Facilitating self-adaptable Inter-Cloud management

G. Kecskemeti*, M. Maurer†, I. Brandic†, A. Kertesz*, Zs. Nemeth*, and S. Dustdar†
*MTA SZTAKI Computer and Automation Research Institute
H-1518 Budapest, P.O. Box 63, Hungary
{kecskemeti, keratt, zsnemeth}@sztaki.hu
†Distributed Systems Group
1040 Vienna, Argentinierstr. 8/181-1, Austria
{ivona, maurer, dustdar}@infosys.tuwien.ac.at

***Abstract*—Cloud Computing infrastructures have been developed as individual islands, and mostly proprietary solutions so far. However, as more and more infrastructure providers apply the technology, users face the inevitable question of using multiple infrastructures in parallel. Federated cloud management systems offer a simplified use of these infrastructures by hiding their proprietary solutions. As the infrastructure becomes more complex underneath these systems, the situations (like system failures, handling of load peaks and slopes) that users cannot easily handle, occur more and more frequently. Therefore, federations need to manage these situations autonomously without user interactions. This paper introduces a methodology to autonomously operate cloud federations by controlling their behavior with the help of knowledge management systems. Such systems do not only suggest reactive actions to comply with established Service Level Agreements (SLA) between provider and consumer, but they also find a balance between the fulfillment of established SLAs and resource consumption. The paper adopts rule-based techniques as its knowledge management solution and provides an extensible rule set for federated clouds built on top of multiple infrastructures.**

***Keywords*-Cloud Computing; Knowledge Management; Autonomous; Federations; Infrastructure as a Service**

I. Introduction

Cloud Computing represents a novel computing paradigm where computing resources are provided on demand following the rules established in form of Service Level Agreements (SLAs). SLAs represent the popular format for the establishment of electronic contracts between consumers and providers stating the terms of use, objectives and penalties to be paid in case objectives are violated. Thus, appropriate management of Cloud Computing infrastructures [7], [8], [10], [11] is the key issue for the success of Cloud Computing as the next generation ICT infrastructure [1], [2], [3]. Thereby, the interaction of the system with humans should be minimized while established SLAs with the customers should not be violated. Since Cloud Computing infrastructures represent mega scale infrastructures comprising up to thousands of physical hosts, there is a high potential of energy waste by overprovisioning resources to keep SLA violation levels as low as possible.

Recent related work presents several concepts for the management of competing priorities of both, prevention of violation of established SLAs while reducing energy consumption of the system. As presented in [4], knowledge management techniques have been used to implement an autonomic control loop, where Cloud infrastructures are autonomously managed in order to keep the balance between SLA violations and resource consumption. Thus, the knowledge management (KM) system suggests reactive actions for preventing possible SLA violations and optimizing resource usage, which results in lower energy consumption. However, reactive actions of the system (as presented in [4], [5]) consider only intra-Cloud management, e.g., application or virtual machine (VM) reconfiguration.

On the other hand, there is considerable work in Cloud federation mechanisms without dealing with self-management issues of the system. A Federated Cloud Management (FCM) architecture proposed in [6] acts as an entry point to cloud federations and incorporates the concepts of meta-brokering, cloud brokering and on-demand service deployment. In this paper, we extend FCM by introducing an integrated system for reactive knowledge management and federation mechanisms suitable for on-demand generation and autonomous management of hybrid clouds. Besides intra-Cloud level (i.e., application and intra-VM management), we also target inter-Cloud management, where VMs are instantiated and destroyed on demand to prevent SLA violations and to minimize resource wastage.

This paper is organized as follows: first, we gather related approaches in Section II. In Section III we introduce the architecture for Cloud federations and provide a short overview on its main components. In Section IV, we present the changes and extensions applied to the architecture to accomplish autonomous behavior. Finally, we conclude our research in Section V.

II. Related work

Bernstein et al. [15] defines two use case scenarios that exemplify the problems faced by users of multi-cloud systems. They define the case of VM Mobility where they identify networking, specific cloud VM management interfaces and the lack of mobility interfaces as the three major obstacles. They also discuss a storage interoperability and federation scenario, in which storage provider replication policies are subject to change when a cloud provider initiates subcontracting. However, they offer interoperability solutions only for low-level functionality of clouds that are not focused on recent user demands, but on solutions for IaaS system operators.

Buyya et al. in [16] suggests a cloud 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 also 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. Finally, they have implemented a CloudSim-based simulation that evaluates the performance of the federations created using InterCloud technologies. Unfortunately, users face most federation-related issues before the execution of their services, therefore the concept of InterClouds cannot be applied in user scenarios this paper is targeting.

RightScale [9] offers a multi-cloud management platform that enables users to exploit the unique capabilities of different clouds, which has a similar view on Cloud federations to our approach. It is able to manage complete deployments of multiple servers across more clouds, using an automation engine that adapts resource allocation as required by system demand or system failures. They provide server templates to automatically install software on other supported cloud infrastructures. They also advertize disaster recovery plans, low-latency access to data, and support for security and SLA requirements. RightScale users can select, migrate and monitor their chosen clouds from a single management environment. They support Amazon Web Services, Eucalyptus Systems, Flexiscale, GoGrid, and VMware. The direct access to IaaS systmes is performed by the so-called Multi-Cloud Engine, which is supposed to perform brokering capabilities related to VM placement. Unfortunately we are not aware of any publications that detail the brokering operations of these components, therefore we cannot provide any deeper comparisons to our approach.

There has been considerable work on energy efficiency in ICT systems. Their common goal is to attain certain performance criteria for reducing energy consumption. Liu et al. [18] show how to save energy by optimizing VM placement via live migration. Meng et al. [19] try to increase efficient resource by provisioning multiple specific VMs together on a physical machine. Some authors as Kalyvianaki [20] focus on optimizing specific resource type as CPU usage, or only deal with homogeneous resources [21]. While most authors assume a theoretical energy model behind their approaches, Yu [22] targets the more basic question of how to effectively measure energy consumption in Cloud computing environments in a scalable way. Besides, none investigated energy efficiency in Cloud federations.

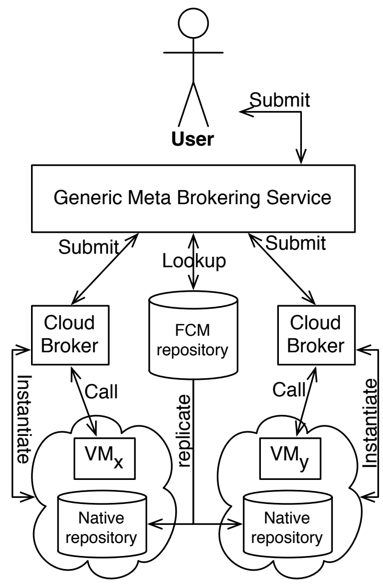

Figure 1. The original Federated Cloud Management architecture

Considering the use of Knowledge Management Systems (KM) and SLAs, Paschke et al. [23] look into a rule based approach in combination with a logical formalism called ContractLog. It specifies rules to trigger after a violation has occurred, e.g., it obliges the provider to pay some penalty, but it does not deal with avoidance of SLA violations. Others inspected the use of ontologies as knowledge bases (KBs), but only at a conceptual level. Koumoutsos et al. [24] view the system in four layers (i.e., business, system, network and device) and break down the SLA into relevant information for each layer, but they give no implementation details. Bahati et al. [25] also use policies, i.e., rules, to achieve autonomic management. They provide a system architecture including a KB and a learning component, and divide all possible states of the system into so called regions, which they assign a certain benefits for being in this region. However, the actions are not structured and lack a coherent approach. They are mixed together into a single rule, which makes them very hard to manage. On the contrary, we provide a well-structured and extendable approach that also investigates how actions are defined to avoid counteracting recommendations of the KM system.

III. Federated Cloud Management architecture

Figure 1 shows the Federated Cloud Management (FCM) architecture (first introduced in [6]), and its connections to the corresponding components that together represent an interoperable solution for establishing a federated cloud environment. Using this architecture, users are able to execute services deployed on cloud infrastructures transparently, in an automated way. Virtual appliances for all services should be stored in a generic repository called FCM Repository, from which they are automatically replicated to the native repositories of the different Infrastructure as a Service cloud providers.

Users are in direct contact with the *Generic Meta Brokering Service* (GMBS – [13]) that allows requesting a service by describing the call with a WSDL, the operation to be called, and its possible input parameters. The GMBS is responsible of selecting a suitable cloud infrastructure for the call, and submitting to a Cloud-Broker (CB) in contact with the selected infrastructure. Selection is based on static data gathered from the *FCM Repository* (e.g., service operations, WSDL, appliance availability), and on dynamic information of special deployment metrics gathered by the Cloud-Brokers (see Section IV-B2). The role of GMBS is to manage autonomously the interconnected cloud infrastructures with the help of the Cloud-Brokers by forming a cloud federation.

Cloud-Brokers are set up externally for each IaaS provider to process service calls and manage VMs in the particular cloud. Each Cloud-Broker [14] has its own queue for storing the incoming service calls, and it manages one virtual machine queue for each virtual appliance (VA). Virtual machine queues represent the resources that can currently serve a virtual appliance specific service call. The main goal of the Cloud-Broker is to manage the virtual machine queues according to their respective service demand. The default virtual machine scheduling is based on the currently available requests in the queue, their historical execution times, and the number of running VMs.

Virtual Machine Handlers are assigned to each virtual machine queue and process the VM creation and destruction requests in the queue. Requests are translated and forwarded to the underlying IaaS system. VM Handlers are infrastructure-specific and built on top of the public interfaces of the underlying IaaS. Finally, the Cloud-Broker manages the incoming service call queue by associating and dispatching calls to VMs created by the VM Handler.

As a background process, the architecture organizes virtual appliance distribution with the *automatic service deployment* component [17]. This component minimizes pre-execution service delivery time to reduce the apparent service execution time in highly dynamic service environments. Service delivery is minimized by decomposing virtual appliances and replicating them according to demand patterns, then rebuilding them on the IaaS system that will host the future virtual machine. This paper does not aim to further discuss the behavior of the ASD, however it relies on its features that reduce virtual appliance replication time and transfer time between the FCM and the native repositories.

IV. Self-adaptable Inter-Cloud management architecture

This paper offers two options to incorporate the concepts of knowledge management (KM) systems into the Federated Cloud Management architecture: local and global. Local integration is applied on a per deployed component basis, e.g., every Cloud-Broker utilizes a separate KM system for its internal purposes. In contrast, global integration is based on a single KM system that controls the autonomous behavior of the architectural components considering the available information from the entire cloud federation. In this section, first, we discuss which integration option is best to follow, then we introduce the extensions made to a KM system in order to perform the integration.

A. Knowledge management integration options

When *local* integration is applied, each knowledge manager can make fine-grained changes – e.g., involving actions on non-public interfaces – on its controlled subsystem. First, the meta-broker can select a different scheduling algorithm if necessitated by SLA violation predictions. Next, the Cloud-Broker can apply a more aggressive VM termination strategy, if the greenness of the architecture is more prioritized. Finally, if the storage requirements of the user are not valid any more, the FCM repository removes unnecessarily decomposed packages (e.g., when the used storage space approaches its SLA boundaries, the repository automatically reduces the occupied storage). However, the locally made reactions to predicted SLA violations might conflict with other system components not aware of the applied changes. These conflicts could cause new SLA violation predictions in other subsystems, where new actions are required to maintain the stability of the system. Consequently, local reactions could cause an *autonomic chain reaction*, where a single SLA violation prediction might lead to an unstable system.

To avoid these chain reactions, we investigated *global* integration (presented in Figure 2) that makes architecture-wide decisions from an external viewpoint. High-level integration is supported by a monitoring solution – deployed next to each subcomponent in the system (GMBS, the various Cloud-Brokers and repositories) – that determines system behavior in relation to the settled SLA terms. Global KM integration aggregates the metrics received from the different monitoring solutions, thus operates on the overall architecture and makes decisions considering the state of the entire system before changing one of its subsystems. However, adaptation actions are restricted to use the public operations of the FCM architecture (e.g., new cloud selection requests, new VM and call associations or repository rearrangements). Consequently, the global integration *exhausts adaptation actions* earlier than the local one, because of metrics aggregation and restricted interface use. For instance, if aggregated data hides the cause of a possible

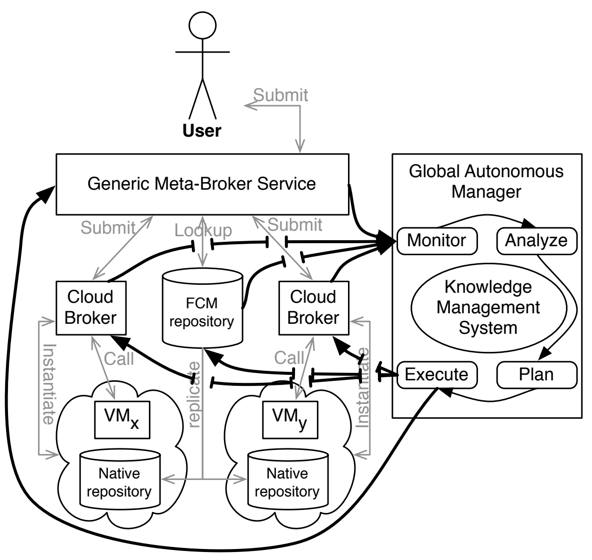

Figure 2. Global integration of the knowledge management system

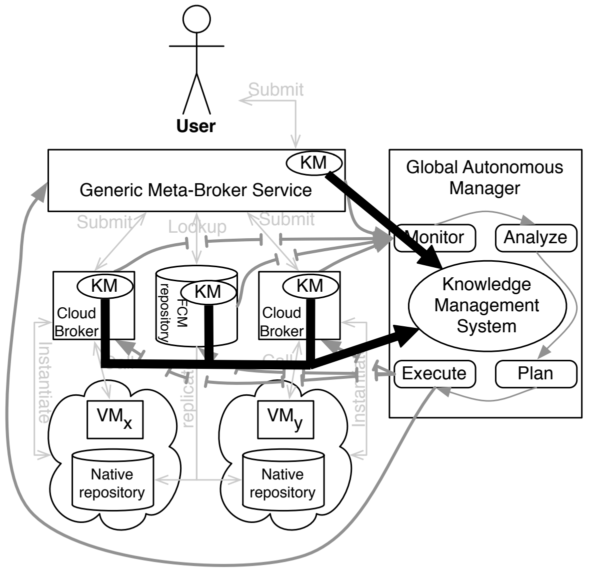

Figure 3. Hybrid integration of the knowledge management system

future SLA violation, then global KM cannot act without user involvement.

In this paper, we propose to use a *hybrid* KM system (revealed in Figure 3) combining both global and local integration options. The hybrid system avoids the disadvantages of the previous solutions by enabling global control over local decisions. In our system, local actions can be preempted by the global KM system by propagating predicted changes in aggregated metrics. Based on predicted changes, the global KM could stop the application of a locally optimal action and prevent the autonomic chain reaction that would follow the local action. On the other hand, if the global system does not stop the locally optimal action, then it enables the execution of more fine-grained actions postponing adaptation action exhaustion.

Action	Involved component	Integration
Reschedule calls	Meta-Broker	Global
Rearrange VM queues	Cloud-Broker	Global
Extend/Shrink VM Queue	Cloud-Broker	Local
Rearrange VA storage	FCM repository	Global
Self-Instantiated Deployment	Service instances	Local

Table I
SUMMARY OF AUTONOMOUS ACTIONS

B. Knowledge management system extensions

This subsection first lists the possible autonomic actions in our KM system, then it analyzes the collected monitoring data that can indicate the need for autonomous behavior. Finally, based on these indicators, we conclude with the rules triggering the adaptation in our FCM components.

1) Actions: Based on the affected components, the architecture applies four basic types of actions on unacceptable behavior. First, at the meta-brokering level, the system can organize a *rescheduling of several service calls*. E.g., the autonomous manager could decide to reschedule a specific amount of queued calls – $c \in Q_x$, where c refers to the call, and Q_x specifies the queue of the Cloud-Broker for IaaS provider x. Consequently, to initiate rescheduling, the knowledge manager specifies the amount of calls (N_{cr}) to be rescheduled and the source cloud (C_s) from which the calls need to be removed. Afterwards, the meta-broker evaluates the new situation for the removed calls, and schedules them to a different cloud, if possible.

Second, at the level of cloud brokering, the system could decide either to *rearrange the VM queues* of different Cloud-Brokers, or alternatively to *extend or shrink the VM queue* of a specific Cloud-Broker. *VM queue rearrangement* requires global KM integration in the system so it can determine the effects of the queue rearrangement on multiple infrastructures. The autonomous manager accomplishes rearrangement by destructing VMs of a particular virtual appliance in a specific cloud and requesting new VMs in another one. Consequently, the autonomous manager selects the virtual appliance (VA_{arr} – appliance to rearrange) that has the most affected VMs. Then it identifies the amount of virtual machines (N_{vmtr}) to be removed from the source cloud (C_s) and instantiated in a more suitable one (C_d).

The queue rearrangement operations have their counterparts also in case of local KM integration. The VM *queue extension and shrinking* operations are local decisions that are supported by energy efficiency related decisions. In case of queue shrinking, some of the virtual machines controlled by the local Cloud-Broker are destructed. However, under bigger loads, virtual machines could be in the process of performing service calls. Therefore, the autonomous manager can choose between the three VM destruction behaviors embedded into the Cloud-Brokers: (i) destroy after call completed, (ii) destroy right after request and put the call back to the local service call queue and finally, (iii) destroy right after request and notify the user about call abortion. As a result, the autonomous manager specifies the number of VMs to extend (N_{ex}) or shrink (N_{shr}) the queue with and the destruction strategy (S_{dest}) to be used.

Third, on the level of the FCM repository, the autonomous manager can make the decision to *rearrange virtual appliance storage* between native repositories. This decision requires the FCM repository either to remove appliances from the native repositories, or to replicate its contents to a new repository. Appliance removal is only feasible, if one of the following cases are met: (i) the hosting cloud will no longer execute the VA, (ii) the hosting cloud can download the VA from third party repositories or finally, (iii) the appliance itself was based on an extensible appliance that is still present in the native repository of the hosting cloud. The objective of the rearrangement is to reduce the storage costs in the federation at the expense of increased virtual machine instantiation time for VMs of the removed appliances. Conclusively, the rearrangement decision should involve the decision on the percentage (N_{repr}) of the reduced or replicated appliances that should participate in the rearrangement process.

Finally, when virtual appliances are built with embedded autonomous capabilities (internal monitoring, KM system etc.), then virtual machines based on them are capable of *self-initiated deployment*. If a service instance gets either overloaded or dysfunctional according to its internal monitoring metrics, then the instance contacts the local Cloud-Broker to instantiate a new virtual machine just like the instance is running in. In case of overloading, the new instance will also be considered for new *Call→VM* associations. In case of dysfunctional instances, the system creates a proxy service inside the original VM replacing the original service instance. This proxy is then used to forward the requests towards the newly created instance until the current VM is destructed.

2) Monitored metrics: After analyzing the various autonomous actions that the KM system can exercise, we investigated the monitoring system and the possible metrics to be collected for the identification of those cases when the architecture encounters unsatisfactory behavior. Currently, we monitor and analyze the behavior of Cloud-Brokers, the FCM repository and individual service instances.

Since Cloud-Brokers represent the behavior of specific IaaS systems, most of the measurements and decisions are made based on their behavior. All measurements are related to the queues of the Cloud-Broker; therefore we summarize their queuing behavior. Cloud-Brokers offer two types of queues: the call queue (Q_x, where x identifies the specific Cloud-Broker that handles the queue) and the VM queues ($VMQ_{x,y}$, where y identifies the specific service – or appliance VA_y – the queued VMs are offering). The members of the call queue represent the service calls that a Cloud-Broker needs to handle in the future (the queue is filled by the meta-broker and emptied by the Cloud-Broker through associating a call with a specific VM). On the other hand, VM queues are handled on a more complex way: they list the currently handled VMs offering a specific service instance. Consequently, the Cloud-Brokers maintain VM queues for all service instances separately. Entries in the VM queues are used to determine the state of the VMs:

$$State : VM \rightarrow \begin{cases} WAITING \\ INIT \\ RUNNING.AVAILABLE \\ RUNNING.ACQUIRED \\ CANCEL \end{cases} \quad (1)$$

- **Waiting:** the underlying cloud infrastructure does not have resources to fulfill this VM request yet.
- **Init:** the VM handler started to create the VM but it has not started up completely yet.
- **Running and available:** the VM is available for use, the Cloud-Broker can associate calls to these VMs only.
- **Running and acquired:** the VM is associated with a call and is processing it currently.
- **For cancellation:** the Cloud-Broker decided to remove the VM and stop hosting it in the underlying infrastructure.

Based on these two queues the monitor collects the metrics listed in the following paragraphs.

To support decisions for *service call rescheduling*, the system monitors the call queue for all available Cloud-Brokers for a specific service call s:

$$q(x, s) := \{c \in Q_x : (type(c) = s)\}, \quad (2)$$

where $type(c)$ defines the kind of the service call c is targeting.

Call throughput measurement of available Cloud-Brokers is also designed to assist *call rescheduling*:

$$throughput(x) := \frac{1}{max_{c \in Q_x}(waitingtime(c))}, \quad (3)$$

where $waitingtime(c)$ expresses the time in sec a service call has been waiting in the specific Q.

We define the average waiting time of a service s by

$$awt(s, Q_x) := \frac{\sum_{c \in q(x,s)} waitingtime(c)}{|q(x, s)|}, \quad (4)$$

and the average waiting time of a queue by

$$awt(Q_x) := \frac{\sum_{c \in Q_x} waitingtime(c)}{|Q_x|}. \quad (5)$$

To distinguish the Cloud-Brokers, where *VM queue rearrangements* could occur, we measure the number of service instances that are offered by a particular infrastructure:

$$vms(x,s) := \big\{vm \in VMQ_{x,s} : \\ State(vm) = RUNNING.AVAILABLE \\ \vee State(vm) = RUNNING.ACQUIRED\big\} \quad (6)$$

The call/VM ratio for a specific service managed by a specific Cloud-Broker:

$$cvmratio(x,s) := \frac{|q(x,s)|}{|vms(x,s)|} \quad (7)$$

This ratio allows the global autonomous manager to plan *VM queue rearrangements* and equalize the service call workload on the federated infrastructures. When applied with the local KM system, this ratio allows the system to decide on *extending and shrinking the VM queues* of particular services and balance the service instances managed by the local Cloud-Broker.

The load of the infrastructure managed by a specific Cloud-Broker:

$$load(x) := \frac{\sum_{\forall s} |vms(x,s)|}{\sum_{\forall s} |VMQ_{x,s}|} \quad (8)$$

The load analysis is used for *VM queue rearrangements* in order to reduce the number of waiting VMs in the federation. When applied locally, along with the call/vm ratio the load analysis is utilized to determine when to *extend or shrink the VM queues* of various services. As a result, Cloud-Brokers could locally reorganize their VM structures that better fit the current call patterns.

To support the remaining autonomous actions, the FCM repository and individual service instances are also monitored. First, the system monitors the accumulated storage costs of a virtual appliance in all the repositories ($r \in R$) in the system (expressed in US dollars/day):

$$stcost(VA_s) := \sum_{\forall r} locstcost(r, VA_s), \quad (9)$$

where $locstcost(r, VA_s)$ signifies the local storage cost at repository r for appliance VA_s. To better identify the possible appliance storage rearrangements the system also analyzes the usage rate of appliances in the different repositories expressed in the number of times the VMs based on the appliance have changed status from $INIT$ to $RUNNING.AVAILABLE$ in a single day ($deployfreq(r, VA_s)$).

Finally, individual services are monitored to support self-instantiated deployment. Here we analyze the service availability (expressed as the % of time that the instance is available for external service calls) of the specific service instance deployed in the same VM where the monitoring system is running.

3) Basic rules for applying actions: We decided to formulate the knowledge base (KB) as a rule-based system. Rules are of the form "WHEN condition THEN action" and can be implemented e.g., using the Java rule engine Drools [26]. We define several rules based on the previously defined measurements and actions, and present them in Drools-related pseudo code. The working memory of the KM system, which is the main class for using the rule engine at runtime, does not only consist of the specified rules, but also of the objects whose knowledge has to be modeled, and that are currently active in the Cloud federation (like a Cloud-Broker, the native repository, different queues, etc.). These objects are typically modeled as Java classes, and thus referred to as Cloud-Broker(), NativeRepository(), etc.

Figure 4 shows the rule for *rescheduling service calls*. Line 1 states the unique name the rule can be identified with in the KB. This way, rules can be dynamically altered or replaced if different global behavior due to changing high-level policies (i.e., changing from energy efficient to SLA performant) is required. Lines 3-4 state the conditions that have to be fulfilled to trigger the actions in lines 6-9. At first, we look for a Cloud-Broker C_s (line 3), whose throughput falls below the average of all the queues' throughputs ($mean()$) plus a multiple of their standard deviation ($std()$, line 4). If such C_s is found, the rule is executed. We have to decide to which Cloud C_d (line 6) to move N_{cr} service calls (line 7), and finally invoke the appropriate public interface methods of the Cloud brokers at stake (lines 8-9). As C_d we choose the Cloud with maximum throughput. The equalizeQs() method (line 7) tries to equal out the average waiting times of the queues of C_s and C_d. It takes the last service call $\hat{s}$ out of Q_s, retrieves its average waiting time $awt(\hat{s}, Q_s)$ and calculates the new estimated average waiting time for Q_s and Q_d by $awt(Q_s) := awt(Q_s) - awt(\hat{s}, Q_s)$ and $awt(Q_d) := awt(Q_d) + awt(\hat{s}, Q_d)$, respectively. Then it adds $\hat{s}$ to Q_d. It continues this procedure as long as $awt(Q_s) \geq awt(Q_d)$, and returns the number of service calls that have been hypothetically added to Q_d. The rule could then either really add the chosen calls to C_d as presented in line 9, or return them to the meta-broker

Figures 5 and 6 show possible rules for removing VAs from a Cloud's native repository due to high local or global costs, respectively. Both rules try to find a repository r and a VA VA_x that have been inserted into the working memory of the rules engine (lines 3-4), and remove the specified VA from the repository (line 8), when certain conditions hold. In Figure 5 the removal action is executed when two conditions hold: First, the local storage cost of the VA at the specified resource exceeds a certain threshold. The threshold is calculated as the average local storage costs at all repositories for the same VA plus a multiple of its standard deviation. Second, the deploy frequency of the VA

```
1 rule "Reschedule calls"
2 WHEN
3   $C_s : Cloudbroker()$
4   $throughput(x) < mean(throughput(.)) + \delta \cdot std(throughput(.))$
5 THEN
6   $C_d := \arg\max throughput(.)$
7   $N_{cr} := equalizeQs(C_s, C_d)$
8   $calls := remove(N_{cr}, C_s)$; //removes last $N_{cr}$ entries in $Q_{C_s}$.
9   $add(calls, C_d)$;
```

Figure 4. Rule for rescheduling calls

```
1 rule "Remove VA from native repository due to high local costs"
2 WHEN
3   $r$ : NativeRepository()
4   $VA_x$ : VirtualAppliance()
5   $locstcost(r, VA_x) > mean(locstcost(., VA_x)) + \delta \cdot std(locstcost(., VA_x))$
6   $deployfreq(r, VA_x) < mean(deployfreq(., VA_x))$
7 THEN
8   $remove(VA_x, r)$ //removes $VA_x$ from native repository $r$
```

Figure 5. Rule for removing VA from native repository of a specific Cloud due to high local costs

at this repository falls below a certain threshold, which is the mean deploy frequency of the VA at all repositories. In short, the VA is instantiated less often than other VAs, but its cost is higher than for other VAs, so the VA should be removed. Figure 6 takes a global perspective and checks whether the overall storage cost for the VA exceeds a certain threshold (defined similarly as with Figure 5, line 5). Then, the VA is removed from the repository that has the lowest deployment frequency (line 6).

The remaining rules can be specified according to the actions and measurements as explained before. However, their specific parameters may have heavy impact on the overall performance of the system. These parameters are to be learned by the KM system. In our future work, we plan to evaluate the system performance with the extension of the simulation engine presented in [4].

```
1 rule "Remove VA from native repository due to high global costs"
2 WHEN
3   $r$ : NativeRepository()
4   $VA_x$ : VirtualAppliance()
5   $stcost(VA_x) > mean(stcost(.)) + \delta \cdot std(stcost(.))$
6   $r_{min} : \arg\min deployfreq(., VA_x)$
7 THEN
8   $remove(VA_x, r_{min})$ //removes $VA_x$ from native repository $r_{min}$
```

Figure 6. Rule for removing VA from native repository of a specific Cloud due to high global costs

V. Conclusion

This paper presented an approach to extend federated cloud management architectures with autonomous behavior. Our research uses knowledge management systems to facilitate the decision making process of the classical monitoring-analysis-planning-execution loop. Using the FCM architecture as the basis of our further investigations, we analyzed different approaches to integrate the knowledge management system within this architecture, and found a hybrid approach that incorporates fine-grained local adaptation operations with options for high-level override. Then this research pinpointed the adaptation actions and their possible effects on cloud federations. Finally, we established metrics that could indicate possible SLA violations in federations, and defined rules that could trigger adaptation actions in the case of predicted violations.

Regarding future works, we plan to investigate more the green aspects in the autonomous behavior of cloud federations. We also aim at defining new rules for advanced action triggers and evaluate the applicability of another knowledge management approaches like case based reasoning. Finally, we also plan to investigate the effects of the autonomous behavior on the overall performance of the cloud federation on an experimental system.

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 under grant agreement N^o RI-283481 (SCI-BUS), also from the Vienna Science and Technology Fund (WWTF) under grant agreement ICT08-018 – Foundations of Self-Governing ICT Infrastructures (FoSII).

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*, 2009, vol. 25, issue. 6, pp. 599–616.

[2] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Computer Communication Review*. vol 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 Engineering*, vol. 15, pp. 313–341, December 2008.

[4] M. Maurer, I. Brandic and R. Sakellariou. Simulating Autonomic SLA Enactment in Clouds using Case Based Reasoning. In *Towards a Service-Based Internet, Third European Conference*, ServiceWave 2010, Ghent, Belgium, December, 2010 pp. 25-37

[5] M. Maurer, I. Brandic and R. Sakellariou. Enacting SLAs in Clouds Using Rules. In *Proceedings of the 17th International Euro-Par Conference on Parallel and Distributed Computing*, Bordeaux, France, September, 2011

[6] A. Cs. Marosi, G. Kecskemeti, A. Kertesz and P. Kacsuk. FCM: an Architecture for Integrating IaaS Cloud Systems. In *Proceedings of The Second International Conference on Cloud Computing, GRIDs, and Virtualization*. Rome, Italy. September, 2011.

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

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

[9] RightScale website. http://www.rightscale.com/, 2011.

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

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

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

[13] A. Kertesz and P. Kacsuk. GMBS: A new middleware service for making grids interoperable. *Future Generation Computer Systems*, 2010, vol. 26, issue 4, pp. 542–553.

[14] A. Cs. Marosi and P. Kacsuk. Workers in the clouds. In *Proceedings of the 19th Euromicro Conference on Parallel, Distributed and Network-Based Processing* (PDP 2011), Ayia Napa, Cyprus, February 9-11, 2011. pp. 519–526

[15] 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* (2008, pp. 328-336)

[16] 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, 20 pages, 2010.

[17] 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.

[18] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang, and Y. Chen. Greencloud: a new architecture for green data center. In *Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*, ICAC-INDST '09, pages 29–38, New York, NY, USA, 2009. ACM.

[19] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis. Efficient resource provisioning in compute clouds via vm multiplexing. In *Proceeding of the 7th international conference on Autonomic computing*, ICAC '10, pages 11–20, New York, NY, USA, 2010. ACM.

[20] E. Kalyvianaki, T. Charalambous, and S. Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th international conference on Autonomic computing*, ICAC '09, pages 117–126, New York, NY, USA, 2009. ACM.

[21] B. Khargharia, S. Hariri, and M. S. Yousif. Autonomic power and performance management for computing systems. *Cluster Computing*, 11(2):167–181, 2008.

[22] Y. Yu and S. Bhatti. Energy measurement for the cloud. *Parallel and Distributed Processing with Applications, International Symposium on*, 0:619–624, 2010.

[23] A. Paschke and M. Bichler. Knowledge representation concepts for automated SLA management. *Decision Support Systems*, 46(1):187–205, 2008.

[24] G. Koumoutsos, S. Denazis, and K. Thramboulidis. SLA e-negotiations, enforcement and management in an autonomic environment. *Modelling Autonomic Communications Environments*, pages 120–125, 2008.

[25] R. M. Bahati and M. A. Bauer. Adapting to run-time changes in policies driving autonomic management. In *ICAS '08: Proceedings of the 4th Int. Conf. on Autonomic and Autonomous Systems*, Washington, DC, USA, 2008. IEEE Computer Society.

[26] Drools, The Business Logic integration Platform. www.drools.org, 2011.