RZ 3851 (# ZUR1307-033) 08/13/2013
Computer Science 8 pages

Research Report

The IBM Performance Simulation Framework for Cloud

Peter Altevogt

Smart Cloud Development, IBM Germany Research & Development GmbH
Boeblingen, Germany

Wolfgang Denzel

Systems Department, IBM Research – Zurich
Rüschlikon, Switzerland

Tibor Kiss

Gamax Kft
Budapest, Hungary

The IBM Performance Simulation Framework for Cloud

Peter Altevogt
Smart Cloud Development
IBM Germany Research & Development GmbH
Boeblingen, Germany

Wolfgang Denzel
Systems Department
IBM Research GmbH, Zurich Research Laboratory
Rueschlikon, Switzerland

Tibor Kiss
Gamax Kft
Budapest, Hungary

***Abstract*—We describe the "IBM Performance Simulation Framework for Cloud", which supports modular, accurate and scalable performance simulations of clouds. The framework treats all hardware and software components of a cloud as first-class citizens. Due to its modular design, it supports a rapid construction of new cloud models by combining already available simulation modules. These models may then be extended or refined by adding new modules as required. The modeling accuracy can be adapted to address cloud performance analysis on various detail levels as well as taking time and resource constraints into account. The framework supports simulations to be parallelized using message-passing technologies to ensure scalability. A careful separation between hardware (infrastructure) modules and modules representing software workflows as well as the introduction of a hierarchy of requests separates the simulation of high-level cloud level workflows from the simulation of hardware components. Finally, we demonstrate how the framework can be applied by simulating image deployment performance in OpenStack managed clouds.**

Keywords—cloud; performance simulation; performance modeling; OpenStack

I. Introduction

Cloud computing is perceived as a game changing technology to provide respectively to consume data center resources [1]. Performance and scalability are key non-functional attributes of cloud services required to enable further growth of cloud computing [2].

To ensure a balanced, workload optimized and scalable design of clouds, a performance engineering approach limited to measurements and tuning only is not sufficient. In general these activities happen too late in the development cycle to address design issues and are too little to address scalability or cover a comprehensive set of benchmark scenarios on various cloud architectures due to time and resource limitations.

Performance modeling and simulation technologies can help to alleviate these issues. In fact, they enable a performance analysis of cloud designs and dynamic capacity planning early in the development cycle, at a much larger scale, with moderate costs and respecting the strict time constraints of an industrial development project. Although performance modeling [3] [4] [5] and simulation technologies [6] [7] are widely used and well established in various branches of information and telecommunication industries, their application to clouds provides some new challenges due to complexity, diversity, agility and scale [8].

The "IBM Performance Simulation Framework for Cloud" addresses these challenges by providing a framework for modular, accurate and scalable performance simulations of clouds. After a discussion of related work available here, we will provide an overview over its design and architecture. Then we will shortly describe our simulation technology and tooling. The main part of our work then focuses on some implementation details, especially on the basic modeling abstractions and on how complex cloud simulation scenarios can be build from more basic building blocks ("Lego bricks"). Finally we outline the application of the framework to model image deployment in various OpenStack [9] managed clouds.

II. Related Work

There are a quite a few simulation frameworks for clouds available and a good survey can be found in [10]. Currently the most popular cloud simulation framework seems to be CloudSim [11] and a number of other simulation tools like CloudAnalyst [12], NetworkCloudSim [13] or EMUSIM [14] based on CloudSim. Unfortunately, CloudSim does not meet our requirements in terms of scalability, accuracy (especially of the hardware modeling) and modularity. Other cloud computing simulators like GroudSim [15] seem to have made some progress concerning execution performance versus CloudSim by replacing a process-based by an event-based simulation, but they still lack parallelizability and provide very

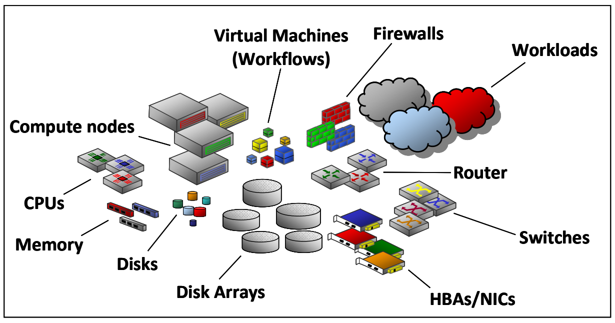


Fig. 1. Samples of modeled cloud components

little infrastructure simulation details. The iCanCloud [16] tool also provides almost no hardware modeling and no modeling of queuing for software resources, which is essential for simulating software workflows in clouds, see section VI.G below. Other tools like DCSim [17], GreenCloud [18] or SPECI [19] furthermore seem to focus on some special, cloud features like resource allocation, energy consumption or resiliency.

There also exists a few cloud related analytic modeling efforts [20] [21], but unfortunately it seems to be too difficult to apply these methods to model clouds with the accuracy and flexibility required.

Our work provides a significant step forward to an industry-strength cloud performance simulation framework capable of addressing performance and scalability problems of real clouds on both hardware and software level with the required accuracy, flexibility, modularity and scalability.

III. Requirements

To address the challenge of supporting the design and deployment of performant and scalable compute clouds, the framework must fulfill various requirements, especially it should

- treat all hardware and software components of a cloud as first-class citizens
- model these components on an appropriate level of detail
- support the rapid construction of new cloud models
- allow for an easy extension or modification of existing models
- enable the adaption of modeling accuracy to address cloud performance analysis on various detail levels as well as taking time and resource constraints into account
- scale-out to enable modeling of large clouds leveraging additional hardware resources if required
- support end-to-end performance analysis of cloud workloads as well as an in-depth performance analysis of some selected cloud components only

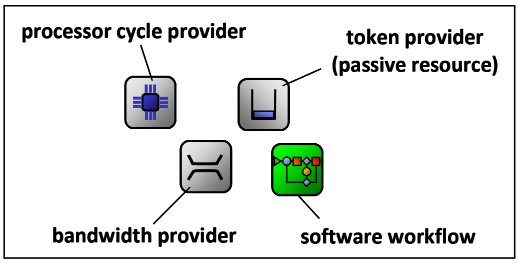


Fig. 2. The fundamental building blocks (basic modules) of the simulation framework

Obviously, some of these requirements are conflicting and the design needs to be sufficiently balanced to address all of them to a certain extend.

IV. Design and Architecture

Although clouds are in general highly complex systems, they frequently consist of a rather small set of fundamentally different building blocks, see Fig. 1. In fact we consider all hardware components being build out of a few fundamental parameterizable building blocks (“Lego bricks”, see Fig. 2), namely a

- bandwidth provider
- processor cycle provider (processor core)
- token provider
- software workflow

The first two represent active resources and the token provider passive ones. Workload requests may queue for active or passive resources, e.g. for some processor cycles for a certain amount of time or for bandwidth to send data. Implementation details of these basic components are described. These fundamental building blocks (modules) can be quite easily combined to create more complex objects (like switches, disks or compute nodes) which again can be combined to create more complex objects like compute racks, disk arrays or even complete data centers respectively clouds. Objects on any level of complexity can be flexibly combined and communicate via messages. They can be replicated (“copy-and-paste”)[1] to enable e.g. the creation of various geographically distributed set of data centers, see Fig. 12.

The request workflows are in general implemented by separate software workflow modules requesting the resources from the appropriate hardware modules.

These workflows are implemented on various levels with the higher level workflows starting lower level workflows, e.g. a cloud level workflow implemented in Virtual Machines may start lower level independent workflows at compute nodes or network storage. This allows an implementation of workloads at cloud level to ignore workflow level details at lower levels,

[1] Because this feature clearly goes beyond the capabilities of ordinary Lego bricks, one might think of the modules here as “Lego++ bricks”.

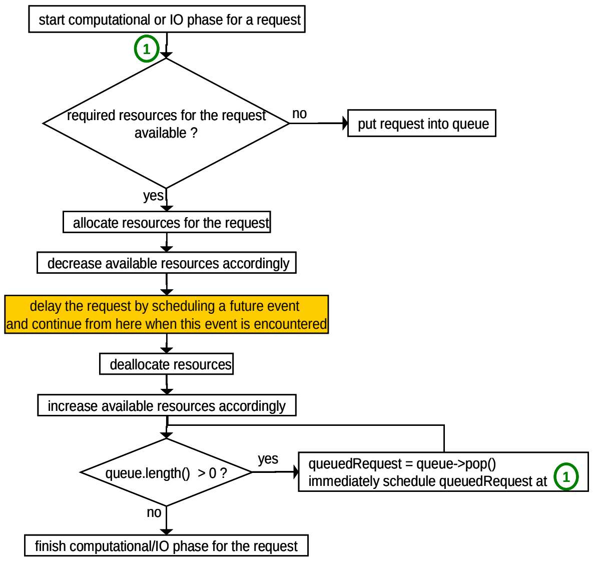

start computational or IO phase for a request
1
required resources for the request available ?
no
put request into queue
yes
allocate resources for the request
decrease available resources accordingly
delay the request by scheduling a future event and continue from here when this event is encountered
deallocate resources
increase available resources accordingly
queue.length() > 0 ?
yes
queuedRequest = queue->pop()
immediately schedule queuedRequest at
1
no
finish computational/IO phase for the request

Fig. 3. The default FCFS arbitration for requests to access resources.

e.g. at disk arrays. On the other hand, this also allows the modification of lower level workflows (e.g. on disk level) without impacting workflows at higher level (e.g. at disk array of cloud level).

Any of the fundamental building blocks may be replaced by a more appropriate one if required, e.g. a more accurate or more coarse-grained one depending on the modeling objectives.

Modularity and the ability to easily replicate modules at any complexity level are the key features of the framework supporting the flexible creation of simulation models for various clouds architectures with moderate effort and time but supporting high accuracy when required.

Scalability is supported by parallelization using the MPI Message Passing Interface [22].

V. Implementation Details

A. *Simulation Technology*

The framework leverages standard discrete-event simulation technologies [6] [7] and uses the OMNEST Network Simulation Framework [23] [24] as a basis for its implementation[2].

B. *Basic Simulation Modules*

The basic simulation modules provide the active (hardware) resources in form of processor cycles or bandwidth associated with disk or network IO as well as passive (software) resources, see Fig. 2.

A request executing in the context of its workflow tries to allocate the resources required to proceed. If it is successful, it

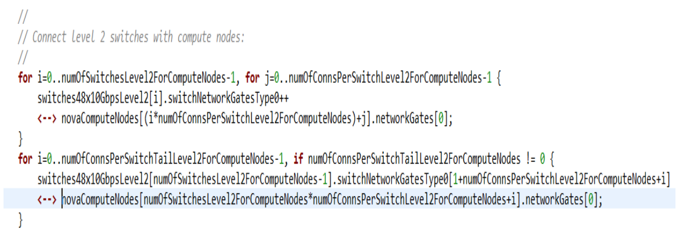

```
//
// Connect level 2 switches with compute nodes:
//
for i=0..numOfSwitchesLevel2ForComputeNodes-1, for j=0..numOfConnsPerSwitchLevel2ForComputeNodes-1 {
    switches48x10GbpsLevel2[i].switchNetworkGatesType0++
    <--> novaComputeNodes[(i*numOfConnsPerSwitchLevel2ForComputeNodes)+j].networkGates[0];
}
for i=0..numOfConnsPerSwitchTailLevel2ForComputeNodes-1, if numOfConnsPerSwitchTailLevel2ForComputeNodes != 0 {
    switches48x10GbpsLevel2[numOfSwitchesLevel2ForComputeNodes-1].switchNetworkGatesType0[1+numOfConnsPerSwitchLevel2ForComputeNodes+i]
    <--> novaComputeNodes[numOfSwitchesLevel2ForComputeNodes*numOfConnsPerSwitchLevel2ForComputeNodes+i].networkGates[0];
}
```

Fig. 4. A NED code snippet for the connection of switches with compute nodes in the context of creating a cloud.

decreases the available resources accordingly and proceeds to be delayed for a specified amount of time by scheduling an appropriate event in the future event list. If not, the request is inserted into a queue for the required resource and waits. When encountering the scheduled event, the request deallocates the requested resources again, increases the available resources accordingly and retrieves the requests waiting in the appropriate queue to be scheduled immediately so that they can proceed to try to allocate the required resources, see Fig. 3. This default FCFS arbitration scheme may be replaced by more advanced ones if required.

C. *Passive Resources*

Passive (software) resources are modeled using a simple combination of a pool of tokens and a blocking queue, see Fig. 5. Owning passive resources may be the prerequisite for requests to access active resources and are essential in modeling so called “critical sections” in software workflows allowing only one concurrent request (thread) in flight.

D. *Workflow Modules*

The workflow of requests is implemented in separate modules enabling an easy reuse of the basic modules providing hardware resources. The workflow modules post requests against the appropriate hardware modules to queue for the resources needed.

E. *Simple Compound Modules*

Compute modules on various levels can be created by combining more basic modules using the OMNEST Network Description (NED) language, see Fig. 4. For definiteness we will describe some compound modules in details below.

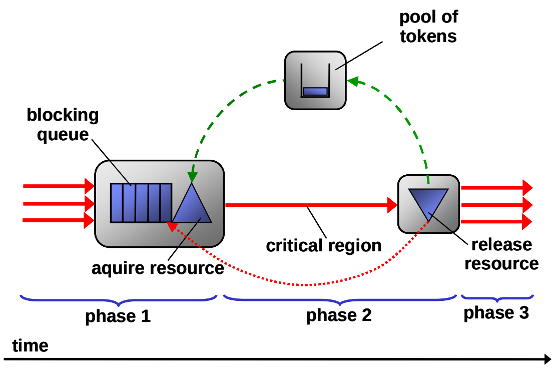


Fig. 5. Modeling a critical region allowing only one request (thread) concurrently in flight using a pool of tokens (passive resources) in combination with a blocking queue.

2 The architecture of our framework is independent of the specific discrete-event simulation tooling, but we found the OMNEST Network Simulation Framework quite suitable here due to its support of modularity and parallelizability.

1) *Network devices*

We introduce a general network device that maybe used to model general interconnects, switches routers or firewalls. For all of these devices, the basic architecture is the one shown in Fig. 6 and they differ e.g. in term of number of ports and various parameters like bandwidths, latencies, service times at controllers and the maximal number of requests in flight at the various components. During initialization, appropriate routing tables are created by the software workflow module. These modules are heavily used to glue cloud components together on various detail levels.

2) *Disk and disk arrays*

Using a generic network device (see above) to model an interconnect, it is quite straightforward to create a module for a disk, see Fig. 7. Different disk types like SATA , SAS or SSDs are then modeled by different parameterizations. Based on this general disk module, we create a generic module for disk arrays, see Fig. 8.

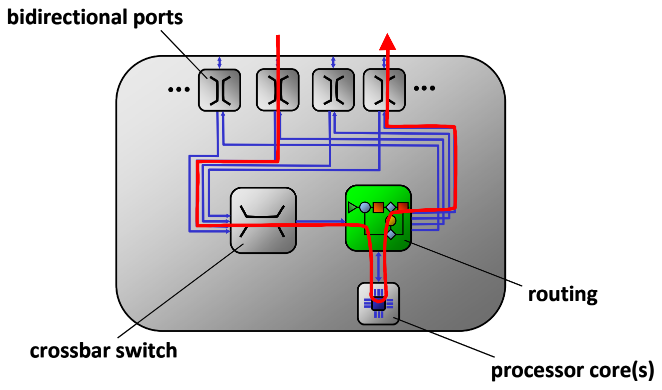


Fig. 6. A general network device consisting of bidirectional ports, a crossbar switch, a software workflow module implementing routing and processor cores. The red line indicates the flow of a request. Using various parameterizations, we can turn this module into e.g. an interconnection module, a switch, a router or a firewall.

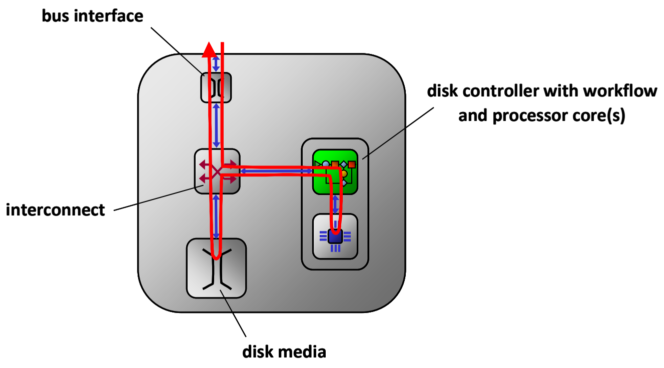


Fig. 7. Disk module consisting of bandwdith providers for modeling the bus interface and disk media, a network device for modeling the intra disk interconnect and the disk controller modeled as a combination of a processor cycle provider (processor core) and a software module implementing the request workflow at the disk.

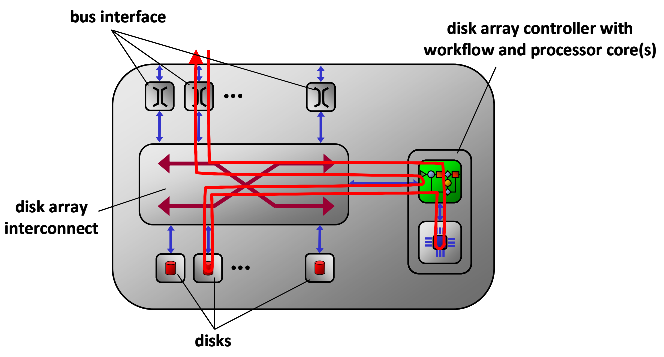


Fig. 8. Disk array module consisting of disks (see Fig. 7), network devices for modeling the bus interface, the disk array interconnect and the disk array controller modeled as a combination of a processor cycle provider (processor core) and a software module implementing the request workflow at the disk array.

3) *Compute nodes*

Compute node modules are built of basic processor cycle and bandwidth providers as well as of network devices, disks and disk arrays, see Fig. 9.

F. Virtual Machines

Virtual Machines (VMs) implement workflows on cloud level, e.g. the deployment of new images in clouds or various application workloads posted against VMs. Each VM implements several computational and IO phases, where each phase is separately parameterized e.g. by the number of required processor cores, IO bandwidth and maximal allowed concurrency.

G. Requests

Each request posted against the cloud is characterized by a set of attributes supporting e.g. the implementation of its

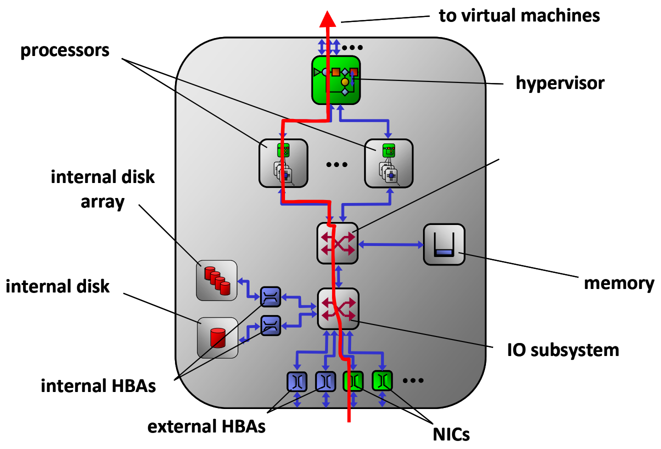


Fig. 9. A compute node module consisting of a software workflow modeling the hypervisor (also containing some passive resources in form of locks, not shown here), processors (consisting of various cores and a software module implementing the request workflow at the processor complex), a passive resource representing memory, various network devices modeling interconnects, bandwidth provider for modeling host bus adapters (HBAs), network interface controllers (NICs) and finally disk respectively disk array modules.

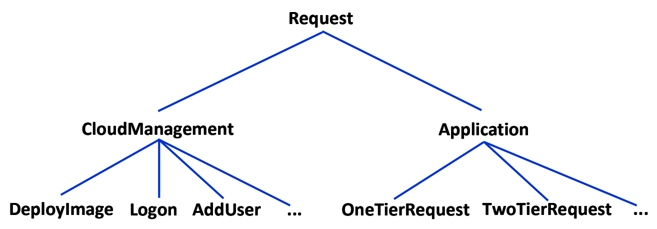


Fig. 10. Request hierarchy with the request of type “Request” at its root

workflow, characterizing its resource consumption and a collection of statistical data.

The requests form a hierarchy with a general request of type “Request” as the root of the tree, see Fig. 10. This supports the type dependent implementation of request workflows, e.g. some VMs may only accept requests of a certain types implementing a special workflow for each type. Hardware resources of course handle requests of any given type.

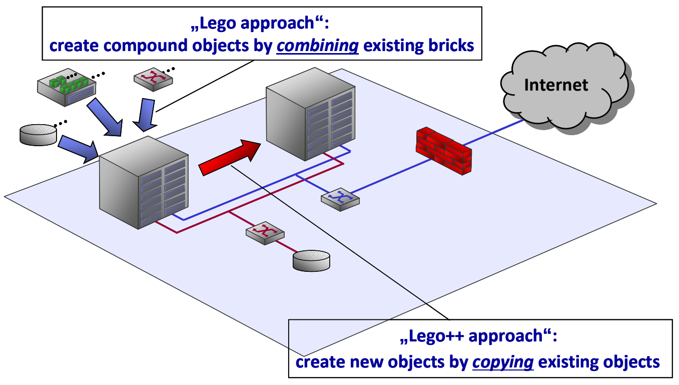
H. Workload Generator

In the workload generator module we have implemented all functionality related to generating, initializing and posting requests of various types against the cloud and collecting all request related statistics. Device related statistics like utilization and queue lengths are collected at the device simulation modules. Therefore, all requests need to return to the workload generator, even if they do not spend any simulation time on their way back.

We currently support the creation of open and closed streams of various request types possibly with bulk (batch) arrivals.

I. Complex Compound Modules

Complex compound modules to model one cloud data center or world wide distributed data centers of a cloud can be


„Lego approach“:
create compound objects by combining existing bricks
Internet
„Lego++ approach“:
create new objects by copying existing objects
Fig. 11. Complex compound modules can be created by combining more basic modules, e.g. a server rack by combining compute nodes with various VMs, disk arrays and network components. Using these racks, a data center can then be created by a copy-and-paste approach.

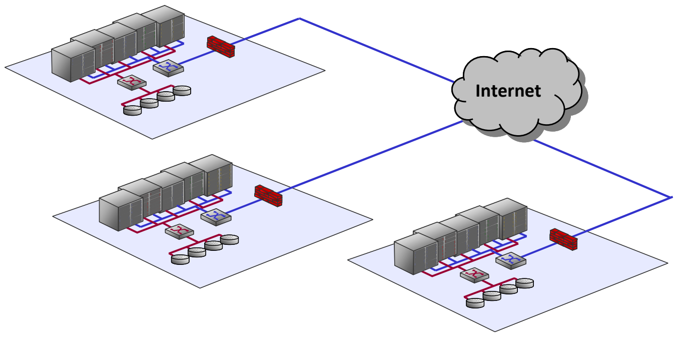
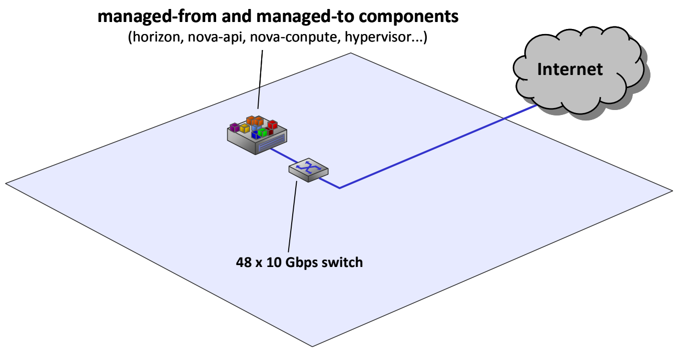


Fig. 12. A cloud consisting of a number of world-wide distributed data centers can also be created by applying the copy-and-paste approach, this time applied on data center level.

build by combining simpler compound modules and then copying the more complex ones, see Fig. 11 and Fig. 12 for more details here.

An important implementation feature is here the usage of abstract interfaces for modules allowing an easy exchange of submodules, e.g. replacing 10 Gbps switches with 40 Gbps ones in a compound data center module.


managed-from and managed-to components
(horizon, nova-api, nova-conpute, hypervisor...)
Internet
48 x 10 Gbps switch
Fig. 13. An OpenStack managed cloud with all OpenStack components on one compute node, i.e. this node is used as the managed-from as well as the managed-to system.

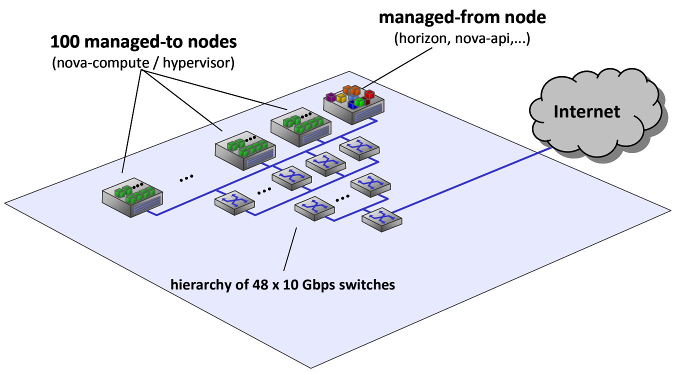


Fig. 14. An OpenStack managed cloud with all OpenStack management components on one compute node, but with separate nodes for the managed-to system

VI. Application Example: OpenStack Image Deployment

In this section we will demonstrate the application of our framework to simulating performance of OpenStack image deployment on various cloud architectures. We will show how simulations can be used to obtain information on the maximal workload an OpenStack managed cloud can support as well as to identify the limiting bottlenecks. In the following subsections, we will describe the steps associated with this simulation effort.

A. *Cloud Architectures*

For simplicity, we will consider here only the following two cloud architectures, see Fig. 13 and Fig. 14 with all

- OpenStack components being installed on only one compute node
- managed-from components of OpenStack being installed on one compute node and having 100 separate compute nodes available as managed-to nodes.

The infrastructure modeled consists of state-of-the-art main stream components like a 10 Gbps switches and NICs, disk arrays with 15K SAS disks and compute nodes with Intel* Xeon* processors[3].

B. *Simulation Scenarios*

The workflow of the OpenStack image deployment scenario modeled is shown in Fig. 15, Fig. 16 and Fig. 17.

C. *Some implementation details*

Our current model includes some idealized modeling assumptions like an almost ideal balancing of deployment workload to all of the compute nodes that might not be fulfilled by the current OpenStack implementation. Furthermore, we currently ignore any thrashing and the resource consumption of already deployed VMs.

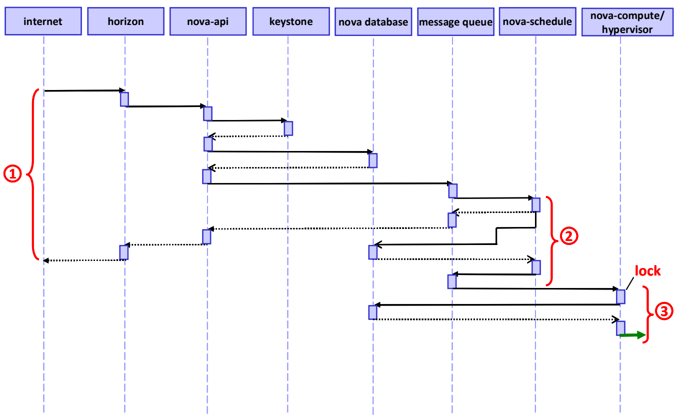


Fig. 15. Workflow phases of the OpenStack image deployment, part 1.

[3] A more detailed specification including the parameters used is beyond the scope of this publication.

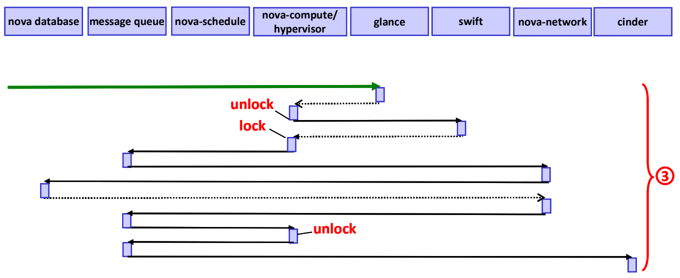


Fig. 16. Workflow phases of the OpenStack image deployment, part 2.

D. *Parameterization and Calibration*

Parameterization is based on internal measurements on real clouds as well as on the analysis of [25] and on specifications of the various hardware components used. Because of the rareness and noisiness of available measurement data for OpenStack image deployment, we calibrate our simulation using relative measurement results, e.g. we use the *increase* of mean response times for requests in bulk arrivals versus the single request times for calibration, see Fig. 18.

E. *Execution Characteristics*

The event throughput measured for our simulation is approximately 10^6 events/sec e.g. on a contemporary laptop and memory consumption for the 100 compute node cloud is

Phase	Components involved											Workflow details
	horizon	nova-api	keystone	nova database	message queue	nova-scheduler	nova-compute hypervisor	glance	swift	nova-network	cinder	
①	✔	✔	✔	✔	✔							- authentication - create instance entry in database - send ack to dashboard - forward request to scheduler
②				✔	✔	✔						- create schedule for instance - update entry in database - forward request to compute / hypervisor
③				✔	✔		✔	✔	✔	✔	✔	- retrieve instance infos from database - setup data for hypervisor - process instance request - get image URI from glance - get image from swift - reserve and allocate network - update instance entry in database - provision storage volume for instance - forward request to compute / hypervisor

Fig. 17. Some details associated with the various phases of the OpenStack image deployment workflow.

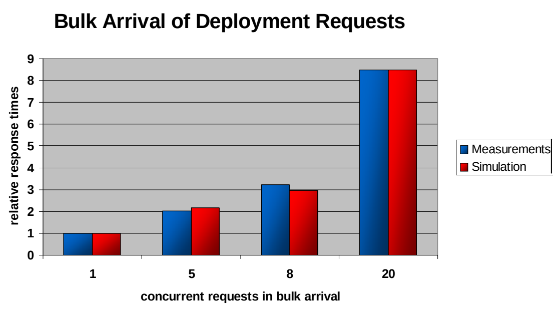


Fig. 18. A number of bulk arrival requests is used for calibration of the simulation.

approximately 2.5 GB. The execution time then depends significantly on the accuracy required and therefore on the number of iterations for each measurement point and various parameters like data segmentation sizes for networking and storage access and the size of time slices for accessing CPU cycles. Using random generators at several places in the simulation (e.g. for the initial placement of a VM), we found that the number of iterations should be at least 50 x the number of concurrent active deployment requests, e.g. 3000 for 60 concurrent active deployment requests. The total number of events processed here is approximately 1.8 x 10^7 resulting in an execution time of approximately 18 seconds[4].

F. *Logging and Tracing*

Scalable logging and measurement facilities are essential for any cloud simulation project. In our framework we support various logging and measurement modes that may furthermore be enabled selectively for each module or a group of modules allowing us to limit the generation of measurement results to the most relevant items only if required.

G. *Results*

As examples for typical simulation results, we present image deployment throughputs and response times for various levels of request concurrency, see Fig. 19 and 0. For the one node cloud architecture, the bottleneck is caused by contention for locks at the hypervisor, see Fig. 21.

As expected, the maximal image deployment throughput at the 100 managed-to node cloud architecture is far below the value of naively extrapolating throughput of the one node case[5]. This is caused by a bottleneck at the storage subsystem of the swift image repository, i.e. the bottleneck is moving from a software resource to a hardware component, see Fig. 22.

Because of the limited data available for parameterization and calibration, these results have to be considered as preliminary and they significantly depend on infrastructure and software workflow details beyond the scope of this publication.

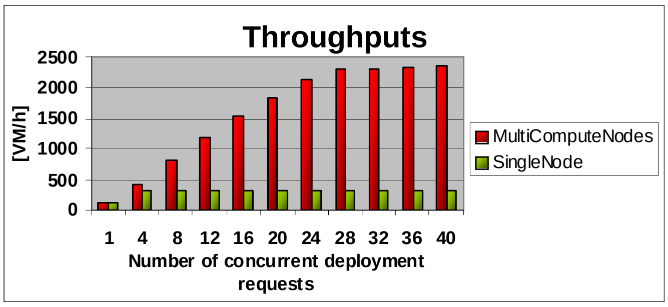


Fig. 19. Image deployment throughputs for various level of concurrency for both cloud architectures under consideration.

[4] Plus several minutes required for the initialization of the simulation modules.

[5] Approximately 300 VM/h x 100 compute node resulting in 30000 VM/h.

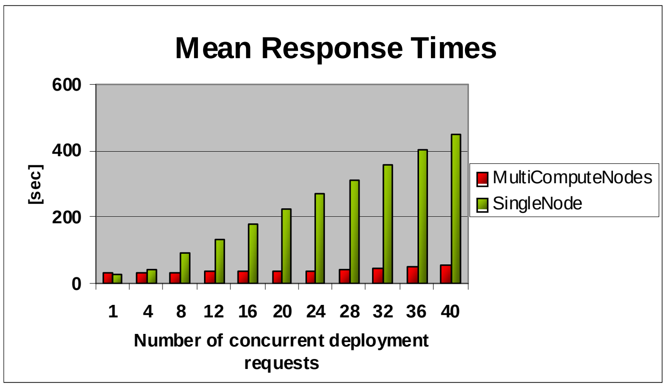


Fig. 20. Image deployment response times for various level of concurrency for both cloud architectures under consideration.

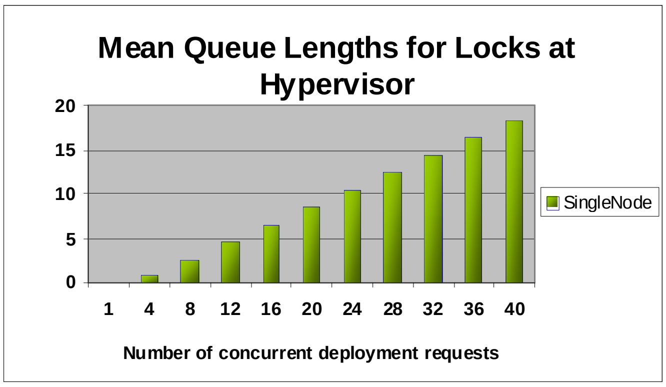


Fig. 21. Queue length for image deployment for various level of concurrency for the single node cloud architecture. No significant queueing occurs here in the multiple node architecture.

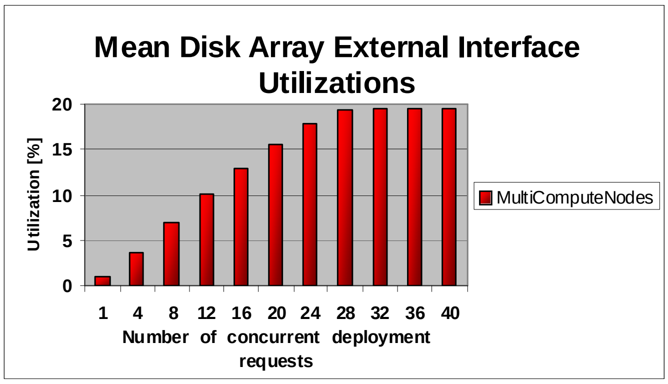


Fig. 22. Utilization of the external interface at the Swift image repository indicating an icreased contention for bandwidth here. In the single node case the utilization here is negligible.

VII. Conclusion

We have designed and implemented a comprehensive, modular, highly scalable, accurate and flexible performance simulation framework and demonstrated how it can be successfully applied to simulate performance of image deployment requests of OpenStack managed clouds.

VIII. Outlook

Future work will focus on leveraging the framework to optimize the design of current and upcoming cloud architectures on hardware and software level. This will most likely result in additional requirements against the framework which may be added to the framework quite easily by exploiting its modular design.

Acknowledgment

The authors would like to thank the IBM management at IBM Germany Research & Development GmbH and IBM Research GmbH, Zurich Research Laboratory for supporting this work over the last years. We especially thank the IBM cloud performance verification team for sharing with us their high-quality cloud performance measurement results.

References

[1] Cloud Computing, http://en.wikipedia.org/wiki/Cloud_computing.

[2] Above the Clouds: A Berkeley View of Cloud Computing, http://www.eecs.berkeley.edu/Pubs/TechRpts/ 2009/EECS-2009-28.pdf .

[3] G. Bloch, S. Greiner, H. de Meer, K.S. Trivedi. Queueing Networks and Markov Chains, Second Edition. Wiley-Interscience, 2006.

[4] L. Kleinrock. Queueing Systems, Volume 1: Theory. John Wiley, 1975.

[5] L. Kleinrock. Queueing Systems, Volume 2: Computer Applications. John Wiley, 1976.

[6] A.M. Law, W.D. Kelton. Simulation Modeling and Analysis, Third Edition. McGraw-Hill, 2000.

[7] J. Banks, J.S. Carson II, B.L. Nelson, D.M. Nicol. Discrete-Event System Simulation, Fourth Edition. Prentice Hall, 2005.

[8] P. Altevogt, W. Denzel, T. Kiss. Proc. of the 2011 Winter Simulation Conference (WSC'11). S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, eds.

[9] OpenStack, http://www.openstack.org.

[10] Wei Z., Yong P., Feng X., Zhonghua D., “Modeling and Simulation of Cloud Computing: A Review”, IEEE Asia Pacific Cloud Computing Congress (APCloudCC), 2012, pp.20-24.

[11] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. 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.

[12] Wickremasinghe, B., Calheiros, R.N. , Buyya, R., “CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications”, IEEE 24th International Conference on Advanced Information Networking and Application, 2010, pp. 446 – 452.

[13] S. K. Garg and R. Buyya, “NetworkCloudSim: modeling parallel applications in cloud simulations,” 4th IEEE International Conference on Utility and Cloud Computing, pp.105-113, 2011.

[14] R. N. Calheiros, M .A. S. Netto, C. A. F. De Rose, and R. Buyya, “EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications,” Software-Practice and Experience, 00: 1-18, 2012.

[15] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer, “GroudSim: an event-based simulation framework for computational grids and clouds,” CoreGRID/ERCIM Workshop on Grids and Clouds. Springer Computer Science Editorial, Ischia, 2010.

[16] A. Nunez, J. L. Vazquez-Poletti, A. C. Caminero, G. G. Castane et al., “iCanCloud: a flexible and scalable cloud infrastructure simulator,” Journal of Grid Computing, Vol.10, No.1, pp.185-209, 2012.

[17] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, “DCSim: a data centre simulation tool for evaluating dynamic virtualized resource management,” The 6th International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standard and the Cloud, 2012.

[18] D. Kliazovich, P. Bouvry, and S. U. Khan, “GreenCloud: a packet-level simulator of energy-aware cloud computing cata centers,” Journal of Supercomputing, special issue on Green Networks, 2011.

[19] I. Sriram, “SPECI, a Simulation Tool Exploring Cloud-Scale Data Centers,” CloudCom’09, LNCS 5931, pp.381-392, 2009.

[20] W. Zhang, X. Huang, N. Chen, W. Wang, H. Zhong, “PaaS-Oriented Performance Modeling for Cloud Computing”, IEEE 36th Annua Computer Software and Applications Conference (COMPSAC), 2012.

[21] H. Khazaei, J. Misic, V. B. Misic, “A Fine-Grained Performance Model of Cloud Computing Centers”, IEEE Transactions on Parallel and Distributed Systems , Vol. X, No. Y, 201Z.

[22] Ewing L. Lusk and Anthony Skjellum, Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation). The MIT Press, Cambridge Massachusetts, 1999.

[23] OMNEST – High-Performance Simulation for All Kinds of Networks. Accessed May 7, 2011. http://www.omnest.com/.

[24] A. Varga, R. Hornig, An overview of the OMNeT++ Simulation Environment. In Proceedings of First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools’08). Marseille, France, March 2008.

[25] P. Feiner, Scaling the Boot Barrier: Identifying & Eliminating Contention in Openstack, http://www.openstack.org/summit/portland-2013/session-videos/presentation/scaling-the-boot-barrier-identifying-and-eliminating-contention-in-openstack.