Autonomic SLA-aware Service Virtualization for Distributed Systems

Attila Kertész, Gábor Kecskeméti
Computer and Automation Research Institute
MTA SZTAKI
H-1518 Budapest, P.O. Box 63, Hungary
attila.kertesz, kecskemeti@sztaki.hu

Ivona Brandic
Distributed Systems Group
Vienna University of Technology
1040 Vienna, Argentinierstr. 8/181-1, Austria
ivona@infosys.tuwien.ac.at

***Abstract*—Cloud Computing builds on the latest achievements of diverse research areas, such as Grid Computing, Service-oriented computing, business processes and virtualization. Managing such heterogeneous environments requires sophisticated interoperation of adaptive coordinating components. In this paper we introduce an SLA-aware Service Virtualization architecture that provides non-functional guarantees in the form of Service Level Agreements and consists of a three-layered infrastructure including agreement negotiation, service brokering and on demand deployment. In order to avoid costly SLA violations, flexible and adaptive SLA attainment strategies are used with a failure propagation approach. We demonstrate the advantages of our proposed solution with a biochemical case study in a Cloud simulation environment.**

***Keywords*-Cloud Computing; SLA-negotiation; Service Brokering; On-demand deployment;**

I. Introduction

Cloud Computing [6] builds on the latest achievements of diverse research areas, such as Grid Computing, Service-oriented computing, business processes and virtualization. Both Grids and Service Based Applications (SBAs) already provide solutions for executing complex user tasks, but they are still lacking non-functional guarantees. The newly emerging demands of users and researchers call for expanding service models with business-oriented utilization (agreement handling) and support for human-provided and computation-intensive services [6]. Providing guarantees in the form of Service Level Agreements (SLAs) are highly studied in Grid Computing [18]. Nevertheless in Clouds, infrastructures are also represented as a service that are not only used but also installed, deployed or replicated with the help of virtualization. These services can appear in complex business processes, which further complicates the fulfillment of SLAs in Clouds. For example, due to changing components, workload and external conditions, hardware and software failures, already established SLAs may be violated. Frequent user interactions with the system during SLA negotiation and service executions (which are usually necessary in case of failures), might turn out to be an obstacle for the success of Cloud Computing. Thus, there is demand for the development of SLA-aware Cloud middleware, and application of appropriate strategies for autonomic SLA attainment. Despite cloud computing's business-orientation, the applicability of Service-level agreements in the Cloud middleware is rarely studied, yet [23]. Most of the existing work address provision of SLA guarantees to the consumer and not necessarily the SLA-based management of loosely coupled Cloud infrastructure. In such systems it is hard to localize where the failures have happen exactly, what the reason is for the failure and which reaction should be taken to solve the problem. Such systems are implemented in a proprietary way, making it almost impossible to exchange the components (e.g. use another version of the broker).

Autonomic Computing is one of the candidate technologies for the implementation of SLA attainment strategies. Autonomic systems require high-level guidance from humans and decide, which steps need to be done to keep the system stable [10]. Such systems constantly adapt themselves to changing environmental conditions. Similar to biological systems (e.g. human body) autonomic systems maintain their state and adjust operations considering changing components, workload, external conditions, hardware and software failures. An important characteristic of an autonomic system is an intelligent closed loop of control. The Autonomic Manager (AM) manages the element's state and behaviour. It is able to sense state changes of the managed resources and to invoke appropriate set of actions to maintain some desired system state. Typically control loops are implemented as MAPE (monitoring, analysis, planning, and execution) functions [10]. The *monitor* collects state information and prepares it for the *analysis*. If deviations to the desired state are discovered during the analysis, the *planner* elaborates change plans, which are passed to the *executor*. However, for the successful implementation of the autonomic principles loosely coupled SLA-based Cloud middleware is required. In such system failure source should be identified based on violated SLAs and located exactly considering different components of a Cloud middleware (virtualization, brokering, negotiation, etc. components). Thus, once the failure is identified Service Level Objectives (SLOs) can be used as a guideline for the autonomic reactions.

In this paper we introduce a novel architecture considering resource provision using a virtualization approach and combining it with the business-oriented utilization used for the SLA agreement handling. Thus, we provide an SLA-coupled infrastructure for on-demand service provision based on SLAs (called SLA-based Service Virtualization – SSV). We also exemplify how autonomic behaviour appears in the architecture in order to cope with changing user requirements and on demand failure handling. The main contributions of this paper include: (i) the presentation of the novel loosely coupled architecture for the *SLA-based Service virtualization* and on-demand resource provision, (ii) description of the architecture including *meta-negotiation, meta-brokering, brokering and automatic service deployment* with respect to its autonomic behaviour, and (iii) the *validation* of the SSV architecture with a biochemical case study in a Cloud simulation environment.

II. Related work

Though Cloud Computing is highly studied and a large body of work has been done trying to define and envision the boundaries of this new area, despite business-orientation, the applicability of Service-level agreements in the Cloud middleware is rarely studied, yet [23]. The envisioned framework in [7] proposes a solution to extend the web service model by introducing and using semantic web services. The need for SLA handling, brokering and deployment also appears in this vision, but they focus on using ontology and knowledge-based approaches. Most of related works consider either virtualization approaches [8] without taking care of agreements or concentrate on SLA management neglecting the appropriate resource virtualizations [21]. The work by Van et. al. [22] studied the applicability of the autonomic computing to Cloud-like systems, but they almost exclusively focus on virtualization issues like Virtual Machine (VM) packing and placement.

Comparing the currently available solutions, autonomic principles are not implemented in a adequate way because of they are lacking an SLA-coupled Cloud infrastructure, where failures and malfunctions can be identified using well defined SLA contracts. Work presented in this paper is the first attempt to develop an SLA-coupled autonomic Cloud infrastructure in an adequate way. Works presented in [19], [17] discuss incorporation of SLA-based resource brokering into existing Grid systems, but they do not deal with virtualization. The Rudder framework [15] facilitates automatic Grid service composition based on semantic service discovery and space based computing. Venugopal et al. propose a negotiation mechanism for advance resource reservation using the alternate offers protocol [26], however, it is assumed that both partners understand the alternate offers protocol.

Current deployment solutions do not leverage their benefits on higher level. For example the Workspace Service (WS) [8] as a Globus incubator project supports wide range of scenarios involving virtual workspaces, virtual clusters and service deployment from installing a large service stack to deploy a single WSRF service if the Virtual Machine image of the service is available. It is designed to support several virtual machines. The XenoServer open platform [20] is an open distributed architecture based on the XEN virtualization technique aiming at global public computing. The platform provides services for server lookup, registry, distributed storage and a widely available virtualization server. Also the VMPlants [14] project proposes an automated virtual machine configuration and creation service which is heavily dependent on software dependency graphs, but this project stays within cluster boundaries.

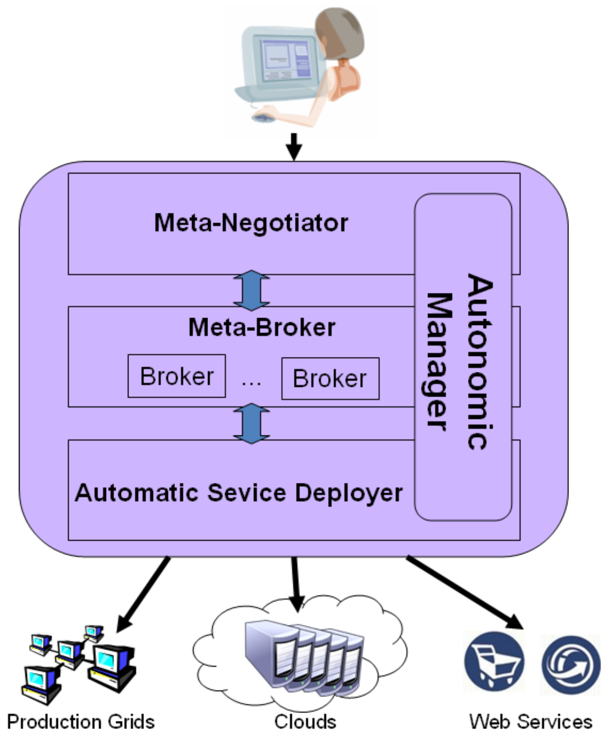

Figure 1. The SSV architecture.

III. The autonomic, SLA-based Service virtualization (SSV) architecture

In this section we present a unified service architecture that builds on three main areas: agreement negotiation, brokering and service deployment using virtualization (more detailed descriptions on the architecture can be read in [11]). We suppose that service providers and service consumers meet on demand and usually do not know about the negotiation protocols, document languages or required infrastructure of the potential partners. Figure 1 shows our

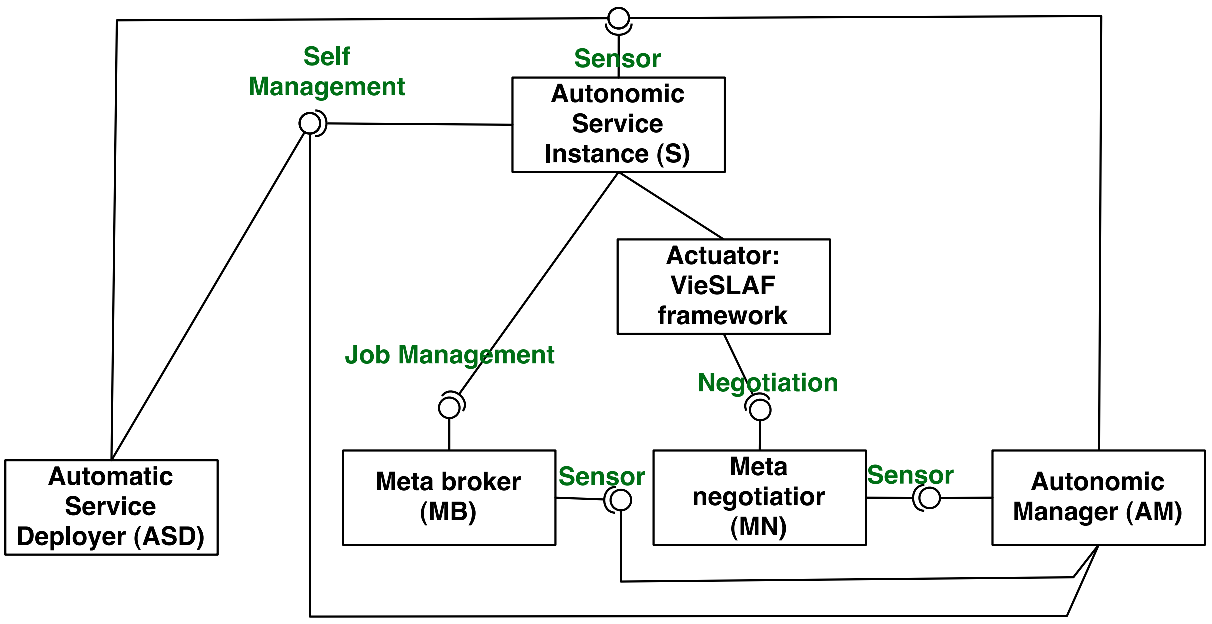

Figure 2. Autonomic components in the SSV.

proposed, general architecture. The main components of the architecture and their roles are gathered in Table I, while the sequence diagram in Figure 3 reveals the interactions of the components and the utilization steps of the architecture:

Agreement negotiation. First the user starts a negotiation for executing a service with certain QoS requirements (specified in a Service Description (SD) with an SLA). Then the Meta-Negotiator (MN) asks the Meta-Broker (MB), if it could execute the service with the specified requirements. Later on MB matches the requirements to the properties of the available brokers and replies with an acceptance or a different offer for renegotiation. After this phase the MN replies with the answer of MB. These initial steps may continue for renegotiations, until both sides agree on the terms.

Service request. After the agreement has been established the user calls the service with the SD and SLA. Then the MN passes SD and the transformed SLA (to the protocol the selected broker (B) understands) to the MB. In the next phase the MB calls the selected Broker with SLA and a possibly translated SD (to the language of the Broker). Then the broker executes the service with respect to the terms of the SLA (if needed deploys the service before execution). Finally the result of the execution is reported to the Broker, the MB, the MN, finally to the User (or workflow engine).

Information collection. While serving requests the architecture also processes background tasks that are not in the ideal execution path, these are also presented on Figure 3. The following background tasks are information collecting procedures that provide accurate information about the current state of the infrastructure up to the meta-broker level. In *"step a"* the Automatic Service Deployer (ASD) monitors the states of the virtual resources and deployed services. Then in *"step b"* ASD reports service availability and properties to its Broker. Finally at *"step c"* all Brokers report available service properties to the MB.

A. Autonomic behaviour in the SSV architecture

The sequence diagram of Figure 3 represents the ideal execution flow of the SSV architecture, therefore it does not reflect cases when unexpected events occur during the operation of each component. The SSV architecture targets these events with the autonomic management interfaces introduced on Figure 2. We distinguish three types of interfaces in our architecture: the *job management interface*, the *negotiation interface* and the *self-management interface*. Negotiation interfaces are typically used by the monitoring processes of brokers and meta-broker during the negotiation phases of the service deployment process. Self-management is needed to re-negotiate established SLAs during service execution. Job management interfaces are necessary for the manipulation of services during execution, for example for the upload of input data, or for the download of output data, and for starting or canceling service executions.

The *Autonomic manager* in the SSV architecture is an abstract component that specifies how self-management is carried out. The manager can be implemented in the different layers of the architecture in order to handle the different unexpected situations. For the identification of failures we use case-based reasoning with a knowledge database, since the identification of failure sources (done by sensors) and mapping failures to appropriate reactions (done by actuators)

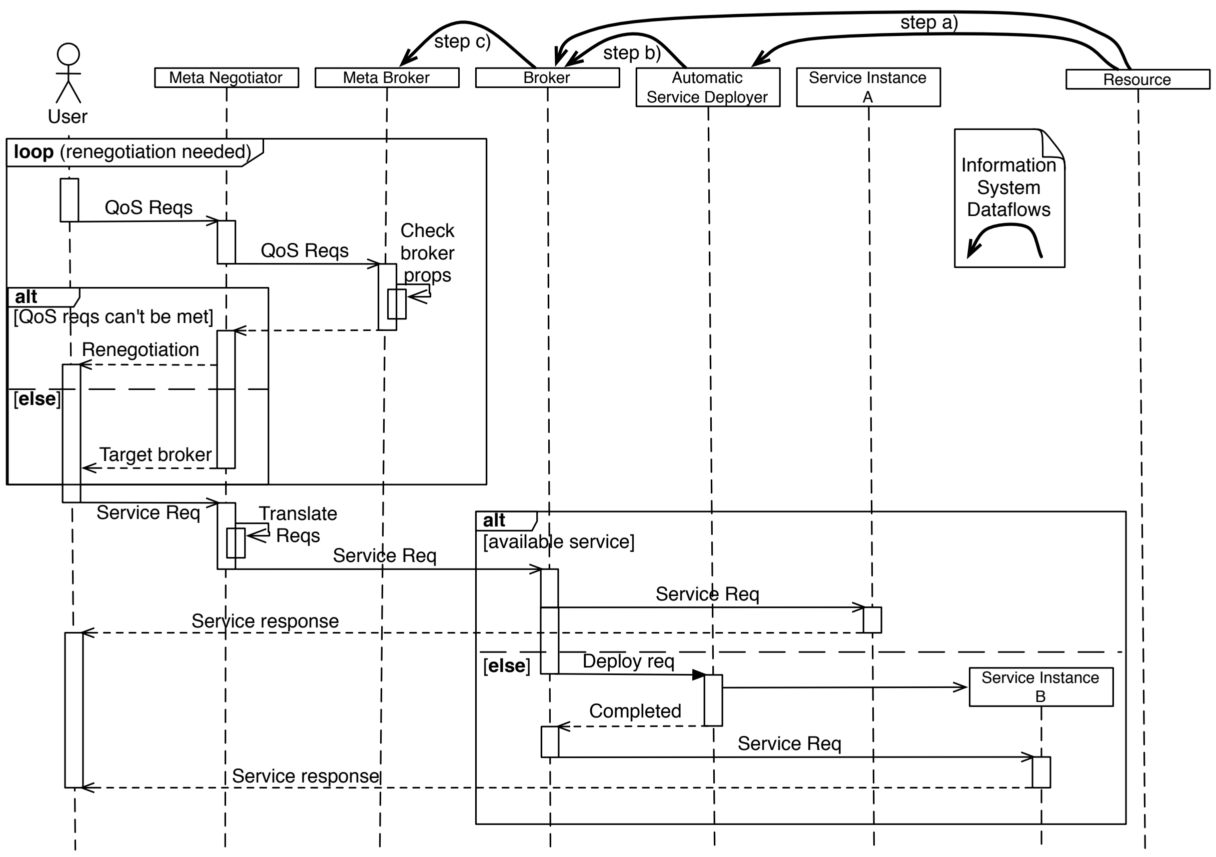

Figure 3. Component interactions in the SSV architecture.

are trivial tasks, and they are part of our ongoing work [1]. However the discussion of the knowledge database and layered notifications is out of scope of this paper, therefore here we mention three typical reactive actions we support in our SSV: namely *negotiation bootstrapping*, *broker breakdown* and *self-initiated deployment*. Negotiation bootstrapping occurs when the architecture needs to select a new negotiation strategy. Broker breakdowns are handled by the Meta-Broker component by initiating renegotiation on an already executed service call. Finally a service instance can guarantee the negotiated SLAs by deploying another service instance and redirecting the calls causing the unexpected service behaviour (more details on these failures and reactive actions can be read in [12]). An example software actuator used for the meta-negotiations is the VieSLAF framework [3], which bridges between the incompatible SLA templates by executing the predefined SLA mappings.

B. Dependencies between the SSV components

During the negotiation process the MB interacts with MN: it receives a service request with the service description and SLA terms, and looks for a deployed service reachable by some broker that is able to fulfil the specified terms. If a service is found, the SLA will be accepted and the and MN notified, otherwise the SLA will be rejected. If the service requirements are matched and only the terms cannot be fulfilled, it could continue the negotiation by modifying the terms and wait for user approval or further modifications.

The Information Collector (IC) component of MB stores the data of the reachable brokers and historical data of the previous submissions. It also communicates with the ASDs, and receives up-to-date data about the available services and predicted invocation times (that might also include service deployment before the actual invocation). This information shows whether the chosen broker is available, or how reliable its services are.

Finally service brokers could instruct ASDs to *deploy* a new service instance. However, deployments could also occur independently from the brokers as explained in the following. After these deployments the ASD has to *notify* the corresponding service brokers about the infrastructure changes. This notification is required, because the IC caches the state of the SBA for scalability. Thus even though a service has just been deployed on a new site, the broker will not direct service requests there. This is especially needed when the deployment was initiated to avoid an SLA violation (e.g. self-initiated deployment).

Table I
ROLES IN SSV ARCHITECTURE

Acronym	Role	Description
U	User	A person, who wants to use a service
MN	Meta-Negotiator	A component that manages Service-level agreements. It mediates between the user and the Meta-Broker, selects appropriate protocols for agreements; negotiates SLA creation, handles fulfilment and violation ([2]).
MB	Meta-Broker	Its role is to select a broker that is capable of executing/deploying a service with the specified user requirements ([13]).
B	Broker	It interacts with virtual or physical resources, and in case the required service needs to be deployed it interacts directly with the ASD.
ASD	Automatic Service Deployment	It installs the required service on the selected resource on demand ([9]).
S	Service	The service that users want to deploy and/or execute
R	Resource	Physical machines, on which virtual machines can be deployed/installed.

IV. EVALUATION OF THE SSV ARCHITECTURE WITH CLOUDSIM

In order to evaluate our proposed SSV solution, we use a typical biochemical application as a case study called TINKER Conformer Generator application [24], gridified and tested on production Grids. The application generates conformers by unconstrained molecular dynamics at high temperature to overcome conformational bias then finishes each conformer by simulated annealing and energy minimization to obtain reliable structures. Its aim is to obtain conformation ensembles to be evaluated by multivariate statistical modeling.

The execution of the application consists of three phases: The first one is performed by a generator service responsible for the generation of input data for parameter studies (PS) in the next phase. The second phase consist of a PS subworkflow, in which three PS services are defined for executing three different algorithms (dynamics, minimization and simulated annealing – we refer to these services and the generator service as TINKERALG), and an additional PS task that collects the outputs of the three threads and compresses them (COLL). Finally in the third phase, a collector service gathers the output files of the PS subworkflows and uploads them in a single compressed file to the remote storage (UPLOAD). These phases contain 6 services, out of which four are parameter study tasks that are executed 50 times. Therefore the execution of the whole workflow means 202 service calls. We set up the simulation environment for executing a similar workflow.

For the evaluation, we have created a general simulation environment, in which all stages of service execution in the SSV architecture can be simulated and coordinated. We have created the simulation environment with the help of the CloudSim toolkit [4] (that includes and extends GridSim). It supports modeling and simulation of large scale Cloud computing infrastructure, including data centers, service brokers and provide scheduling and allocations policies. Our general simulation architecture that builds both on GridSim and on CloudSim, can be seen in Figure 4. On the left-bottom part we can see the GridSim components used for the simulated Grid infrastructures, and on the right-bottom part we can find CloudSim components. Grid resources can be defined with different Grid types, they consist of more machines, to which workloads can be set, while Cloud resources are organized into Datacenters, on which Virtual machines can be deployed. Here service requests are called as cloudlets, which can be executed on virtual machines. On top of this simulated Grid and Cloud infrastructures we can set up brokers. Grid brokers can be connected to one or more resources, on which they can execute so-called gridlets (ie. service requests). Different properties can be set to these brokers and various scheduling policies can also be defined. A Cloud broker can be connected to a data center with one or more virtual machines, and it is able to create and destroy virtual machines during simulation, and execute cloudlets on these virtual machines. The Simulator class is a CloudSim entity that can generate a requested number of service requests with different properties, start and run time. It is connected to the created brokers and able to submit these requests to them (so is acts as a user or workflow engine). It is also connected to the Grid Meta-Broker Service through its web service interface and able to call its matchmaking service for broker selection.

We submitted the simulated workflow in three phases: in the first round 61 service requests for input generation, then 90 for executing various TINKER algorithms, finally in the third round 51 calls for output preparation. The simulation environment was set up similarly to the real Grid environment we used for testing the TINKER workflow application. Estimating the real sizes of these distributed environments, we set up four simulated Grids (GILDA, VOCE, SEEGRID and BIOMED [5]) with 2, 4, 6 and 8 resources (each of them had 4 machines). Out of the 202 jobs 151 had special requirements: they use the TINKER library available in the last three Grids, which means these calls need to be submitted to these environments, or to Cloud resources (with pre-deployed TINKER environments). The simulated execution time of the 150 parameter study services were set to 30 minutes, the first generator service to 90

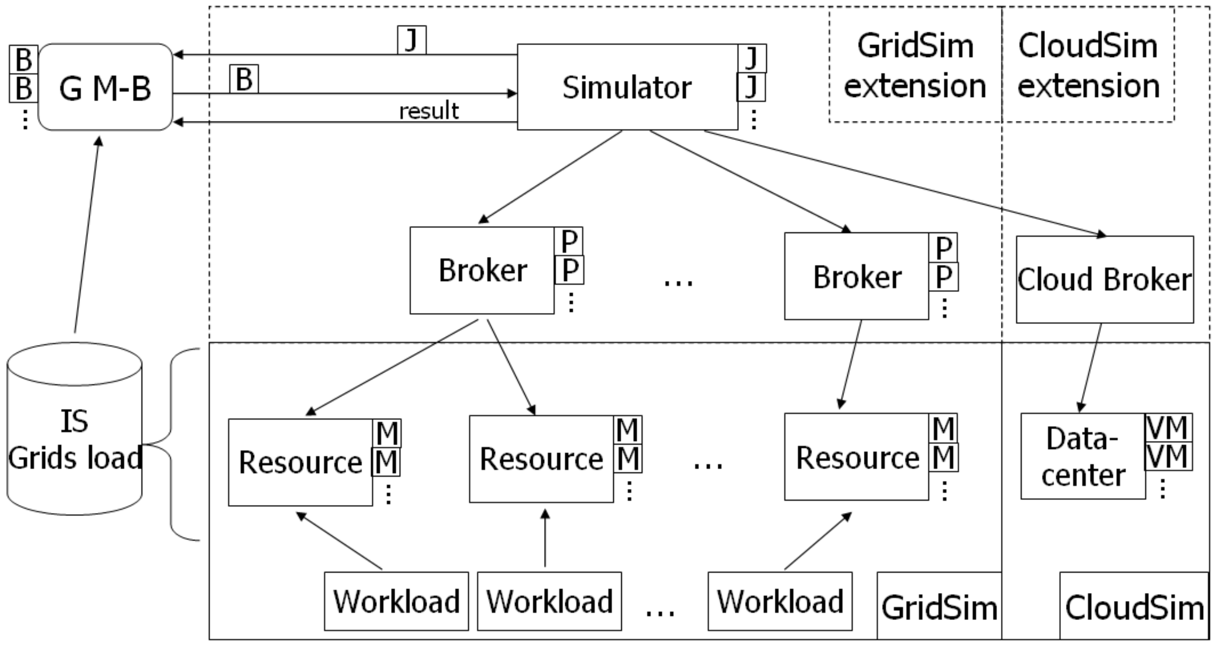

Figure 4. Simulation architecture with CloudSim.

minutes, and the other 51 were set to 10 minutes. All of the four brokers (set to each simulated Grid one-by-one) used random resource selection policy, and all the resources had background workload, for which the traces were taken from the Grid Workloads Archive (GWA) [25] (we used the GWA-T-11 LCG Grid log file). In our simulation architecture we used 20 nodes (called resources in the simulation), therefore we partitioned the logs and created 20 workload files (out of the possible 170 according to the number of nodes in the log). The sorting of the job data to files from the original log file were done continuously, and their arrival times have not been modified, and the run time of the jobs also remained the same. According to these workload files the load of the simulated environments are shown in Figure 5 (which are also similar to the load experienced on the real Grids). One Cloud broker has also been set up. It managed four virtual machines deployed on a data center with four hosts of dual-core CPUs. In each simulation all the jobs were sent to the Meta-Broker to select an available broker for submission. It takes into account the actual background load and the previous performance results of the brokers for selection. If the selected Grid broker had a background load that exceeded a predefined threshold value, it selected the Cloud broker instead.

Out of the 202 workflow services 150 use TINKER binaries (three different algorithms are executed 50 times). These requirements can be formulated in SLA terms, therefore each service of the workflow has an SLA request. If one of these requests are sent to the Cloud broker, it has to check if a virtual machine (VM) has already been created that is able to fulfil this request. If there is no such VM, it deploys one on-the-fly. For the evaluation we used three different Cloud broker configurations: in the first one four pre-deployed VMs are used – one for each TINKER algorithm (TINKERALG) and one for data collecting (capable for both COLL and UPLOAD, used by the last 51 jobs). In the second case we used only one pre-deployed VM, and deployed the rest on-the-fly, when the first call arrived with an SLA. Finally in the third case, when a VM received more then 15 requests the Cloud broker duplicated it (in order to minimize the overall execution time).

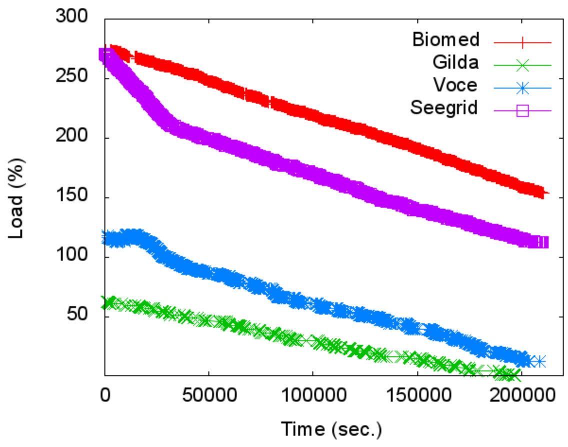

Figure 5. Workload of simulated Grids.

Regarding on-demand deployment, we have created 4 virtual appliances encapsulating the four different services our TINKER workflow is based on (namely TINKERALG, COLL and UPLOAD – we defined them in the beginning of this section). Then we have reduced the size of the created appliances with ASD's virtual appliance optimization facility. Finally we have deployed each service 50 times on an 8 node (32 CPU) Eucalyptus [16] cluster, and measured the interval between the deployment request and the service's first availability. Table II shows the measurement results for the TINKERALG, COLL and UPLOAD images. These latencies were also applied in the simulation environment within the Cloud broker.

Table II
DEPLOYMENT TIMES OF THE DIFFERENT SERVICES IN THE TINKER APPLICATION.

Service	Average deployment time	Standard deviation
GEN	8.2 sec	1.34 sec
TINKERALG	8.3 sec	1.48 sec
COLL	6.9 sec	0.84 sec
UPLOAD	6.9 sec	1.21 sec

In order to evaluate the performance of our proposed SSV solution we compare it to a general meta-brokering architecture used in Grid environments. Using this approach we created four simulations: in the first one we use only grid brokers by the Meta-Broker (denoted by MB in the figures) to reach grid resources of the simulated Grids. In the second, third and fourth case we extend the matchmaking of the Meta-Broker (in order to simulate the whole SSV): when the background load of the selected grid broker exceeds 113%, it selects the Cloud broker instead. In the second case the Cloud broker has four pre-deployed VMs (4VMs), while in the third case only one, and later creates three more as described before (1+3VMs), and in the fourth it has one pre-deployed and creates 7 more on demand (1+3+4VMs). In Figure 6 and 7 we can see the evaluation results denoting the average and detailed execution times of the service requests respectively.

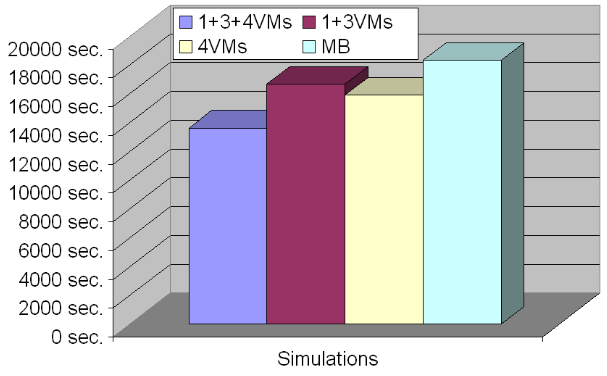

Figure 6. Average request run times.

From these results we can clearly see that the simulated SSV architecture overperforms the former (purely) Grid meta-brokering solution. Comparing the different deployment strategies we can see that on demand deployment introduces some overhead (4VMs was faster then 1+3VMs), but service duplication (1+3+4VMs) can enhance the performance and help to avoid SLA violations with additional VM deployment costs.

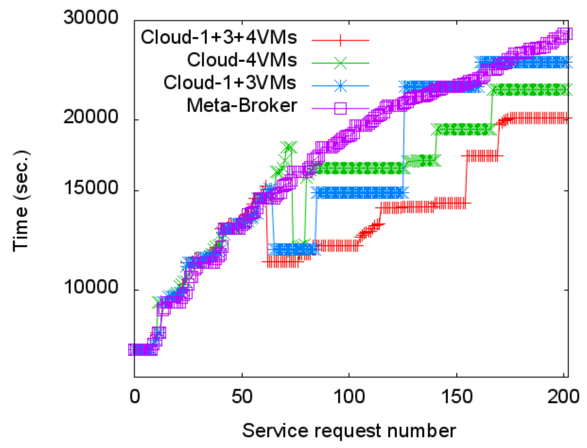

Figure 7. Detailed run times of requests.

V. CONCLUSION

In heterogeneous, distributed service-based environments such as Grids and Clouds, there is an emerging need for transparent, business-oriented autonomic service execution. In the future more and more companies will face the problem of unforeseen, occasional demand for a high number of computing resources. In this paper we have investigated how such problems could arise, and proposed a novel approach called Service-level agreement-based Service Virtualization (SSV). The presented general, conceptual architecture is built on three main components: the Meta-Negotiator responsible for agreement negotiations, the Meta-Broker for selecting the proper execution environment, and the Automatic Service Deployer for service virtualization and on-demand deployment. We have also discussed how the principles of autonomic computing are incorporated to the SSV architecture to cope with the error-prone virtualization environments. The proposed service virtualization architecture is validated in a simulation environment based on CloudSim, using a general biochemical application as a case study. The evaluation results clearly fulfill the expected utilization gains compared to a less heterogeneous Grid solution.

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 EGI-InSPIRE project (contract number RI-261323) and the Vienna Science and Technology Fund

(WWTF) under agreement ICT08-018, FoSII – Foundations of Self-governing ICT Infrastructures.

References

[1] I. Brandic, V. C. Emeakaroha, M. Maurer, S. Acs, A. Kertesz, G. Kecskemeti, S. Dustdar. LAYSI: A Layered Approach for SLA-Violation Propagation in Self-manageable Cloud Infrastructures. In *Proc. of the First IEEE International Workshop on Emerging Applications for Cloud Computing*, Seoul, Korea, 2010.

[2] I. Brandic, D. Music, S. Dustdar. Service Mediation and Negotiation Bootstrapping as First Achievements Towards Self-adaptable Grid and Cloud Services. In *Proceedings of Grids meet Autonomic Computing Workshop*. ACM. 2009.

[3] I. Brandic, D. Music, P. Leitner, S. Dustdar. VieSLAF Framework: Enabling Adaptive and Versatile SLA-Management. In *the 6th International Workshop on Grid Economics and Business Models 2009 (Gecon09)*, 2009.

[4] R. Buyya, R. Ranjan and R. N. Calheiros, Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities, in proc. of the 7th High Performance Computing and Simulation Conference, 2009.

[5] Enabling Grids for E-sciencE (EGEE) project website: http://public.eu-egee.org/

[6] 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.

[7] R. Howard and L. Kerschberg. A knowledge-based framework for dynamic semantic web services brokering and management. In *DEXA '04: Proceedings of the Database and Expert Systems Applications, 15th International Workshop*, pp. 174–178, IEEE Computer Society, 2004.

[8] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the grid. *Sci. Program.*, 13(4):265–275, 2005.

[9] G. Kecskemeti, P. Kacsuk, G. Terstyanszky, T. Kiss, T. Delaitre. Automatic service deployment using virtualisation. In *Proc. of 16th Euromicro International Conference on Parallel, Distributed and network-based Processing*. IEEE Computer Society. 2008.

[10] J.O. Kephart, D.M. Chess. The vision of autonomic computing. *Computer* . 36:(1) pp. 41-50, Jan 2003.

[11] A. Kertesz, G. Kecskemeti, and I. Brandic. An SLA-based resource virtualization approach for on-demand service provision. In *Proceedings of the 3rd international Workshop on Virtualization Technologies in Distributed Computing (Barcelona, Spain, June 15 - 15, 2009)*. ACM, New York, pp. 27-34, 2009.

[12] A. Kertesz, G. Kecskemeti, and I. Brandic. Autonomic Resource Virtualization in Cloud-like Environments. In *Technical Report, TUV-1841-2009-04*. Distributed Systems Group, Institute for Information Systems, Vienna University of Technology, 2009.

[13] A. Kertesz and P. Kacsuk. GMBS: A New Middleware Service for Making Grids Interoperable. *Future Generation Computer Systems*, Volume 26, Issue 4, pp. 542–553, 2010.

[14] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 2004. IEEE Computer Society.

[15] Z. Li and M. Parashar. An infrastructure for dynamic composition of grid services. In *GRID '06: Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, pages 315–316, Washington, DC, USA, 2006. IEEE Computer Society.

[16] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *CCGRID*, pages 124–131. IEEE Computer Society, 2009.

[17] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, and K. Krishnakumar. A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing. In *Proceedings of the 2005 European Grid Computing Conference (EGC 2005)*, February 2005.

[18] M. Parkin, D. Kuo, J. Brooke, and A. MacCulloch. Challenges in eu grid contracts. In *Proceedings of the 4th eChallenges Conference*, pp. 67–75, 2006.

[19] D. M. Quan and J. Altmann. Mapping a group of jobs in the error recovery of the grid-based workflow within sla context. *Advanced Information Networking and Applications, International Conference on*, 0:986–993, 2007.

[20] D. Reed, I. Pratt, P. Menage, S. Early, and N. Stratford. Xenoservers: Accountable execution of untrusted programs. In *In Workshop on Hot Topics in Operating Systems*, pages 136–141, 1999.

[21] M. Surridge, S. Taylor, D. De Roure, and E. Zaluska. Experiences with gria – industrial applications on a web services grid. In *E-SCIENCE '05: Proc. of the First International Conference on e-Science and Grid Computing*, pp. 98–105, 2005. IEEE Computer Society.

[22] H. N. Van, F. D. Tran, and J. Menaud. Autonomic virtual resource management for service hosting platforms. In *Proceedings of the ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 1–8, 2009.

[23] C. A. Yfoulis, and A. Gounaris. Honoring SLAs on cloud computing services: a control perspective. In *Proceedings of the European Control Conference*, 2009.

[24] TINKER Conformer Generator workflow: http://www.lpds.sztaki.hu/gasuc/index.php?m=6&r=12

[25] The Grid Workloads Archive website: http://gwa.ewi.tudelft.nl

[26] S. Venugopal, R. Buyya, and L. Winton. A grid service broker for scheduling e-science applications on global data grids. *Concurrency and Computation: Practice and Experience*, 18(6):685–699, 2006.