One Click Cloud Orchestrator: bringing Complex Applications Effortlessly to the Clouds*

Gabor Kecskemeti, Mark Gergely, Adam Visegradi, Zsolt Nemeth, Jozsef Kovacs and Peter Kacsuk

Institute for Computer Science and Control, Hungarian Academy of Sciences.
1111 Budapest, Kende u. 13-17, Hungary.
kecskemeti.gabor@sztaki.mta.hu

Abstract. Infrastructure cloud systems offer basic functionalities only for managing complex virtual infrastructures. These functionalities demand low-level understanding of applications and their infrastructural needs. Recent research has identified several techniques aimed at enabling the semi-automated management and use of those applications that span across multiple virtual machines. Even with these efforts however, a truly flexible and end-user oriented approach is missing. This paper presents the One Click Cloud Orchestrator that not only allows higher level of automation than it was possible before, but it also allows end-users to focus on their problems instead of the complex cloud infrastructures needed for them. To accomplish these goals the paper reveals the novel building blocks of our new orchestrator from the components closely related to infrastructure cloud to the ways virtual infrastructures are modeled. Finally, we show our initial evaluation and study on how the orchestrator fulfills the high level requirements of end-users.

1 Introduction

Infrastructure as a service (IaaS) cloud systems allow automated construction and maintenance of virtual infrastructures [2]. Such infrastructures exploit the concept of virtualization and use virtual machines (VMs) as the smallest building block. Therefore, in their core, IaaS systems enable the creation, management and destruction of virtual machines through a convenient and machine accessible API. The reliability and possibility of virtually infinite sized infrastructure leases of commercial IaaSs lead to their fast adoption and widespread use.

Unfortunately, even with these IaaS functionalities, setting up and using complex virtual infrastructures is the privilege of a few because of several reasons: (*i*) current IaaS APIs barely manage more than single VMs, but (*ii*) even if they do so, they are mostly focused on network management among user controlled VMs.

* The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 283481 (SCI-BUS) and no. 608886 (CloudSME) as well as from the KMR 12-1-2012-0119 (Cloud akkreditációs szolgáltatás indítása, CLAKK).

Thus, IaaS systems have severely limited applicability because they require deep system administration knowledge. This highlights the need for techniques capable of automating the creation and management of large-scale applications deployed over potentially thousands of virtual machines without knowledge about how particular virtual machines or their networking is set up [17].

Recent research answered these needs with the cloud orchestrator concept [3, 8] that shifts the center of attention from sole VMs to the required functionalities. To reduce the needed networking knowledge, orchestrators also expect the description of dependencies between the different functional blocks of a large-scale application. Although this description greatly reduces the expertise needed to operate complex infrastructures, there are still several outstanding issues (e.g., VM creation conforming to required functionalities, cross VM or cross-cloud error resilience, autonomous scaling techniques that not only consider application load but other properties – like cost, energy – as well, high level user interfaces).

In this paper, we propose a new orchestrator technique – called the One Click Cloud Orchestrator (OCCO) – that targets these issues with novel approaches. Our technique is based on a virtual machine management technique independent of infrastructure. Next, OCCO encompasses several software delivery approaches from custom and on the fly virtual machine construction (e.g., with Chef) to supporting user built virtual machine images that are optimized for a particular purpose. The proposed orchestrator also incorporates a unified infrastructure state model (which allows the system to determine what functionalities are missing or perform below expectations). Finally, on top of these components, OCCO offers customizable techniques for automated infrastructure maintenance (ranging from simple multi-VM infrastructure creation, to highly available and scalable application management).

To reveal the capabilities of OCCO, we have investigated several academic use case scenarios that would require such advanced orchestrators. We have selected a scenario that is capable to run parametric study based scientific workflow applications in a built-to-order virtual infrastructure. Afterwards, we implemented a prototype system to use it for evaluating the applicability of our findings. We showed that the prototype is capable to hide the infrastructure details and can manage scientific workloads automatically while it also increased the productivity of scientists with no computing infrastructure management knowledge.

The rest of the paper is organized as follows. First, in Section 2, we shortly overview the currently available orchestrator solutions. Afterwards, Section 3 provides a discussion on the architecture devised for our One Click Cloud Orchestrator. Later, we reveal a prototype implementation of the new orchestrator in Section 4. Then, the last section provides a conclusion with our closing thoughts and future plans to enhance our orchestrator.

2 Related Works

One type of orchestration tools covers the development and operations aspects. These tools, also called as configuration management tools, are aimed at au-

tomating development and system administration tasks such as delivery, testing and maintenance releases to improve reliability and security but the same mechanisms can perform orchestration activities such as creating, deploying and managing virtual machines. These are well known and are briefly listed here: Saltstack [15], Puppet [14], Chef [4], Docker [7], Juju [16] and Cloudify [5]. Beyond these general-purpose utilities there is another category of orchestration tools with specific aims, that we discuss in the following paragraphs.

Liu et al. [11] propose a data-centric approach (Data-centric Management Framework, DMF) to cloud orchestration where cloud resources are modeled as structured data that can be queried by a declarative language, and updated with well-defined transactional semantics. This data centric approach is further advanced by an additional Cloud Orchestration Policy Engine (COPE) in [12]. COPE takes policy specifications (of system wide constraints and goals) and cloud system states and then optimizes compute, storage and network resource allocations within the cloud such that provider operational objectives and customer SLAs can be better met.

Orchestrator [9] is aimed at sensor-rich mobile platforms where it enables multiple, context aware applications that simultaneously run and share highly scarce and dynamic resources. Applications submit high-level context specifications and comply with Orchestrator's resource allocation. Resource selection and binding is postponed until resources' availability is sufficiently explored. The major innovation of Orchestrator, the notion of active resource use orchestration, is explored in [10]. In this scheme resource needs are decoupled from the actual binding to physical resources and can be changed dynamically at runtime. Opposed to passive resource use orchestration, where the resource needs are programmed in the application, this approach provides adaptivity via demand based, selective use of alternative plans for application requests.

Merwe at al. define a Cloud Control Architecture for a ubiquitous cloud computing infrastructure [6]. The Cloud Control Architecture follows a layered design where orchestration is realized as a separate layer and interconnects the Service Abstraction (presents service logic to the users) and Intelligence (gathers information about the cloud infrastructure) and derives abstract knowledge. The Orchestration layer collects both the requests from Service Abstraction and actual data from Intelligence and makes decision about initial placements, resource allocation, resource adjustment and movement of resources.

Lorincz et al. present a very different way or resource orchestration in Pixie: resource tickets [13]. A ticket is an abstraction for a certain part (capacity) of a resource and all orchestration actions are mediated via the tickets. Tickets are generated by resource allocators and managed by resource brokers. A ticket provides information about the resource, the allocated capacity and the timeframe. Resources can be manipulated by operations on tickets such as join (increasing resource capacity), split (sharing), revoke or redeem (collecting specific tickets for a certain operation) just to mention a few. This approach also decouples actual resources from resource requests and gives a great flexibility in planning, advance requests and adaptation.

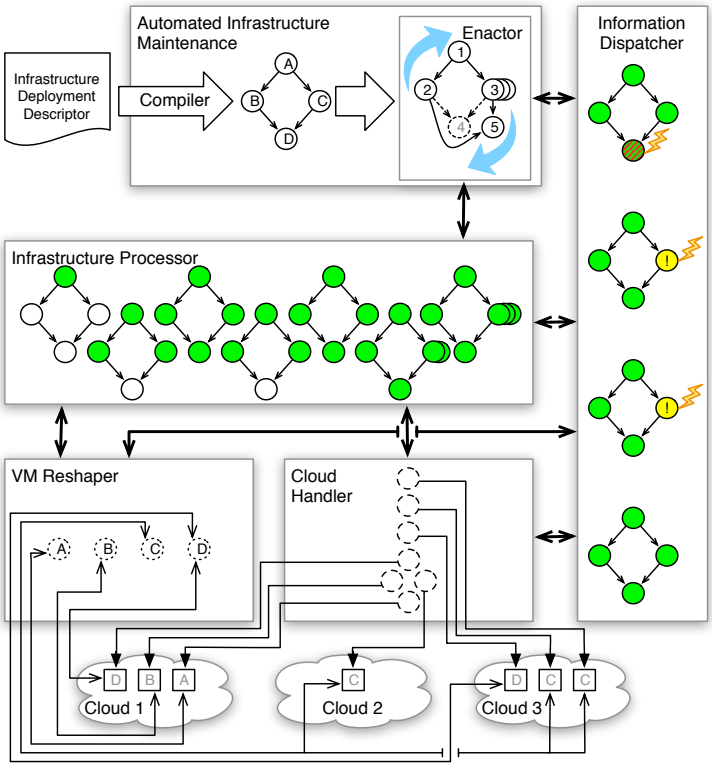

Fig. 1. Internal behavior of OCCO

3 Architecture

3.1 The view of an infrastructure maintainer

This sub-section reveals the internal components of our architecture and how these components interact to automatically operate a virtual infrastructure described by an infrastructure maintainer. In the scope of this paper, the term *infrastructure maintainer* refers to those users of OCCO who have the capabilities to describe a virtual infrastructure and its expected behavior. To understand the design considerations of OCCO and the required knowledge of infrastructure maintainers, Figure 1 shows the main components of our proposed orchestrator. These components are illustrated as boxes with gray boundaries in the figure. The behavior of each component is exemplified inside the box through the operation of a simple virtual infrastructure. In the following we give an overview of the components, then each component is going to be described in detail using the examples shown in the component's boxes.

OCCO has five major components: (*i*) Automated Infrastructure Maintenance – infrastructure descriptor processing and VM management initiator; (*ii*) In-

frastructure Processor – internal depiction of a virtual infrastructure (groups VMs with a shared aim); (*iii*) Cloud Handler – abstracts IaaS functionality (e.g., VM creation) for federated and interoperable use of clouds; (*iv*) VM Reshaper – ensures awaited functionalities for VMs; and (*v*) Information Dispatcher – decouples the information producer and consumer roles across the architecture. Except for Automated Infrastructure Maintenance, these components have internal interfaces only (e.g., not even offered for an infrastructure maintainer).

Automated Infrastructure Maintenance. This component is the only one that sees the operated infrastructure with all of its complexity. It basically allows two major operations: (*i*) submission of new virtual infrastructure requests and (*ii*) destruction of already existing virtual infrastructures.

For the submission interface OCCO expects an Infrastructure Deployment Descriptor as an input. This descriptor is defined by an infrastructure maintainer and contains vital information to construct and operate a virtual infrastructure. First of all, the descriptor contains a list of node types needed to build a virtual infrastructure (in Figure 1, types are shown as capital letters in the range of *A*–*D*). Then, it specifies the dependencies between these types of nodes (*directed edges* between nodes in the figure). These dependencies allow the Automated Infrastructure Maintenance component to determine which node types need to be instantiated first – in cases when there is a loop in the dependency graph, the infrastructure maintainer should specify node types that could be deployed earlier than others. Finally, the descriptor also includes rules for error resolution (e.g., what to do when nodes are failing, under- or over-provisioned).

After the submission interface receives the descriptor, it is immediately *compiled* into an internal representation (the figure shows this as a non-colored graph with annotated node types). In case of compilation failure immediate feedback is provided to the infrastructure maintainer allowing easy development and debugging of newly created or modified deployment descriptors. On the other hand, successful compilation leads to the enactment of the virtual infrastructure.

The *enactor* subcomponent is the fundamental component within the orchestrator. During infrastructure construction, the enactor pushes node requests to the Infrastructure Processor in the sequence determined by dependencies (the figure shows this sequence as numbers in the nodes within the enactor). After the sequence is pushed and the requested infrastructure is created, the enactor continuously monitors the state of the infrastructure to detect errors and resolve them with the rules in the descriptor. As an example, rules could define the necessary actions – like node re-instantiation, dependency re-evaluation – when a particular kind of node becomes inaccessible. Such error resilience is exemplified through the node type *D* (in the figure, step 4 is a faulty node and step 5 re-instantiates it). The rules also allow the scaling of the described virtual infrastructure. Scaling rules define the number of node instances depending on the virtual infrastructure's state. The instance number is expressed as a function of some property of a node type (e.g., the collective CPU load of all instances with the same functionality) or time (on workdays we need more resources than

on holidays). Unfortunately, simple rules could result in oscillations. The enactor eradicates this behavior through complex rules (e.g., ones that stop asking for more instances unless some time has already passed since the last instance number change). Scaling is exemplified in the figure with the node type C (in Figure 1, see the multi instance node configuration behind step 3).

The enactor maintains the virtual infrastructure completely autonomously unless a change is needed in the Infrastructure Deployment Descriptor. In such case, first, the infrastructure maintainer updates the descriptor, and then the Automated Infrastructure Maintenance component compiles the new internal representation, and finally the enactor switches to a transitional mode. In this mode the enactor evaluates the differences of the old and the new internal representation of the descriptor. If the evaluation finds new error resolution rules only then the enactor ensures the infrastructure's conformance with them (e.g., if a new scaling rule needs fewer instances for the same load then the excess instances are terminated via the Infrastructure Processor) and it returns to normal operation. If the evaluation finds new node types and dependencies also, then the currently operated virtual infrastructure is restructured according to the new deployment descriptor.

Finally, the destruction of a virtual infrastructure can also be ordered. During the destruction, the enactor pushes node destruction requests for all previously created nodes to the Infrastructure Processor. The order of node destruction is reversed compared to node creation so every node type can count on its dependencies during its existence.

Infrastructure Processor. OCCO creates an abstraction for virtual infrastructures with this component. As discussed before, the Infrastructure Processor receives node creation or destruction requests from the enactor. When the first creation request is received for a virtual infrastructure, this component prepares an *administrative group* for the future virtual infrastructure. Nodes of the virtual infrastructure can share information between each other through this administrative group (e.g., allowing newly created nodes to retrieve the dynamic properties – like IP addresses – of existing ones). Depending on the underlying systems utilized by the implementation these administrative groups can be mapped to lower level concepts (e.g., if Chef is used behind the VM Reshaper component, then administrative groups can be implemented through Chef's `environment`s).

Node creation requests are processed as follows. First, the processor ensures that the VM Reshaper knows about the node type that is going to be instantiated. In case Chef is behind the VM Reshaper, then the processor checks for the presence of the type's recipe. If the recipe is not present, then the processor pushes the recipe of the type to the reshaper. The pushed recipe could be retrieved either from another Chef server or from the Infrastructure Deployment Descriptor's extended node type definition. Once the reshaper knows the node type, the Infrastructure Processor sends a contextualized VM request to the Cloud Handler component. Within the contextualization information the processor places a reference to the previously created administrative group and the

expected node type of the future VM. Figure 1 exemplifies processed requests for creation with green filled circles. The example shows various stages of a virtual infrastructure's operation (from the initial phases on the left, to the final developments in the right side of the Infrastructure Processor's box). These stages show how an infrastructure is constructed and how it is adopted to errors and problematic situations identified by the enactor.

In contrast to node creation, *node destruction* requests are directly forwarded to the Cloud Handler component. If the last node is destructed in a virtual infrastructure then the Infrastructure Processor also destroys its administrative group automatically.

Cloud Handler. As its basic functionality, this component provides an abstraction over IaaS functionalities and allows the creation, monitoring and destruction of virtual machines. For these functionalities, it offers a plugin architecture that can be implemented with several IaaS interfaces (currently we aim at supporting at least OCCI and EC2 interfaces). These plugins are expected to serve all concurrently available requests as soon as they can manage. To increase the throughput and flexibility of the deployed virtual infrastructure, the Cloud Handler also offers VM scheduling across multiple clouds. If this functionality is used, cloud selection criteria can be either specified by the infrastructure maintainer – e.g., as a guideline – or by the user who initiated the virtual infrastructure. The Cloud Handler always expects some selection criteria for each VM (e.g., a static cloud mapping has to be specified in every deployment descriptor).

Our example in Figure 1 shows the order of the VM requests (shown as dashed circles) that arrive to the handler (the first request is at the bottom, the last is at the top, parallel requests are shown side by side). Cloud to VM request association is shown with arrows between the request and the cloud. At the end of arrows, little squares represent the actual VMs created in the clouds. Each VM shows its contextualized node type with gray letters (A–D).

VM Reshaper. This component manages the deployed software and its configuration on the node level. This functionality is well developed and even commercial tools are available to the public. Our VM Reshaper component therefore offers interfaces to these widely available tools – e.g., [4, 5, 7, 14, 15]. These software tools use their proprietary node type definitions (e.g., so called `recipe`s in Chef and `manifest`s in Puppet). For node types already described, the VM Reshaper allows the reuse of these proprietary definitions stored at even external – but accessible – locations (thus, regular node type definitions are just references to these proprietary definitions). On the other hand, new node types can be defined in the infrastructure deployment descriptor in the extended node type definition. The form of these definitions allows the Infrastructure Processor to select a VM Reshaper with matching node management tools behind (e.g., in case a recipe is given as an extended node type definition then Chef will be the tool used). It is expected that advanced infrastructure maintainers could create such node type definitions for custom applications.

Returning to our example in Figure 1, node type definitions are presented with dotted circles within the VM Reshaper component. With arrows between the VMs and type definitions, the figure also reveals that the VMs contact the VM Reshaper to retrieve and apply the node type definitions. These activities ensure the presence and correct configuration of the software components necessary for each VM to fulfill its role within the OCCO operated virtual infrastructure.

Information Dispatcher. In order to make accurate decisions on the state of the ordered virtual infrastructure, our proposed architecture has a common interface to reach the diverse information sources from which the state can be composed. In order to reduce redundancy and structural bottlenecks, requests to our dispatcher component are practically directly forwarded to relevant information sources. The minimal processing done inside the dispatcher is limited to two activities: (*i*) request transformation and (*ii*) information aggregation. For the first activity, the dispatcher transforms the – sometimes abstract or conceptual – requests to the actual information pieces accessible from the various components and underlying clouds of the OCCO (e.g., request for node D load can be translated to the CPU utilization of the VM hosted in Cloud 1 or Cloud 3 in Figure 1). The second activity happens when the dispatcher receives requests to information that is available only as a composite. In such cases, the dispatcher forwards the request to all relevant OCCO components and if necessary to the virtual infrastructure. Upon receiving their response, the dispatcher calculates an aggregated value of the responses and presents this as a response to the original request. For example, using our running example of Figure 1, a request for node C load will be computed as an average of the CPU utilization of all VMs hosting node type C in Cloud 2 and 3. In OCCO, generic transformation and aggregation rules can be specified by the deployer of the Information Dispatcher while specific rules for the particular kind of virtual infrastructure are given in the Infrastructure Deployment Descriptor by the infrastructure maintainer.

In Figure 1, within the box of Information Dispatcher, we show how querying this component can help with understanding the state of the operating virtual infrastructure by showing three scenarios. We expect that the enactor regularly queries the dispatcher. In the top graph within the dispatcher's box, we see that a query to the dispatcher is sent to check the availability of each node in the virtual infrastructure. This query is then forwarded to all participating virtual machines. Unfortunately, in this scenario, the dispatcher is not receiving node D's response, thus it is reported unavailable (represented with red and green stripes within node D's circle). As this would render the virtual infrastructure unusable, the enactor will immediately request a new node for type D through the Infrastructure Processor. Similarly, in the middle two graphs we see requests for load of node type C. When a single VM performs this type, the dispatcher transforms this request to CPU load request on that VM. If the load is too high (shown with an exclamation mark in the respective node of the figure) and it is expected that a single VM cannot handle the anticipated load, the enactor will increase the node count for type C. This will make later requests

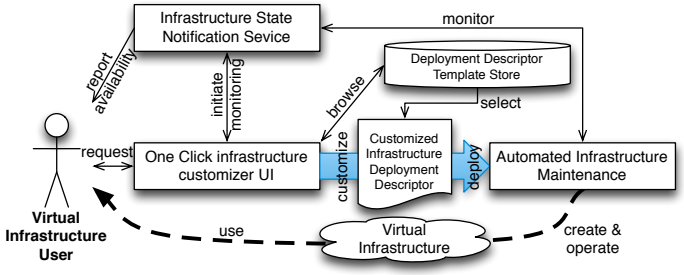

Fig. 2. User's relation to OCCO

to the dispatcher as composite. In the third graph it is also shown that even a composite request reports unmanageably high loads and thus the enactor will again increase the node count of type *C*.

3.2 The view of a virtual infrastructure user

After infrastructure maintainers complete an infrastructure deployment descriptor, they can publish it in OCCO's template store. After publication, the described kinds of infrastructures are going to be available for regular cloud users with the need of deploying complex but easily maintainable virtual infrastructures. Figure 2 reveals the interfaces and the use case OCCO offers for these regular users.

The figure shows that regular users are expected to interface with OCCO through a graphical user interface that allows browsing and customizing deployment descriptors. This interface supports the user in the selection of the appropriate kind of virtual infrastructure based on textual descriptions accompanied with templates in the store. Once a template is selected, users receive a list of customization options that were added as hints for the GUI in the deployment descriptor by the infrastructure maintainer. These hints could range from the supported IaaS providers to the possibility to specify an initial size of the custom infrastructure, but hints could also include pricing and cost allowances.

When the customization is done, users can request the deployment of their virtual infrastructure via the GUI. After the request is made, the monitoring of the requested infrastructure is initiated at the notification service. This service has two purposes: (*i*) let the user know when the requested infrastructure is completely available and (*ii*) monitor the changes – introduced by the infrastructure maintainer – of the deployment template and propagate them to the Automated Infrastructure Maintenance component. The first purpose allows users to immediately use the prepared infrastructure when it is ready. The notification service can trigger automated actions (so the user's application can react to infrastructure availability immediately) or it can also send emails to interested parties. The second purpose ensures that infrastructures are updated transparently to their latest, most secure and stable versions the particular maintainer can produce.

4 Evaluation

In order to test the concept of the new orchestrator, to perform analysis of the internal operation and to provide a demonstration platform, we have implemented a prototype of OCCO. It is currently limited to a single infrastructure deployment descriptor template, and it is publicly available[1] for users.

The infrastructure template aims to provide a distributed computing infrastructure (DCI) with a science gateway attached as a front-end. The DCI is implemented by a BOINC [1] based Desktop Grid project with a molecular docking simulator called autodock. As an extra functionality, the BOINC project is associated with a public IP address, therefore the user can attach his/her own BOINC client to the server. Using automatically deployed and configured BOINC clients in virtual machines, computational resources are automatically attached to this BOINC project. Our descriptor template allows the customization of the number of computational resources. Computing jobs arrive to the BOINC project as work units with the help of the WS-PGRADE/gUSE science gateway (also automatically deployed as a node of the virtual infrastructure). Overall, the prototype shows how a complete gateway plus DCI with resources can be deployed by OCCO and how the components attach to each other. Detailed description of a similar infrastructure is shown at `http://doc.desktopgrid.hu/doku.php?id=scenario:unidg` with a different application.

In the prototype's welcome- and request submission page (see Figure 3) the user is not only requested to fill in the list of customization options, but he/she must also provide some details about him/her for identification and for justification. After a request is submitted, the prototype first asks for approval by the SZTAKI cloud administrators then initiates the infrastructure's creation with the Automated Infrastructure Maintenance component. Once the infrastructure is created the notification service generates an email with all the authentication and access details to the new infrastructure (e.g., url of the science gateway and of the BOINC project plus user/password for login). With these details, users just need to login to the gateway, submit a prepared autodock workflow with their inputs and inspect the operation (i.e. how the jobs are flowing through the infrastructure and processed by the BOINC clients). To prevent SZTAKI's IaaS from overload the OCCO created virtual infrastructures have a limited lifetime. Our notification service sends an email to the infrastrucure's user before the shutdown procedure is initiated.

As the aim of the prototype implementation is to demonstrate and test the OCCO concept, we implemented the most crucial components with basic functionality only. The current Automated Infrastructure Maintenance component only provides virtual infrastructure creation and termination facilities. The simple VM Reshaper can handle prepared VM images with pre-installed applications and expects these applications to be configurable through IaaS contextualization methods. Our Cloud Handler is already capable to support multiple IaaS clouds as long as they offer EC2 interfaces.

[1] http://desktopgrid.hu/oc-public

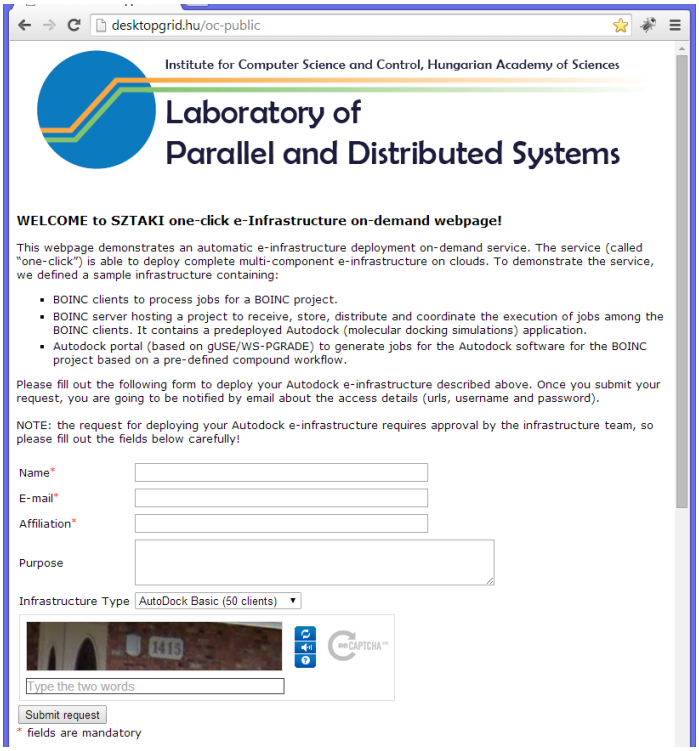

Fig. 3. Request submission page of the OCCO prototype

5 Conclusions and Future Work

Through the analysis of this paper we have found several issues with currently existing scientific and commercial cloud orchestrator offerings. Namely, recent solutions lack functionality with regards to function oriented VM creation, error resilience across VMs or even clouds and high level user orientation with such advanced but hidden features like automatic scaling of entire virtual infrastructures. To remedy these issues, we have proposed the OCCO architecture that builds on the strengths of past solutions (e.g. Chef). We have shown the behavior of OCCO from the point of view of both a regular cloud user and also a virtual infrastructure template maintainer. In the discussions we have shown the way maintainers can describe virtual infrastructures. Finally, we have presented our initial prototype implementation of the architecture which already shows the high potential of the architecture and available as a public service for the scientific community with access to the SZTAKI cloud infrastructure.

Other than implementing a more complete and openly downloadable version of OCCO, we also identified several future research areas. First, error resilience and scaling is only based on simple reactive rules, in the future we plan to incorporate proactive approaches combined with learning techniques. Next, decisions

on cloud use are made on a per VM request basis. However, in some cases (e.g. expected significant network activities between particular nodes), it would be beneficial to make decisions on which cloud to use considering more information about the operating virtual infrastructure. Finally, we are planning to increase the reliability and failure handling of the internal components by introducing atomic operations and cross-component transactions.

References

[1] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th International Workshop on Grid Computing (GRID 2004)*, pages 4–10, Pittsburgh, PA, USA, November 2004. IEEE Computer Society.

[2] 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.

[3] M. Caballer, I. Blanquer, G. Molto, and C. de Alfonso. Dynamic management of virtual infrastructures. *Journal of Grid Computing*, pages 1–18, DOI: 10.1007/s10723-014-9296-5, 2014.

[4] Chef. `http://www.getchef.com/`.

[5] Cloudify. `http://www.cloudifysource.org/`.

[6] J. Van der Merwe, K. Ramakrishnan, M. Fairchild, A. Flavel, J. Houle, H. A. Lagar-Cavilla, and J. Mulligan. Towards a ubiquitous cloud computing infrastructure. In *Proceedings of the IEEE Workshop on Local and Metropolitan Area Networks (LANMAN)*, 2010.

[7] Docker. `https://www.docker.io/`.

[8] R. Dukaric and M. B. Juric. Towards a unified taxonomy and architecture of cloud frameworks. *Future Generation Comp. Sys.*, 29(5):1196–1210, 2013.

[9] S. Kang et al. Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments. In *IEEE International Conference on Pervasive Computing and Communications*, 2010.

[10] Y. Lee, C. Min, Y. Ju, S. Kang, Y. Rhee, and J. Song. An active resource orchestration framework for pan-scale sensor-rich environments. *IEEE Transactions on Mobile Computing*, 13(3), 2014.

[11] C. Liu et al. Cloud resource orchestration: A data-centric approach. In *Proceedings of the biennial Conference on Innovative Data Systems Research (CIDR)*, 2011.

[12] C. Liu, B. Thau Loo, and Y. Mao. Declarative automated cloud resource orchestration. In *2nd ACM Symposium on Cloud Computing*, 2011.

[13] K. Lorincz, B. r. Chen, J. Waterman, G. Werner-Allen, and M. Welsh. Resource aware programming in the pixie os. In *SenSys*, pages 211–224, 2008.

[14] Puppet. `http://puppetlabs.com/`.

[15] SaltStack. `http://www.saltstack.com/`.

[16] Ubuntu. Juju. `http://juju.ubuntu.com`.

[17] J. Wettinger, M. Behrendt, T. Binz, U. Breitenbücher, G. Breiter, F. Leymann, S. Moser, I. Schwertle, and T. Spatzier. Integrating configuration management with model-driven cloud management based on tosca. In *3rd International Conference on Cloud Computing and Service Science*, pages 437–446, 2013.