Integrated Monitoring Approach for Seamless Service Provisioning in Federated Clouds

A. Kertesz, G. Kecskemeti, A. Marosi
Computer and Automation Research Institute
MTA SZTAKI
H-1518 Budapest, P.O. Box 63, Hungary
{*keratt, kecskemeti, atisu*}*@sztaki.hu*

M. Oriol, X. Franch, J. Marco
Universitat Politecnica de Catalunya
ESSI – UPC
08034 Barcelona, c/Jordi Girona 1-3, Spain
{*moriol, jmarco*}*@lsi.upc.edu*
franch@essi.upc.edu

***Abstract*—Cloud Computing offers simple and cost effective outsourcing in dynamic service environments, and allows the construction of service-based applications using virtualization. By aggregating the capabilities of various IaaS cloud providers, federated clouds can be built. Managing such a distributed, heterogeneous environment requires sophisticated interoperation of adaptive coordinating components. In this paper we introduce an integrated federated management and monitoring approach that enables autonomous service provisioning in federated clouds. In this architecture, cloud brokers manage the number and the location of the utilized virtual machines for the received service requests. In order to provide seamless service executions, a state of the art monitoring solution is proposed that supports cloud selection performed by the management layer of the architecture. Our solution is able to cope with highly dynamic service executions by federating heterogeneous cloud infrastructures in a transparent and autonomous manner.**

***Keywords*-Cloud Computing; Service Monitoring; Cloud Brokering; On-demand deployment;**

I. Introduction

Cloud Computing [1] 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 [3] 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 applications) 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 public and private IaaS systems co-exist provided by public service providers (like Amazon [4] or RackSpace [5]) or by smaller scale privately managed infrastructures. Cloud solutions are also spreading fast in the academia with the emerging open-source tools, such as Eucalyptus [6] and OpenNebula [7].

However, user demands are frequently overextending the boundaries of a single cloud system. In these cases, they need to handle the differences between the 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 the different cloud providers as they would do with a single cloud system. This paper proposes an architecture to construct federated cloud systems that not only offers a single interface for its users but it automatically manages their virtual machines independently from the currently applied cloud system. We argue that efficient cloud selection in federated clouds requires a cloud monitoring subsystem that determines the actual health status of the available IaaS systems.

We present an architecture that incorporates the concepts of on-demand service deployment, cloud brokering and meta-brokering, supported by an integrated monitoring solution. The meta-brokering component allows the system to interconnect the 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.

Related works have identified several shortcomings in the current cloud infrastructures [2]: e.g. federated clouds face the issue of scalability, self-management and losing

complete control on computing costs. Our solution aims at these problems by allowing users to utilize meta-brokering between public, academic and private cloud systems as a result lowering their operation costs. Our architecture serves as an entry point to this cloud federation by providing transparent service execution for users. The following challenges are of great importance for such a mediator solution: varying load of user requests, enabling virtualized management of applications, establishing interoperability, minimizing cloud usage costs and enhancing provider selection. Therefore the main contributions of this paper are: (i) an advanced, integrated monitoring solution to support meta-brokering decisions for user requirement-based service provisioning, and (ii) a holistic view of interoperable federated clouds managed by a multi-level resource management architecture.

This paper is organized as follows: first, we gather related approaches in Section II. In Section III we introduce our proposed architecture and discuss its main components in three subsections. In Section IV, we present a simplified scenario that we use to exemplify our approach. Finally, we conclude our research in Section V.

II. Related work

In 2009, Amazon Web Services launched Amazon CloudWatch [8], 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 is that this solution is tied to a specific IaaS cloud system and introduces a monetary overhead, since the service charges a fixed hourly rate for each monitored instance.

Yigitbasi et. al. [15] introduced a solution for cloud performance monitoring 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 necessary require additional deployments.

Regarding federated management of different infrastructures, GridBot [10] represents an approach for execution of bags-of-tasks on multiple grids, clusters, and volunteer computing 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.

M. Schmidt et al. [9] 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 (”cross-cloud”) they choose BitTorrent. They also propose to use layered virtual machine images for virtual appliances consisting of three layers: user, vendor and base. By using the layers and a copy-on-write method they were able to avoid the retransmission of images already present at the destination and thus decrease instantiation time and network utilization. The authors only investigated distribution methods within the boundaries of a single data center, going beyond that remained future work.

With respect to monitoring the provisioned services, the existing technical approaches found in the literature to gather the required data can be classified into two big categories. On one hand, some proposals rely on the use of monitoring directives as follows: Introducing monitoring directives into the services themselves using Aspect Oriented Programming (AOP), and weaving the monitoring code into the execution process, which is commonly defined in BPEL [16], [17]. 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. [18] 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 [19], [20]. 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.

III. Integrated monitoring approach for service provisioning in clouds

Figure 1 shows the architecture of the Integrated Monitoring Approach for Seamless Service Provisioning (IMA4SSP). 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-broker*” component. This component is capable to decide

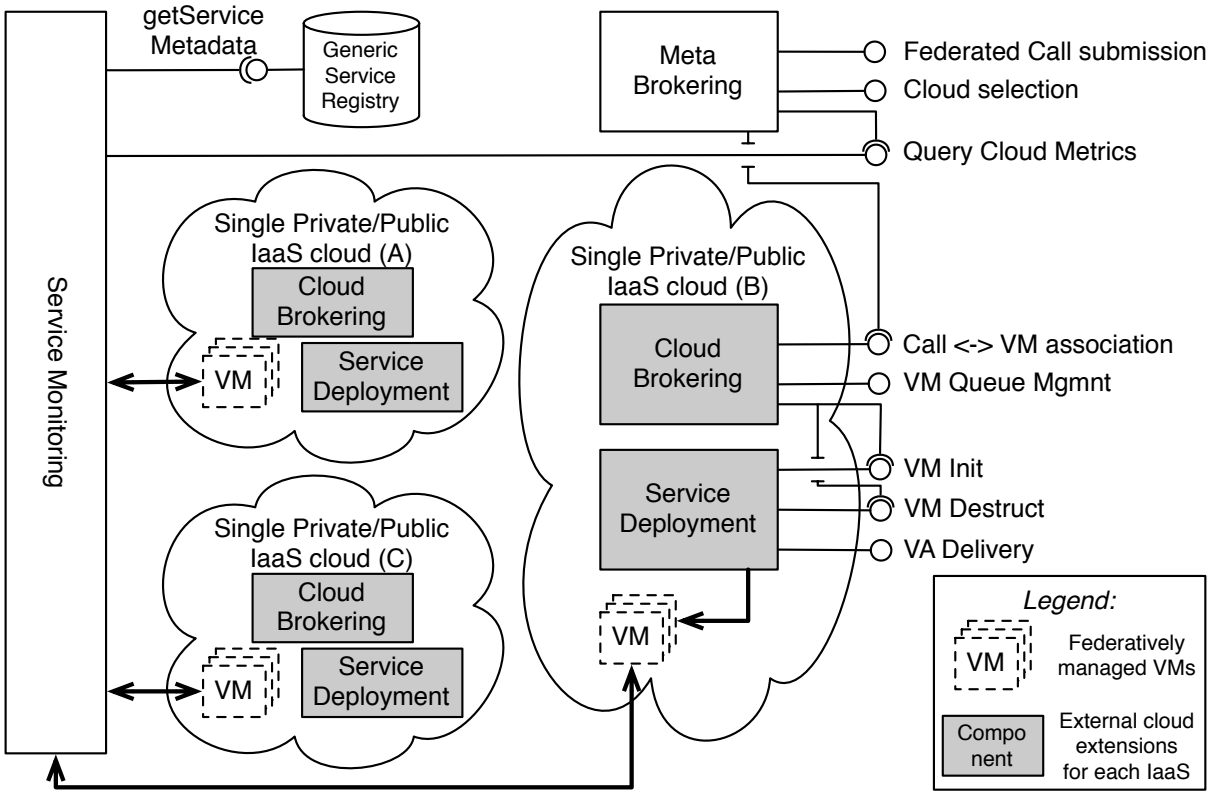

Figure 1. The IMA4SSP architecture

between the use of various "*cloud brokers*" based on metrics gathered from our "*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.

We restrict our solution to support standard and stateless web services described by WSDLs [11]. 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 on these services (including WSDLs and their virtual machine images or *virtual appliances* – VA). When a service is deployed on a new host, the service deployment component registers its new endpoint to the GSR. Upon decommissioning, these endpoint registrations are removed from the registry. During operation, the *SALMon* [14] monitoring subsystem allows the components in IMA4SSP to order regular testing on the deployed services according to pre-defined metrics based on the service availability data from the registry.

In our system, users send service calls as request submissions to the *Meta-Brokering* component (later realized by Generic Meta-Broker Service – GMBS). "*Federated call submissions*" specify the requested service with a WSDL, 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 *CloudBroker* (CB) for further submission, otherwise rejects the request. Based on service usage patterns (e.g. average service response time, call frequency) the GMBS orders the monitoring of the deployed service from SALMon. The monitoring results allow sophisticated matchmaking algorithms based on static data gathered from the GSR and on dynamic information of special metrics (referred as "*query cloud metrics*" in Figure 1) gathered by SALMon and the cloud brokers. GMBS forms a cloud federation by enabling the autonomous management of the interconnected cloud infrastructures through cloud brokers.

CloudBrokers are dedicated to specific IaaS systems and offer a queue for incoming service calls. They also manage one virtual machine queue for each virtual appliance. Virtual machine queues represent the resources that currently can serve a specific service call. VM queues allow CBs to schedule members of the incoming service queue to specific virtual machines ("*Call* $\Leftrightarrow$ *VM Association*"). The main goal of CB is to "*manage the virtual machine queues*" (instantiate and destruct them using service deployment – see Figure 1) according to their respective service demand. The default

virtual machine scheduling is based on the currently available requests in the incoming service queue, their historical execution times, and the number of running VMs. The secondary task of CB involves the dynamic creation and destruction of the various queues. In the following subsections we detail the main components of the architecture.

A. Meta-brokering approach for interoperating clouds

As we already mentioned in the beginning of this section, brokering takes place at two levels in this architecture: the service call is first submitted to a meta-brokering component implemented and names as the Generic Meta-Broker Service (GMBS – which is a revised and extended version of the Grid Meta-Broker Service described in [12]), where a high-level decision is made to which cloud infrastructure the call should be forwarded. Then the service call is placed in the queue of the selected cloud broker, where a lower level brokering is carried out to select the VM that performs the actual service execution.

Now, let us turn our attention to the role of GMBS. 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 broker. This decision making is based on aggregated static and dynamic data stored by the Information Collector (IC) component in a local database. The Information System (IS) Agent is implemented as a listener service of GMBS, and it is responsible for regularly updating static information from the GSR repository on service availability, dynamic information on service and cloud reliability provided by SALMon (further discussed in Section III-C), and aggregated dynamic information collected from the CloudBrokers (CB) including average VA deployment- and service execution time. The Invoker component forwards the service call to the selected CB and receives the service response.

Each CB is described by an XML-based Broker Property Description Language (BPDL) document containing basic broker properties (e.g. name), and the gathered dynamic properties. The scheduling-related attributes are typically stored in the PerformanceMetrics field of BPDL. More information on this document format can be read in [12]. Namely, the following data are stored in the BPDL of each CB:

- Static availability information on specific virtual appliances in native repositories collected from the GSR;
- average VA deployment time and average service execution time for each VA queried from the cloud brokers;
- and dynamic reliability information expressed by metrics collected from SALMon.

The scheduling process first filters the CBs by checking VA availability in the native cloud repository, then a rank is calculated for each broker based on the collected dynamic data. Finally, the CB with the highest rank is selected for forwarding the service call.

B. Cloud brokering and automatic service deployment

The CloudBroker, which is an extended version of the system described in [13], handles and dispatches service calls 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 GSR 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 VA. New resource requests are inserted to the queue of the appropriate VA, while the need for resource destruction is indicated by the shortening of 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 to the IaaS system.

The service call queue stores incoming service requests and a reference in the GSR to a VA for each request. There is a single service call queue in each CloudBroker, while there are many VM queues. Dynamic requirements for the VA may be specified with the service call: additional resources (CPU, memory and disk), and an UUID to identify service calls originating from the same requestor. If dynamic requirements are present, then the CloudBroker 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 CloudBroker 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 CloudBroker also schedules service call requests to VM's and manages the VM lifecycle. If a service call cannot be associated to any VM, the CloudBroker may decide to start a new VM to serve the request. The VM creation and destruction decisions are based on the following:

- The number of running VM's available to handle the service call;
- the number of waiting service calls for the VA in the service call queue;
- the average execution time of service calls;
- the average deployment time of VA's;
- and SLA constraints (e.g. total budget, deadline);
- the billing period of the IaaS system.

If a destruction is needed, shutdown is performed shortly before the end of the billing period with regard to the average

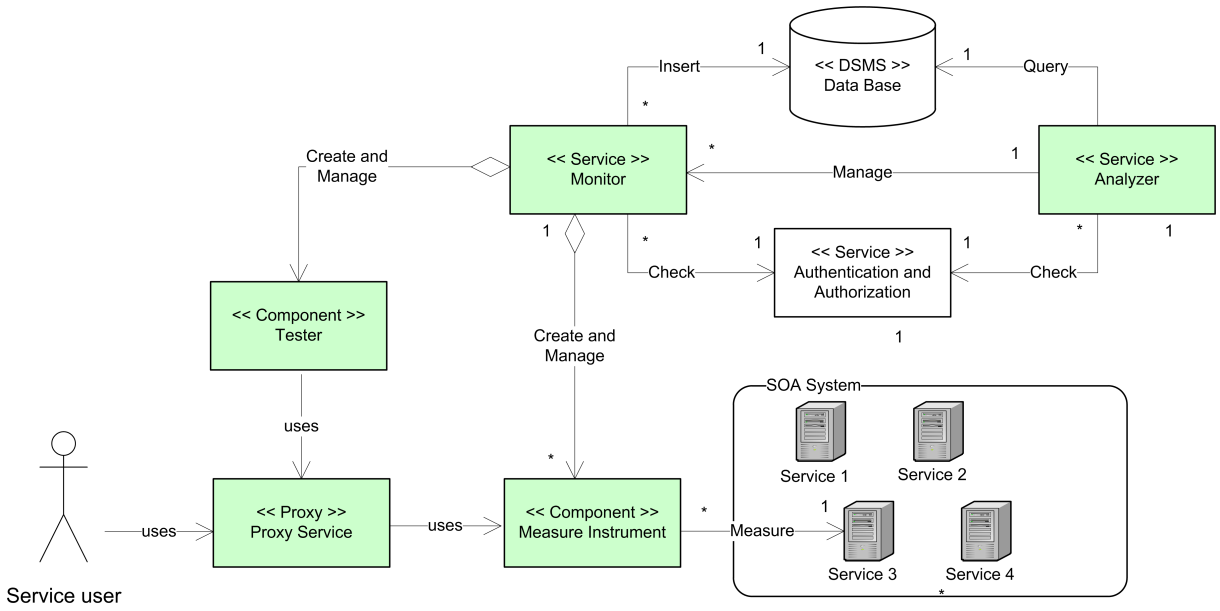

Figure 2. SALMon framework

decommission time in the IaaS.

IaaS systems require virtual appliances to be stored in their native repositories, because only the previously stored appliances are usable to instantiate virtual machines. The architecture organizes the distribution of user created appliances with the help of the Automatic Service Deployment (ASD – [23]) component. To optimize service executions in highly dynamic service environments, the system organizes virtual appliance distribution by automatically decomposing and replicating appliances. To support the rebuilding of decomposed VAs, the ASD requires appliances to embed minimal manageable virtual appliances (MMVA). 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 current state of its instances (e.g. provide access to their CPU load, free disk space and network usage);
- And, it is optimally sized: only those files present in the appliance that are required to offer the previously two properties.

As a result, the ASD only replicates MMVAs to native repositories. Then, the VM Handler controls virtual appliance rebuilding using minimal manageable virtual appliances. Consequently, the VM Handler applies the following strategy when it instantiates a virtual appliance that is not available in the native repository. First, it instantiates the MMVA. Then, the it requests the MMVA's content management interfaces to download the appliance parts – not present in the native repository – from the GSR. Therefore, the appliance is rebuilt in the virtual machine instantiated for the MMVA. Finally, the VM is ready to serve the scheduled requests from the service call queue.

C. Enhanced service health monitoring with SALMon

SALMon [14] is a service monitoring framework that has been integrated into our proposed 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 accordingly to stated conditions and notify the results to the interested parties, which in this case is the Information System (IS) agent from the GMBS.

One of the main characteristics of SALMon is that it combines both passive monitoring and testing approaches, being able to configure each method accordingly to the preferences of the user. In our integrated monitoring solution, 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 calculate the QoS over the obtained results. Another important characteristic is that the architecture is able to support any kind of service technology (e.g. SOAP-based web services, RESTFul, etc.), which in the context of the heterogeneity of the cloud is an important aspect to address. The framework has been implemented as a Service-Based Application itself, hence it enables an easy integration with other frameworks. SALMon provides the following two services: the Monitor, responsible to retrieve the value of the required quality metrics of the services; and the Analyzer, which is in charge of the evaluation of conditions over these metrics. These services have the required capabilities in order to monitor services running at different cloud infrastructure providers.

In our IMA4SSP solution, only the monitor service of SALMon is used in order to provide run-time values of the dynamic QoS. To offer the required support for any kind of service, the monitor manages several measure instruments. A Measure Instrument is a component that implements the logic needed in order to obtain the value of a concrete basic quality metric (e.g., Current Response Time, Current Availability, Accuracy of a service or operation. Derived quality metrics are calculated from the set of basic quality metrics retrieved from the measure instruments using an aggregator function in a defined time interval (maximum, minimum, average). 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. In this sense, the Monitor service stays above the technological details, and just creates the different measure instruments to obtain the QoS.

The architecture of SALMon is depicted in Figure 2. As stated, the main artifacts are the Analyzer and the Monitor Service. The Analyzer service makes usage of the Monitor service to obtain the QoS and evaluate the satisfaction of conditions, whereas the Monitor creates and manages several Measure Instruments to actually gather the QoS. In the passive monitoring approach, instead of invoking the services directly, the user would invoke a proxy that have these Measure Instruments deployed. In the testing approach, the Monitor creates and manages the Tester component, which is responsible to invoke the service periodically. These invocations are performed through the same proxy and using hence the same measure instruments to retrieve the QoS.

IV. Simplified usage scenario of IMA4SSP

As we discussed in the previous subsection, SALMon is capable of monitoring services running at various cloud infrastructure providers. The monitoring target can be defined using specific monitoring metrics. In order to exemplify the operation of our proposed IMA4SSP solution, we describe a simplified usage scenario shown in Figure 3 that we have set up as a proof of concept installation. In this scenario SALMon uses a special test service called Minimal Metric Monitoring Service (M3S) for monitoring cloud reliability, instead of monitoring different service methods stored in the Generic Service Registry. This reference M3S test service has two methods representing three monitoring metrics:

1) Availability: a generalized ping test (e.g. getting the WSDL of the test service) this shows if the service is up and running;
2) Computational capability: measured by a compute method that performs a 5 minute-operation (the result is normalized compared to a reference hardware setup) the response time of this method represents the computational speed of the cloud;

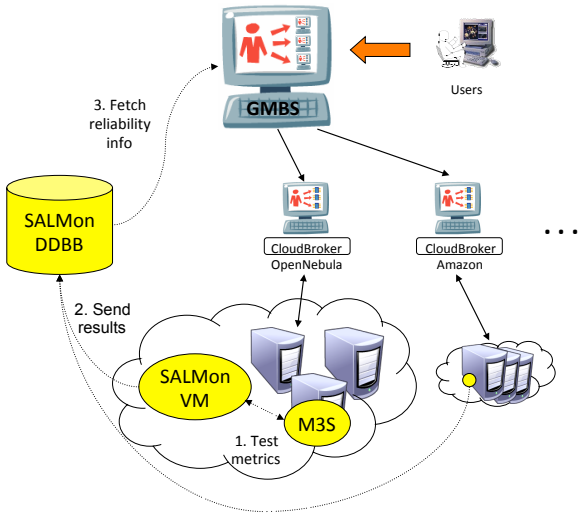

Figure 3. Simplified usage scenario of IMA4SSP

3) Data transfer capability: measured by a transfer method that uploads and downloads a 10 MB file to a predefined public storage location the response time of this method shows the transfer speed of the cloud.

This monitoring test service is prepared in advance, and pre-deployed on the managed IaaS providers. Meanwhile SALMon is configured to monitor the methods of the deployed M3S test services, and the monitoring test cases are set in a way that minimum, maximum, average and latest metric values can be gathered and fetched by GMBS. In a real world private cloud infrastructure we experienced that service methods of the deployed VAs are not always reachable from outside the cloud (though some providers make it available on request). Therefore we separated the monitoring and data management components of SALMon, and created a VM from the monitoring part and a public service from the data management part. For seamless operation, we place the SALMon VM inside the cloud and perform the interactions with the M3S there.

As shown in Figure 3, during operation SALMon performs the monitoring of the M3S methods continuously (Step 1) in an IaaS cloud, and reports the monitored metric values to a central database (Step 2). The IS Agent of GMBS regularly gets the monitored values and updates them in the appropriate BPDL fields of the responsible CloudBroker (Step 3).

Since keeping the monitoring VMs in the cloud can be costly, we have extended the IS Agent component of GMBS to initiate the deployment and decommission of these VMs in order to minimize monitoring costs. The monitored metric values reported to the DDBB have timestamps, therefore these data become outdated after a predefined time interval. When the IS Agent finds that the retrieved metric value of a cloud is outdated, it contacts the VM Handler part of the

responsible CloudBroker, and initiates a M3S VM than a SALMon VM deployment, and calls the appropriate method of the deployed VM to start monitoring. When metric values with new timestamps are read from the DDBB, the IS Agent contacts the VM Handler again, to decommission the monitoring VMs.

V. Conclusion

In this paper, we have presented an architecture that offered federated cloud management and utilized a sophisticated service monitoring approach to evaluate basic cloud health status. The architecture uses the Generic Meta-Broker Service as the entry point for the users of the cloud federation. The GMBS service decides the most suitable cloud to perform the service requests of the user by investigating the current state of the clouds according to the Generic Service Registry and the health metrics collected by the SALMon service monitoring subsystem. We also presented the concept of the CloudBroker that is capable of handling service requests and managing virtual machines within a single IaaS cloud system. Finally, we discussed a simplified scenario for exemplifying the operation of our proposed solution using a minimal metric monitoring service.

Our future work targets the evaluation of the IMA4SSP architecture with real user scenarios including ordinary services deployed at different cloud providers. If ordinary services are supported, the architecture should also take into consideration the possible expenses of monitoring. Therefore, we plan to investigate approaches for metric collection for services with long call processing times. Finally, health metrics on ordinary services will also enable better decisions in the lower level components of the architecture. Consequently, we will also explore the integration options of the monitoring system to the CloudBroker and Automatic Service Deployment components of the architecture.

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 SCI-BUS FP7 project under grant agreement 283481.

References

[1] 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.

[2] 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.

[3] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. Automated Software Engg., vol. 15, pp. 313–341, December 2008.

[4] Amazon Web Services LLC. Amazon elastic compute cloud. http://aws.amazon.com/ec2/, 2009.

[5] Rackspace Cloud. http://www.rackspace.com/cloud/, 2011.

[6] Eucalyptus cloud. http://www.eucalyptus.com/, 2011.

[7] OpenNebula cloud. http://opennebula.org/, 2011.

[8] Amazon CloudWatch. http://aws.amazon.com/cloudwatch/, 2009.

[9] 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.

[10] 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.

[11] The World Wide Web Consortium. http://www.w3.org/TR/wsdl, 2009.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] L. Baresi, S. Guinea. Self-supervising BPEL Processes. In IEEE Transactions on Software Engineering, IEEE computer Society Digital Library, 2010.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] D. Devaurs, K. Musaraj, F. De Marchi, and M. Hacid. Timed Transition Discovery from Web Service conversation Logs. In 20th International Conference on Advanced Information Systems Engineering (CAISE'08), 2008.

[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] G. Kecskemeti, G. Terstyanszky, P. Kacsuk, and Zs. Nemeth. An Approach for Virtual Appliance Distribution for Service Deployment. Future Generation Computer Systems, 2011, vol. 27, issue 3, pp 280–289.