FCM: an Architecture for Integrating IaaS Cloud Systems

Attila Csaba Marosi, Gabor Kecskemeti, Attila Kertesz, Peter Kacsuk
MTA SZTAKI
Computer and Automation Research Institute
of the Hungarian Academy of Sciences
H-1528 Budapest, P.O.Box 63, Hungary
Email: {atisu, kecskemeti, keratt, kacsuk}@sztaki.hu

***Abstract*—Cloud Computing builds on the latest achievements of diverse research areas, such as Grid Computing, Service-oriented computing, business processes and virtualization. In this paper, we reveal open research issues by envisaging a federated cloud that aggregates capabilities of various IaaS cloud providers. We propose a Federated Cloud Management architecture that acts as an entry point to cloud federations and incorporates the concepts of meta-brokering, cloud brokering and on-demand service deployment. The meta-brokering component provides transparent service execution for the users by allowing the system to interconnect the various cloud broker solutions available in the system. Cloud brokers manage the number and the location of the utilized virtual machines for the received service requests. In order to fast track the virtual machine instantiation, our architecture uses the automatic service deployment component that is capable of optimizing service delivery by encapsulating services as virtual appliances in order to allow their decomposition and replication among the various IaaS cloud infrastructures. Our solution is able to cope with highly dynamic service executions by federating heterogeneous cloud infrastructures in a transparent and autonomous manner.**

***Keywords*—cloud federation; cloud brokering; IaaS; virtual appliance.**

I. Introduction

Highly dynamic service environments [1] require a novel infrastructure that can handle the on demand deployment and decommission of service instances. Cloud Computing [2] 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. Virtual appliances (VA) encapsulate metadata (e.g., network requirements) with 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 and to accomplish dynamic service environments users frequently envisage a federated cloud that aggregates capabilities of various IaaS cloud providers. These IaaS systems are either offered by public service providers (like Amazon [3] or RackSpace [4]) or by smaller scale privately managed infrastructures. We propose an autonomic resource management solution that 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.

This paper proposes a layered architecture that incorporates the concepts of meta-brokering, cloud brokers and automated, on-demand service deployment. The meta-brokering component allows the system to interconnect the various cloud brokers available in the system. The cloud broker component is responsible for managing the virtual machine instances of the particular virtual appliances hosted on a specific infrastructure as a service provider. Our architecture organizes the virtual appliance distribution with the automatic service deployment component that can decompose virtual appliances to smaller parts. With the help of the minimal manageable virtual appliances the Virtual Machine Handler rebuilds these decomposed parts in the IaaS system chosen by the meta-broker. As a result, the cloud broker component uses the VM Handler to maintain the number of virtual machines according to the demand.

Related works have identified several shortcomings in the current cloud infrastructures [5]: e.g., federated clouds will face the issue of scalability and self-

 ISBN: 978-1-61208-153-3

management similarly to Grid systems, or users of the cloud systems should be in control of their computing costs. We propose an architecture that aims at both of these problems by allowing users to utilize meta-brokering between public and private cloud systems as a result lowering their operation costs. Our architecture also handles the issue of scalability by offering the cloud brokers that manage the virtual machines according to the actual demands of the user applications.

This paper is organized as follows: first, we introduce the related research results in Section II. Then, we discuss an advanced use case in Section III that involves our proposed architecture and discusses its advantages in contrast to previous research results. Next, we detail the operational roles of the brokering components in our architecture in Section III-A and Section III-B. Afterwards, in Section IV, we discuss an optimization approach to rebuild virtual appliances within the virtual machine that is used to execute them. Finally, we conclude our research in Section V.

II. Related Work

Matthias Schmidt et al. [6] 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.

There are several related works focusing on providing dynamic pool of resources. Paul Marshall et al. [7] describe an approach for developing an "elastic site" model where batch schedulers, storage and web services can utilize such resources. They introduce different basic policies for allocating resources, that can be "on-demand" meaning resources are allocated when a service call or task arrives, "steady stream" assumes steady utilization, thus leaves some elastic resources continuously running, regardless of the (temporary) shortage of tasks, or "bursts" for fluctuating load. They concentrate on dynamically increasing and decreasing the number of resources, but rely on third party logic for balancing load among the allocated resources. Constantino Vázquez et al. [8] are building complex grid infrastructures on top of IaaS cloud systems, that allow them to adjust the number of grid resources dynamically. They focus on the capability of using resources from different cloud providers and on the capability of providing resources for different grid middleware, but meta-scheduling between the utilized infrastructures and developing a model, that considers the different cloud provider characteristics is not addressed.

In 2009, Amazon Web Services launched Amazon CloudWatch [9], that is a supplementary service for Amazon EC2 instances that provides monitoring services for running virtual machine instances. It allows to gather 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.

Mohsen Amini et al. [10] are focusing on so called marketing-oriented scheduling policies, that can provision extra resources when the local cluster resources are not sufficient to meet the user requirements. Former scheduling policies used in grids are not working effectively in cloud environments, mainly because Infrastructure as a Service providers are charging users in a pay-as-you-go manner in an hourly basis for computational resources. To find the trade-off between to buy acquired additional resources from IaaS and reuse existing local infrastructure resources he proposes two scheduling policies (cost and time optimization scheduling policies) for mixed (commercial and non-commercial) resource environments. Basically two different approaches were identified on provisioning commercial resources. The first approach is offered by the IaaS providers at resource provisioning level (user/application constraints are neglected: deadline, budget, etc.), the other approach deploys resources focusing at user level (time and/or cost minimization, estimating the workload in advance, etc.).

III. Federated Cloud Management Architecture

Figure 1 shows the Federated Cloud Management (FCM) architecture and its connections to the corresponding components that together represent an interoperable solution for establishing a federated cloud environment. The FCM targets the problem area outlined in the Introduction, and provides solutions for most of the listed open issues. In the following, we exemplify

 ISBN: 978-1-61208-153-3

the interaction of the main components of this solution through a low level use case.

In this scenario we restrict our solution to support standard stateless web services described with WSDL [11]. Using this solution, 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 that they are automatically replicated to the native repositories of the different Infrastructure as a Service cloud providers.

When a user sends a service call to the system, he/she submits a request to the "*Generic Meta-Broker Service*" (GMBS) specifying the requested service with a WSDL, the operation to be called, and its possible input parameters. The GMBS checks if the service has an uploaded VA in the generic repository, then it selects a suitable CloudBroker for further submission. The matchmaking is based on static data gathered from the "*FCM Repository*" (e.g., service operations, WSDL), and on dynamic information of special deployment metrics gathered by the CloudBrokers. Currently we use the average VA deployment time and the average service execution time for each VA. VA deployment time assumes that the native repository already has the requested VA, thus includes only the service provision time on a specific IaaS cloud. The role of GMBS is to manage autonomously the interconnected cloud infrastructures with the help of the CloudBrokers by forming a federation.

Each "*CloudBroker*" has an own queue for storing the incoming service calls (called Q_1 and Q_2 in Figure 1), and manages one virtual machine queue for each VA ($VA_x \rightarrow VMQ_x$). Virtual machine queues represent the resources that currently can serve a virtual appliance specific service call. The main goal of the CloudBroker 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 (n, m, o, p) of running VMs. The secondary task of the CloudBroker involves the dynamic creation and destruction of the various VMQs.

Virtual Machine Handler ("*VM Handler*") components are assigned to each virtual machine queue. These components process the virtual machine creation and destruction requests placed in the queue. The requests are translated and forwarded to the corresponding IaaS system ($Cloud_a$). This component is a cloud infrastructure-specific one, that uses the public interface of the managed infrastructure.

Independently from the virtual machine scheduling process the CloudBroker also handles the queue of the incoming service calls. As a result, these calls are dispatched to the available VMs created in the previously discussed manner.

In order to optimize service executions in highly dynamic service environments, our architecture organizes the virtual appliance distribution as a background process with the automatic service deployment component that can decompose virtual appliances to smaller parts. With the help of the minimal manageable virtual appliances (MMVA – further discussed in Section IV) the Virtual Machine Handler is able to rebuild these decomposed parts in the IaaS system on demand, that results in faster VA deployment and in a reduced storage requirement in the native repositories.

In the following, subsections we detail how resource management is carried out in this architecture. At the top-level, a meta-broker is used to select from the available cloud providers based on performance metrics, while at the bottom-level, IaaS-specific CloudBrokers are used to schedule VA instantiation and deliver the service calls to the clouds.

A. Top-level Brokering in FCM

As we already mentioned in the scenario discussed in the previous section, brokering takes place at two levels in the FCM architecture: the service call is first submitted to the Generic Meta-Broker Service (GMBS – that is a revised and extended version of the Grid Meta-Broker Service described in [12]), where a top-level decision is made to that cloud infrastructure the call should be forwarded. Then the service call is placed in the queue of the selected CloudBroker, where the bottom-level brokering is carried out to select the VM that performs the actual service execution. This bottom-level brokering and the detailed introduction of the architecture of the CloudBroker is discussed later in Section III-B.

Now, let us turn our attention to the role of GMBS. An overview of its architecture is shown in Figure 2. 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 FCM Repository on service availability, and aggregated dynamic information collected from the Cloud-

 ISBN: 978-1-61208-153-3

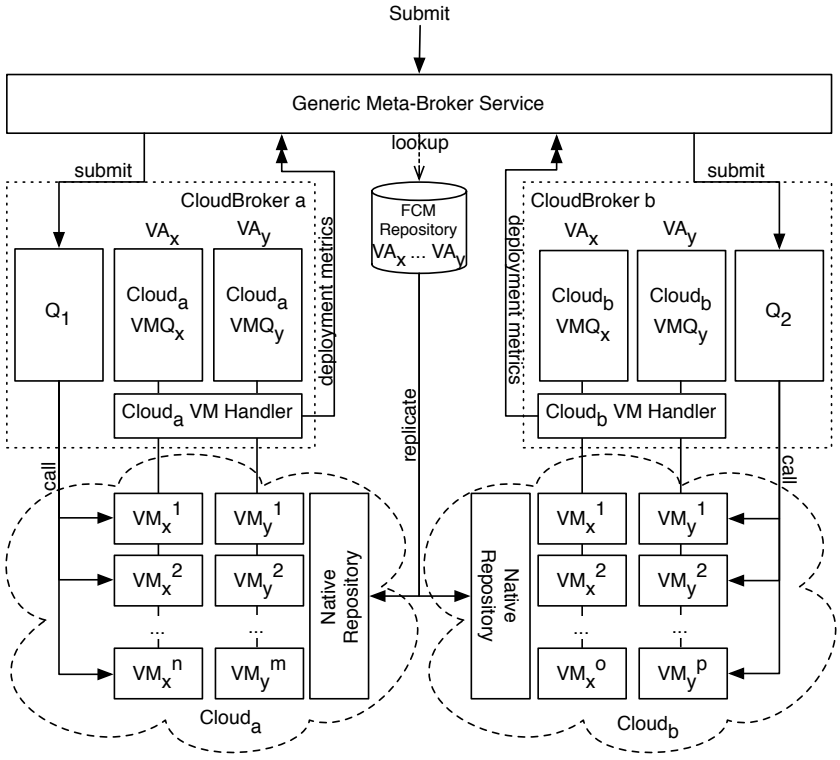

Fig. 1. The Federated Cloud Management architecture

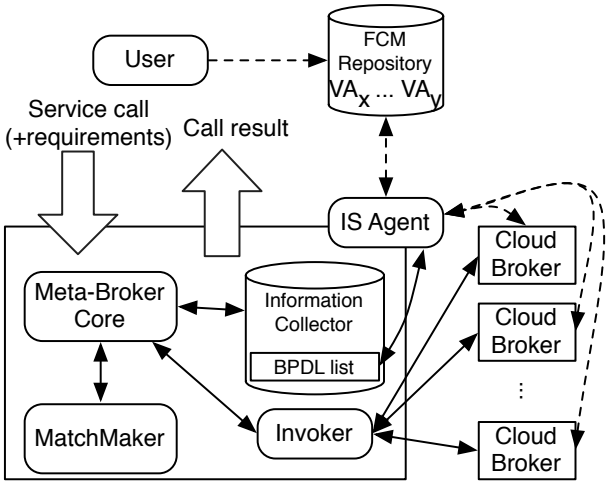

Fig. 2. The architecture of the Generic Meta-Broker Service

Brokers including average VA deployment and service execution times. The Invoker component forwards the service call to the selected CloudBroker and receives the service response.

Each CloudBroker is described by an XML-based Broker Property Description Language (BPDL) document containing basic broker properties (e.g., name), and the gathered aggregated 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 CloudBroker:

- Estimated availability time for a specific virtual appliance in a native repository – collected from the FCM Repository;
- average VA deployment time and average service execution time for each VA – queried from the CloudBroker;

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

B. CloudBroker

The CloudBroker handles and dispatches service calls to resources and performs resource management within a single IaaS system, it is an extended version of the

system described in [13].

The architecture of the CloudBroker is shown in Figure 1. Its first task is to dynamically create or destroy virtual machines (VM_x^i) and VM queues (VMQ_x) for the different used virtual appliances. To do that, first, the VA has to be replicated to the native repository of the IaaS system from the FCM Repository (an alternative method is discussed in Section IV). Alongside the appliance, the FCM Repository also stores additional static requirements about its future instances, like its minimum resource demands (disk, CPU and memory), that are needed by the CloudBroker. This data is not replicated to the native repository, rather the FCM Repository is queried.

A VM queue stores references to resources capable of handling a specific service call, thus instances of a specific VA. New resource requests are new entries inserted into the queue of the appropriate VA, while resource destruction requests are modification of entries representing an already running resource. The entries are managed by the VM Handler, that is a cloud fabric specific component designed to interact with the public interface of a single IaaS system. It simply translates and forwards requests to the public interface of the IaaS system ($Cloud_a$). Each VA contains a monitoring component deployed, that allows the CloudBroker to monitor the basic status (CPU, disk and memory usage) of the running resources along the average deployment time for each VA and average service execution times. These data can be queried by the IS Agent of the GMBS.

The service call queue (Q_1 and Q_2) stores incoming service requests and, for each request, reference to a VA in the FCM Repository. There is a single service call queue in each CloudBroker, while there are many VM queues. If the native repository does not contain the requested VA it is replicated first. Dynamic requirements for the VA may be specified with the service call:

- Additional resources (CPU, memory and disk);
- an UUID, that allows to identify service calls originating from the same entity.

The UUID will allow to meet SLA constraints later, e.g., to enforce a total cost limit on public clouds for service calls of the entity, or to be in compliance with deadlines. If any dynamic requirements are present the CloudBroker treats the VA as a new VA type, thus creating a new VM queue and starts a VM. The service calls may now be dispatched to the appropriate VMs. Most IaaS systems offer predefined classes of resources (CPU, memory and disk capacity) not adjustable by the user, in this case the CloudBroker will select the resource class that has at least the requested resources available. This may lead to allocating excess resources in some cases (e.g., the resource class has twice the memory requested to meet the CPU number requirement).

The CloudBroker also performs the scheduling of service call requests to VA's and the life-cycle management of resources. Scheduling decision is made based on the monitoring information gathered from the resources. If the service request cannot be scheduled to any resource the CloudBroker may decide to start a new VM capable of serving the request. The decision is 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);

VM decommission is also based on the above, but the CloudBroker takes into account the billing period of the IaaS system, shutdown is performed only shortly before the end of the period with regard to the average decommission time for the system.

IV. Virtual Appliance Delivery Optimization

IaaS systems require virtual appliances to be stored in their native repositories. Only those virtual appliances, that were previously stored in these repositories, can be used to instantiate virtual machines. Our architecture allows users to upload their virtual appliances to the FCM Repository that behaves as an active repository and handles the distribution of the appliances to the native repositories according to [14]. As an active repository, the FCM repository identifies the common parts of the appliances and decomposes them into smaller packages that allow appliance delivery and rebuilding from multiple repositories.

Central virtual appliance storage would require the VM Handler to first download the entire appliance from the FCM repository to a native one, then instantiate the appliance with the IaaS system. To avoid the first transfer, but keep the convenience for the users of our architecture, we have investigated options to rebuild virtual appliances in already running virtual machines. We have identified two distinct approaches for rebuilding: (i) native appliance reuse, (ii) minimal manageable virtual appliances. The first approach utilizes already available virtual appliances in the native repositories and extends them towards the required virtual appliance. In this article, we do not aim at this approach because it requires the investigation of the publicly available

 ISBN: 978-1-61208-153-3

appliances in order to find the appliance most suitable for extension.

The second approach proposes the minimal manageable virtual appliance that we define as basic appliance with the following three properties:

- Offers content management interfaces to add, configure and remove new appliance parts.
- Offers monitoring interfaces to analyze the current state of its instances (e.g., provide access to their CPU load, free disk space and network usage).
- Optimally sized: only those files present in the appliance that are required to offer their extensibility with the previously mentioned interfaces.

As a result, our architecture only needs to replicate the MMVAs to every native repository. If the FCM repository identifies high demands for specific virtual appliance parts, then the active repository functionality automatically replicates the appliance to those IaaS systems where most requests were originated from.

Our VM Handler is prepared to control virtual appliance rebuilding using minimal manageable virtual appliances. Consequently, the VM Handler applies a new strategy when it receives a virtual appliance instantiation request for a specific appliance that is not available in the native repository. This strategy starts with the instantiation of the MMVA. Next, the Handler waits until the virtual machine of the MMVA has started up. Then, it requests the content management interfaces to add the parts of the specific appliance that were identified as unique during the decomposition of the MMVA and the specific appliance. As a result, the specific appliance is rebuilt and ready to serve the scheduled service requests in the virtual machine instantiated for the MMVA.

V. Conclusion and Future Works

In this paper, we proposed a Federated Cloud Management solution that acts as an entry point to cloud federations. Its architecture incorporates the concepts of meta-brokering, cloud brokering and on-demand service deployment – their interaction is exemplified through a low-level use case. The meta-brokering component provides transparent service execution for the users by allowing the system to interconnect the various cloud broker solutions managed by aggregating capabilities of these IaaS cloud providers. We have shown how CloudBrokers manage the number and the location of the utilized virtual machines for the various service requests they receive. In order to fast track the virtual machine instantiation, our architecture uses the automatic service deployment component that is capable of optimizing its delivery by decomposing and replicating it among the various IaaS cloud infrastructures. Regarding future works, we plan to investigate various scenarios that arise during handling federated cloud infrastructures using the FCM architecture (e.g., the interactions and interoperation of public and private IaaS systems).

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 261556 (EDGI).

References

[1] 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. [Online]. Available: http://portal.acm.org/citation.cfm?id=1459074.1459084

[2] 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.

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

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

[5] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 50–55, December 2008. [Online]. Available: http://doi.acm.org/10.1145/1496091.1496100

[6] 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*, ser. PDP '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 567–574. [Online]. Available: http://dx.doi.org/10.1109/PDP.2010.39

[7] P. Marshall, K. Keahey, and T. Freeman, "Elastic site: Using clouds to elastically extend site resources," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, ser. CCGRID '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 43–52. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2010.80

[8] C. Vázquez, E. Huedo, R. S. Montero, and I. M. Llorente, "On the use of clouds for grid resource provisioning," *Future Gener. Comput. Syst.*, vol. 27, pp. 600–605, May 2011. [Online]. Available: http://dx.doi.org/10.1016/j.future.2010.10.003

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

[10] M. A. Salehi and R. Buyya, "Adapting market-oriented scheduling policies for cloud computing."

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

[12] A. Kertész and P. Kacsuk, "Gmbs: A new middleware service for making grids interoperable," *Future Gener. Comput. Syst.*, vol. 26, pp. 542–553, April 2010. [Online]. Available: http://dx.doi.org/10.1016/j.future.2009.10.007

[13] A. C. Marosi and P. Kacsuk, "Workers in the clouds," in *PDP*, Y. Cotronis, M. Danelutto, and G. A. Papadopoulos, Eds. IEEE Computer Society, 2011, pp. 519–526.

[14] G. Kecskeméti, G. Terstyánszky, P. Kacsuk, and Z. Németh, "An approach for virtual appliance distribution for service deployment," *Future Gener. Comput. Syst.*, vol. 27, pp. 280–289, March 2011. [Online]. Available: http://dx.doi.org/10.1016/j.future.2010.09.009

 ISBN: 978-1-61208-153-3