt.ro

by autonomicity, programmability, asynchronicity, and failure-proneness, while the communication medium is usually unreliable.

Distributed systems enable a form of parallel computing, namely **distributed computing**, where computation and data is geographically spread, but the applications have parallel tasks processing the same or different parts of the data at the same time.

Designing and testing platforms and application on top of them require therefore specific configurations which due to the nature of the DS may be out of reach of researchers and software engineers. In addition, real-life systems may not offer the complexity and specific setup required by some applications. In situ experiments need therefore to be replaced with a more economic and flexible alternative in which the DS can still be properly modeled. The complexity of DSs (e.g., clouds) make theoretical models not viable as the large number of parameters required to model heterogeneity, dynamism, exascale, and Big Data leads to systems which are too complex to be modeled through mathematics alone.

Simulations offer the fastest path from idea to its testing. They enable users to get preliminary results from partial implementations and to improve their algorithms, applications, and platforms quickly and under various settings and assumptions. They also allow experiments on thousands of configurations fast at no cost and without having to wait in line for available resources. Finally, they allow users to bypass any technical challenges posed by the platform and DS letting them focus on the application itself. Figure 1 depicts the usual simulation flow from idea, experimental setup and model to scientific results.

Despite their advantages, simulations face several challenges including:

- **Validity**: Results obtained through simulations should match or be close to those obtained in real-life experiments. Approximations in any simulation should be quantifiable such that any result would be mapped to its real-life equivalent. For instance, Virtual Machine (VM) boot and stop times which could be ignored in simulations should not impact the outcome of real-life deployment as predicted by the simulation. Furthermore, the accuracy of underlying is essential in validating the experiments. Extensive tests of SimGrid and comparison with other simulators have outlined strange behaviors in the network modeling of simulators such as OptorSim, GridSim, and CloudSim [18].
- **Scalability**: Any simulation should scale with the experiment size to allow fast exploration of scenarios of several orders of magnitude. For instance, the simu-

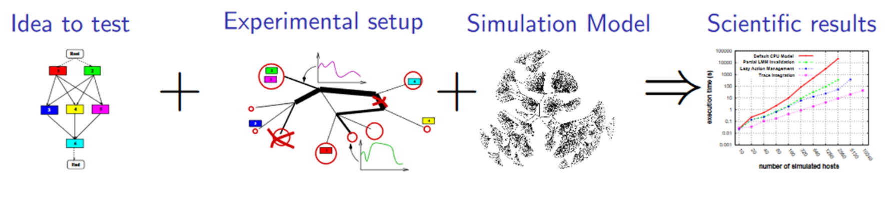

Fig. 1 Simulation flow [7].

lation time of a single scenario should not exceed in any case the validation on real-life DSs.

- **Tools**: Simulation results are usually numerically encoded and contain lots of data unreadable in raw format. Furthermore, if multiple scenarios are tested the visual analysis of the raw data is practically impossible. Hence, automated tools for visual analysis are required. In addition, to avoid the time consuming manual generation of hundreds or thousands of experiment settings, automatic generation based on customized parameters is necessary.
- **Applicability**: Simulations should match the user requirements and objectives. Hence, the underlying models should closely match real-life scenarios while the simulation output should match the desired goals.

Fundamentally, to enable a good simulation sound **models** are required across the simulated platform layers. These models are essential both for the validity and applicability of the simulation. *In this chapter we focus on cloud simulators as clouds are widely researched and new algorithms for numerous problems are developed constantly*. Despite being around for more than a decade clouds have yet to unveil their full potential with Big Data and Internet of Things promising new challenges for clouds. Simulations will play an essential role in driving the next wave of algorithms for topics including Big Data processing, job scheduling in hybrid systems and architectures, and Quality of Service assurance.

The rest of the chapter introduces cloud computing and cloud simulators (cf. Sect. 2, then it moves on to discuss the assessment methods of a distributed application (cf. Sect. 3, gives an overview of platform cross layer models (cf. Sect. 4) and details each of them (cf. Sect. 5, Sect. 6, and Sect 7), discusses the importance of simulation data (cf. Sect. 8), concluding with general remarks (cf. Sect. 9) and few chapter control questions and problems (cf. Sect. 10). The model layers are presented by mirroring the ones existing in the SimGrid simulator [3] and SchIaaS extension [6].

2 Cloud Computing

According to NIST, ”cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction“ [19].

Cloud computing is characterized by:

- **On demand access** to storage, computational and network resources, platforms, and applications
- **Broad network access**
- **Pay-per-use policy** varying between resource types and providers. Examples include per hour, per minute, per Gb, per request

- **Resource pooling** with virtually unlimited resources for users
- **Rapid elasticity** which allows users to horizontally (e.g., adding new/removing VMs) or vertically (e.g., adding new/removing VM cores and memory) scale based on demand
- **New programming paradigms** such as MapReduce, Hadoop, NoSQL (Cassandra, MongoDB)
- **Big Data** where the large cloud data centers can now store PBs of data coming from the research community (e.g., astrophysical data, meteorological data) or industry (e.g., social network data, banking data) .

All these features are stored in a layered cloud stack (cf. Fig. 2). The Infrastructure as a Service (IaaS) offers direct access to virtualized resources being targeted as network architects and cloud administrators. The Platform as a Service (PaaS) offers OS level functions to application developers, and finally, Software as a Service (SaaS) targets end users by exposing fully fledged applications. The *as a Service* model extends beyond these initial three layers and comprises Data as a Service (DaaS) to emphasize the Big Data stored and accessible in clouds, Network as a Service (NaaS), Communication as a Service (CaaS), and Monitoring as a Service (MaaS)

To simulate such complex environments various simulators have been proposed. Next, we briefly compare them before looking at what a good simulation should offer and what are the cloud simulation layers.

2.1 Cloud Simulators

Over the years many simulators, some of them short lived, others widely used in literature and well documented and validated, have been devised. Many of them started as grid simulators have slowly evolved in generic cloud simulators [3]. Oth-

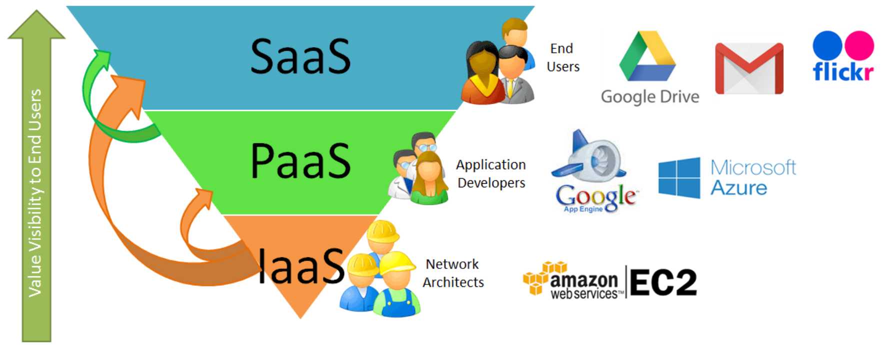

Fig. 2 Cloud stack.

ers [2] have evolved into cloud simulators by borrowing from other the underlying models, or were designed from scratch [20].

Despite many of the simulators being generic, there are also corner cases where custom built solutions are required. As example IoT applications are becoming better integrated with the cloud and they give birth to another layer between the cloud and the IoT devices called *fog*. A simulator that addresses this corner case and based on CloudSim is presented in [13].

For a detailed classification of cloud simulators we direct the readers to [1] and [24].

Table 1 Recommended cloud simulators depending on user priorities [24]

Scenario	CloudSim	GreenCloud	iCanCloud	NetworkCloudSim	CloudAnalyst	GoundSim	CDOSim	MDCSim	GDCSim	SPECI	BigHouse	TeachCloud	SimGrid
Simple state sharing problem / High number of Nodes (>10k Nodes)	✓	×	×	×	×	✓	×	×	×	✓	✓	✓	✓
Energy efficiency / Energy aware scheduling	×	✓	✓	×	×	×	×	×	×	×	×	×	×
Energy efficiency / Cooling	×	×	×	×	×	×	×	×	✓	×	×	×	×
High availability / Fault tolerance	✓	✓	×	×	×	✓	×	×	×	×	×	×	✓
Network modeling and network-aware scheduling	×	✓	×	✓	×	×	×	✓	×	✓	×	×	✓
Workload planning / evaluation	✓	✓	×	×	✓	×	×	✓	×	×	✓	×	✓
Workflow Modeling	✓	✓	×	×	×	✓	×	×	×	×	×	×	✓
Resource allocation	✓	✓	✓	✓	✓	×	×	✓	×	×	✓	✓	✓
Service brokering	✓	×	✓	×	✓	×	✓	×	×	×	×	×	×
Storage modeling	×	×	✓	×	×	×	×	×	×	×	✓	✓	✓
GUI / Easy Use	×	×	✓	×	✓	×	×	×	×	✓	×	✓	✓
High request/job load (>10k requests)	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓
MapReduce Application Modeling/Data Replication	✓	✓	×	×	×	×	×	×	×	×	✓	✓	✓

Currently, CloudSim is one of the most used cloud simulator and there are a large number of related projects like [27] [22] [25] and others. Its main advantage is that it offers the building blocks for modeling complex cloud infrastructure and applications that run atop them. The entire code is written in Java.

A different simulator which focuses on DSs but which has support for clouds is SimGrid [3]. It relies on well tested models across the simulated system stack and extensions are built frequently. The code is written in C but has Java, Ruby, and Lua bindings which makes it suitable for a wider audience. SimGrid does not offer cloud support directly but does expose a VM migration and execution model. Based on it, cloud extensions have been proposed with an updated list available at [4].

One main drawback of today's simulators is that they do not have accurate models for system/application failures. In a DS, components fail all the time and those failures can affect the overall system performance. Another major drawback is that there is no extensive validation for many of the simulators and thus there can be big discrepancies between results obtained using the simulators and experiments on real cloud infrastructures due to poor or simplistic models.

3 Assessment of Applications

When assessing cloud applications users are usually interested in its **correctness** and **performance**. Each is quantified through various metrics depending on the application objectives. Hence correctness can be modeled through the absence of crashes, race conditions or deadlocks. Performance indicators usually address makespan, throughput, energy consumption or running costs. A comprehensive overview of client side objectives is given in our previous work [10].

Due to the complexity of the environment the **correction study** for a cloud application relies on model-checking as it allows an exhaustive automated exploration of the state space to identify issues. Instead, **performance studies** rely on simulations to test prototype applications on system models in scenarios unavailable on real-life systems and where math is insufficient to understand the behavior of the applications. An alternative to simulations could be emulations which rely on testing real-life applications on synthetic systems.

In simulations a key requirement is the **reproducibility** of simulation results which allows users to rerun the same experimental setup described in a paper to benchmark on a different data set or to compare with a different approach. However, one of the main problems in literature is the lack of sufficient details and that of publicly available source codes on which the experiments were based on.

Another important aspect when running simulations is to have access to **standard tools** which users can learn quickly without having to code their own software or to learn several simulators for different simulation objectives. In practice, there are lots of custom made shot lived simulators which do not provide insight on the models used to simulate the cloud systems or make assumptions which may not be valid. Furthermore, their validity has not been thoroughly tested despite being used in many research papers [7].

By having failures inserted in the simulations, we can observe how application behaves under abnormal conditions and establish limits in the amount of failures that our application can cope with.

4 Modeling Layers

When designing a cloud simulation and simulator, the entire cloud stack needs to be considered to ensure a proper representation of the real-life environment. For each of the layers, accurate models need to be implemented and validated on large amounts of data to ensure the validity of the subsequent simulations.

Hence, the following minimal list of models should be implemented:

1. Bare metal models
    - Hardware models for CPU, network, memory;
2. Virtualization models
    - Models for hypervisors;
3. Cloud models
    - IaaS level management similar to existing providers' APIs;
    - OS level (PaaS) models for proper abstractions and resource management;
    - Models for simplistic yet accurate process/application representation.

Next, we give an overview of each of them by providing implementation examples with reference to the SimGrid simulator.

5 Hardware Model

At the hardware model level simulators need to consider CPU and network.

CPU processing speed s is usually expressed in flops (floating point operations per second) which means that processes running on them should be specified in terms of required floating point operations r. At this level modeling is trivial since in order to get the execution time of a process we simply have to compute r/s.

In resource sharing environments it is possible to model CPU sharing and to introduce CPU availability traces to model the fluctuation of the CPU speed.

However, modern architecture are usually parallel machines with multiple cores and processors. These architectures are usually modeled by having an array a which describes the number of floating point operations that each processor has to execute and a matrix B which describes the communication pattern [15]. This enables the modeling of:

- Fully parallel tasks: $a \neq 0$ and $B = 0$;
- Data redistribution tasks: $a = 0$ and $B \neq 0$;
- Tasks with communication: $a \neq 0$ and $B \neq 0$.

The model can be further extended to account for inter-processor cache sharing, memory, and compiler/OS particularities.

Network modeling is more complex and needs to account for latency, bandwidth sharing, and TCP congestion in order to obtain realistic simulations. Several simulation models exist in literature:

- **Delay-based** models: are the simplest network models. They allow the modeling of communication time through statistical models, constants (e.g., latency), and geographical coordinate systems to account for geographic proximity. The motivation behind these models is that end-to-end delay greatly affects the performance of applications running on the network [11]. One of their main drawbacks is that they ignore network congestion and assume large bisection bandwidth (i.e., the available bandwidth between endpoints).

- **Packet level** models: capture the behavior and interaction of individual packets through complex models. Examples of simulators taking this approach include GTNetS [21], NS2 [16] which simulate the entire protocol stack.
- **Flow level** models: simulate the entire communication flow as a single entity $T_{i,j}(S) = L_{i,j} + S/B_{i,j}$, where S represents the message size, $L_{i,j}$ is the latency between endpoints i and j, and $B_{i,j}$ represents the bandwidth. The model assumes a steady-state and bandwidth sharing each time a new flow appears or disappears. Given a flow ϕ_k and the capacity of the link C_j the constraint is to have $\sum_k \phi_k < C_j$. Several algorithms including Max-Min fairness, Vegas, and Reno exist. In case of Max-Min fairness the objective is to $\max\min(\phi_k)$ with the equilibrium reached when increasing any flow ϕ_l decreases a given flow ϕ_k. As such it tries to give a fair share to all flows sharing the link.

Besides models for simulating data flows the hardware model also comprises of models of the topology. In SimGrid for instance, a DS is represented as a static topology as seen in the following simple example[1] which defines an autonomous system with full routing comprised of three machines linked together as in Fig. 3:

```
<platform version="3">
 <AS  id="AS0"  routing="Full">
  <host id="Horus" core="4" power="8095000000"
   availability_file="horus_avail.trace"
   state_file="horus.state" />
  <host id="Osyris" core="4" power="8095000000"/>
  <host id="Isis" core="4" power="8095000000"/>

  <link id="link1" bandwidth="125000000"
   latency="0.000100" bandwidth_file="link1.bw"
   latency_file="link1.lat"/>
  <link id="link2" bandwidth="125000000"
   latency="0.000100"/>
  <link id="link3" bandwidth="125000000"
   latency="0.000500"/>

  <route src="Horus" dst="Osyris"><link_ctn id="link1"/>
  </route>
  <route src="Horus" dst="Isis"><link_ctn id="link2"/>
  </route>
  <route src="Osyris" dst="Isis"><link_ctn id="link3"/>
  </route>
 </AS>
</platform>
```

[1] Full documentation available at: http://simgrid.gforge.inria.fr/simgrid/3.12/doc/platform.html

where power is in flops, latency is in seconds, and bandwidth is in bytes/second. While the topology is fixed there is the option to define traces for CPU, bandwidth and latency fluctuations (e.g., the cases of horus host and link2), and availability periods (e.g., simulate failures).

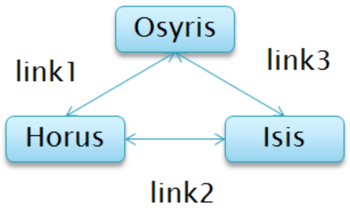

Fig. 3 Simple DS topology.

6 Hypervisor Model

While the previous hardware models enable the simulation of DS such as grids, clouds require a virtualization layer where the hypervisor (e.g., Xen, KVM, VMWare) can create and execute VMs.

To enable a seamless transition from simulation to real-life deployment models should mimic the real systems. Hence the user level API should match that of existing hypervisors with functions for starting, stopping, pausing, and resuming VMs.

Simulators such as SimGrid implement [14] such APIs and offer models for live migration as well.

To enable **VM management** two resource constraint problem need to be solved, at physical level, and at virtualized level. In SimGrid for instance, VMs are seen as an ordinary task executed on the physical machine. Basically, to place VMs along side regular tasks the simulator first computes the share of the host for each of them. Then, for each VM it computes the shares of tasks running on them using the allocated shared by the host as maximum. For instance if a host has a capacity C and there are 2 VMs and one task allocated to it it first solves the constraint $S_{VM_1} + S_{VM_2} + S_t < C$, where S_* represents the share of the host to be allocated. Second, once S_{VM_1} and S_{VM_2} are determined, assuming VM_1 will execute 2 tasks and VM_2 one task, it solves the constraints $S_{t_1} + S_{t_2} < S_{VM_1}$ and $S_{t_3} < S_{VM_2}$. In addition, task priorities and VM CPU usage capping can be specified.

Once a hypervisor model is in place, the **live migration** of VMs needs to be modeled too. This capability is at the core of activities involving system maintenance, load balancing, energy efficiency, reconfiguration and fault tolerance. Some simulators such as CloudSim let users specify the migration time but this approach is simplistic. Default live migration policies can be overridden hence allowing for some flexibility and testing of new algorithms.

In SimGrid, the precopy live migration **algorithm** is implemented, however in literature other well-known algorithms such as post copy and hybrid exist. A detailed overview and analysis of their performance is given in [23]. The reason for implementing the precopy algorithm is its popularity among well-known hypervisors such as Xen, KVM, and VMWare.

- Precopy: the algorithm iteratively copies memory pages of the VM from the source host to the destination. First, it copies all memory pages. At subsequent steps it copies only the modified pages, and repeats this step until the number of modified pages is small enough. At this stage it stops the VM and copies the remaining dirty pages to the destination. Finally, it restarts the VM at the destination. The entire process takes from few ms to seconds. The algorithm is reliable and robust as the entire process can be rolled back if the migration fails.
- Postcopy: the algorithm first stops the VM and then copies using demand and pre-paging techniques over the network. First, the VM makes some initial preparation of resources. Then, the VM is stopped and the execution states are transferred and switched on at the destination host to resume the VM. During this phase the VM is down. After the states have been transferred and the VM has resumed the memory page will be copied. In this algorithm the transferred VM will start immediately but will suffer from performance penalties from network page faults. The performance of this algorithm is highly dependent on the workload and hence choosing it requires a deep analysis with different workloads.
- Hybrid: the algorithm is a special version of postcopy where a limited number of precopy stages are applied a priori. The algorithm is useful in cases where we want to balance the reliability of precopy with speed of postcopy.

Depending on whether or not users want to investigate live migration algorithms simulators can offer extensible constructs to enable their validation by relying on the hardware models.

7 Cloud Model

With a virtualization model in place simulators can be augmented with support for cloud models. These models should mimic the layered cloud architecture at IaaS and PaaS with support for running applications at each one. Simulators should be generic and extensible to allow the insertion of new cloud engines. Popular cloud IaaS models include the Amazon EC2 model of instances and billing. The complexity and level of Amazon EC2 services has enabled a vast collection of EC2 compatible APIs in various cloud software platforms such as Eucalyptus, OpenStack, and OpenNebula to name a few.

Contrary to the hypervisor layer, in the cloud layer users handle instances not VMs. These instances have several characteristics including type and billing model, and are automatically placed on hosts by the hypervisor. In SimGrid, users can access cloud IaaS APIs through the SchIaas extension, while PaaS level resource management for bag-of-tasks and workflow applications can be handled through SimSchlouder.

7.1 Infrastructure Model

In SimGrid, the cloud topology including compute and storage services, instance types, instance images, and the physical infrastructure to host the VMs algorithms is defined in a file similar to the simple example below:

```
<clouds version="1">
 <cloud  id="myCloud">
  <storage id="myStorage"
    engine="org.simgrid.schiaas.engine.storage.rise.Rise">
    <config controller="Horus"/>
  </storage>

  <compute
    engine="org.simgrid.schiaas.engine.compute.rice.Rice">
   <config controller="Horus" image_storage="myStorage"
     image_caching="PRE inter_boot_delay="10"/>

   <instance_type id="small" core="1" memory="1000"
    disk="1690"/>
   <instance_type id="medium" core="2" memory="1000"
    disk="1690"/>
   <instance_type id="large" core="4" memory="1000"
    disk="1690"/>

   <image id="myImage" size="1073741824"/>

   <host id="Osyris"/>
   <host id="Isis"/>
  </compute>
 </cloud>
</clouds>
```

The IaaS model usually has two views. The cloud client view available to end users where compute instances and storage can be handled; and the cloud provider view where cloud IaaS administrators handle VM to host placement and other cloud infrastructure management activities. In SimGrid, the provider view is handled by default by the RICE (Reduced Implementation of Compute Engine) and RISE (Reduced Implementation of Storage Engine) engines.

7.2 Platform Model

At PaaS level simulators usually provide functionality and models for simulating application execution. Resource management for simulating bag-of-tasks and workflow applications is an example of such functionality.

At this level users can test scheduling algorithms on various applications, and cloud and infrastructure topologies by relying on the simulator models for computation, communication, virtualization, and cloud. For simulators such as SimGrid this is the where users take advantage of the full simulator stack to propose new models for cloud resource management.

7.3 Application Model

To simulate applications we require simplistic yet comprehensive models for them. Required information should be mapped on the underlying models, namely on the computation and communication models. The following simple example specifies a process that will spawn a job with 10 tasks with predefined size in floating point operations and communication size in bytes.

```
<process host="Horus" function="cloud.schiaas.Master">
 <!-- Number of tasks -->
 <argument value="10"/>
 <!-- Computation size of tasks -->
 <argument value="5e10"/>
 <!-- Communication size of tasks -->
 <argument value="1000000"/>
 <!-- Number of slave processes -->
 <argument value="10"/>
</process>
```

8 Simulation Data

Traces for platform and application bring the simulated application and DS closer to the behavior of real-life systems. **Traces** can be either **synthetic** or from **real-life** systems. Synthetic traces are based on statistical analysis of real-life systems and capture variations which on real traces may not be visible. A detailed overview of synthetic data and how to generate it for DS is given in [9]. A comparative study – from more than 2 decades ago – between the two trace types has outlined no significant differences in algorithm behavior [17].

There are many large trace sources from companies like Wikipedia [26], Google [12] as well as traces from various parallel [8] and grid systems [5]. Despite their advantages, one downside is that a large portion of the simulation time is spent reading the traces from disk as those traces can have hundreds of GB in size.

9 Conclusion

In this chapter we have emphasized the importance of simulators and simulation models. Real-life systems require a huge amount of effort to configure the environment (application and platform), to run tests, and to analyze results.

In contrast, simulators allow users a fast and reliable method to evaluate applications ran on a specific platform. Over the years, the diversity of simulators has increased leading to general purpose and specialized simulators on energy efficiency, network modeling, network-aware scheduling, workload planning, resource allocation, service brokering, storage modeling (cf. Table 1).

Building cloud simulations and simulators consists in implementing bare metal models, virtualization models, and cloud models.

The bare metal model is represented by hardware model level. Here has to be taken in consideration CPU and network. Modeling CPU is a simpler than network modeling which has to take into account latency, bandwidth, and TCP congestion.

At virtualization level, the hypervisor model can create and execute VMs. The most well known algorithms implemented at this level are precopy, postcopy and hybrid.

Cloud models should mimic the layered cloud architecture at the IaaS and PaaS layers. Simulators need to be generic and extensible to allow extensions and customized behavior. The IaaS model is composed of two main views: client view and provider view. The PaaS level simulators provide functionality and models for simulating application execution. The application model is mapped on the underlying models to enable users to take full advantage of the simulation environment.

Simulation data is represented either by real-life data or by synthetic data. Synthetic data is based on statistical analysis of real-life systems. It can captures variations which may not be visible on real traces.

In conclusion, with the increase complexity of DSs we expect simulators to play a crucial role in both research and development industry by enabling applications to be tested in scenarios not covered by the limitations of real-life systems.

10 Chapter Control Questions and Problems

1. Explain why theoretical models may not be suitable for DS.
2. Explain why we need custom simulators for the cloud environment.
3. Enumerate at least three cloud simulators.
4. Enumerate and explain the drawbacks of current cloud simulators.

5. Define what the *correctness study* and the *performance study* are and how they are done.
6. Enumerate the modeling layers of the cloud environment that need to be considered when designing a cloud simulator. Please give a short description for each layer.
7. Install SimGrid Simulator, implement and run master/workers example. All the necessary documentation can be found on the official website[2]
8. Implement and run ping-pong example.

Acknowledgment

The work of the first author has been partially funded by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS/CCCDI - UEFISCDI, project number PN-III-P3-3.6-H2020-2016-0005, within PNCDI III.

The work of the second author has been partially funded by the EU H2020 VI-SEEM project under contract no. 675121.

The work of the third and forth authors has been partially funded by the EU H2020 CloudLightning project under grant no. 643946.

References

1. A. Ahmed and A. S. Sabyasachi. Cloud computing simulators: A detailed survey and future direction, Feb 2014.
2. Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, C&#x00e9;sar A. F. De Rose, and Rajkumar Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50, January 2011.
3. Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, June 2014.
4. Simgrid Cloud. Virtualization / cloud abstractions in simgrid, 2016. http://simgrid.gforge.inria.fr/contrib/clouds-sg-doc.php.
5. A. Iosup e al. Grid workload archive, 2016. http://gwa.ewi.tudelft.nl/.
6. Gossa Julien et al. Iaas simulation upon simgrid, 2015. http://schiaas.gforge.inria.fr/.
7. Quinson Martin et al. Simgrid 101: Getting started to the simgrid project, January 2015. http://simgrid.gforge.inria.fr/tutorials/simgrid-101.pdf.
8. D. Feitelson. Parallel workload archive, 2016. http://www.cs.huji.ac.il/labs/parallel/workload/.
9. Dror G Feitelson. *Workload modeling for computer systems performance evaluation*. Cambridge University Press, Cambridge, 2015.
10. Marc Eduard Frîncu, Stéphane Genaud, and Julien Gossa. Client-side resource management on the cloud: survey and future directions. *IJCC*, 4(3):234–257, 2015.
11. Mo Ghorbanzadeh, Ahmed Abdelhadi, and Charles Clancy. *Delay-Based Backhaul Modeling*, pages 179–240. 2017.

[2] http://simgrid.gforge.inria.fr/simgrid/latest/doc/group__MSG__examples.html

12. Google. Google traces, 2016. https://github.com/google/cluster-data.
13. Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *CoRR*, abs/1606.02007, 2016.
14. T. Hirofuchi, A. Lebre, and L. Pouilloux. Simgrid vm: Virtual machine support for a simulation framework of distributed systems. *IEEE Transactions on Cloud Computing*, PP(99):1–1, 2015.
15. S. Hunold, H. Casanova, and F. Suter. From simulation to experiment: A case study on multiprocessor task scheduling. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 665–672, 2011.
16. ISI. The network simulator, November 2016. http://www.isi.edu/nsnam/ns/.
17. Virginia Lo, Jens Mache, and Kurt Windisch. *A comparative study of real workload traces and synthetic workload models for parallel job scheduling*, pages 25–46. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.
18. Simgrid Models. Getting started with simgrid models, 2016. http://simgrid.gforge.inria.fr/tutorials/surf-101.pdf.
19. NIST. Cloud computing, 2016. https://www.nist.gov/itl/cloud-computing.
20. Alberto Núñez, Jose L. Vázquez-Poletti, Agustin C. Caminero, Gabriel G. Castañé, Jesus Carretero, and Ignacio M. Llorente. icancloud: A flexible and scalable cloud infrastructure simulator. *J. Grid Comput.*, 10(1):185–209, March 2012.
21. G. F. Riley. Large-scale network simulations with gtnets. In *Simulation Conference, 2003. Proceedings of the 2003 Winter*, volume 1, pages 676–684 Vol.1, 2003.
22. Parnia Samimi, Youness Teimouri, and Muriati Mukhtar. A combinatorial double auction resource allocation model in cloud computing. *Information Sciences*, 357:201 – 216, 2016.
23. S. A. R. Shah, A. H. Jaikar, and S. Y. Noh. A performance analysis of precopy, postcopy and hybrid live vm migration algorithms in scientific cloud computing environment. In *High Performance Computing Simulation (HPCS), 2015 International Conference on*, pages 229–236, 2015.
24. Mohamed Abu Sharkh, Ali Kanso, Abdallah Shami, and Peter hln. Building a cloud on earth: A study of cloud computing data center simulators. *Computer Networks*, 108:78 – 96, 2016.
25. Thiago Teixeira Sá, Rodrigo N. Calheiros, and Danielo G. Gomes. *CloudReports: An Extensible Simulation Tool for Energy-Aware Cloud Computing Environments*, pages 127–142. Springer International Publishing, Cham, 2014.
26. Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009.
27. B. Wickremasinghe, R. N. Calheiros, and R. Buyya. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 446–452, April 2010.