Accepted Manuscript

MiCADO–Microservice-based Cloud Application-level Dynamic Orchestrator

Tamas Kiss, Peter Kacsuk, Jozsef Kovacs, Botond Rakoczi, Akos Hajnal, Attila Farkas, Gregoire Gesmier, Gabor Terstyanszky

PII: S0167-739X(17)31050-6
DOI: https://doi.org/10.1016/j.future.2017.09.050
Reference: FUTURE 3706

To appear in: *Future Generation Computer Systems*

Received date : 20 May 2017
Revised date : 17 July 2017
Accepted date : 20 September 2017

Please cite this article as: T. Kiss, P. Kacsuk, J. Kovacs, B. Rakoczi, A. Hajnal, A. Farkas, G. Gesmier, G. Terstyanszky, MiCADO–Microservice-based Cloud Application-level Dynamic Orchestrator, *Future Generation Computer Systems* (2017), https://doi.org/10.1016/j.future.2017.09.050

MiCADO –Microservice-based Cloud Application-level Dynamic Orchestrator

Tamas Kiss[a], Peter Kacsuk[a,b], Jozsef Kovacs[b], Botond Rakoczi[a], Akos Hajnal[b], Attila Farkas[b], Gregoire Gesmier[a], Gabor Terstyanszky[a]
[a] Centre for Parallel Computing, University of Westminster, London United Kingdom
[b] MTA-SZTAKI, Budapest, Hungary

Corresponding author:
Tamas Kiss
University of Westminster
115 New Cavendish Street
W1W 6UW London, United Kingdom
email: kisst@wmin.ac.uk
phone: +44-20-79115000

Abstract: Various scientific and commercial applications require automated scalability and orchestration on cloud computing resources. However, extending applications with such automated scalability on an individual basis is not feasible. This paper investigates how such automated orchestration can be added to cloud applications without major reengineering of the application code. We suggest a generic architecture for an application level cloud orchestration framework, called MiCADO that supports various application scenarios on multiple heterogeneous federated clouds. Besides the generic architecture description, the paper also presents the first MiCADO reference implementation, and explains how the scalability of the Data Avenue service that is applied for data transfer in WS-PGRADE/gUSE based science gateways, can be improved. Performance evaluation of the implemented scalability based on up and downscaling experiments is presented.

Keywords: cloud orchestration, science gateway, automated scalability, Data Avenue

1. Introduction

Many scientific and commercial applications require access to computation, data or network resources based on dynamically changing requirements. Applications running on distributed computing infrastructures, such as grids or clouds, typically fall into this category. End-users can access these applications via desktop or web-based high-level user interfaces, such as science/business gateways. When executing applications or accessing services via high-level user environments, users and providers both require these applications or services to dynamically adjust to fluctuations in demand and serve end-users at required quality of service (performance, reliability, security, etc.) and at optimized cost.

Cloud computing has the potential to fulfil these requirements. In order to support the development of a large number of applications that are capable utilising the dynamic and scalable nature of IaaS (Infrastructure as a Service) clouds, advanced tools are required that support application developers. Custom developing cloud awareness to every single application is not a viable solution and would require significant training and development efforts that especially smaller organisations, such as SMEs (Small and Medium-sized Enterprises) cannot afford.

One of well-documented benefits of cloud computing [1] is its ability to supply a variable amount of resources (computational power, storage, network capacity), which can scale dynamically up and down, forming the supply side of Figure 1. On the demand side, we can see applications that are likely to be formed of one or more services. Services can be either in-house developed or provided by external suppliers or open-source communities. Services could also be shared between applications.

Services consume *baseline* and *variable resources.* B*aseline resources* can be defined as resources consumed by the component in idle state. *Variable resources* are consumed when

the component performs its duties. This can include heavy computation or storage utilisation based on application requirements, which can vary significantly based on the nature of the application. Overall resource demand of an application is the sum of baseline and variable resources.

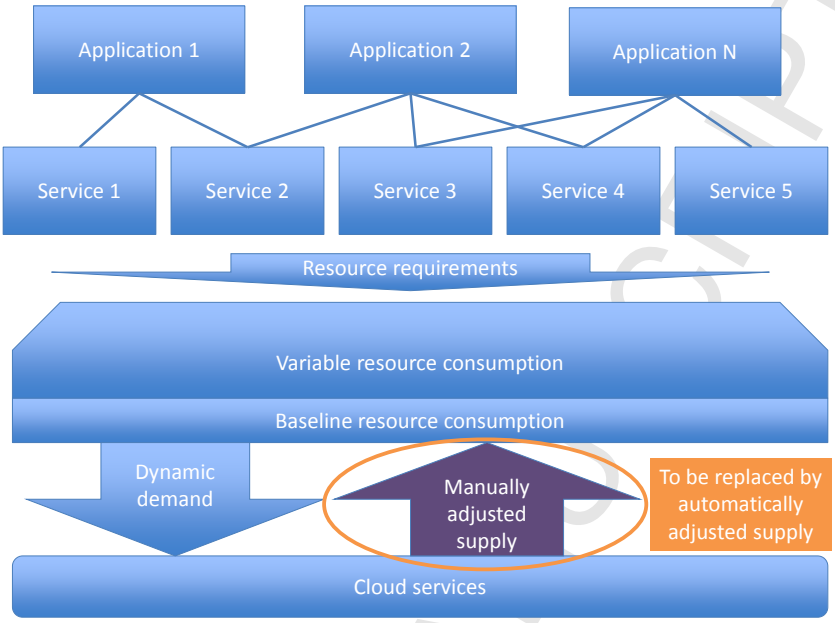

Fig. 1 Resource demand and supply of cloud applications

On the resource provision side cloud computing already offers rapid elasticity and dynamic scalability. IaaS clouds can scale up or down on demand. However, the dynamic and intelligent utilisation of such scalability from the perspective of cloud applications is not trivial. Many legacy applications have been migrated to cloud infrastructures that only consume and run on a predefined static set of resources. More cloud-aware applications have also been developed that offer dynamic scalability based on user demands or application characteristics. However, these applications have typically been custom developed requiring significant effort and low level cloud computing expertise to implement. On the other hand, typical application patterns can be relatively easily identified that support a large number of similar applications and can use rather similar underlying mechanisms from dynamic clouds.

As shown in Figure 1, cloud service providers are indifferent to the resource needs of applications since they have no means of predicting their capacity demand. In practice, the operator of an application requests cloud resources based on predictive estimates (typically, worst-case estimates), but after being commissioned, these resources remain static without operator intervention. If demand exceeds supply at a given point of time, applications do not function within their required parameters. If supply exceeds demand, resources are wasted, which generally has a cost impact.

This work investigates possibilities of replacing manually adjusted supply of cloud services with an automatically adjusted supply, as also illustrated in Figure 1. The aim is to create a framework, where automatically adjusted cloud service supply can be arranged, based on application demands. Such framework would allow cloud application developers to build cost and performance optimization mechanisms into their application code through a high-level API (Application Programming Interface). The suggested solution is based on microservices [8] and their dynamic orchestration in a cloud computing environment.

The rest of this paper defines a generic microservices-based architecture for application-level cloud orchestration, called MiCADO (Microservices-based Cloud Application level Dynamic

Orchestrator), and describes its first reference implementation utilizing container-based open source cloud technologies. In order to illustrate how to handle such variable resource demand, the example of a science gateway supported by the first version of the MiCADO architecture is introduced. The scalability problem is investigated and solved in connection with the WS-PGRADE/gUSE [2] gateway framework, and its particular service that needs such scalability is data staging. This example is generic enough to appear in many different science gateways, and also the MiCADO approach created to solve this problem is sufficiently generic to support a large class of application types. The paper significantly extends the MiCADO concept first described in [17], and presents its first implementation and benchmarks.

2. Related Work

The problem of application-level orchestration has been recognized and a number of solutions have been designed and implemented. A short overview is provided below highlighting the current limitations of these solutions.

Marpaung et al. [3] discuss Altocumulus, AppScale, Cloudify and mOSAIC. Altocumulus focuses on deploying web applications to a variety of public clouds, which limits its usability in private or hybrid clouds. It does not provide monitoring or dynamic changes to services. AppScale is an open-source product that supports execution of Google Application Engine applications and therefore restricted to this particular technology. mOSAIC provides a set of APIs to application developers to tackle cloud deployment issues. The limitation of mOSAIC is the implementation of these APIs as the application developer needs to integrate these to application components.

Cloudify [4] combines a TOSCA (Topology and Orchestration Specification for Cloud Applications) editor with deployment and orchestrator. It provides access to multiple clouds and a complete framework to describe microservices and execute them either in Docker containers or on cloud metal. Cloudify also provides dynamic service upscaling and downscaling based on microservice dependent parameters, for example number of transactions, number of threads, etc. Cloudify does not provide a container portability framework, nor do its metrics span dockerised microservices and the cloud metal executing them.

Pham, Tchana et al. discuss the problem of distributed applications [5]. They focus on application orchestration that can span several different clouds. They recognize the need to deliver microservice-specific parameters, for example port numbers and IP addresses, to other microservices, and provide a description language framework to do this. However, this solution does not support service discovery tools and dynamic relocation of microservices.

Amazon CloudFormation [6] provides to system administration developers an easier way for the collection, creation and management of related AWS (Amazon Web Services) resources through templates. These templates describe the AWS resources and associated dependencies. After the deployment of AWS resources, CloudFormation ensures the start of services in the correct order. As limitation, CloudFormation is specific to AWS and does not support run-time orchestration.

OpenTosca [7] provides an open source ecosystem for the Topology and Orchestration Specification for Cloud Applications (TOSCA) developed by Stuttgart University. OpenTosca is divided into three parts: a TOSCA runtime environment (OpenTosca container), a graphical modelling TOSCA tool (Winery) and a self-service portal for the application available in the container (Vinothek). Although OpenTosca is a generic framework, it does not support run-time orchestration.

Although effective in particular narrow circumstances, none of these systems provide fully-automated, cloud-agnostic solution for a wide range of applications and for wide variety of clouds in a way that MiCADO aims.

3. MiCADO concept and architecture

The generic life-cycle of cloud-aware applications based on the MiCADO concept is illustrated on Figure 2. The aim is to provide dynamic and automated resource supply for applications, as it was described in Section 1. This process can be divided into two major phases: optimised deployment (1) and run-time orchestration (2).

In phase 1, application developers/operators need to provide a high-level description of their applications. This description, besides application topology, also includes various QoS (Quality of Service) parameters such as cost and performance requirements, and desired security policies. This description is passed on to the *Coordination/Orchestration* component of MiCADO. This component collaborates with the *Security facilitator* in order to translate user defined security policies into specific security solutions, and also with the *Optimisation decision maker* in order to translate performance and cost related parameters into actual deployment values. Following this, the *Coordination/Orchestration* component passes on the deployment instructions to MiCADO's *Deployment executor* that deploys the services required to run the application on the targeted cloud infrastructure.

Once the application is deployed in the cloud, run-time orchestration (phase 2) starts. MiCADO continuously collects various metrics from the running application and passes it on to the *Coordination/orchestration* component. This data is analysed from performance/cost aspects by the *Optimisation decision maker* and from security enforcement point of view by the *Security facilitator*. If adjustment is required, the *Deployment executor* is called to scale up or down the infrastructure, and to execute the required changes.

Users also have the possibility to adjust any requirement, either security or performance/cost related, during runtime. If the user provides a modified description with updated QoS parameters, then it is passed on to the *Coordination/Orchestration* component that analyses the received data and instructs the *Deployment executor* to modify the infrastructure.

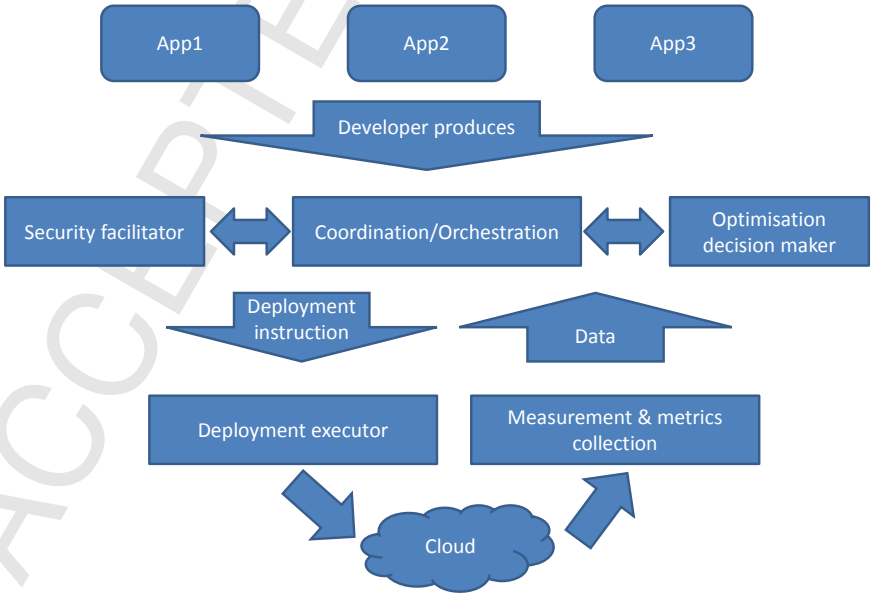

Fig. 2 Generic life-cycle of cloud-aware applications orchestrated by MiCADO

Based on the above described life-cycle model, a layered MiCADO architecture is defined below. This infrastructure framework is generic in the sense that we do not specify the actual

components required and do not discuss their specific implementation here. The identified layers and their desired functionality can be implemented in various ways using different technologies and services that may already exist. Our aim with the generic architecture definition is to identify a modular and pluggable framework where different functionalities can be delivered by different components on-demand, and where these components can be easily substituted. The resulting solution will be technology neutral that will not be depending on one particular component implementation.

The layers of such framework that support the dynamic application level orchestration of cloud applications are illustrated in Figure 3. The suggested generic framework is based on the concept of microservices, as defined for example by Balalaie [8]. Cloud computing is a natural platform for microservices that provide decoupling of independent components from a monolithic application. Cloud enables execution and resource allocation of these independent components based on their specific needs. One microservice might require a lot of storage while another could be CPU intensive. Cloud execution offers the possibility to optimise resource allocation and thus resource cost dynamically. The alternative would be to allocate a monolithic infrastructure, the size of which large enough to be sufficient for worst-case requirements scenario. However, most of the time, the worst case scenario is not present and allocated resources of the monolithic infrastructures are wasted.

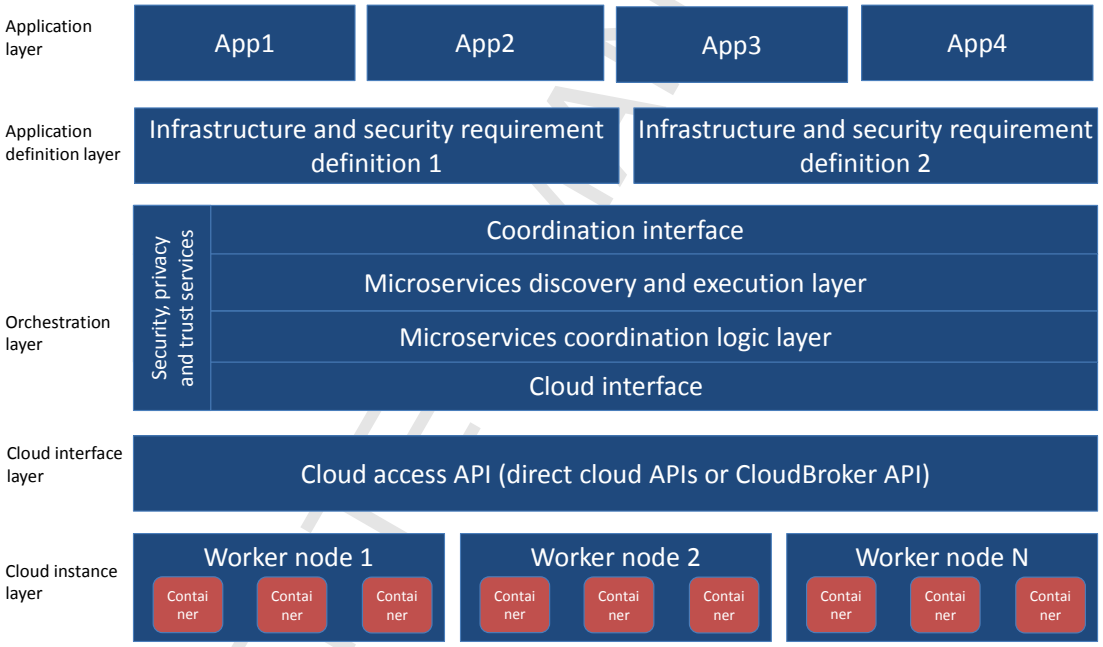

Fig. 3 MiCADO generic architecture framework

The layers of the MiCADO generic architecture (from top to bottom), based on the above described microservices-based concept are as follows:

1. **Application layer**. It contains actual application code and data to make an application definition (layer 2) functioning in such a way that a desired functionality is reached. For example, this layer could populate database with initial data, and configure HTTP server with look and feel and application logic.
2. **Application definition layer**. This layer forms the basis to define a functional architecture of the application using application templates. At this level software components and their requirements (both infrastructure and QoS specifications) as well as their interconnectivity are defined using application descriptions uploaded to a public repository. As the infrastructure is agnostic to the actual application using it, the

application template can be shared with any application that requires such an environment.

3. Orchestration layer

This layer is divided into four horizontal and one vertical sub-layers. The horizontal sub-layers are:

a. **Coordination interface API.** It provides access to orchestration control and decouples the orchestration layer from the application definition. This set of APIs enables application developers to utilise the dynamic orchestration capabilities of the underlying layer and supports the convenient development of dynamically and automatically scalable cloud-based applications by embedding these API calls into application code.

b. **Microservices discovery and execution layer**. This layer manages the execution of microservices and keeps track of services running. Execution management combines both start-up and shut down of microservices. Service management gathers information about currently running services, such as service name, IP address and port where the service is reachable and optional service tags to help in service coordination.

c. **Microservices coordination logic layer**. With large infrastructures and to reap the benefits from cloud-based execution, it becomes necessary to understand how the current execution environment is performing. Information needs to be gathered and processed. If bottlenecks are detected or the currently running infrastructure appears underutilised, it may be necessary to either launch or shut down cloud instances, and possibly move microservices from one physical worker node to another.

d. **Cloud interface API** is to abstract cloud access from layers above. Cloud access APIs can be complex interfaces, as they typically cater for a large number of services provided by the cloud provider. On the other hand, the microservices execution and coordination logic layers (see 3b and 3c) only need to shut down and start instances. Abstracting this to a cloud interface API simplifies implementation of aforementioned layers, and if new Cloud access APIs are implemented, only this layer needs to change.

e. Security, privacy and trust services: The orchestration layer also includes a vertical sub-layer that deals with security, privacy and trust related services for advanced security policy management. These services span multiple levels of the orchestration layer, as it is illustrated on Figure 3. The main aim is to shield application developers from detailed security management. To achieve this, the security, privacy and trust services of the orchestration layer take the general security policies defined at the Application definition layer, as well as security credentials for the application domain. These inputs will then be used by the special purpose security policy enforcement services to enforce the security policies at orchestration level.

4. **Cloud interface layer**. This layer provides means to launch and shut down cloud instances. There can be one or more cloud interfaces to support multiple clouds. Besides directly accessing cloud APIs, generic cloud access services, such as the CloudBroker platform [18] can also be used at this layer to support accessing multiple, heterogeneous and distributed clouds via a uniform access layer.
5. **Cloud instance layer**. It contains cloud instances provided by IaaS cloud providers. These instances can run various containers that execute actual microservices. This layer typically represents state-of-the-art of cloud technology, as provided by various public or private cloud providers.

The full MiCADO framework is currently being investigated and implemented in the COLA (Cloud Orchestration at the Level of Application) project funded by the European Commission [22]. This paper focuses on the horizontal sub-layers of the orchestration layer, and describes the first experiences when implementing these layers of MiCADO.

4. MiCADO prototype implementation

The MiCADO prototype implementation targets to solve a very wide class of application types where a certain service is used by a variable number of users for a variable amount of data. As a result the load of the service rapidly changes in time. If the allocated resources do not match the load requirements, either the service's response time will be too slow (too few cloud resources are provided for the service), or the cost to pay for the service will be much more than it is necessary (too many cloud resources are provided for the service). Web applications typically belong to this class of applications since the number of users visiting a certain web page might be unpredictable and changes rapidly.

Such a service where the usage dynamically changes is the Data Avenue data staging service [19]. Data Avenue enables transferring of files or even directories of files between different data storages having various storage access protocols. The major storage access protocols Data Avenue supports are HTTP, HTTPS, SFTP (SSH File Transfer Protocol), GSIFTP (FTP enhanced to use GSI – Grid Security Infrastructure), SRM (Storage resource Management), iRODS (Integrated Rule-Oriented Data System), and S3 (Amazon simple Storage Service). Data Avenue also enables to upload and download files and directories between the user's computer and a data storage (using the storage access protocols mentioned above). Currently Data Avenue is a public service that is accessible via a web page at https://data-avenue.eu/en_GB/. Data Avenue is also used in gUSE/WS-PGRADE portals either as a portlet or inside WS-PGRADE workflows to enable file transfer based communication between workflow nodes. Data Avenue client is part of the standard WS-PGRADE distribution and used by more than 30 large projects in Europe. For example, the Vialactea project, that investigates the star formation processes of the Milky Way, deployed its own Data Avenue server to support its WS-PGRADE gateway in data transfer [27]. The use of the Data Avenue service makes WS-PGRADE workflows highly portable across different DCIs (Distributed Computing Infrastructures) since Data Avenue based data staging is independent from where the data files are stored and where the workflow nodes are executed.

The original version of the Data Avenue service did not use cloud technology and it worked fine as long as only a few users used it and only a small number of file transfers were initiated via the service. However, if a large number of users want to use this service in a large number of science gateways, it will be a bottleneck. To avoid this problem we developed a cloud version of the service based on the MiCADO concept. This version of the service is highly scalable because it can be automatically replicated whenever the usage of the service exceeds a pre-defined threshold. Once the load is reduced below this threshold, under-loaded replicas can be removed from the cloud. In this section we show how this scalable version of the Data Avenue service has been created and how this solution fits to the concept of MiCADO.

The architecture of the scalable Data Avenue service is a three-layered architecture as shown in Figure 4. The bottom layer contains the application itself that runs in the worker nodes of a Docker Swarm cluster [14]. These are denoted as Data nodes in Figure 4 emphasizing that this MiCADO architecture can support not only the Data Avenue application but any other application having similar characteristics. Applications to be deployed in a scalable way in the MiCADO architecture should be placed in Docker containers inside these Data nodes.

The Central layer of the MiCADO prototype architecture contains those services that make the Data node application scalable. These services include:

1. Swarm master node
2. Consul [20]
3. Prometheus [21] extended with an Alert manager and Alert executor service

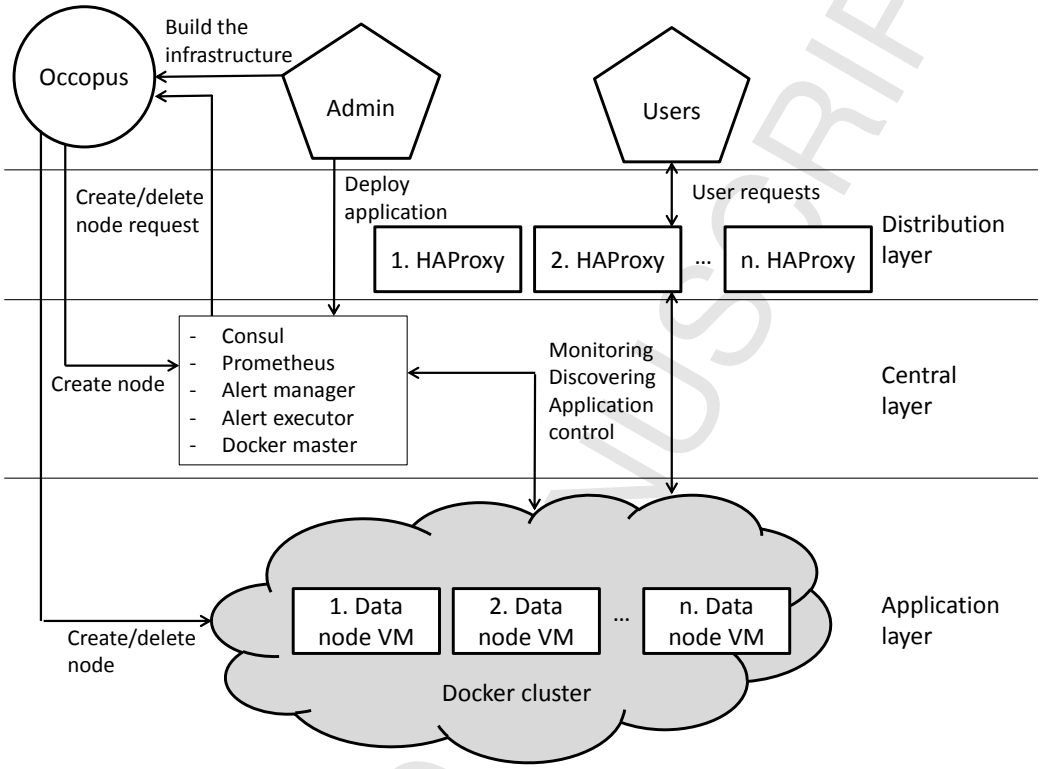

Fig4 MiCADO prototype architecture

An important design aspect of MiCADO was to use only open source tools that can be replaced later if needed by similar type of services with little effort. Therefore, all the services used in this prototype MiCADO architecture are based on open source solutions. Docker Swarm cluster is a widely used docker-based infrastructure-like service where the Docker master node is responsible for the dynamic creation and management of Docker containers at the worker nodes of the Swarm cluster. The registration of the workers into the cluster is done via the Consul discovery service that is another well-known and widely used open source service to organize Docker containers into a cluster. Every time a new node is created, first it is unknown to the other nodes. It is the Consul service that provides information about the active nodes to the central services of MiCADO.

In order to take the decision when the number of workers in the Swarm cluster should be increased or decreased, we need a monitoring tool to observe the load of the worker nodes. Such mature monitoring tool is Prometheus that is responsible for collecting information about the load of the worker nodes. The collected data will be used to create alerts. These alerts are events which if triggered then they are sent out to other services of MiCADO. The alerts are generated by the Alert manager service. Prometheus provides only a framework to create the Alert manager service therefore it was our task to develop the alert conditions and code. Similarly the Alert executor service was developed by us based on the Prometheus framework concept. The Alert executor instructs the Occopus cloud orchestrator [25] to deploy new Swarm worker nodes or remove existing ones depending on the information provided by the Alert manager service.

Occopus is a cloud orchestrator service that can deploy complex virtual infrastructures in many different types of clouds including single and heterogeneous multi-cloud systems. After deploying the services it can also manage the component services of the deployed infrastructure, regularly checking their health conditions (responsiveness for various types of messages). In MiCADO, Occopus has two major roles. First, Occopus is used to set up the three-layer MiCADO infrastructure in a selected cloud. Once the infrastructure is created, the next task of Occopus is to process the requests of the Alert executor and based on those requests deploy new Swarm worker nodes or remove existing ones.

The third layer of the MiCADO prototype, the Distribution layer realizes load balancing among the Data node services. It contains several HAProxy [24] services. This farm of nodes shares the incoming traffic and routes it to the Data nodes in a balanced way. This helps managing traffic and spreading incoming traffic spikes between the Data nodes. In this way MiCADO ensures that the Data nodes are well balanced and the load is equally shared by every worker of the Swarm cluster. The current solution uses round robin technique however least connections or predictive node techniques can also be used. This layer will remain transparent from the users' point of view and offers additional features such as caching, http compression and SSL (Secure Socket Layer) offloading. We selected HAProxy, the high performance TCP/HTTP load balancer because of its reliability and well documented features. Please notice that the Distribution layer distributes user data transfer requests only and not the actual data to be transferred. As a result the traffic and processing load of the HAProxy services is much less than the load of the Data nodes. Nevertheless, if there are a very large number of users, a single HAProxy service can be a bottleneck and hence the same kind of scalability approach is applied for the HAProxy services as for the Data node services. Prometheus collects load information on HAProxy services too and the Alert Executor can instruct Occopus to deploy a new HAProxy service or remove an existing one. The new ones are registered into MiCADO via the Consul service as in case of the Swarm worker nodes.

Comparing Figure 3 and 4 we can easily identify how the MiCADO prototype in Figure 4 implements the MiCADO concept and architecture shown in Figure 3. The **Microservices discovery and execution layer** that manages the execution of microservices and keeps track of services running, is implemented by Occopus. It has these tasks combined with the start-up and shut down of microservices (Data node and HAProxy). It gathers information about currently running services, such as service name, IP address and port where the service is reachable, and optional service tags to help in service coordination, as described in Section 3.

The **Microservices coordination logic layer** is realised by the Central and Distribution layers of the MiCADO prototype architecture. To understand how the current execution environment is performing the Central layer uses Prometheus to collect information about the various services. This information is processed by the Alert manager. If bottlenecks are detected or the currently running infrastructure appears underutilised, Alert Executor instructs Occopus to launch or shut down cloud instances. Finally, it is the task of the Swarm master to move microservices from one physical worker node to another.

The **Cloud interface API** of MiCADO is to abstract cloud access from layers above. This is implemented by the different cloud handler plugins of Occopus. Currently the following cloud interfaces are supported by Occopus: CloudSigma, Amazon EC2 (Elastic Compute Cloud), OpenStack NOVA, OpenNebula EC2, OCCI (Open Cloud Computing Interface), CloudBroker and Microsoft Azure. This has the advantage that Occopus can deploy the newly required Data nodes in a cloud different from the one where the MiCADO prototype runs. In this way the MiCADO prototype can exploit even heterogeneous multi-cloud

systems. If the originally used cloud system is overloaded new Data nodes can be deployed either on other clouds within a cloud alliance like the EGI FedCloud (European Grid Initiative Federated Cloud) [26] system or on commercial cloud systems.

The **Coordination interface API** of MiCADO provides access to orchestration control and decouples the orchestration layer from the application definition. This API is currently the Occopus infrastructure descriptor language. In the near future the COLA project considers TOSCA to implement this layer.

There are several advantages of the developed MiCADO prototype. First, it can support a large set of applications which have the common feature of serving large, variable number of users with variable size of data. To replace the currently prototyped Data Avenue service with any other similar type of services is extremely easy since it is used inside a Docker container and it does not require any changes in the MiCADO architecture. The price for this flexibility is that the application implementing the required service should be dockerized before being used in MiCADO. Second, MiCADO guarantees that the service is optimised, i.e. MiCADO creates new service instances in Data nodes when the load increases and removes some of them if the load drops. Third, MiCADO guarantees the well-balanced usage of these service instances based on the Distribution layer that directs user requests in an even way to the different service instances.

Based on the MiCADO concept, WS-PGRADE/gUSE gateways running in clouds can be extended with a scalable Data Avenue (DA) service. Such a DA service is installed within a MiCADO architecture that can be attached to the gateway. Occopus can be used to deploy both the WS-PGRADE/gUSE gateway and the attached MiCADO with the DA service. These set of services guarantee that the users of WS-PGRADE/gUSE can make any necessary file transfer without performance loss even if the number of users is increased in a significant way.

5. Performance evaluation

In order to evaluate the performance of the MiCADO prototype implementation described in Section 4, a set of experiments have been designed and implemented on the CloudSigma public cloud. Different phases, including building up the MiCADO infrastructure, and also scaling up and down the application nodes have been measured using the Data Avenue application. These experiments provided evidence for the automated scalability features provided by MiCADO.

The first task is to build the MiCADO infrastructure, including Data nodes (the nodes that host the Data Avenue application and a MySQL database node required by DA), load balancers (HAProxy), and monitoring infrastructure (Prometheus). This task is executed by Occopus based on a pre-defined infrastructure descriptor file. The operator first needs to import the node descriptor files containing the cloud interface type and endpoint of the target Cloud API where the nodes should be built. After importing the necessary files, MiCADO is started through the Occopus-Rest-API by submitting the infrastructure descriptor file which specifies the relation between the virtual machines, and the number of instances that will be created. Application specific variables such as port numbers and database credentials are also defined in this file. Next, Occopus creates the base infrastructure without the user application. Overall, it took 320 seconds on our target cloud to build the infrastructure, with Figure 5 illustrating how this time is broken down between the different node types.

Once the components are successfully connected to each other, the DA application can be started. Since MiCADO works with Docker Swarm, the applications can be started as Docker services. To do so, the operator needs to start DA in global service mode through the Swarm

service locally or remotely. Global mode will assure that every node in the Docker cluster will start the application automatically in one instance per node. This gives us the advantage that when we scale up the application layer the new node will work just as the other ones in the Docker cluster. Deploying the application in this way took 140 seconds in our experiment.

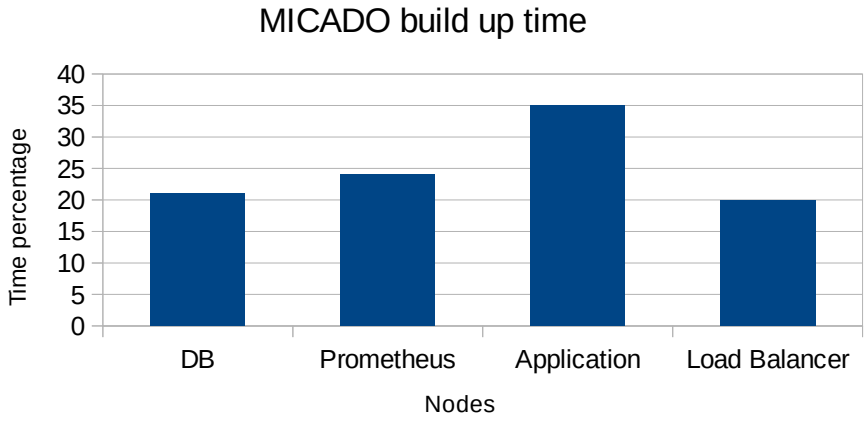

Fig. 5: Time to build the MiCADO infrastructure

After the MiCADO infrastructure was built and the DA application was successfully started, we tested and measured the automated scale-up and scale-down features of MiCADO. First, we put load on the cluster by transferring large volumes of data through the application nodes that run DA. To make the application cluster overloaded, 1 GB files from multiple different sources have been transferred. In the current MiCADO prototype there is an alert defined that in case the load in the Docker cluster exceeds 60% of the available resources, MiCADO automatically creates a new DA node to scale the infrastructure up. In our experiment, it took MiCADO approximately 300 seconds to connect this new node to the cluster and make the application available. On the other hand destroying existing virtual machines when decreasing the load on the cluster, took only 12 seconds in average. Figure 6 summarises the above described scale-up and down times and compares these to creating and destroying the complete infrastructure.

Operation	time (sec)
create infrasturcture	320
destroy infrastructure	15
scale up app node	300
scale down app node	12

Fig. 6 Infrastructure and node creation/destroy

The next benchmark was designed to demonstrate MiCADO in actual usage, with scale-up and scale-down events. Figure 7 shows three graphs. These graphs were generated by Grafana [15] based on the input provided by the Prometheus monitor. On the top, the average CPU utilization in the application cluster is illustrated. In the middle graph, the CPU usage of the individual nodes in the application cluster are shown with different colours for each node (which in average gives us the upper graph). Finally, the bottom graph indicates the number of nodes in the Docker cluster.

Overall, the figure illustrates how the number of application nodes changed depending on the actual load on the cluster. At the beginning (15:34) there was no load on the cluster which could serve user requests perfectly with the help of only one VM (Virtual Machine), shown in

green on the middle graph. As the file transfers have started, the load soon reached 100% of CPU usage on the only available node. After MiCADO ascertained that the load remains high (the load needs to be constantly above the alert value of 60% for a certain amount of time, in our case for 180 seconds to avoid unnecessary scaling-up/down), it fired an alert which called Occopus to instruct scaling up the application cluster. The new node (blue) was connected to cluster at 15:43 and after the virtual machine finished the boot process it started the DA application automatically, to decrease the load on the first node that was previously overloaded (15:45). The same up-scaling event can be seen on the figure at 15:49 as two nodes still struggled to serve user requests, and an additional third node was connected to the Docker cluster.

On the "CPU/node" graph we can check that the application nodes actually share the load between them, while their load are close to each other. Also worth mentioning that new nodes are connected to the cluster with instant 100% of load at the beginning. The reason for this is the boot process of the virtual machines. When the nodes are booting they are at full load, and when they finish starting the Docker containers, they remain on the actual load coming from the application.

At the end of this test we successfully transferred 12GB of data using three DA services as maximum in parallel. The transfer took place between 15:36 and 15:56, as we can see in Figure 7. From that point, MiCADO started to scale down the application cluster. The number of nodes decreased by one every time when Prometheus fired up the alert telling Occopus that the cluster is under-loaded. At 16:06 the number of nodes decreased back to the minimum of one application node. It is important to note that while the under-loaded alert in Prometheus was still firing, Occopus did not scale below the minimum number of nodes, which was one in our case, to make sure that the application is reachable at any time.

Fig. 7 Resource optimization, on both scale up and down events with MiCADO

The above described experiment clearly illustrates that MiCADO works as it is expected. The application was scaled automatically on demand to adopt to actual load. In this experiment we adjusted the time interval which MiCADO had to wait between two scaling events before

scaling again: 300 seconds for scaling up and 60 seconds for scaling down. These time ranges were selected to fit the DA application where large file transfers may require more time to finish, while after they finished, the load expected to drop dramatically allowing scaling down more quickly. As these numbers are application specific they can be set to make sure the scaling events happen in the expected time intervals.

6. Industry relevance

As part of the COLA European project, several application scenarios are investigated and large scale industry trials are being implemented involving SMEs and public sector organisations. These trials are applying the MiCADO framework in order to provide automatic scalability of applications in the targeted areas. The project develops three full scale demonstrators and twenty further proof of concept solutions. This section gives a short overview of three of these trials in order to illustrate the industry relevance and significance of the developed solution. The implementation of these trials is currently in progress and detailed results will be reported in follow-up papers.

Scalable evacuation planning service. One of the critical issues in developing evacuation simulations is speed. As the size and detail of a simulation increases so does the time taken to process each scenario. Saker Solutions is a UK based company with a mission to expand the benefits achieved from the use of simulation modelling. Saker developed SakerGrid [23], a Desktop Grid based solution to support the efficient execution of large scale simulation scenarios and significantly reduce the timeframe for users to undertake model experimentation. However, SakerGrid is limited by the number of physical ‘nodes’ that are present on the network. When decisions need to be taken in a short timeframe, or when there is a need to run a plethora of replications in order to obtain a statistically valid result, the ability to burst onto a cloud from SakerGrid would be highly beneficial. Within the COLA project Saker will use MiCADO to burst out to the cloud on demand for scaling up simulations if required, and also quickly scaling back when economic factors make it necessary.

Social media data analytics for public sector organisations. The spread of Social Media created vast amount of information available in public sources regarding people’s preferences and opinions. Monitoring this information requires significant computation and storage resources and has become a critical aspect for both private companies and also for the public sector. Aragon Regional Government in Spain is willing to develop communication channels with citizens to become aware of their opinion about the local government’s services and the ways these can be improved. Authorities also want to offer entrepreneurs and companies in the region with information that can be used to improve business or develop new ones. To fulfil this objective, regional government uses Eccobuzz, a web application for Social Media data mining, competitive intelligence and brand and media management developed by Spanish Company Inycom. One of the challenges facing such application is fluctuation of computing load required. The information collected from the Internet grows exponentially and there is uncertainty regarding how much information will be collected by the crawlers to be processed later. Furthermore, end users can change the configuration of the system increasing unexpectedly the number of crawlers and their launching frequency. Within the COLA project Eccobuzz will be reengineered into microservices based architecture and extended with MiCADO to support dynamic resource demand.

Scalable simulations for manufacturing SMEs. The European CloudSME project [9] developed a cloud-based platform to run computation and data intensive manufacturing and engineering simulations on heterogeneous cloud computing resources [10]. The solution is

based on the CloudBroker multi-cloud platform and the gUSE science gateway framework [11]. Several industry applications, e.g. open source (OpenFoam) and proprietary (TransAT) fluid dynamics simulation software, design and validation of scanned foot images for foot insole production [12], and business process modelling applications [13] have been ported to and operated on the platform. During the COLA project, start-up company CloudSME UG will prototype several of these applications using MiCADO. In each scenario, the aim is to optimise resource consumption of the simulation application by minimising costs while still providing significantly enhanced performance for its clients.

Industry partners of COLA estimate that utilising MiCADO they can increase the efficiency of their business processes by 20-70%, increase customer satisfaction by 20-60%, decrease time to product and market by 10-50%, and increase their profit by 10-100% (depending on application areas and company profiles).

7. Conclusion and future work

This paper described the generic concept and the first reference implementation of MiCADO that supports the automated scalability of cloud applications. Initial benchmarks and experiments were also successfully conducted using the Data Avenue file transfer application and demonstrating that the MiCADO is capable of scaling up and down the application cluster.

Future work will be conducted on two fronts. On the one hand, MiCADO will be further developed within the European COLA project. The current implementation is only the very first prototype that handles only simple alerts and operates based on simple policies. In COLA, MiCADO will be significantly extended to provide scalability based on more complex optimisation scenarios, for example to optimise not only for performance but also for cost. COLA also considers complex user policy enforcement, for example orchestrating the application depending on security policy specifications. The Application Definition layer of MiCADO will be specifically developed based on COLA Application Description templates using the TOSCA standard [16].

On the other hand, the scalability of the DA service and its usability in connection to science gateways will be further investigated. There are many projects and communities in Europe that actively use WS-PGRADE/gUSE gateways. For these communities the scalable Data Avenue service is a good candidate to improve their data staging mechanism. Furthermore, the scalable Data Avenue service can easily be connected to other kind of gateway services making them also capable of providing scalable data staging with good performance.

As we showed in Section 4, Data Avenue can easily be replaced with other kind of services that are already used in existing gateways. This gives the opportunity that various gateways can significantly improve their services if they replace their current non-scalable services with scalable ones. The only task they have to do is to dockerize the application that implements the service.

8. Acknowledgment

This work was funded by the COLA Cloud Orchestration at the level of Applications Project No. 731574 project.

References

[1] Thomas, D. (2009). Cloud Computing — Benefits and Challenges! The Journal of Object Technology, 8(3), 37.

[2] P. Kacsuk et al., “WS-PGRADE/gUSE generic DCI gateway framework for a large variety of user communities” J. Grid Comput., vol. 10, no. 4, pp. 601-630, Dec. 2012.

[3] Marpaung, Sain, & Hoon-Jae Lee. (2013). Survey on middleware systems in cloud computing integration. Advanced Communication Technology (ICACT), 2013 15th International Conference on Advanced Communications Technology, 709-712.
[4] Cloudify [online] Available from: <http://getcloudify.org> [Accessed 16/03/2016]
[5] Manh Pham, L., Tchana, A., Donsez, D., Zurczak, V., Gibello, P., & De Palma, N. (2015). An adaptable framework to deploy complex applications onto multi-cloud platforms. The Institute of Electrical and Electronics Engineers, Inc. (IEEE) Conference Proceedings, 169-174.
[6] AWS CloudFormation [online] Available from: <https://aws.amazon.com/cloudformation/> [Accessed 16/03/2016]
[7] Binz, Breiter, Leyman, & Spatzier. (2012). Portable Cloud Services Using TOSCA. Internet Computing, IEEE, 16(3), 80-85.
[8] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2015). Migrating to Cloud-Native Architectures Using Microservices: An Experience Report
[9] Kiss, T, Dagdeviren, H, Taylor, JES, Anagnostou, A, Fantini, N. (2015). Business Models for Cloud Computing: Experiences from Developing Modeling & Simulation as a Service Applications in Industry, Proceedings of the 2015 Winter Simulation Conference, Pages 2656-2667, ISBN: 978-1-4673-9741-4, IEEE Press
[10] Taylor SJE, Kiss T, Terstyanszky G, Kacsuk P, Fantini N. (2014). Cloud Computing for Simulation in Manufacturing and Engineering: Introducing the CloudSME Simulation Platform. Proceedings of the 2014 Annual Simulation Symposium, ANSS '14, Society for Computer Simulation International: San Diego, CA, USA, 2014; 12:1–12:8.
[11] Balasko A, Farkas Z, Kacsuk P. Building Science Gateways by Utilizing the Generic WS-PGRADE/gUSE Workflow System, Computer Science Jun 2013; 14(2):307, doi:10.7494/csci.2013.14.2.307
[12] Gabor Terstyanszky, Tamas Kiss, Simon Taylor, Anastasia Anagnostou, Miguel Subira, Giuseppe Padula, Enrique De Meer Alonso, Jose Manuel Martin Rapun: Validating Scanned Foot Images and Designing Customized Insoles on the Cloud, in proceedings of HICSS-49 2016, Havaii International Conference on System Sciences, January 5-8 2016, Pages 3288 – 3296, ISSN 1530-1605, DOI 10.1109/HICSS.2016.411, Published by IEEE
[13] Terstyanszky, G., Kiss, T., Ozkok, A and Isler, V: Simulating Business Processes of Manufacturing SMEs on the Cloud. PeerJ. 4, p. e2509v1, 2016
[14] Docker Documentation, [online] Available from: < https://docs.docker.com/> [Accessed 03/05/2017]
[15] Grafana, the open platform for analytics and monitoring, [online] Available from: <https://grafana.com/> [Accessed 03/05/2017]
[16] Pierantoni G, Kiss T, Terstyanszky G. Towards Application Templates Suporting Quality of Service, To be published in proceedings of IWSG 2017, 19-21 June, 2017, Poznan, Poland.
[17] Visti H, Kiss T, Terstyanszky G, Gesmier G, Winter S: MiCADO – Towards a Microservice-based Cloud Application-level Dynamic Orchestrator, To be published in proceedings of IWSG 2016, 8-10 June, 2016, Rome, Italy
[18] CloudBroker GmbH. "CloudBroker Platform". [Online]. Available: http://cloudbroker.com/platform/. [Accessed: 7 Mar 2017]
[19] Hajnal A, Farkas Z, Kacsuk P: Data Avenue, Remote Storage Resource Management in WS-PGRADE, in proceedings of IWSG 2014, 6th International Workshop on Science Gateways, IEEE, 25 August 2014, DOI: 10.1109/IWSG.2014.7
[20] Consul web page [Online] Available: http://www.mammatustech.com/Microservice-Service-Discovery-with-Consul [Accessed: 5 May 2017]
[21] Prometheus web page [Online] Available: https://prometheus.io/ [Accessed: 5 May 2017]
[22] COLA Project web page: [Online] Available: http://www.project-cola.eu/ [Accessed: 5 May 2017]
[23] Kite S, Wood C, Taylor SJE, Mustafee N: Sakergrid: Simulation experimentation using grid enabled simulation software, Proceedings of the 2011 Winter Simulation Conference (WSC), 11-14 Dec. 2011, DOI: 10.1109/WSC.2011.6147939, Phoenix, AZ, USA, 9 February 2012, IEEE.
[24] HAProxy web page: [Online] Available: http://www.haproxy.org/ [Accessed: 5 May 2017]
[25] Kecskemeti G, Gergely M, Visegradi A, Nemeth Z, Kovacs J, Kacsuk P: One Click Cloud Orchestrator: Bringing Complex Applications Effortlessly to the Clouds, In: Euro-Par 2014. Lecture Notes in Computer Science (8806). Springer, Cham, pp. 38-49. ISBN 978-3-319-14312-5 10.1007/978-3-319-14313-2_4
[26] EGI Cloud Compute: [Online] Available: https://www.egi.eu/services/cloud-compute/ [Accessed: 5 May 2017]
[27] The Vialactea Science Gateway, Available: http://vialactea-sg.oact.inaf.it:8080/ [Accessed: 17 July 2017]

Dr Tamas Kiss is a Reader in Distributed Computing at the Department of Computer Science and Deputy Director of the University Research Centre for Parallel Computing at the University of Westminster. He holds Masters Degrees in Electrical Engineering, and Computer Science and Mathematics, and PhD in Distributed Computing. His research interests include distributed and parallel computing, cloud, cluster and grid computing. He has attracted over £20 Million research funding and leads national and European research projects related to enterprise applications of cloud computing technologies. He co-authored more than 100 papers in journals, conference proceedings and as chapters of edited books.

Prof. Dr. Peter KACSUK is the Director of the Laboratory of the Parallel and Distributed Systems in the Computer and Automation Research Institute of the Hungarian Academy of Sciences (MTA SZTAKI). He received his MSc and university doctorate degrees from the Technical University of Budapest in 1976 and 1984, respectively. He received the kandidat degree (equivalent to PhD) from the Hungarian Academy in 1989. He habilitated at the University of Vienna in 1997. He received his professor title from the Hungarian President in 1999 and the Doctor of Academy degree (DSc) from the Hungarian Academy of Sciences in 2001. He served as full professor at the University of Miskolc and at the Eötvös Lóránd University of Science Budapest. He has been a part-time full professor at the University of Westminster (London, UK) since 2001. He has published two books, two lecture notes and more than 350 scientific papers on parallel computer architectures, parallel software engineering, Grid and Cloud computing. He is editor-in-chief of the Journal of Grid Computing published by Springer.

Dr. Jozsef Kovacs is a Senior Research Fellow at the Laboratory of Parallel and Distributed Systems (LPDS) at the Institute for Computer Science and Control (SZTAKI) of the Hungarian Academy of Sciences (MTA). He got his BSc (1997), MSc (2001) and PhD (2008) in the field of parallel computing. His research topic was parallel debugging and checkpointing, clusters, grids and desktop grid systems, web portals. Recently, he is focusing on cloud and container computing especially on infrastructure orchestration and management. He gave numerous scientific presentations and lectures at conferences, universities and research institutes in many places in Europe and outside. He is reviewer at several scientific journals and holds various positions on conferences. He is author and co-author of more than 60 scientific publications including conference papers, book chapters and journals.

Botond Rakoczi is a research associate at University of Westminster where he is one of the lead developers of the COLA project. He has extensive experience in cloud computing and network related technologies. He hold a B.S. degree in computer science, an Oracle Database developer and a CISCO networking degree as well. His was previously part of the Hungarian Research Institute (MTA SZTAKI) where he gained cutting edge knowledge in infrastructure development and took part in research projects as well. Currently he is researching auto-scaling possibilities in virtualized infrastructures.

Akos Hajnal graduated from the Technical University of Budapest in 1996 in Electrical Engineering and received his PhD in Information Science and Technology in 2013 at Eötvös Loránd University. He is working at the Institute for Computer Science and Control of the Hungarian Academy of Sciences (MTA SZTAKI) as a research fellow at the Laboratory of Parallel and Distributed Systems. His research interests focus on software engineering, parallel computing, grids, clouds and software testing.

Attila Farkas is a DevOps engineer at the Laboratory of the Parallel and Distributed Systems in the Computer and Automation Research Institute of the Hungarian Academy of Sciences (MTA SZTAKI). He received his Computer Science Engineering BSc degree from Obuda University, John von Neumann Faculty of Informatics (OE NIK) in 2017. He also achieved second place of OE NIK's student research competition with his paper "Hybrid research and educational IT platform based on container technology" in 2017. He has been involved in the Docker@SZTAKI project addressing Industry 4.0 application domain, and the "Cloud Orchestration at the Level of Application" (COLA) European H2020 projects.

Grégoire Gesmier is a Research Associate at the University of Westminster. During his Bachelor Degree in computing science, which took place at the University of Besançon in

France he did a placement as part of his grade at the University of Westminster. After graduating, in 2014, he joined the ranks of the CPC (Centre for Parallel Computing) of the University where he worked on different project funded under the FP7 body. He worked on developing and supporting application on different technology, such as Cloud computing, Desktop Grid and Cluster.

Professor Gabor Terstyanszky is the Director of the University Research Centre for Parallel Computing (CPC) Centre at the University of Westminster. The Centre specialises in research on large-scale distributed computing infrastructures (DCIs) addressing research challenges of High Performance Computing (HPC) and High-Throughput Computing (HTC), such as clouds, clusters and grids whereby large numbers of computing resources are used to process computationally-demanding problems. Gabor Terstyanszky's research interests include distributed and parallel computing focusing on targeting research issues of Big Data and Cloud Computing. He has been involved in more than 15 research projects as either Principal or Co-Investigator. He also had several research grants at various universities in Germany, Spain, and United Kingdom. He published over 140 papers at conferences and journals. He was member of programming committees of several conferences and workshops. He supervised more than 10 PhD students. He has taught several BSc and MSc modules on distributed computing such as Distributed Computing, Service Oriented Architecture, Web Services. Currently, he is teaching the Client Server Architecture and Advanced Big Data Analytics module.

.

- a generic microservices-based architecture for application-level cloud orchestration, called MiCADO (Microservices-based Cloud Application level Dynamic Orchestrator) is defined
- first reference implementation of MiCADO utilizing container-based open source cloud technologies is described
- an example of a science gateway supported by the first version of the MiCADO architecture is introduced
- performance benchmarks are presented showing the MiCADO supported implementation of the Data Avenue service of the WS-PGRADE/gUSE science gateway framework