JoGC manuscript No.
(will be inserted by the editor)

Enhancing Federated Cloud Management with an Integrated Service Monitoring Approach

A. Kertesz · G. Kecskemeti · M. Oriol · P. Kotcauer · S. Acs · M. Rodríguez · O. Mercè · A. Cs. Marosi · J. Marco · X. Franch

Received: date / Accepted: date

Abstract Cloud Computing enables the construction and the provisioning of virtualized service-based applications in a simple and cost effective outsourcing to dynamic service environments. Cloud Federations envisage a distributed, heterogeneous environment consisting of various cloud infrastructures by aggregating different IaaS provider capabilities coming from both the commercial and the academic area. In this paper, we introduce a federated cloud management solution that operates the federation through utilizing cloud-brokers for various IaaS providers. In order to enable an enhanced provider selection and inter-cloud service executions, an integrated monitoring approach is proposed which is capable of measuring the availability and reliability of the provisioned services in different providers. To this end, a minimal metric monitoring service has been designed and used together with a service monitoring solution to measure cloud performance. The transparent and cost effective operation on commercial clouds and the capability to simultaneously monitor both private and public clouds were the major design goals of this integrated cloud monitoring approach. Finally, the evaluation of our proposed solution is presented on different private IaaS systems participating in federations.

Keywords Cloud Computing · Cloud Federation · Service Monitoring · Cloud Brokering

A. Kertesz, P. Kotcauer, S. Acs, A. Marosi
MTA SZTAKI,
H-1518 Budapest, P.O. Box 63, Hungary
E-mail: {kertesz.attila,kecskemeti.gabor,kotcauer.peter, acs.sandor,marosi.attila}@sztaki.mta.hu

G. Kecskemeti
Universität Innsbruck, 6020, Innsbruck, Technikerstraße 21a
on leave from MTA SZTAKI
E-mail: gabor@dps.uibk.ac.at

M. Oriol, M. Rodríguez, O. Mercè, J. Marco, X. Franch
Universitat Politècnica de Catalunya,
08034 Barcelona, c/Jordi Girona 1-3, Spain
E-mail: moriol@lsi.upc.edu, {nebrios2,oscar.mp10}@gmail.com, jmarco@lsi.upc.edu, franch@essi.upc.edu

1 Introduction

Cloud Computing [5,23] offers simple and cost effective outsourcing in dynamic service environments and allows the construction of service-based applications extensible with the latest achievements of diverse research areas, such as Grid Computing, Service-oriented computing, business processes and virtualization. Cloud-based, highly dynamic service environments [11] require a novel infrastructure that incorporates a high-level monitoring approach to support autonomous, on demand deployment and decommission of service instances. Virtual appliances (VA) encapsulate a complete software system (e.g. operating system, software libraries and the deployable services themselves) prepared for execution in virtual machines (VM). Infrastructure as a Service (IaaS) cloud systems provide access to remote computing infrastructures by allowing their users to instantiate virtual appliances on their virtualized resources as virtual machines. Nowadays, several IaaS systems co-exist and they are independently offered by several public service providers (like Amazon [43] or RackSpace [49]) or by smaller scale privately managed infrastructures. Cloud solutions are also spreading fast in academia with the emerging open-source tools, such as Eucalyptus [45] and OpenNebula [48,32], but these solutions can hardly interoperate.

Related works have identified several shortcomings in the current cloud infrastructures [34]: e.g. federated clouds face the issue of scalability, self-management and

lost of complete control on computing costs. The ever growing user demands call for overextending the boundaries of a single cloud system. In these cases, users need to handle the differences between various cloud providers and have to negotiate their requirements with multiple parties. Federated clouds aim at supporting these users by providing a single interface on which they can transparently handle different cloud providers, as they would do with a single cloud system. Therefore it is essential to construct federated cloud systems that not only offer a single interface for their users, but also automatically manage their virtual machines independently from the currently applied cloud system. Recent studies (eg. [12]) have also shown that significant performance differences can be experienced on acquired virtual resources in Clouds. Therefore an efficient cloud selection in a federated environment requires a cloud monitoring subsystem that determines the actual status of available IaaS systems.

To overcome these challenges, we propose an architecture that copes with the varying load of user requests, enables virtualized management of applications, enhances provider selection, establishes interoperability and allows users to reduce their operating costs by simultaneously exploiting public, academic and private cloud systems. This architecture incorporates the concepts of meta-brokering, cloud-brokering and on-demand service deployment, supported by a sophisticated monitoring solution. Our architecture serves as an entry point to the entire cloud federation by providing transparent service execution for users. Our meta-brokering component allows the system to interconnect various cloud-brokers available in the system. It is also responsible for selecting a proper execution environment managed by a cloud-broker. This selection process relies on a sophisticated monitoring component, which provides up-to-date service availability and infrastructure reliability based on specific monitoring metrics. The cloud-broker component is responsible for managing the virtual machine instances of the particular virtual appliances hosted on a specific IaaS provider. Our architecture also organizes virtual appliance distribution with its automatic service deployment component that can decompose and deliver virtual appliances in smaller parts.

Therefore the main contributions of this paper are: (i) a holistic view of interoperable federated clouds with integrated service monitoring solution managed by a multi-level resource management architecture, (ii) the introduction of an incorporated cloud service monitoring solution together with a minimal metric monitoring service to measure cloud performance on a cost effective and provider independent way, and (iii) the evaluation of the proposed integrated monitoring solution on private IaaS systems with the help of the minimal metric monitoring service.

This paper is organized as follows: first, we gather related works in Section 2. In Section 3, we introduce our proposed architecture and discuss its main components. In Section 4, we introduce the minimal metric monitoring service, and in Section 5, we present the evaluation of our approach in different private clouds. Finally, we conclude our research in Section 6.

2 Related work

In this section we describe the related works relevant to our findings. First, we describe the different cloud federation approaches found in the literature, then we describe the evaluation mechanisms used for calculating the performance of the cloud as the basis to establish the most convenient deployment strategies.

2.1 Cloud federation approaches

Cloud federation refers to a mesh of cloud providers that are interconnected based on open standards to provide a universal decentralized computing environment, where everything is driven by constraints and agreements in a ubiquitous, multi-provider infrastructure. Next, we summarize the relevant related works in this field.

Buyya et al. [6] suggest a federation-oriented, just-in-time, opportunistic and scalable application services provisioning environment called InterCloud. They envision utility oriented federated IaaS systems that are able to predict application service behavior for intelligent down and up-scaling infrastructures. They list the research issues of flexible service-to-resource mapping, user and resource centric QoS optimization, integration with in-house systems of enterprises, scalable monitoring of system components. They present a market-oriented approach to offer InterClouds including cloud exchanges and brokers that bring together producers and consumers. Producers are offering domain specific enterprise Clouds that are connected and managed within the federation with their Cloud Coordinator component. Celesti et al. [9] proposed an approach for the federation establishment considering generic cloud architectures according to a three-phase model, representing an architectural solution for federation by means of a Cross-Cloud Federation Manager (CCFM), a software component in charge of executing the three main functionalities required for a federation. In particular, the component explicitly manages: i) the discovery phase in

which information about other clouds are received and sent, ii) the match-making phase performing the best choice of the provider according to some utility measure and iii) the authentication phase creating a secure channel between the federated clouds.

Bernstein et al. [1] define two scenarios that exemplify the problems of multi-cloud systems: (i) VM Mobility, where they identify the networking, the specific cloud VM management interfaces and the lack of mobility interfaces as the three major obstacles, and (ii) storage interoperability and federation scenario, in which storage provider replication policies are subject to change when a cloud provider initiates subcontracting. Marshall et al. proposed an IaaS cloud solution to elastically extend physical clusters with cloud resources [26]. They created a so called elastic site manager on top of Nimbus, which interfaces directly with local cluster managers and three different policies were examined for elastic site addition. GridBot [31] represents an approach for execution of bags-of-tasks on multiple clusters, volunteer and service grids. It has a Workload Manager component that is responsible for brokering among these environments, which is similar to our approach, but we rather target multi-cloud solutions and focus on highly dynamic service executions instead of tasks more suitable for volunteer grids. Cuomo et al. introduced a volunteer-based approach called Cloud@Home to form a federation in [10]. This solution is only applicable to providers, who are willing to voluntarily share their resources. Their work is focusing on providing reliable service provisioning despite the high degree of heterogeneity existing in such systems.

Regarding recent Cloud Computing projects, the OPTIMIS project [13] identified that commercial solutions in the field of Cloud Computing have mainly focused on providing functionalities at levels close to the infrastructure, and higher-level solutions, like Platform-as-a-Service (PaaS) environments are limited to a single infrastructure provider. Their goal is to build an improved cloud service ecosystem that supports higher-level concerns and non-functional aspects to achieve a wider adoption of Cloud Computing. The project followed a holistic approach for multiple coexisting cloud architectures and they target cloud service life-cycle optimization including cost, trust, risk and economic goals. They also planed to enable market-oriented multi-cloud architectures with clarified legislative background.

The Reservoir project [28] approach can be exemplified by the electric grid approach: for one facility to dynamically acquire electricity from a neighboring facility to meet a spike in demand. In this vision, disparate datacenters should be federated in order to provide a seemingly infinite service computing utility. Regarding their architectural view, a Reservoir Cloud consists of different Reservoir Sites (RS) operated by different IPs. Each RS has resources that are partitioned into isolated Virtual Execution Environments (VEE). Service applications may use VEE hosts from different RSs simultaneously. Each application is deployed with a service manifest that formally defines its SLA contract. Virtual Execution Environment Managers (VEEM) interact with VEEs, Service Managers and other VEEMs to enable federations to be formed. A VEEM gathers interacting VEEs into a VEE group that serves a service application. This implies that a Reservoir service stack has to be present on the resources/sites of IPs. Its federated IaaS cloud management model presented in [29] argues that commercial cloud providers could also temporarily lease excess capacities during high-demand periods. They investigate the problems faced by federated cloud management solutions: (i) dynamic service elasticity, (ii) admission control, (iii) policy-driven placement optimization, (iv) cross-cloud virtual networks (v) cross-cloud monitoring, and (vi) cross-cloud live migration.

The Contrail project [8] proposes an SLA-centered federated approach to Clouds. Its goal is to minimize the burden on the user with eliminating provider lock-in by exploiting resources belonging to different cloud providers regardless the kind of technology they use, and to increase the efficiency of using Cloud platforms by performing both a vertical and a horizontal integration. It follows an open-source approach toward technology and standards, and supports user authentication and applications deployment by providing extended SLA management functionalities. Its federation architecture acts as a bridge among the users and the cloud providers, and has three layers. The top layer (called Interface) provides ways to interact with the federation. The middle layer (called Core) contains modules that fulfill the functional and non-functional requirements of the federation. The federation runtime manager (FRM) operates in this layer, which uses a set of heuristics that consider different aspects to govern the federation, such as to minimize economical cost and to maximize performance levels. Finally, the bottom layer (called Adapters) contains the modules that retrieve information and operate on different cloud providers.

The mOSAIC project [27] offers the specification of service requirements in terms of a cloud ontology via an innovative API. The implementation of this approach will offer a higher degree of portability and vendor independence. It also provides application programming interfaces for building applications using services from multiple cloud providers and plans to realize a

self-adaptive distributed scheduling platform composed of multiple agents implemented as intelligent feedback control loops to support policy-based scheduling and expose self-healing capabilities. They plan to foster competition between cloud providers by enabling the selection of best-fitting cloud services to actual user needs and efficiently outsource computations.

Our proposed approach also tackles the interoperability problems of cloud federations, but it also goes beyond this state-of-the-art by providing a generic solution for monitoring service provisioning in different IaaS systems.

2.2 Evaluation of cloud performance

Cloud federation approaches follow different deployment strategies based on the evaluation of the performance of the available clouds. We distinguish these strategies either if they are based on offline or online performance data.

Offline performance data. M. Schmidt et al. [30] investigate different strategies for distributing virtual machine images within a data center: unicast, multicast, binary tree distribution and peer-to-peer distribution based on BitTorrent. They found the multicast method the most efficient, but in order to be able to distribute images over network boundaries (for a so called cross-cloud solution), they have chosen BitTorrent. The authors only investigated distribution methods within the boundaries of a single data center, going beyond that remained future work.

Online performance data. With respect to online performance data, several monitoring solutions gather the QoS of the different cloud systems. We distinguish between those that monitor at the infrastructure level, and those that monitor at the service or application level.

Monitoring the infrastructure. Regarding commercial cloud monitoring solutions, Amazon Web Services launched Amazon CloudWatch [42] in 2009, which is a supplementary service for Amazon EC2 instances that provides monitoring services for running virtual machine instances. It allows gathering information about the different characteristics (traffic shape, load, disk utilization, etc.) of resources, and based on that, users and services are able to dynamically start or release instances to match demand as utilization goes over or below predefined thresholds. The main shortcoming of this solution is its strong bounds to a specific IaaS, because it introduces a monetary overhead by charging every monitored instance by an hourly rate.

Nagios XI [47] is an infrastructure monitoring solution that also addresses clouds (the Amazon EC2 interface is supported). It is a robust, comprehensive, business-oriented solution that is capable of monitoring a wide area of system components including services, operating systems and network components. Even though it has an open source core, it can be very costly to use it in cloud federation. Our solution is focusing more on monitoring of service component metrics.

The Cerebrata Azure Diagnostics Manager [44] is a monitoring component of the Azure Platform designed for monitoring the performance of Azure applications. It can be regarded as a data and event logging system usable in the Azure system only. Therefore, it is not suitable for utilization in arbitrary providers of a federation.

An academic approach for cloud performance monitoring is introduced by Yigitbasi et. al. [36], called C-Meter. Using this framework, workloads can be submitted to target clouds to analyze their performances. On the contrary, our monitoring solution examines the real, running applications instead of workloads, and does not necessarily require additional deployments.

Another solution is presented by Baur et al. in [4]. In their approach, they present an integrated monitoring solution for heterogeneous Grid infrastructures, which aggregates and provides the monitoring data from different Grids. To do so, they apply transformation rules to the monitored data of each Grid in order to get an homogeneous data model. However, contrary to our solution, they require each Grid to have its own Grid monitoring service to collect the data.

Monitoring the service level. Regarding monitoring of the provisioned services, the existing technical approaches found in the literature to gather the required data can be classified into two big categories. On the one hand, some proposals rely on the use of monitoring directives embedded into the services themselves using Aspect Oriented Programming (AOP), and weaving the monitoring code into the execution process, which is commonly defined in BPEL [3,37]. The advantages of this solution are a result of those of AOP, which isolates the monitoring code from the business logic as an aspect, providing low coupling and the ability to add/modify the monitoring rules without affecting the core code of the service. However, in the context of deploying the service over cloud infrastructures, changes over the monitoring rules would require dynamic weaving processes on runtime, which might be somehow difficult if the cloud does not provide the required artifacts for inserting these directives on the execution chain of the service engine. For instance, Zhou et al. [38] make usage of Model-Driven techniques to automatically generate monitoring code for Axis. As advantage, this solution seems to be more efficient than the previous one

since there is no weaving process. However, this approach depends on the technology used for service deployment, in this case the engine, where the service is installed.

On the other hand, other proposals use a proxy that intercepts the messages to add monitoring capabilities to the system without the need to be so intrusive into the service or its engine and hence, being independent of the technologies chosen in the implementation of the services [2,35]. In this case, the same monitoring tool can be used for all kind of services deployed in a cloud. Its main drawback is that if the architecture is not properly built, the proxy can generate a bottleneck affecting negatively the response time of the monitored services.

Monitoring multiple levels. A more sophisticated solution is GMonE [21], a cloud monitoring platform aimed at monitoring the different service levels (from SaaS to IaaS) to support the lifecycle of services deployed in the cloud. However, their solution requires the implementation of plug-ins that are highly coupled to the infrastructure of each cloud provider, which makes it unsuitable for cloud federations.

3 Federated cloud management with integrated service monitoring

Figure 1 shows the Federated Cloud Management (FCM) [20] architecture extended with an integrated service monitoring approach. The figure reveals the interfaces of our components and their relations with the currently available IaaS systems. Our solution offers interoperable access to a federated cloud environment through the interface of the *meta-brokering* component. This component is capable to decide between various *cloud-brokers* based on metrics gathered from a *service monitoring* subsystem. Cloud-brokers extend the current IaaS functionality by analyzing and dispatching service requests. Based on service demand patterns, they also use the *service deployment* component to deploy or decommission the requested services as virtual machines in specific IaaS systems. The generic integrated solution highlighted in this figure can be used to monitor any existing component in the infrastructure of the providers participating in the cloud federation, and provide this information to the upper decision making layers of the architecture.

In its present state, the FCM architecture is mostly focused on the handling of stateless web services. As a result, we have investigated monitoring solutions that can also handle these kind of services [18,16,7]. Because of its unique QoS attribute monitoring capabilities, we have selected SALMon (Service Level Agreement Monitor [25]) to act as the Service Monitoring layer of FCM. SALMon's advanced capabilities (relevant to the FCM architecture) are discussed in detail in Section 3.3.

In this architecture users are able to execute services deployed on cloud infrastructures transparently, in an automated way. The *Generic Service Registry* (GSR – see Figure 1) contains information of these services (including WSDLs [50] and their virtual machine images or *virtual appliances*). When a service is deployed on a new host, the service deployment component registers its new endpoint to the service registry. Upon decommissioning, these endpoint registrations are removed. During operation, the SALMon monitoring subsystem allows the components in FCM to order regular QoS evaluation on the deployed services according to predefined metrics coupled with the service's description in GSR.

In our system, users send service calls as request submissions to the *Meta-Brokering* layer realized by Generic Meta-Broker Service (GMBS). *Federated call submissions* specify the requested service, the operation to be called, and its possible input parameters. The GMBS checks if the service is registered to the GSR, and if so, it selects a suitable *Cloud-Broker* for further submission, otherwise rejects the request. Based on service usage patterns (e.g. average service response time, call frequency) the GMBS requests the monitoring of service instances via SALMon. The monitoring results are used by its matchmaking algorithm that combines the just received dynamic data with information gathered from the registry and with status information on cloud-brokers and SALMon (gathered with the *query cloud metrics* function in Figure 1). GMBS forms a cloud federation by enabling the autonomous management of the interconnected cloud infrastructures through cloud-brokers.

Cloud-brokers are dedicated to specific IaaS systems and offer a queue for incoming service calls. Incoming service calls are scheduled to virtual machines available in VM queues (*Call* $\Leftrightarrow$ *VM Association*). The automated *management of these virtual machine queues* is the main goal of our cloud-brokers. Members of the VM queues represent those VMs that are ready to serve a particular service call. For every virtual appliance (i.e. kind of service) a VM queue is maintained. To meet the respective service demand, the cloud-broker decides the amount of required VMs meeting the actual request load. If necessary, the cloud-broker requests VM instantiation or decommission from the service deployment component – see Figure 1. The default virtual machine scheduling is based on the currently available requests in the incoming service call queue, their historical execution times, and the number of running VMs. The

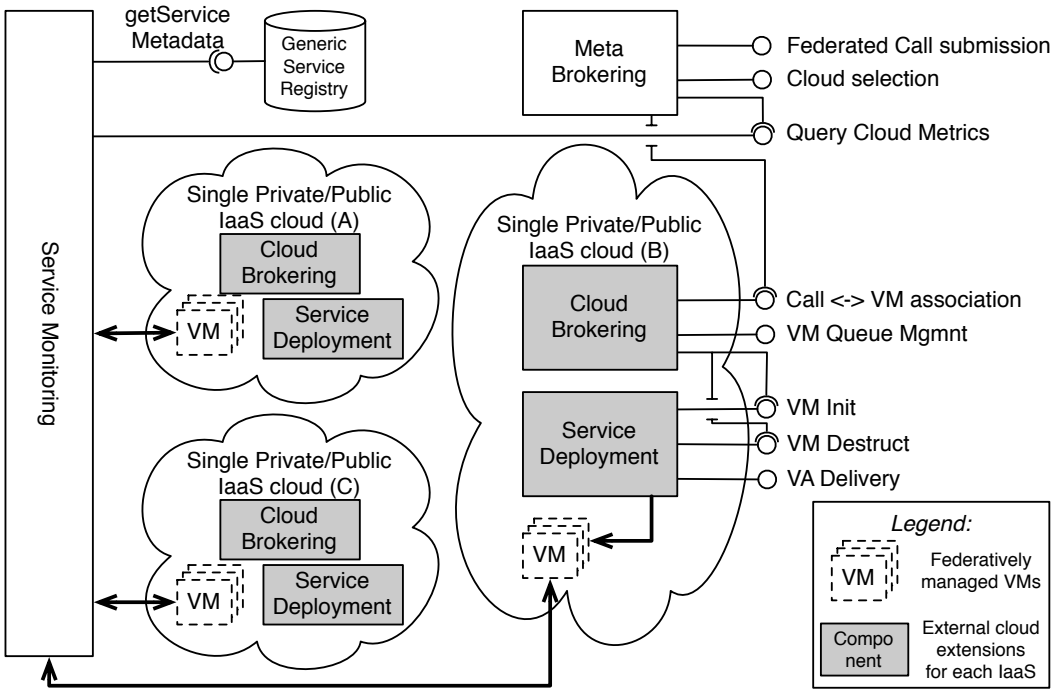

Fig. 1 The FCM architecture with enhanced monitoring

secondary task of a Cloud-Broker involves the dynamic creation and destruction of the various queues.

The following subsections provide a detailed overview on the main components of the architecture.

3.1 Meta-brokering approach for federating clouds

As we already mentioned in the beginning of this section, brokering takes place at two levels in this architecture: the service request is first submitted to a meta-brokering component implemented by the Generic Meta-Broker Service (which is a revised and extended version of the Grid Meta-Broker Service described in [17]), where a high-level decision is made to which cloud infrastructure the call should be forwarded. Then the service call is queued at the selected Cloud-Broker, where lower level brokering selects the VM that will perform the actual service execution.

Next we shortly summarize the role of GMBS within FCM. This meta-brokering service has five major components. The Meta-Broker Core is responsible for managing the interaction with the other components and handling user interactions. The MatchMaker component performs the scheduling of the calls by selecting a suitable cloud-broker. This decision making is based on aggregated static and dynamic data stored by the Information Collector component in a local database. The Information System (IS) Agent is implemented as a listener service of the meta-broker, and it is responsible for regularly updating static information gathered from the Generic Service Registry on service availability, dynamic information on service and cloud reliability provided by SALMon (further discussed in Section 3.3), and aggregated dynamic information collected from the cloud-brokers including average virtual appliance deployment and service execution time. The Invoker component forwards the service call to the selected Cloud-Broker and receives the service response.

Each Cloud-Broker is described by an XML-based Broker Property Description Language (BPDL) document containing basic broker properties (e.g. name, managed cloud infrastructure), and the gathered dynamic properties. The scheduling-related attributes are also stored in the description language document. More information on this document format can be read in [17]. Namely, the following data are stored in the BPDLs of each Cloud-Broker:

- Static virtual appliance availability information for each native repository according to the GSR registry;
- average VM deployment time and average service execution time for each virtual appliance provided by the cloud-brokers;
- and dynamic reliability information expressed by metrics collected by SALMon.

The scheduling process first filters the brokers by checking virtual appliance availability in the native cloud repository, then a rank is calculated for each broker

based on the collected dynamic data. Finally, the cloud-broker with the highest rank is selected for managing the actual service request.

3.2 Cloud-brokering and automated service deployment in FCM

The Cloud-Broker, which is an extended version of the system described in [19], handles and dispatches service calls (i.e. requests) to resources and performs resource management within a single IaaS system. It dynamically creates and destroys virtual machines and VM queues of different virtual appliances. Virtual machine creation is supported in the registry by storing additional static requirements (e.g. its minimum disk, CPU or memory requirements) about each appliance's future instances.

A VM queue lists resources capable of handling specific service calls, thus instances of a specific virtual appliance. New resource requests are inserted to the queue of the appropriate appliance, while the need for resource destruction is indicated by shortening the queue. Resource entries are managed by the VM Handler that is designed to interact with the public interface of a specific IaaS system. It translates queue changes, as VM creation and destruction requests, towards the IaaS system. The VM creation process is further detailed in the last paragraph of this subsection.

The service call queue stores incoming service requests and a reference in the GSR registry to an appliance for each call. There is a single service call queue in each Cloud-Broker, while there are many VM queues. Dynamic requirements for a virtual appliance may be specified with the service call: additional resources (CPU, memory and disk), and a unique id to identify service calls originating from the same requester. If a unique dynamic requirement is specified, then the Cloud-Broker creates a new VM queue for them and starts the newly requested VM. Most IaaS systems offer predefined classes of VMs (CPU, memory and disk capacity) not adjustable by the user, therefore the Cloud-Broker selects the VM class that offers the requested extra resources. This may lead to allocating excess resources in some cases (e.g. the VM class that meets the extra CPU requirement offers twice the requested memory). The Cloud-Broker also schedules service call requests to VM's and manages the VM life-cycle. If a service call cannot be associated to any VM, the Cloud-Broker may decide to start a new one for the request. The VM creation and destruction decisions are based on the following:

- The number of running VMs available to handle the service call;
- the number of waiting service requests for the appliance in the service call queue;
- execution time metrics of service calls provided by the monitoring service;
- deployment time metrics of virtual appliances provided by the monitoring service;
- SLA constraints (e.g. total budget, deadline);
- and the billing period of the IaaS system.

If a destruction is needed, shutting down is performed shortly before the end of the billing period of an IaaS cloud with regard to its average decommission time. The billing period is generally published by commercial clouds in their SLA terms, and it is used to determine the minimal time interval in which the users have to pay for using the required resources. In academic clouds, resource usage quotas can be taken into account for the same purpose.

IaaS systems require virtual appliances (VA) to be stored in their native repositories, because only appliances available in native repositories are usable to instantiate virtual machines. FCM organizes the distribution of user created appliances with the help of the Automatic Service Deployment (ASD) [14] component. To meet the demands of highly dynamic service environments, appliance distribution is optimized by automatically decomposing and replicating appliances. To support the rebuilding of decomposed VAs, the ASD requires appliances to embed minimal manageable virtual appliances (MMVA - [15]). These special appliances meet the following properties:

- Provide content management interfaces to add, configure and remove new appliance parts;
- Offer monitoring interfaces to analyze the state of their instances (e.g. provide access to their CPU load, free disk space and network usage);
- And, they are optimally sized: only those files that are required to offer the previously mentioned two properties are present in the MMVAs.

As a result, the ASD always replicates the MMVAs to native repositories. When a native repository does not hold the necessary appliance for the current VM creation request, the VM Handler uses these minimal manageable appliances to reconstruct appliances locally. Consequently, the VM Handler applies the following strategy if it faces a non available appliance. First, it instantiates an MMVA within a VM suitable for the non available appliance. Then, using the new VM's content management interfaces, the VM handler requests the download of the complementary appliance parts to the new VM. These parts are not present in the native repository, so the download operation will use the GSR registry. Therefore, the appliance is rebuilt in

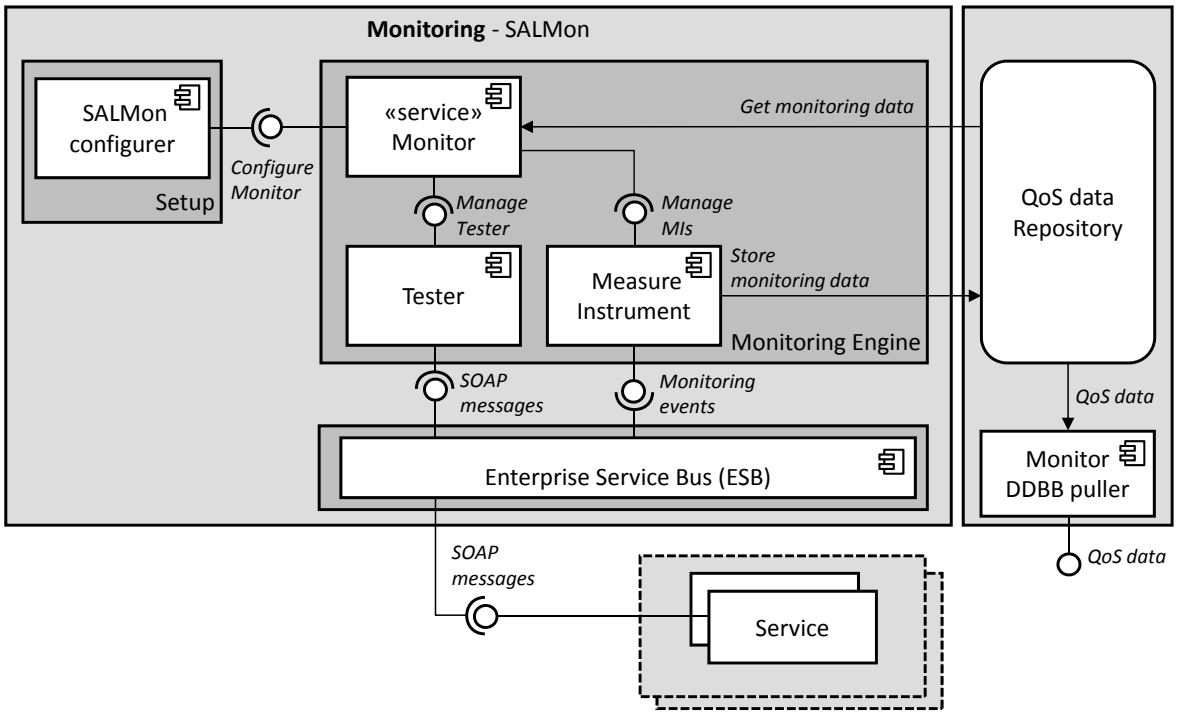

Fig. 2 SALMon framework

a virtual machine originally based on the MMVA. Finally, the VM is ready to serve the scheduled requests from the service call queue.

3.3 Cloud service monitoring with SALMon

SALMon [25] is a service monitoring framework that has been integrated into our proposed FCM architecture in order to gather reliability information on the managed IaaS clouds. It is focused on monitoring the QoS of software services, and is able to evaluate them according to pre-defined conditions, and to notify the results to the interested parties, which is the IS Agent of the GMBS in our case.

The main features of SALMon that justify its use to monitor the cloud infrastructure are (see [25] for details):

Technology independent. Some monitoring solutions are attached to a particular service technology when monitoring the service layer (e.g. BPEL monitoring [3,37], SOAP-based [38], etc.), whereas others are attached to a particular cloud when monitoring the infrastructure layer (e.g. Amazon CloudWatch [42]). SALMon, in contrast, may operate on any available technology with minor changes. The architecture of SALMon decouples the different aspects of monitoring and their technological dependencies are isolated, which allows an easy extension to different infrastructures (see implementation details at [25]). In order to interoperate with FCM, we have extended SALMon to be able to monitor services deployed in the cloud.

Easily interoperable with other frameworks. Not all monitoring solutions are easily interoperable. To this aim, SALMon has been developed as a Service-Based Application itself. On one hand, providing the monitoring solution as a service facilitates the user to monitor the cloud system easily by just deploying it and using the service without worrying about technical details about the instrumentation of the underlying technologies. On the other hand, standard web service protocols ensures the integration capabilities with any framework able to deal with web service technologies. This approach is similar to the solution proposed by Truong et al. [33] in monitoring Grids. However, in their solution, they required each Grid provider to implement and provide the monitoring service of their own Grid system to monitor metrics at the infrastructure layer. In contrast, we overcome this obstacle by complementing SALMon with the M3S service (see section 4). SALMon can be easily deployed on the cloud as any other service by the service deployment component, and communicate with the other FCM services through standard SOAP-based protocols.

Easily extensible with new metrics. The monitor is composed of several measure instruments. Each one is responsible for calculating a specific quality metric. Hence, new metrics can be computed by implementing the corresponding measure instrument and adding it to the monitor. By doing so, the monitor is able to compute new metrics as they are required by the FCM in a federated cloud. A similar feature is also present in other cloud monitoring solutions that uses plug-ins to conduct each monitoring task [21]. However, in contrast to SALMon, they have not been implemented as a Service-Based Application.

Combines passive monitoring and on-line testing. SALMon combines both passive monitoring and on-line testing approaches, being able to configure each method according to the preferences of the user. In FCM, SALMon is used for testing purposes in order to gather the QoS of the constituent services deployed in the cloud. This approach consists of periodically invoking a set of methods of the target service and calculating the QoS over the obtained results. The advantage of the testing approach is that it is not intrusive with the real invocations of the service.

When measuring a cloud system's behavior, the SALMon service evaluates the services deployed in the particular cloud infrastructure. However, some measurements (e.g. network related metrics) are heavily dependent on SALMon's connections to the particular cloud. Thus, to allow informed decision making in GMBS, this work proposes to eliminate the difficulties of SALMon's connectivity (cloud resources are usually behind firewalls) by deploying SALMon into the measured cloud infrastructures. As a result, we have prepared SALMon to be executed in a virtual machine to be deployed to each cloud managed by FCM. To this aim, the new components *SALMon configurer* and a Database query tool named *Monitor DDBB (Dynamic DataBase Binding) poller* have been implemented. The SALMon configurer enables SALMon to be configured dynamically on the cloud, whereas the Monitor DDBB poller is used to retrieve the measured QoS in a separated repository outside the cloud.

The different components of SALMon involved in this framework are depicted in Figure 2:

SALMon configurer. It is the component that configures and starts the *Monitor Service* component of SALMon. The SALMon configurer includes a configuration file named Monitoring Management Document (MMD [24]), which is an XML file that specifies all the required information to configure the monitor dynamically (i.e., services, operations, metrics and the testing or passive monitoring approach). When a new SALMon virtual machine starts up, the GMBS ensures the automatic creation and use of an MMD document that points to newly deployed services.

Monitor Service. It is responsible for managing all the monitoring processes. During a testing approach, it periodically activates the tester component which will perform the tests. It also creates and manages the *Measure Instruments*, which are responsible to obtain the specified *QoS Data*, when the services are invoked.

ESB. The invocations of the services are performed with the *Enterprise Service Bus (ESB)* (i.e., instead of invoking the services directly, all requests and responses of the service are sent through the ESB). When an invocation is intercepted by the ESB, it notifies it to the *Measure Instruments* to compute the QoS. In such a manner, when an invocation is performed, the ESB notifies it to the Measure Instruments in order to compute the QoS transparently and seamlessly to the target service, and not attached to a particular technology.

Measure Instrument. This is the component that implements the logic required to obtain the value of a concrete basic quality metric. The derived metrics are calculated from the set of basic quality metrics by computing the required formula (e.g. average and maximums). The *Measure Instruments* are activated by the *Monitor Service* component based on the quality metrics to measure. Since measure instruments are the core components that actually retrieve the values of the basic metrics, these components are technologically dependent on the kind of service they are monitoring. The list of Measure Instruments that are currently implemented in SALMon includes: availability, response time, execution time and round trip time. Moreover, by combining the Measure Instruments implemented with the M3S, SALMon is able to compute also infrastructure-related metrics such as CPU and network performance.

Publisher Service. It implements the observer pattern for services in a Service-Based Application. This component is used when new measures are obtained for notifying the *meta-brokering* service. Using the observer pattern, SALMon can be decommissioned as soon as the values are retrieved, which reduces any possible overhead due to the consumption of resources. This pattern requires that the subscribed service (the observer, ie. the GMBS in this case) implements the required interface to receive such a notification. This is achieved by defining a common interface with the notify method.

QoS Data It is the repository where the gathered QoS is stored. It is located outside the cloud to provide the access to the data after the VM is destroyed.

Monitor DDBB Poller This is the controller to access the data stored in the QoS Data repository. The Monitor DDBB Poller is used by the GMBS to obtain the required QoS.

4 The Minimal Metric Monitoring Service

The effects of multi-tenancy are observable even with the strongly isolated virtualized environments of an IaaS system. E.g., one could observe degrading connectivity in a virtual machine if network-heavy virtual machines are introduced to the underlying virtual machine monitor (the next section provides further examples and evidence for these effects). Thus multi-tenancy could have significant effects on the reliability information of a service instance deployed in cloud infrastructures. These effects are imposed as seemingly added noise to the measurements of SALMon. To cancel the effects of this noise, we propose to detect the effects of multi-tenancy with a basic service that we refer to as the Minimal Metric Monitoring Service (M3S).

M3S allows SALMon to determine the basic characteristics of the VM in which the M3S was deployed. GMBS uses these measurements both to evaluate the performance and to detect the effects of the internal provisioning policies in a cloud. For performance, individual measurements on the M3S service allows direct comparison of providers that offer the same kind of virtual machine types (e.g., similar VMs to the EC2 type named "*M1.small*"). As a result, the system is capable to evaluate and to choose among both public and private clouds based on the same kind of metrics. Fortunately, these individual measurements are also subject to the effects of multi-tenancy. Thus, to detect the internal provisioning policies, the Information Collector component of GMBS statistically analyses and aggregates measurements (e.g. it calculates their standard deviation). Taking into account such dynamic information at the meta- and cloud-brokering layers can result in better inter- and intra-cloud management. E.g., for services other than the M3S, the GMBS uses these aggregated values to cancel the effects of multi-tenancy in the measurements.

As with any monitoring system, M3S cannot avoid the introduction of some overhead to the system in overall (e.g. it increases the chances of having under-provisioned virtual machines in the system). To reduce its impact, we have designed M3S to function in its own virtual machine and to be lightweight. Being in a separate virtual machine is crucial, as it minimizes the effects of M3S on other virtual machines hosted in the same cloud system. The service is also lightweight in terms of its appliance size and in resource usage. M3S's appliance is also minimized, so whenever it needs to be deployed to take new measurements it is promptly available and does not cause significant delays in the decision making processes of the various layers in FCM. The size is minimized by providing only the minimal functionality required to determine basic VM characteristics.

The M3S service offers 4 methods to evaluate the basic capabilities of its hosting VM. SALMon uses the response times of these four methods to express the reliability of the particular cloud that runs the M3S VM:

1. The method `Ping()` is a generalized ping test to check the *availability of the service*. This method returns a simple empty object to notify that the service is up and running.
2. The method `StressCpu()` performs several mathematical calculations in a large loop over a predefined set of variables, consisting on integer and floating point numbers in order to determine *the computational capability of a given VM*. The calculations include sums, multiplications, divisions, modulos, etc. We use the response time of this method to estimate the computational speed of the VM in which M3S is deployed in. This estimate gives a general overview on the performance of those VMs that have the same type as the M3S VM. Using the historical values of this performance estimate GMBS could even determine if there are multi-tenancy or under-provisioning issues at the particular provider.
3. The method `StressInputBandwidth(input)` is used to compute the download transfer speed of the system – thus determines its *inbound data transfer capability* – , which receives from the invocations of SALMon a considerably sized input to read. This input consists of a pregenerated dataset of 6 MBytes. Based on our experiments, 6 MBytes are enough for rough bandwidth estimates, nevertheless this measurement cannot significantly influence the monthly data transfer bill of GMBS users concerning commercial clouds.
4. The method `StressOutputBandwidth()` is used to compute the upload transfer speed of the system – that we later refer as the *outbound data transfer capability* – . This method responds with a 6-MBytes-long string to SALMon.

Figure 3 exemplifies the operation of our proposed integrated FCM solution. This figure reveals that the GMBS continuously collects the monitored reliability

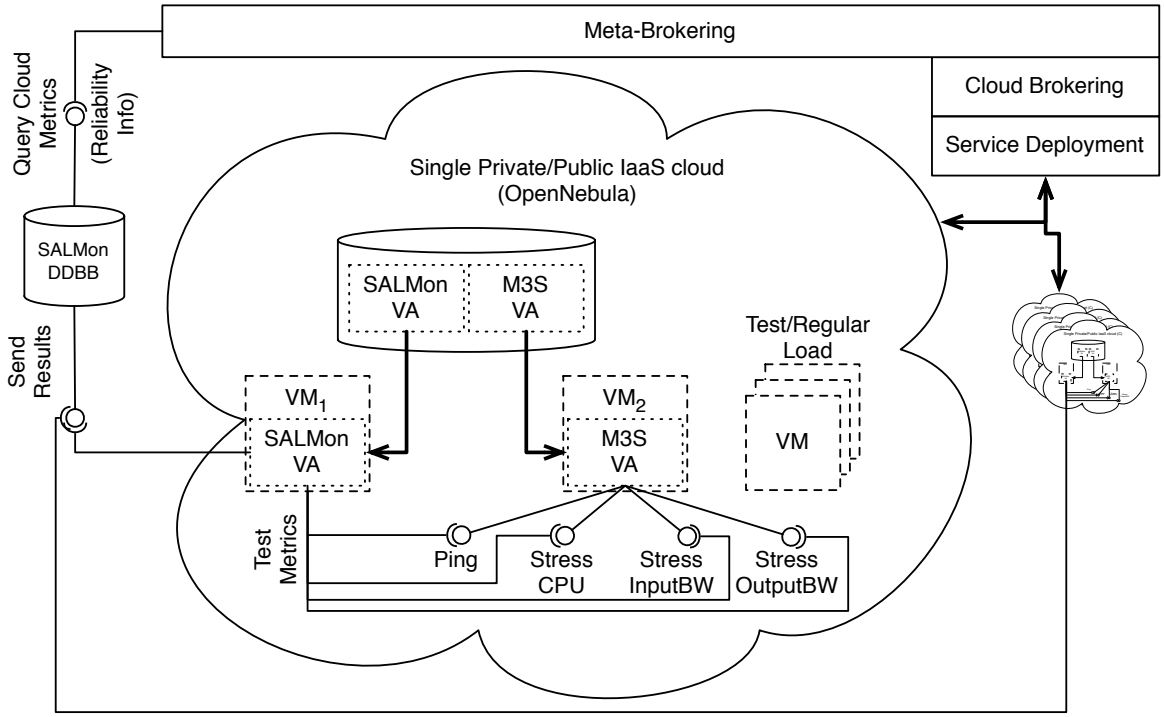

Fig. 3 Integrated monitoring in FCM with M3S and SALMon

information from the SALMon DDBB. If the collected information is due to expire, the GMBS initiates its metric revival phase. As a result, it instantiates a new SALMon and M3S VM in the cloud infrastructure represented with the almost expired data. To do so, GMBS contacts the VM Handler part of the appropriate Cloud-Broker to initiate a new deployment of the monitoring VMs. During this deployment, the new SALMon VM is configured to monitor the methods of the deployed M3S test appliance. After deployment, the GMBS calls the SALMon Configurer method of the new SALMon VM to start monitoring (as shown in Figure 3 with arrows on the right). Consequently, SALMon performs periodic monitoring of the M3S methods using its monitoring test cases obtained from a predefined MMD with the required details, then it reports the metric values and their aggregates (e.g. average or minimum/maximum) to the DDBB. The IS Agent of GMBS (at the meta-brokering layer of FCM) regularly queries the monitored values and updates them in the appropriate description document fields of the responsible Cloud-Broker. Since keeping the monitoring VMs in the cloud can be expensive, we have extended the IS Agent to also initiate the decommission of these VMs after the new metric values become available in the DDBB.

5 Evaluation of our proposed integrated solution

In this section, we present the evaluation of our proposed solution in three phases. First, we show that data collected from an M3S service in a private cloud correlates with the latent load of physical machine that hosts the M3S virtual machine. Second, we present the federative use of SalMon and M3S by collecting and analyzing their metrics from three cloud providers in two cloud federations. Finally, we offer an outlook on how the FCM's monitoring extension could improve user experience with heterogeneous cloud federations.

5.1 Detecting latent load with M3S metrics

The LPDS laboratory of MTA SZTAKI runs an OpenNebula 3.6 [48] based cloud infrastructure [46], which is partitioned in two parts: a production service and an experimental one. Both services use KVM-based virtualization, and support the following interfaces: OCCI, EC2 and the SunStone WEB frontend. Both services are built on hardware with equivalent performance (e.g., the experimental service consists of 4 hosts including 64 CPU cores, having together 152 GBs RAM and 4.3 TBs storage). The only difference between the two is the guarantees they provide. The experimental service is deployed for LPDS cloud developers and enables imme-

diate reconfiguration of the entire infrastructure setup and therefore it does not guarantee properly performing VMs at all times. This reconfiguration capability enables us to perform detailed measurements for testing all M3S functionalities. In this section we present measurements that were taken while the experimental service was under our exclusive control.

5.1.1 Deployment of the monitoring components

To automate the evaluation, we focused our attention on the behavior of the IS Agent component of the GMBS. This component was separated from GMBS for the experiments to reduce the interferences that could possibly be caused by other GMBS/FCM components and the various cloud systems that GMBS connects with. An *independent metric collector script* was developed to manage the instances of SALMon and M3S virtual appliances with the help of the Cloud-Broker. Upon request, this script first instantiates M3S, then SALMon in the target cloud. During the initialization of the SALMon VM, the script instructs SALMon to monitor the VM of the M3S service and forward the results to DDBB. The script then continuously monitors the contents of DDBB and waits until at least one new measurement is available for each of the monitored metrics of the M3S service. After each metric is updated for the target cloud the script ensures the termination of both the M3S and the SALMon VMs. This final step allows minimizing the cost of monitoring, but still maintains recent data in the DDBB to be used during the decisions regarding the target cloud by the GMBS.

In order to determine the usability of the reported metrics, we have checked how these metrics behave under various background load on the experimental private cloud service of LPDS. To imitate the latent load in cloud infrastructures, we have generated the following kinds of artificial load during our measurements: (*i*) with normal load on the hosts of the private cloud , (*ii*) with an increased network load present on the infrastructure and (*iii*) with an increased CPU load present on the hosts of the cloud. Under *normal load conditions* some other developers run several virtual machines for their experiments, but the CPU load never reached over 50% (this load was actually present in all three cases as it represents more closely the production use of a cloud infrastructure). Under *network load situation*, we have introduced continuous transfers on between two physical nodes (a disk image sized 8GBs was transferred over and over again) of our experimental cloud. During the *increased CPU load* scenario, we have deliberately created an under-provisioning situation on one of the physical machines (i.e. we allocated more virtual CPUs on the machine than it had in reality and we also ensured that they run compute-intensive operations).

In both cases with increased load, the preparation of the testing environment is crucial, because too heavy load would render our experiments useless (i.e., it is not expected from any provider to sacrifice its user's operations with such a high level of under-provisioning), in contrast too little load could result no significant changes in our measurements (thus GMBS cannot differentiate between clouds). During network transfer overload, too heavy load may prevent transferring the VA images to the hosts. It is evident that no measurements could be done in these cases, therefore we avoided such transfer loads. Regarding CPU load manipulations, OpenNebula differentiates two parameters we can vary: the 'CPU' – that is reserved by OpenNebula on a physical host for a VM, and the 'vCPU' – that is the number of CPUs a VM can actually utilize. Unfortunately, in VM requests, when one specifies the 'CPU' parameter only, the new VM will end up with an indefinite number of processors. To avoid this unexpected behavior, we have ensured that the Cloud-Broker issues VM requests that specify both parameters (for M3S and SALMon we used the setup of $CPU = 0.01$ – to fit in even heavily under-provisioned environments – and $vCPU = 1$).

Although the scenarios could strongly influence M3S and SALMon behavior, their actual deployment might result in measurement errors. E.g., when the network load is applied it should not occur in an isolated part of the cloud, the network components used by M3S and SALMon should be also influenced. Also, we should avoid deploying SALMon and M3S on the same host as networking between such virtual machines is not comparable to regular networking capabilities. Similarly to the increased network load situation, the increased CPU load should also happen on the host where M3S is deployed (thus influencing its performance metrics). Therefore, our *independent metric collector script* enforced the Cloud-Broker to instantiate the M3S services on one of the nodes that actually experienced the artificial load. While the VM of SALMon has been deployed on another machine, which actually also served as the virtual machine image repository of the private cloud, therefore this host has been used to transfer the M3S VM to the utilised host during deployment. This SALMon VM was instructed to measure the response times (in milliseconds) of the previously defined methods on the M3S VM. With these preparations, our experimental cloud was ready for measurements.

Table 1 Average and median values of the evaluations.

	StressInBW		**StressCPU**		**StressOutBW**		**Ping**	
	Avg (ms)	Med (ms)	Avg (ms)	Med (ms)	Avg (ms)	Med (ms)	Avg (ms)	Med (ms)
Normal	5,97	5	1672,30	1657,5	87,94	68,67	2,91	2,75
Net load	6,39	6,5	1704,42	1659,5	86,47	74	2,89	3
CPU load	17,69	8,5	2319,27	2289,25	121,93	76	2,71	2,5

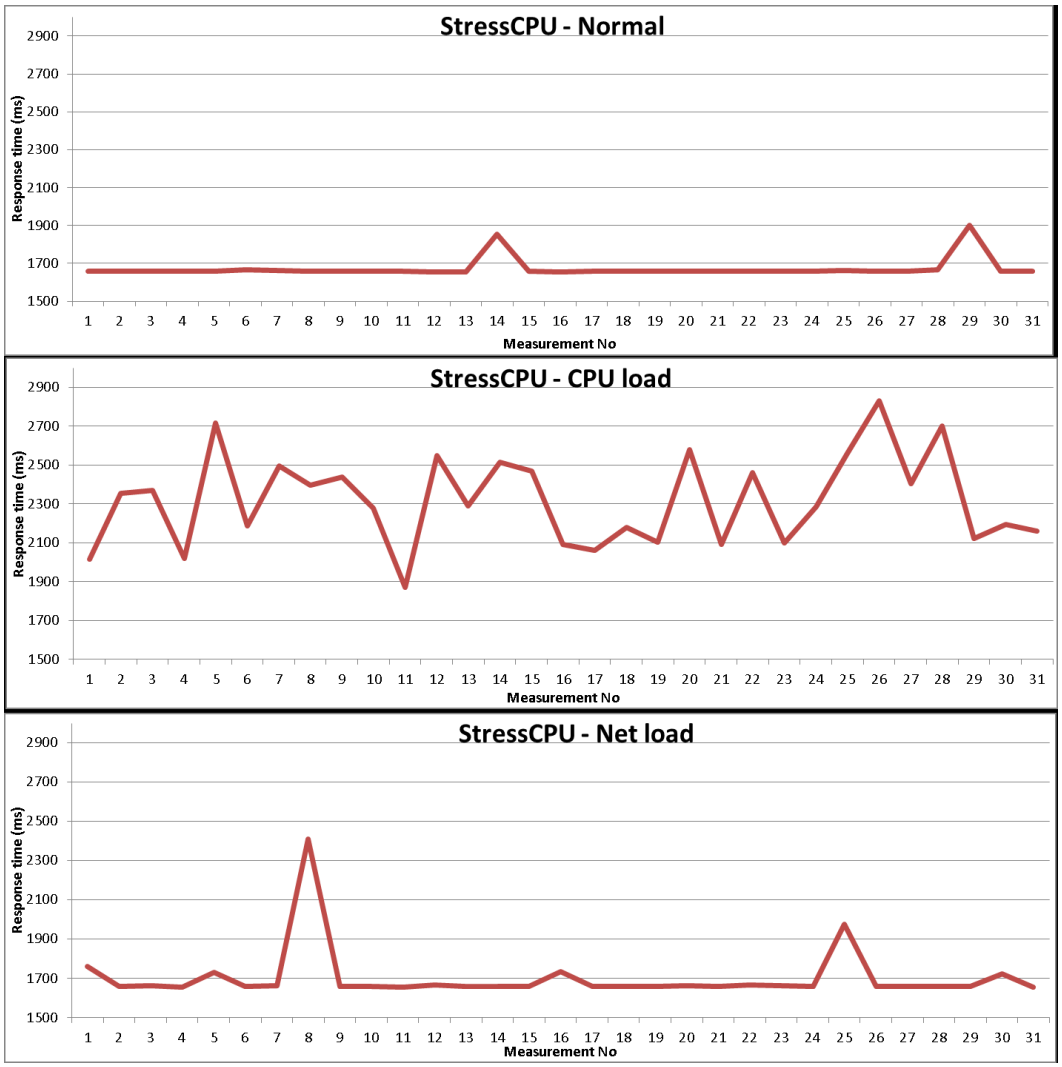

Fig. 4 Evaluation results for CPU intensive tests in the LPDS cloud

5.1.2 Measurements in the LPDS cloud

While running our artificial load setups we have repeatedly run the IS Agent of GMBS until the statistical evaluation of the measurement results became more stable (i.e. we could eliminate the discrepancies because of the continuously present normal load). Besides this paper, the evaluation process is also exemplified through a video available in [52]. In Table 1, we summarize the results with basic statistical measures. The table's columns show the average and median response times of all M3S functions depending on the artificial load applied. The table presents the increased network load situation with its "*Net load*" row, while the row titled "*CPU load*" reveals the data collected during CPU under-provisioning in the infrastructure. The rest of the section gives a detailed discussion on the behavior and properties of the metrics collected on the clouds via M3S.

In Figure 4 we can see that for the `StressCpu()` method of M3S in the first and second phases (ie. normal and increased network load) of an evaluation run we measured around the same CPU response times (that are shown in milliseconds). The figure shows that for

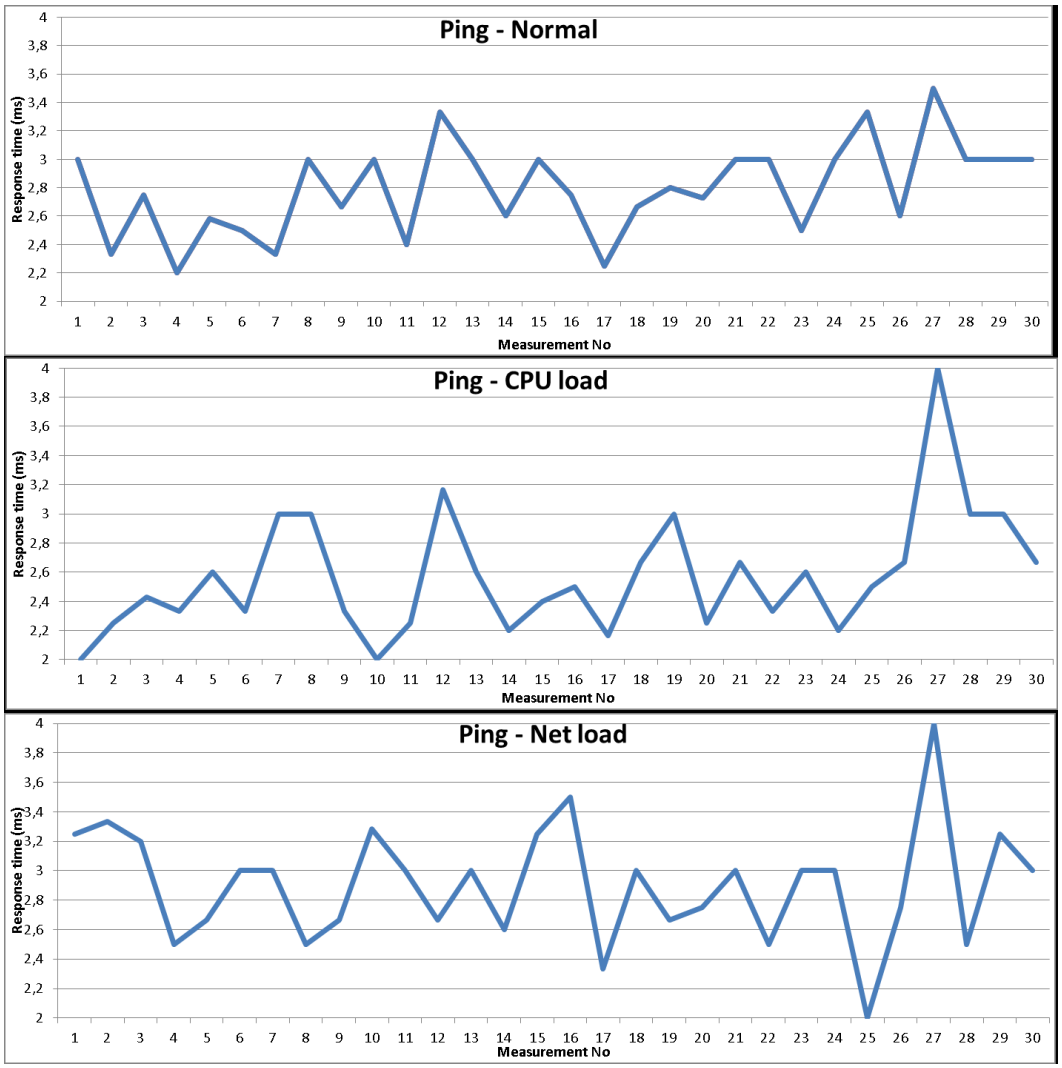

Fig. 5 Evaluation results for service availability tests in the LPDS cloud

a few cases the background load of our experimental cloud was increased for short periods (e.g. Measurement No. 14 under normal load), but otherwise under both load situations the measurements remained stable (their standard deviation is really small – 55ms). In contrast, in the increased CPU load scenario, the M3S service not only responded around 1.5 slower, but the standard deviation of its response times have increased significantly (to 235ms).

Figure 5 shows the response times for the `Ping()` method of M3S during all three artificial loads situations. Due to the applied granularity of measurements the standard deviation of each measurement run is much higher (i.e. even under normal load it is 327ns – 12% of the average measured ping response time). However, even with such diversity in measurement results the standard deviation gives us a hint for under-provisioning situations (e.g. it raises to 397ns and 417ns for network and CPU load situations respectively).

Finally, the way the independent metric collector is created allowed us to also measure and publish the deployment times of the M3S and SALMon services in the particular cloud infrastructure. As stated in Section 3.2, the average deployment time of virtual appliances are reported by the VM Handler of the cloud-brokers to the GSR registry, to help decision making for brokering operations. We have gathered these deployment times measured during the evaluation runs, and summarized them in Table 2. In addition to the previous evaluation runs we executed an additional phase, in which we further increased the network load on the M3S host (there were 10 times as many parallel transfers as before in our regular increased network load situation). This phase is denoted in the last row of this table.

Table 2 Average deployment times of the monitoring appliances.

Evaluation phases	Deployment time (ms)
Normal	200,92
CPU load	228,2
Network load	245,45
Higher network load	379,69

Analysis. As we have expected, the results show that the combined raise in deployment time of the M3S and in response time for the median ping and bandwidth stressing functions could be a good indicator for network under-provisioning. Also, the increase in median CPU load alone could be used to indicate CPU under-provisioning. Unfortunately, our used bandwidth stressing functionalities also dependent on the CPU on some level. This dependency is revealed in the increased responds times of the bandwidth stressing functions during heavy CPU load situations. Therefore, the current set of measurements cannot detect concurrently occurring CPU and network under-provisioning. As new versions of the M3S will be produced this constraint is planned to be removed allowing the GMBS to take more sophisticated decisions. Measuring these metrics in all participant cloud infrastructures of a cloud federation can contribute to a better performing execution environment selection at the upper layers of the FCM architecture, resulting in a higher level of user satisfaction.

What we learned from the first round of measurements performed on the LPDS local cloud is that other running VMs in the cloud infrastructure can cause some performance degradation to a user's application, which is proved by our measurements depicted in Figures 4 and 5. Since cloud providers usually do not give access to information on the total number of running VMs (and their dynamic load) in their datacenters, there is a definite need for a monitoring solution capable of providing such information.

5.2 M3S metrics in the scope of cloud federations

After we finished examining the behavior of M3S in a controlled environment, we have set up larger scale experiments that would support the M3S's integration into the FCM architecture. We aimed at collecting M3S measurements for various cloud systems to enable the FCM architecture to make more informed decisions, while federating them into a single cloud formation. To identify the cloud infrastructures that we could experiment on, we have checked out the various federative partners of the private LPDS cloud. The LPDS cloud is participating in two larger scale cloud federations: one that is formed by the EGI Cloud Federation task force [39], and the other one is formed by the SZTAKI Cloud project [41].

5.2.1 The used infrastructures

First, we have contacted the *EGI Cloud Federation* to determine which of their participants we can use for our experiments. To reveal how M3S handles heterogeneous environments, we have aimed at an EGI site with a different cloud setup as our local experimental cloud. We received positive answer from the CESNET cloud [40]. During our measurements, this cloud had 10 nodes, 2 of which were reserved for EGI FedCloud. The hardware parameters of this cloud were: 24 cores, 96 GB RAM, 1 TB local storage (with RAID 0), and InfiniBand Mellanox MT26428 QDR. Its software stack uses OpenNebula v. 3.6 with nodes having Debian 6 Squeeze, XEN 4.1.2 hypervisor (with tap2 drivers), and GPFS or NFS shared storage (44 TB, mounted IPoIB). Though this cloud is also based on the OpenNebula middleware, it uses a different virtual machine monitor (namely XEN), which required the transformation of the originally KVM based virtual appliances of the M3S and SALMon services. These transformed appliances were registered to the Generic Service Registry of FCM (see Figure 1) to allow seamless deployment independently from the applied virtual machine monitor in the actually monitored cloud infrastructure.

Second, the LPDS Cloud also participates in a larger private federation of SZTAKI formed by the internal project called "*SZTAKI Cloud*". This federation offers a central infrastructure service for SZTAKI that federates with the individually maintained infrastructures of the various research laboratories in SZTAKI. The central service runs a different version of OpenNebula (version 3.8) allowing us to experiment M3S behavior in an even more heterogeneous environment, when the cloud toolset is not equivalent. Therefore we used the SZTAKI Cloud central infrastructure to perform the third series of measurements. This infrastructure consists of 448 CPU cores, 1.75 TBs RAM and 66 TBs storage.

In addition to the examined academic Clouds, we have also performed preliminary evaluations on the Windows Azure platform [51]. However, the performance characteristics of the Azure cloud renders the results practically incomparable, because even deploying a single instance of the M3S service took around 10 minutes. Thus, in the next section we only focused our measurements to FCM-based academic cloud federation.

5.2.2 Measurements

On the previously detailed infrastructures we have also performed the same measurements that we introduced for the LPDS Cloud in the previous subsection. As the artificial CPU and network load could not fit in to our usage quotas, we have only executed measurements under the regular load of the particular cloud. The comparison of the appropriate test cases can be seen in Figure 6. The figure represents the local LPDS cloud with the label "LPDS", the Czech cloud as "CESNET" and the SZTAKI Cloud federation as "SZTAKI".

Analysis. In order to summarize the evaluation results we can state that they are heavily dependent on the hardware characteristics, overall load and utilization, and the policies (SLAs) of the actual cloud provider. For example, the minimum values for the `StressCPU()` measurements reveal the clear performance difference between the individual cores of the clouds (LPDS uses Intel(R) Xeon(R) CPU E5420 @ 2.50GHz CPUs, CESNET uses Intel Xeon E5649@2.53GHz CPUs and SZTAKI uses AMD Opteron(TM) Processor 6272 @2.1GHz CPUs) and their standard deviation reveals the usage pattern of the infrastructure and the signs for under or over-provisioning. This situation can be clearly observed through the SZTAKI Cloud that has several processing power slowdowns (e.g. see measurements around iteration 15). In those cases the CPUs were under-provisioned, and our measurements also reflect this situation.

Based on our deployment time measurements, we managed to show that deploying similar services at different providers in a federation can result in highly different startup times, therefore for highly dynamic and often upgraded service appliances our proposed FCM solution can save significant time for the users by efficient provider selection and VM management with the help of its integrated monitoring solutions.

6 Conclusion and future work

The growing number of user communities in Cloud Computing calls for overextending the boundaries of single cloud systems. Federated clouds aim at supporting these users by providing a single interface on which they can transparently handle the different cloud providers as they would do with a single system. In this paper, we have presented an architecture that offers federated cloud management and utilizes a sophisticated service monitoring approach to evaluate basic cloud reliability status, and to perform seamless service provisioning over multiple cloud providers.

The architecture uses the Generic Meta-Broker Service as the entry point for the users of the cloud federation. This service selects the most suitable cloud provider to perform the service requests of the user by investigating the current state of the participating clouds according to the information stored in a generic service registry and the reliability metrics collected by the integrated SALMon service monitoring framework. We also presented the concept of the Cloud-Broker that is capable of handling service requests and managing virtual machines within a single IaaS cloud system. We have created a minimal metric monitoring service, which is capable of measuring infrastructure reliability together with the integrated SALMon framework in public and private clouds.

Finally, we have evaluated our integrated federated management solution using the minimal metric monitoring service and SALMon to monitor service provisioning reliability in three different private cloud infrastructures. The presented evaluation results show that both service reliability and responsiveness do vary over time and load conditions, and these measures can be used by our federated cloud management solution to select better execution environments for achieving a higher level of user satisfaction.

Our future work aims at applying the proposed approach in other cloud federations including commercial solutions. Since our SZTAKI Cloud project has recently launched its federated Cloud service, now we are able to manage local clouds of different laboratories by taking into account the monitored service performance data. Regarding cloud-brokering, we would like to rely more tightly on the information provided by the monitoring service, e.g., VM deployment and service execution times. Using these and historic information collected about the services, we would like to forecast load and required VM count incorporating e.g., delays caused by VM deployment times and maximum time limit constraints for services calls waiting in the service queue. In these additional evaluations we will further examine the performance of centralized components in FCM (such as the GMBS and GSR), and propose replicated or decentralized versions for better scalability.

7 Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube), and from the SZTAKI Cloud project financed by the Computer and Automation Research Institute, Hungarian Academy of Sciences (MTA SZTAKI).

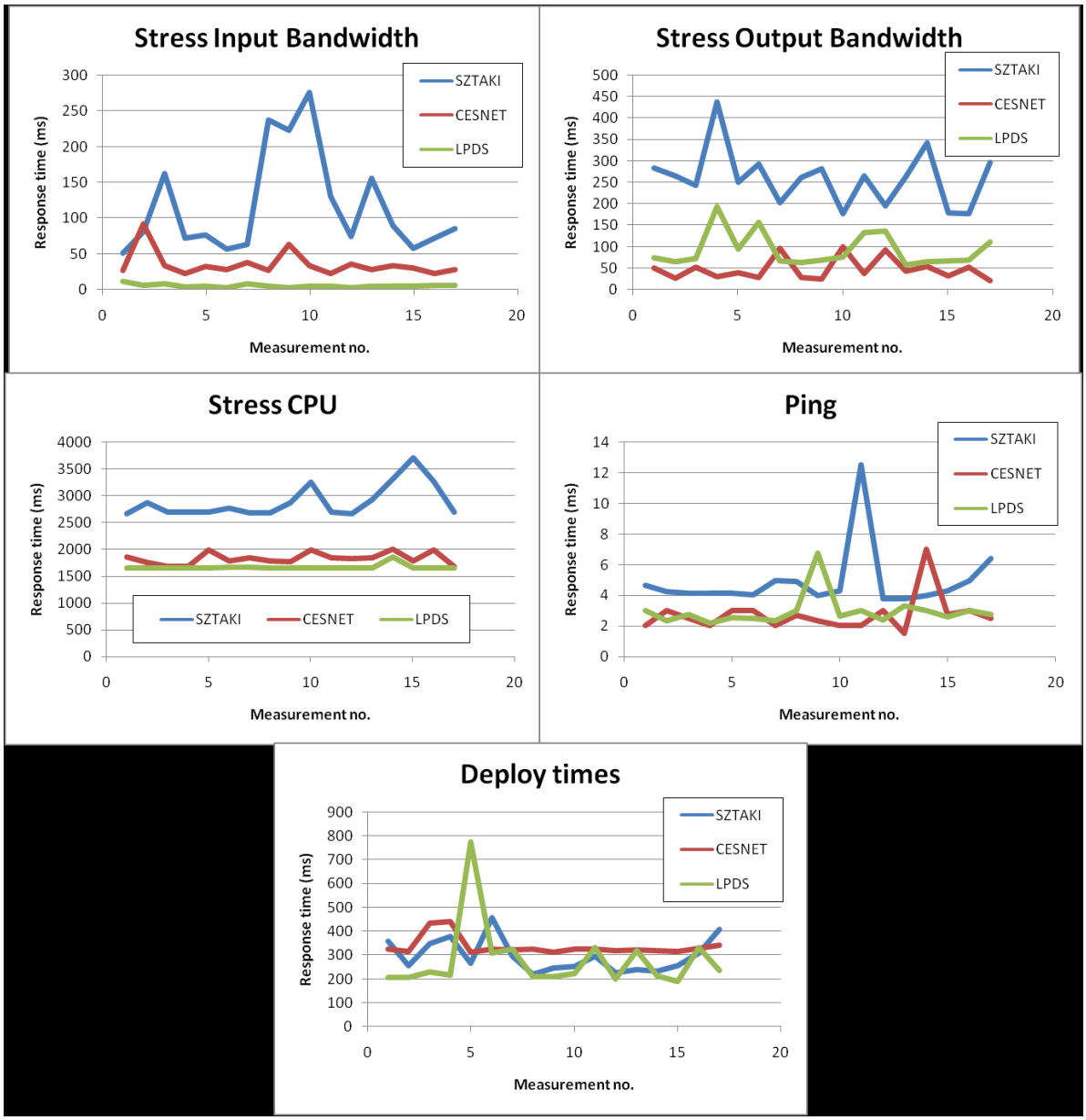

Fig. 6 Comparison figures of measurements in 3 Clouds

References

1. D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond and M. Morrow. Blueprint for the Intercloud – Protocols and Formats for Cloud Computing Interoperability. In Proceedings of The Fourth International Conference on Internet and Web Applications and Services, pp. 328–336, 2008.
2. E. Badidi, L. Esmahi, M. A. Serhani and M. Elkoutbi. WS-QoSM: A Broker-based Architecture for Web Services QoS Management. Innovations in Information Technology, pp. 1–5, 2006.
3. L. Baresi, S. Guinea. Self-supervising BPEL Processes. In IEEE Transactions on Software Engineering, IEEE computer Society Digital Library, 2010.
4. T. Baur, R. Breu, T. Kalman, T. Lindinger, A. Milbert, G. Poghosyan, H.Reiser, M. Romberg. An Interoperable Grid Information System for Integrated Resource Monitoring Based on Virtual Organizations. Journal of Grid Computing, Volume 7, Issue 3, pp. 319–333, September 2009.
5. R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, vol. 25, no. 6, pp. 599–616, June 2009.
6. R. Buyya, R. Ranjan, and R. N. Calheiros. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. Lecture Notes in Computer Science: Algorithms and Architectures for Parallel Processing. Volume 6081, 2010.
7. O. Cabrera and X. Franch. A Quality Model for Analysing Web Service Monitoring Tools. In proc. of the Sixth IEEE International Conference on Research

Challenges in Information Science, RCIS 2012, Valencia, Spain, 16-18 May 2012.
8. E. Carlini, M. Coppola, P. Dazzi, L. Ricci, and G. Righetti. Cloud Federations in Contrail. Euro-Par 2011 Workshops, LNCS 7155, pp. 159168, 2012.
9. Celesti, A., Tusa, F., Villari, M., and Puliafito, A. (2010). How to Enhance Cloud Architectures to Enable Cross-Federation. In IEEE 3rd Conference on Cloud Computing (CLOUD). 2010.
10. A. Cuomo, G. Di Modica, S. Distefano, A. Puliafito, M. Rak, O. Tomarchio, S. Venticinque, U. Villano. An SLA-based Broker for Cloud Infrastructures. Journal of Grid Computing, Volume 11, Issue 1, pp. 1–25, March 2013.
11. E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive servicebased applications. Automated Software Engg., vol. 15, pp. 313–341, December 2008.
12. R. R. Exposito, G. L. Taboada, S. Ramos, J. Gonzalez-Dominguez, J. Tourino, R. Doallo. Analysis of I/O Performance on an Amazon EC2 Cluster Compute and High I/O Platform. Journal of Grid Computing, Online First, March 2013. DOI: 10.1007/s10723-013-9250-y.
13. A. J. Ferrer et. al. OPTIMIS: a Holistic Approach to Cloud Service Provisioning. Future Generation Computer Systems, vol. 28, pp. 66–77, 2012.
14. G. Kecskemeti, G. Terstyanszky, P. Kacsuk, and Zs. Nemeth. An Approach for Virtual Appliance Distribution for Service Deployment. Future Generation Computer Systems, vol. 27, issue 3, pp 280–289, 2011.
15. G. Kecskemeti, G. Terstyanszky, P. Kacsuk and Zs. Nemeth. Towards Efficient Virtual Appliance Delivery with Minimal Manageable Virtual Appliances. *to appear* in IEEE Transactions on Services Computing, DOI: 10.1109/TSC.2013.12.
16. A. Keller and H. Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. Journal of Network and Systems Management, 11(1), pp. 57–81, 2003.
17. A. Kertesz and P. Kacsuk. GMBS: A new middleware service for making grids interoperable. Future Gener. Comput. Syst., vol. 26, pp. 542–553, April 2010.
18. Z. Li, Y. Jin and J. Han. A runtime monitoring and validation framework for web service interactions. In proc. of Australian Software Engineering Conference, 2006.
19. A. Cs. Marosi and P. Kacsuk. Workers in the clouds. In PDP2011, Y. Cotronis, M. Danelutto, and G. A. Papadopoulos, Eds. IEEE Computer Society, pp. 519–26, 2011.
20. A. Cs. Marosi, G. Kecskemeti, A. Kertesz, P. Kacsuk. FCM: an Architecture for Integrating IaaS Cloud Systems. In proc. of the Second International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2011), IARIA, pp. 7-12, Rome, Italy, 2011.
21. J. Montes, A. Sanchez, B. Memishi, M. Perez, G. Antoniu. GMonE: A complete approach to cloud monitoring. Future Generation Computer Systems, In Press, Corrected Proof, Available online 5 March 2013.
22. H. R. Motahari-Nezhad, R. Saint-Paul, B. Benatallah, and F. Casati. Deriving protocol models from imperfect service conversation logs. IEEE Transactions on Knowledge and Data Engineering (TKDE), 2008.
23. B. P. Rimal, A. Jukan, D. Katsaros, Y. Goeleven. Architectural Requirements for Cloud Computing Systems: An Enterprise Cloud Approach. Journal of Grid Computing, Volume 9, Issue 1, pp. 3–26, March 2011.
24. C. Muller, M. Oriol, M. Rodriguez, X. Franch, J. Marco, M. Resinas and A. Ruiz-Cortes. SALMonADA: A platform for Monitoring and Explaining Violations of WS-Agreement-compliant Documents. In proc. of the 4th International Workshop on Principles of Engineering Service-Oriented Systems (PESOS'12), 2012.
25. M. Oriol, X. Franch, J. Marco, D. Ameller. Monitoring adaptable soa-systems using salmon. In Workshop on Service Monitoring, Adaptation and Beyond (Mona+). pp. 19–28, 2008.
26. P. Marshall, K. Keahey and T. Freeman. Elastic Site: Using Clouds to Elastically Extend Site Resources. T. IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2010), Melbourne, Australia. May 2010.
27. D. Petcu, C. Craciun, M. Neagul, M. Rak, I. Lazcanotegui. Building an Interoperability API for Sky Computing. In proc. of the Second International Workshop on Cloud Computing Interoperability and Services (InterCloud 2011), IEEE CS, pp. 405-412, 2011.
28. B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin. I. Lloriente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda W. Emmerich, F. Galan. The RESERVOIR Model and Architecture for Open Federated Cloud Computing. IBM Journal of Research and Development, 53(4), 2009.
29. B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Munoz and G. Toffetti. Reservoir - When One Cloud is not enough. Computer, vol. 44, i. 3, pp. 44-51, 2011.
30. M. Schmidt, N. Fallenbeck, M. Smith, and B. Freisleben. Efficient distribution of virtual machines for cloud computing. In Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, IEEE Computer Society, pp. 567–574, 2010.
31. M. Silberstein, A. Sharov, D. Geiger, and A. Schuster. GridBot, execution of bags of tasks in multiple grids. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09), 2009.
32. B. Sotomayor, R.S. Montero, I.M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. Internet Computing, vol. 13, no. 5, pp. 14–22, IEEE, 2009.
33. H. Truong, T. Fahringer, S. Dustdar. Dynamic Instrumentation, Performance Monitoring and Analysis of Grid Scientific Workflows Jounal of Grid Computing (JOGC), vol. 3, pp. 1-18, 2005.
34. Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commun. Rev. 39, 1, pp. 50–55, 2008.
35. X. Wang; H. Wang, Y. Wang. A Monitoring Framework for Multi-Cluster Environment Using Enterprise Service Bus. International Conference on Management and Service Science, 2009.
36. N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann. C-Meter: A Framework for Performance Analysis of Computing Clouds. In the International Workshop on Cloud Computing (Cloud 2009), 2009.
37. P. Zhang, B. Li, H. Muccini and M. Sun. An Approach to Monitor Scenario-Based Temporal Properties in Web Service Compositions. In Advanced Web and Network Technologies, and Applications, 2008.

38. C. Zhou, L. T. Chia and B. S. Lee. DAML-QoS ontology for web services. In IEEE International Conference on Web Services, pp. 472–479, 2004.
39. EGI Federated Clouds Task Force. https://wiki.egi.eu/wiki/Fedcloud-tf:FederatedCloudsTaskForce, 2012.
40. CESNET Czech academic network operator. http://www.ces.net/about/, 2012.
41. SZTAKI Cloud. http://cloud.sztaki.hu/en/home, 2012.
42. Amazon CloudWatch. http://aws.amazon.com/cloud-watch/, 2009.
43. Amazon Web Services LLC. Amazon elastic compute cloud. http://aws.amazon.com/ec2/, 2009.
44. Cerebrata Azure Diagnostics Manager. http://www.cere-brata.com/Products/AzureDiagnosticsManager, 2011.
45. Eucalyptus cloud. http://www.eucalyptus.com/, 2011.
46. LPDS laboratory website. http://www.lpds.sztaki.hu, 2012.
47. Nagios XI monitoring solution. http://www.nagios.com/products/nagiosxi/, 2012.
48. OpenNebula cloud. http://opennebula.org/, 2011.
49. Rackspace Cloud. http://www.rackspace.com/cloud/, 2011.
50. The World Wide Web Consortium. http://www.w3.org/TR/wsdl, 2009.
51. Windows Azure Platform. http://www.windowsazure.com, 2012.
52. Video demonstration of the monitoring capability integrated to FCM. http://www.youtube.com/watch?v=uIewqw_FJQc, April 2013.