Orchestrated service deployment, maintenance, and debugging in IaaS clouds for crowd computing[*]

Robert Lovas

Institute for Computer Science and Control, Hungarian Academy of Sciences
Kende u. 13-17, Budapest 1111, Hungary

robert.lovas@sztaki.mta.hu

Abstract. The One Click Cloud Orchestrator framework provides facilities to create and maintain virtual e-infrastructures, such as crowd or volunteer computing platforms, on various cloud systems. Complex service deployment and maintenance scenarios supported by such cloud orchestrators pose new challenges since software engineers and testers face (among others) the probe effect, the irreproducibility, the completeness problem, and also the large state-space to be handled during the debugging phase. In this paper, a highly automated debugging methodology, the 'cloudified' macrostep-by-macrostep concept, is discussed focusing on the automatic generation of successive consistent global states for cloud based complex service deployment and maintenance processes. The paper outlines an on-demand crowd computing platform deployment use case on an Infrastructure as a service (IaaS) cloud. The use case leverages on OCCO and its energy consumption evaluation is presented as one of top issues regarding e-infrastructure operation.

Keywords: cloud, orchestration, debugging

1 Introduction

Infrastructure as a service (IaaS) cloud systems allow automated construction and maintenance of virtual infrastructures 1 applying the concept of virtual machines (VMs) as the fundamental building block. Thus, IaaS systems enable the creation, management and destruction of VMs but the current IaaS solutions barely manage multiple VMs or focus only on network management among multiple VMs. Recent research efforts were able to address and answer these issues with the cloud orchestrator concept 23. On the other hand, complex service deployment and maintenance scenarios of such cloud orchestrators still pose new challenges since software engineers and testers must face (among others) the probe effect, the irreproducibility, the

[*] The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreements no. 608886 (CloudSME), and no. 312297 (IDGF-SP).

completeness problem, and also the large state-space to be somehow handled during the debugging phase.

For instance, it seems a given orchestrated cloud deployment scenario always generates correct results on a particular cloud platform or on a set of cloud platforms in hybrid and federated clouds (where the software engineers originally developed and deployed their services) but often fails on other cloud platforms operated by other IaaS providers. Mostly, the reason for this behaviour is the varying relative speeds of deployment tasks together with the untested race conditions. The different timing conditions might be occurring more frequently on cloud-based platforms then on dedicated clusters or traditional supercomputers because of the different implementation of the underlying operating systems/communications layers and the unpredictable network traffic, CPU loads or other dynamical changes. The above described phenomenon can be very crucial because one cannot ensure that the cloud based deployment always capture the same nodes with almost the same timing conditions in case of (re)deployment or VM failure.

The only way to prove the cloud platform agnostic feature of complex deployment and maintenance strategies is to leverage on advanced systematic debugging methods in order to find the timing/architecture dependent failures in the designed deployment description and orchestrator. For this purpose I applied and also extended the macrostep-based systematic debugging methodology that has been introduced originally for message passing parallel programs developed in the P-GRADE graphical programming environment 4. The experimental prototype is designed for the One Click Cloud Orchestrator 5 (OCCO) framework and the presented work partly relies on the achievements related to the HARNESS metadebugger 6. Thus, the presented work attempts to overcome the limitation of existing debugging solutions 78910 and extend some advanced debugging methods from parallel and distributed systems towards IaaS clouds.

In this paper, a highly automated debugging methodology and an experimental toolset are discussed focusing on the automatic generation of successive consistent global states for cloud based complex service deployment and maintenance, called cloudified macrostep-by-macrostep. The second part of the paper outlines an on-demand crowd computing platform deployment use case and example for OCCO with energy consumption evaluation on an IaaS cloud.

2 Background: One Click Cloud Orchestrator

This section gives a short overview of the OCCO components (see Fig. 1) that allow the creation and fault tolerant maintenance of virtual e-infrastructures, such as crowd computing platforms (see Section 5), capable of hosting various applications. In order to achieve its goals, OCCO contains two kinds of components: (i) end user oriented components, (ii) components focusing on the definition and inspection of ever-evolving virtual e-infrastructures.

The end user oriented components are the template store and the notification service. With the template store, OCCO allows virtual e-infrastructure designers to create such infrastructure descriptions that are easily customizable by end users (operators) and also capable to describe the peculiarities of virtual e-infrastructures at the same time. With the exception of the customization options, we will not discuss further the details of the virtual e-infrastructure description as it is out of scope. The customization options are specified in the description as hints attached to the attributes that the user should be able to change. The other user oriented component of OCCO is the notification service, that plays crucial role after the infrastructure is customized by the user and its creation is requested from the virtual e-infrastructure management related components. The notification service enables automated reactions to particular infrastructure maintenance related activities, e.g., when the infrastructure first becomes available or when it has scaled to allow higher processing power or availability.

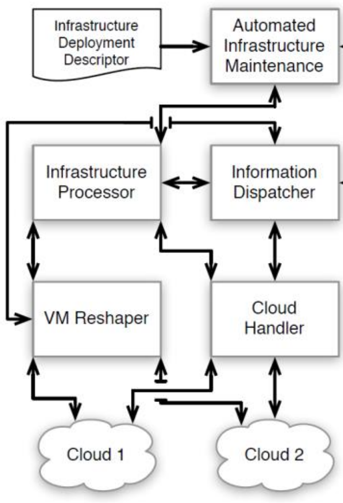

Fig. 1. High level overview on OCCO virtual e-infrastructure management components

The components actually managing the virtual e-infrastructures for the end user (see Fig. 1): (i) Automated Infrastructure Maintenance, (ii) Infrastructure Processor, (iii) Cloud Handler, (iv) VM Reshaper, and (v) Information Dispatcher.

The Automated Infrastructure Maintenance component is responsible for understanding the customized deployment descriptors. But this component does not only provide the descriptor processing capabilities but it also offers dependency resolution (so the nodes of the particular instantiated infrastructures are instantiated in the proper order), scalability and error resilience rule evaluation and enactment. Therefore, the end user does not have to intervene in its infrastructure's internal operations.

The Infrastructure Processor component of OCCO is used to ensure that the definitions of the infrastructure nodes are propagated to the VM Reshaper, which allows runtime reconfiguration of a virtual machine to meet a particular node description. In addition, the Infrastructure Processor sends such virtual machine requests to the Cloud Handler that ensures the intended role of the VMs after their creation. Next, the

Cloud Handler is responsible of selecting a cloud infrastructure that will host a particular VM, and interfacing with the cloud infrastructure provider.

Finally, the Information Dispatcher component allows the Automated Infrastructure Maintenance component to determine the current state of the e-infrastructure to be used during the scaling and error resolution rule evaluation process.

The next sections will show how Infrastructure Processor component can be controlled when the new macrostep based debugger is in action.

3 Problems arising during orchestration

The following diagram (see Fig. 2) illustrates a part of the OCCO in use; the Automated Infrastructure Maintenance, the Information Dispatcher, and the Infrastructure Processor together with the introduced cloudified macrostep debugger. In the example, there is a 4-VM basic deployment with A, B, C, D nodes in the dependency graph defined in the Infrastructure Deployment Descriptor where node C is scalable according to the user or performance/availability needs. Macrosteps 1-3 are the major phases of deployment (see the definition of macrosteps later) and Step 3' shows a VM crash with automatic recovery in Macrostep 4 as a part of maintenance phase. Finally, in Macrostep 5 two more nodes are launched (also in the maintenance phase).

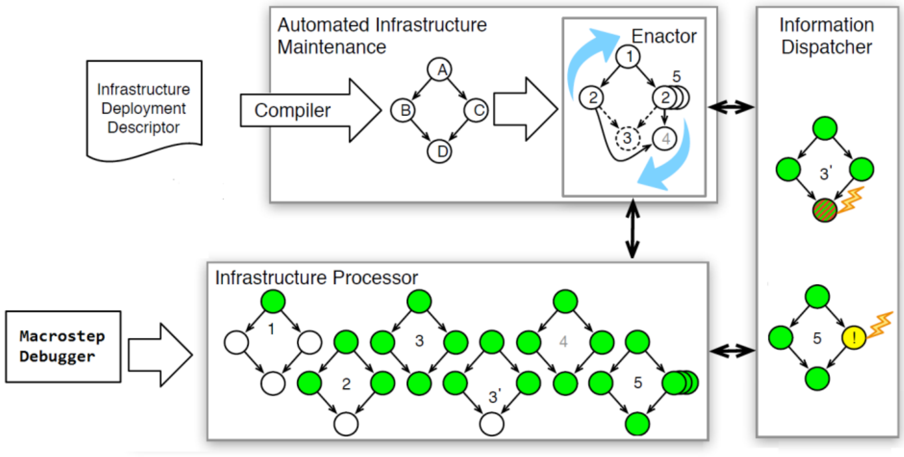

Fig. 2. Selected OCCO components with the cloudified macrostep debugger in use

It is easy to imagine a real-life situation when one has to take into consideration the large number of mostly concurrent notifications, particularly for synchronisation and information exchanging purposes, that are required among the individual processes in the Infrastructure processor dealing with the (re)deployment of VMs individually in a more complex scenario.

The following issues make debugging of cloud deployment and maintenance much more difficult than traditional sequential debugging:

1. To address the problem of the *large number of hierarchical deployment steps with dependencies and dynamic changes* of the deployment, since the replacement of a node (due to a crashed or unreachable VM) may require e.g. VM reshaping of its neighbour VMs,
2. The *non-deterministic behaviour* of cloud environment makes the actual behaviour of the deployment dependent on actual speed of the individual VM deployments due to the distinct CPU speeds, varying operating system and hypervisor effects, and unpredictable I/O and communication delays in *the heterogeneous and ever-changing cloud environment*. It requires the OCCO distributed debugger to provide facilities to detect those situations, and to help the software developer somehow evaluate correctness properties based on the deployment specification for various possible execution timings during the debugging phase. It also required techniques to allow reproducible and coherent observation of such error situations.
3. *Constructing consistent global states* must be also considered in general because the evaluation of erroneous situations depends on accurate observations. In general, the accurate observation can only be approximately achieved in any distributed system, by remote observation due to the absence of global system states. To solve this problem, OCCO distributed debugger provides strategies for the observation of consistent deployment and maintenance states leveraging on the cloudified macrostep-by-macrostep execution (see the details in Section 4).
4. *Probe-effect* due to the observation and control mechanisms is a well-known phenomenon: any observation may affect the system. Therefore, the OCCO debugger relies on techniques that give efficient control over occurring race conditions and time-dependent circumstances; the debugger is able to gain detailed information about the structure (dependencies) of deployment steps/inter-VM notifications.

In order to handle these problems in different debugging sessions, I present the description for a novel, automatic generation of successive global consistent states, called cloudified macrostep based deployment execution in the next section.

4 Solution: Cloudified macrostep based deployment

The fundamental ideas of the further developed macrostep debugging methodology can be summarized by the following concepts:

1. cloudified collective breakpoints,
2. enhanced macrosteps,
3. cloudified macrostep-by-macrostep deployment mode,
4. deployment execution tree,

In the rest of this section, these concepts are described as well as some implementation issues.

The fundamental aim of cloudified macrostep-based deployment is the generation of consistent cuts (or global states) for orchestrated deployments and maintenance in OCCO. The idea of cloudified macrostep is based on the concept of *collective break-*

points; a collective breakpoint consists of a finite number of single breakpoints placed in different (deployment) processes, and the collective breakpoint is hit if all the breakpoints belonging the collective breakpoint are hit.

Fig. 3 shows a complex, synthetic example for illustration purposes with several collective breakpoints such as NAR^1_1- NAS^1_2- NAS^1_3- NAR^1_4, which are placed on the inter-VM notification primitives related to sender (NAS), receiver (NAR) or alternative/collective receiver (NACR) methods in each VM deployment process in the Infrastructure processor component of OCCO. These inter-VM notifications are typically synchronised actions, and indexed by the corresponding VM deployment process number (lower index) and a serial number (upper index) in Figure 3. The set (region) of executed lower level deployment steps between two consecutive collective breakpoints is called a *macrostep*. A detailed generic definition of macrostep is given in 1112.

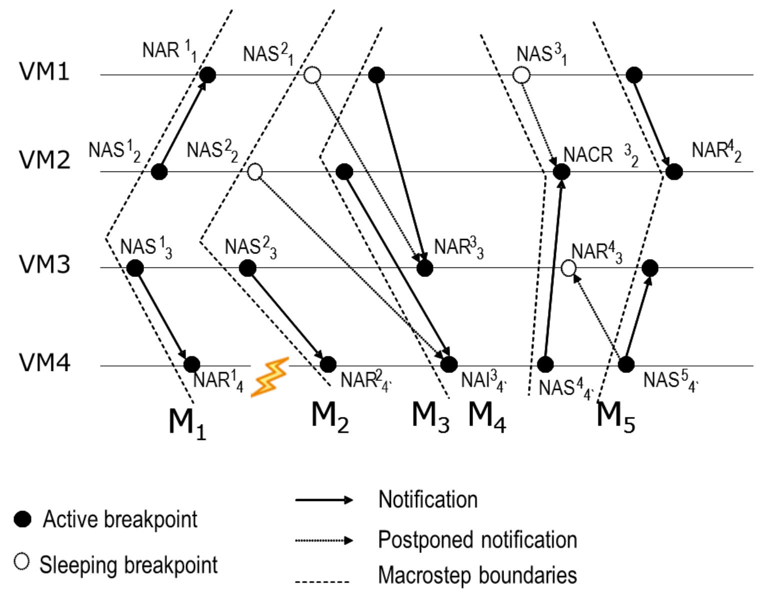

Fig. 3. Illustration for macrostep based execution of complex deployment with inter-VM notifications

A single breakpoint of the collective breakpoint is called *active* if it was hit in a macrostep and its associated inter-VM notification can be completed (e.g. see NAS^2_2 in Fig. 3). On the other hand, a breakpoint is called sleeping if it was hit in a collective breakpoint but its associated notification cannot be completed during the next macrostep thus, it will be a part of the next collective breakpoint. For example, a send instruction (NAS^2_1) of a given VM deployment process (VM1) wants to notify another VM deployment process (VM4) synchronously, but it is communicating (engaged) with a 3rd VM deployment process (VM3). That is why, the breakpoint placed at NAS^2_1 operation is a sleeping breakpoint and can be found in the next collective breakpoint. Similarly to this, NAR34’ is also a sleeping breakpoint, since it must wait for NAS^4_5 .

Please note that in this example the VM4 crashed (or became unavailable) between M1 and M2, therefore the NAS^1_3- NAR^1_4 notification must be replicated in macrostep M2 with NAS^2_3- $NAR^2_{4'}$.

The ***cloudified macrostep-by-macrostep deployment mode*** of orchestrated cloud service deployments can be defined as follows; in each macrostep every VM deployment process is forced to run until a collective breakpoint is hit. Thus, the boundaries of the macrosteps (see Fig. 3, M_1, M_2, …) are defined by a series of global breakpoint set, and the consecutive consistent global states of complex deployment are generated automatically.

At replay, the progress of VM deployments is controlled by the stored collective breakpoints and the orchestrated deployment is automatically executed again macrostep-by-macrostep as in the original deployment phase.

In order to ensure the correct replay, according to the original macrostep concept the debugger should store the history of collective breakpoints, the acceptance order of messages at alternative/collective receiver actions and the external input parameters.

Additionally, in the cloud environment the debugger also stores the details about the events corresponding to reconfigurations; when e.g. a new VM is deployed, released or failed anywhere in the cloud as well as the versions of packages used for updating the VMs by the VM reshaper component in OCCO. To handle the dynamic, elastic and fault tolerant behaviour of OCCO based systems, the basic concept is the following. During the initialisation the macrostep debugger it places some so-called ‘system breakpoints’ in the OCCO Infrastructure Processor (see Fig. 2) in order to detect all changes/reconfiguration of deployment in advance.

At replay, the progress of deployment tasks are controlled by the stored collective breakpoints, reconfiguration events, and stored versions of external applied packages, then the deployment and maintenance are automatically executed again macrostep-by-macrostep as in the original deployment phase. The debugger is also responsible for grabbing/releasing VMs with the help of Cloud Handler in OCCO (if it is needed).

The deployment execution path is a graph whose nodes represent the boundaries of macrosteps (i.e. consistent global states) and the directed arcs indicates the possible macrosteps (i.e. the possible state transitions between consecutive global states). The *deployment execution tree* is a generalization of the deployment execution path; it contains all the possible deployment execution paths of an orchestrated deployment assuming that the non-determinism is inherited either at the notification actions at alternative/collective receiver side (NACR) or a random failure of an arbitrary set of VMs. Nodes of the deployment execution tree can be of four types: (i) Root node, (ii) Alternative nodes, (iii) Deterministic nodes, (iv) Termination node.

Breakpoints can be placed at the nodes of the deployment execution tree. Such breakpoints are called deployment meta-breakpoints. The role of deployment meta-breakpoints is analogous with the role of the breakpoints of sequential programs. A breakpoint in a sequential program means to run the program until the breakpoint is hit. Similarly, a deployment meta-breakpoint at a node of the deployment execution tree means to place the collective breakpoint belonging to that node and run the deployment until the collective breakpoint is hit. Replay guarantees that the collective breakpoint will be hit and the deployment will be stopped at the requested node.

Testing can be also supported by traversing exhaustively the deployment execution tree with all the possible execution paths in it. Therefore, the deployment execution tree represents a search space that should be explored completely using injections of various combinations of VM errors systematically. Accordingly, systematic testing and debugging of an orchestrated IaaS deployment require (i) generation of its deployment execution tree (ii) exhaustive traverse of its deployment execution tree. With the help of the cloudified macrostep-by-macrostep concept both of these issues can be solved and implemented in a similar way as they have been implemented in DIWIDE 11.

5 Experiments: Energy efficient crowd computing e-infrastructure with OCCO

This section addresses the testing of OCCO further and performing different analysis with an on-demand demonstration platform for crowd computing purposes.

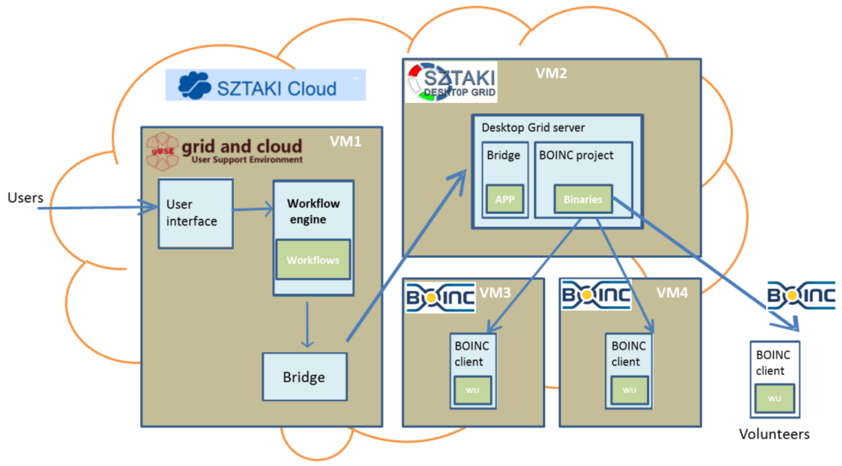

Fig. 4. OCCO complex test case: BOINC based crowd computing platform with cloud burst and science gateway

The infrastructure template is aimed at providing a distributed computing infrastructure (DCI) for crowd computing with a science gateway attached as a workflow-based frontend. The DCI is implemented by a BOINC 13 based SZTAKI Desktop Grid 14 with a molecular docking simulator. As an extra functionality, the BOINC project is associated with a public IP address, therefore the home or PC lab computers from university campuses may also attach external BOINC clients to the server. Using automatically deployed and configured BOINC clients in VMs, the deployed computational resources are able to automatically join a BOINC project. The number of clients can be customized in the descriptor template. Computing jobs arrive to the BOINC project as work units (WUs) with the help of the gUSE science gateway and bridge 15 that are also automatically deployed additionally to the DCI. Overall, the demonstration system shows how a complete gateway, bridge together with a partly virtualized and operational BOINC e-infrastructure can be deployed on the SZTAKI cloud by OCCO and how the components attach to each other (see Fig. 4).

The energy consumption is always an issue when one talks about crowd (or volunteer) computing involving resources from the enterprises/institutes or home users. SZTAKI Desktop Grid team performed measurements about the effect of crowd computing applications on the power consumption of the cloud that hosts the BOINC server with its workers (clients) using One Click Cloud Orchestrator.

The scenario addresses the use case when a given number of workunits of the application are executed on a cloud environment in order to handle QoS related problems (e.g. the longtail effect) of the crowd computing applications 16.

The experiments were carried out on the OpenNebula (version 4.2.0) based SZTAKI Cloud. The compute nodes in the cloud are heterogeneous (8 to 64 CPU core machines), interconnected with 4x10 GbE/48x1GbE switch, and a 32 TB high performance iSCSI storage. The 1 or 2 RU high servers are built in a rack, and two independent power sources serve the 24/7 operation of the cloud with one PowerWare 9155 UPS and one PowerWare 9130 UPS. Both UPS devices provide high level monitoring facilities that have been used during the experiment via SNMP. For gathering, aggregating, and visualising the monitored data from the UPS devices and from the cloud system itself the SZTAKI Desktop Grid team used Zabbix version 2.0.3.

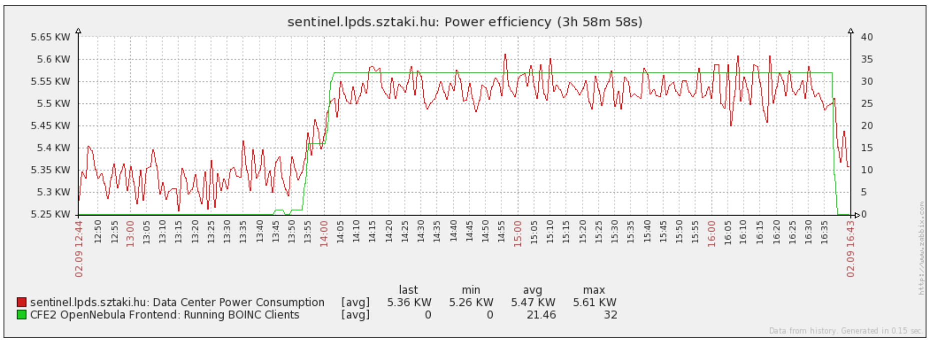

Fig. 5. Power consumption (red) and number of running BOINC clients (green)

In the experiment 32 BOINC clients (with 1 virtual CPUs and 512 MB RAM for each client) have been launched through the EC2 interface of the cloud using OCCO. The clients attached to the BOINC server and executed continuously the application in a 4-hour timeframe.

As it can be seen in Fig. 5 the 32 BOINC clients increased the average power consumption from the 5325 Watt to the level of 5550 Watt. It means approximately 225 Watt extra consumption for the 32 BOINC clients, i.e. 7 Watt per client. However, it is more interesting that the 32 clients caused only 4.2% increase in the overall power consumption contrary to the approximately 33% higher CPU utilisation of the entire cloud (including some overhead e.g. from the cloud hypervisor system); the number of CPU cores increased from the level 20s to the level of 80s from the available 182 CPU cores shown on Fig. 6.

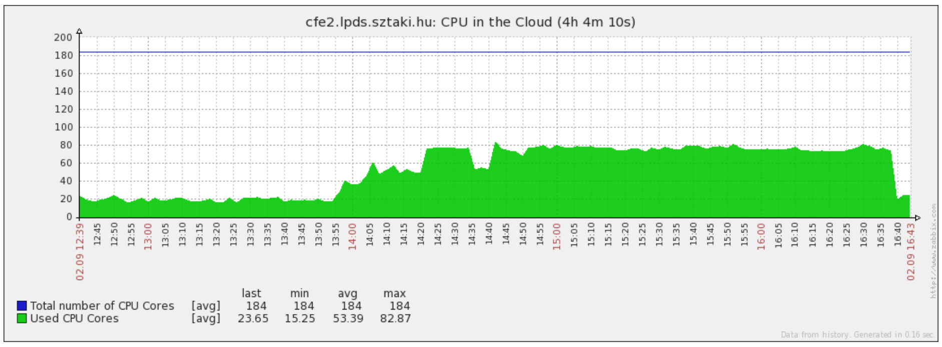

Fig. 6. Total number of CPU cores in the cloud vs. used CPU cores

6 Related and future work

In general, cloud providers do not offer high level and advanced debugging facilities to their user that are similar to the described macrostep based concept. There are some related attempts for HPC applications 7 but the typical example is the remote cloud debugging 8 feature for Windows Azure Cloud Services 10 that can be considered the most basic functionality of all distributed debugger. Another example is the Cloud Debugger 9 from Google, where the creation of snapshots is allowed to the software engineers for Java applications but not for multiple, orchestrated VMs.

Concerning the future work; the time and resource requirements of the exhaustive debugging can be decreased by magnitude of orders when debugger takes the advantage the large number of resources included in cloud environment by starting more (even hundreds) test scenarios at the same time with different deployment execution paths. However, due to the combinatorial explosion of the deployment execution tree the complete exhausted testing and the elimination of all defects of a real size cloud deployment scenario is still impossible even with simultaneous discovering of deployment execution paths using large-scale cloud platforms. Hence, the debugging and testing phases must be stopped at a certain point, where the quality parameters of the deployment are reliable enough to release it as a beta or a final version. Including the widespread Rayleigh model in the proposed solution is a good candidate for the estimation of error density.

Moreover, to improve the efficiency of macrostep-based debugging methodology, two well-funded model checking techniques have been already introduced in parallel debugging, such as simulation of program by its coloured Petri-net model, and program verification using temporal logic specification 12. The adaptation of these formal methods into the OCCO debugger in order to steer the debugging session towards suspicious situations and detect bugs automatically would help significantly reduce the necessary user interactions.

7 References

1. S. Bhardwaj, L. Jain, and S. Jain. Cloud computing: A study of infrastructure as a service (IaaS). International Journal of engineering and information Technology, 2(1):60-63, 2010.
2. M. Caballer, I. Blanquer, G. Molto, and C. de Alfonso: Dynamic management of virtual infrastructures. Journal of Grid Computing, 13(1):53-70, 2015.
3. R. Dukaric and M. B. Juric: Towards a unified taxonomy and architecture of cloud frameworks. Future Generation Computer Systems, 29(5):1196-1210, 2013.
4. P. Kacsuk, et. al: P-GRADE: a Grid Programming Environment, Journal of Grid Computing, 1(2):171-197, 2003
5. Gábor Kecskémeti, et. al: One Click Cloud Orchestrator: Bringing Complex Applications Effortlessly to the Clouds, Euro-Par 2014: Parallel Processing Workshops, Lecture Notes in Computer Science Vol. 8806, pp. 38-49, 2014
6. R. Lovas, V. Sunderam: Extension of macrostep debugging methodology towards metacomputing applications, In: Computational Science - ICCS 2001, Lecture Notes in Computer Science, Vol. 2074, pp. 263-272, 2001
7. J. Zhang, et. al: CDebugger: A scalable parallel debugger with dynamic communication topology configuration, Proceedings of the 2011 International Conference on Cloud and Service Computing (CSC '11), pp. 228-234, 2011
8. Junjie Cai, et. al: Remote Debugging in a Cloud Computing Environment, Patent US20140366004, 2014
9. Cloud Debugger, https://cloud.google.com/tools/cloud-debugger (May 19, 2015)
10. Debugging Cloud Services, https://msdn.microsoft.com/en-us/library/azure/ee405479.aspx (January 26, 2015)
11. P. Kacsuk, R. Lovas, and J. Kovács. Systematic Debugging of Parallel Programs in DIWIDE Based on Collective Breakpoints and Macrosteps. In 5th Euro-Par Conference, Lecture Notes in Computer Science, Vol. 1685, pp. 90-97, 1999.
12. R. Lovas, P. Kacsuk: Correctness Debugging of Message Passing Programs Using Model Verification Techniques. Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, Vol. 4757, pp. 335-343, 2007.
13. D. P. Anderson: BOINC: A system for public-resource computing and storage. In Proceedings of the 5th International Workshop on Grid Computing (GRID 2004), pp. 4-10, 2004.
14. P. Kacsuk, et. al: SZTAKI Desktop Grid (SZDG): A Flexible and Scalable Desktop Grid System, Journal of Grid Computing, 7(4):439-461, 2009
15. P. Kacsuk, et. al: WS-PGRADE/gUSE Generic DCI Gateway Framework for a Large Variety of User Communities, Journal of Grid Computing, 10(4), pp. 601-630, 2012
16. S. Delamare, et, al: SpeQuloS: A QoS Service for BoT Applications Using Best Effort Distributed Computing Infrastructures. Proc. Of International Symposium on High Performance Distributed Computing (HPDC'2012), pp.173-186. ACM press, 2012.