2010 34th Annual IEEE Computer Software and Applications Conference Workshops

LAYSI: A Layered Approach for SLA-Violation Propagation in Self-manageable Cloud Infrastructures

Ivona Brandic, Vincent C. Emeakaroha,
Michael Maurer, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
Vienna, Austria
{*ivona,vincent,maurer,dustdar*}*@infosys.tuwien.ac.at*

Sandor Acs, Attila Kertesz, Gabor Kecskemeti
MTA SZTAKI, P.O. Box 63
1518 Budapest, Hungary
{*acs,attila.kertesz,kecskemeti*}*@sztaki.hu*

***Abstract*—Cloud computing represents a promising computing paradigm where computing resources have to be allocated to software for their execution. Self-manageable Cloud infrastructures are required to achieve that level of flexibility on one hand, and to comply to users' requirements specified by means of Service Level Agreements (SLAs) on the other. Such infrastructures should automatically respond to changing component, workload, and environmental conditions minimizing user interactions with the system and preventing violations of agreed SLAs. However, identification of sources responsible for the possible SLA violation and the decision about the reactive actions necessary to prevent SLA violation is far from trivial. First, in this paper we present a novel approach for mapping low-level resource metrics to SLA parameters necessary for the identification of failure sources. Second, we devise a layered Cloud architecture for the bottom-up propagation of failures to the layer, which can react to sensed SLA violation threats. Moreover, we present a communication model for the propagation of SLA violation threats to the appropriate layer of the Cloud infrastructure, which includes negotiators, brokers, and automatic service deployer.**

***Keywords*-Cloud Computing; SLA management; autonomic computing;**

I. Introduction

Cloud computing can be defined as the convergence and evolution of several concepts from virtualization, distributed application design, Grid and enterprise IT management to enable a more flexible approach for deploying and scaling applications [3], [19], [18]. Service provisioning in the Cloud is based on Service Level Agreements (SLAs) representing a contract signed between the customer and the service provider including the non-functional requirements of the service specified as Quality of Service (QoS). SLA considers obligations, service pricing, and penalties in case of agreement violations.

Flexible and reliable management of SLA agreements is of paramount importance for both, Cloud providers and consumers. On one hand, preventions of SLA violations ahead of time can avoid unnecessary penalties a provider has to pay in case of violations. Sometimes, simple actions like migrating VMs to available nodes can prevent SLA violations. On the other hand, based on flexible and timely reactions to possible SLA violations, interactions with the users can be minimized increasing the chance for Cloud computing to take roots as a flexible and reliable form of on demand computing.

However, current Cloud infrastructures lack appropriate mechanisms for the self-management of SLAs. Large body of work concentrates on monitoring of resource metrics of Cloud resources, which however cannot be easily mapped to SLA parameters [1], [2]. There is also considerable body of work done in the area of SLA management in general, which however is not related to Cloud infrastructures [15]. Thus, very little work has been done on identifications of SLA violations ahead of time, before they happen. Furthermore, there is a lack of appropriate mechanisms to identify which components of the Cloud infrastructure have to react in order to avert SLA violations.

In this paper we present *LAYSI - A Layered Approach for Prevention of SLA-Violations in Self-manageable Cloud Infrastructures*, which is embedded into the *FoSII project (Foundations of Self-governing ICT Infrastructures)* [8], an ongoing research project developing self-adaptable Cloud services. The *LAYSI* framework represents one of the building blocks of the *FoSII* infrastructure facilitating future SLA violation detection and propagation of the reactive actions to the appropriate layer of the Cloud infrastructure. We discuss a layered Cloud architecture utilizing hierarchically and loosely coupled components like negotiator, broker or automatic service deployer. For the decision making we use knowledge databases proposing reactive actions by utilizing case based reasoning - a process of solving problems based on past experience. Based on the novel communication model we present how possible SLA violations can be identified and propagated to the layer of the Cloud infrastructure, which can execute appropriate reactive actions in order to advert SLA violations.

The main contributions of this paper are: (i) discussion on the solution for mapping low-level resource metrics to SLA parameters; (ii) description of the integrated SLA-aware Cloud architecture suitable for the propagation of the SLA violation threats; (iii) concept for the realization

978-0-7695-4105-1/10 $26.00 © 2010 IEEE
DOI 10.1109/COMPSACW.2010.70

of the knowledge database using case based reasoning; (iv) architecture for the autonomic management and propagation of SLA violation threats.

The rest of this paper is organized as follows: Section II presents the related work. In Section III we present the architecture for the autonomic management of Cloud services and the approach for mapping low-level resource metrics to SLA parameters. In Section IV we discuss the LAYSI architecture. In particular we discuss the concept of knowledge databases and the SLA manager responsible for the autonomic management of SLA violation threats. Section V presents our conclusions and describes the future work.

II. Related Work

We classify related work into (i) monitoring of Cloud/Grid/Web services [1], [2]; (ii) SLA management including QoS management [9], [6], [14]; (iii) and self-management of Cloud/Grid/SOA services [15]. Since there is very little work on monitoring, SLA management, and self-mamagement in Cloud systems we look particularly into related areas, i.e., Grid and SOA based systems.

GridRM is an open-source project trying to provide a unified way of accessing different monitored data sources. Every domain needs a Java-based gateway to collect and normalize events from the local monitoring system. However, it does not provide mapping of monitored values to SLA parameters [1].

Frutos et al. [9] discuss the main approach of the EU project BREIN [6]: to develop a framework, which extends the characteristics of computational Grids by driving their usage into new target areas in the business domain. BREIN deals with the provision of the basic infrastructure these new business models need: enterprise system interoperability, flexible relationships, dynamicity in business processes, security mechanisms, and enhanced SLA and contract management. However, BREIN applies SLA management to Grids, whereas we target SLA management in Clouds. Koller et al. [14] discuss autonomous QoS management using a proxy-like approach. The implementation is based on WS-Agreement. Thereby, SLAs can be exploited to define certain QoS parameters that a service has to maintain during its interaction with a specific customer. However, their approach is limited to Web services and does not consider requirements of Cloud Computing infrastructures like scalability.

Based on the defined workflow adaptations as MAPE[1] decision making [15], Lee et al. discuss the application of autonomic computing to the adaptive management of Grid workflows.

III. FoSII Infrastructure

In this section we present an overview of the *FoSII* infrastructure and its relation to the *LAYSI* framework. In particular we describe the mapping of low level metrics to high level SLAs. Thereafter, we discuss the SLA-based layered Cloud infrastructure.

[1]Monitoring, Analysis, Planning, Execution

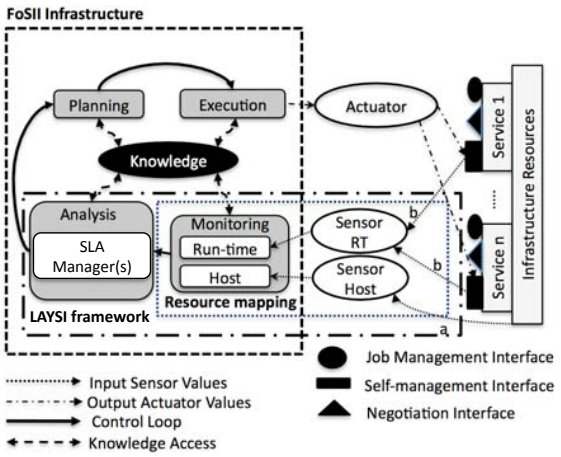

Figure 1. FoSII infrastructure

A. FoSII overview

The FoSII infrastructure is used to manage self-adaptable Cloud services following the MAPE lifecycle. Each FoSII service implements three interfaces: (i) negotiation interface necessary for the establishment of SLA agreements, (ii) job-management interface necessary to start the job, upload data, and similar job management actions, and (iii) self-management interface necessary to devise actions in order to prevent SLA violations.

The self-management interface shown in Figure 1 is implemented by each Cloud service and specifies operations for sensing changes of the desired state and for reacting to those changes. The host monitor sensors continuously monitor the infrastructure resource metrics (input sensor values arrow *a* in Figure 1) and provide the autonomic manager with the current resource status. The run-time monitor sensors sense future SLA violation threats (input sensor values arrow *b* in Figure 1) based on resource usage experiences and predefined threat thresholds. The mapping between the sensed host values and the values of the SLA parameters is described next.

B. Mapping of Low level Metrics to High-level SLAs

In order to explain our mapping approach we consider the Service Level Objectives (SLOs) as shown in Table I including incoming bandwidth, outgoing bandwidth, storage, and availability.

As shown in Figure 1 we distinguish between *host monitor* and *runtime monitor*. Resources are monitored by the *host monitor* using arbitrary monitoring tools (e.g. Ganglia [17]). Resource metrics include, e.g., down-time, up-time, available storage. Based on the predefined mappings stored in a database, monitored metrics are periodically mapped to

SLA Parameter	Value
Incoming Bandwidth (IB)	$>$ 10 Mbit/s
Outgoing Bandwidth (OB)	$>$ 12 Mbit/s
Storage (St)	$>$ 1024 GB
Availability (Av)	$\geq$ 99%

Table I
SAMPLE SLA PARAMETER OBJECTIVES

the SLA parameters. An example SLA parameter is service availability Av, (as shown in Table I), which is calculated using the resource metrics $downtime$ and $uptime$ and the mapping rule looks like the following:

$Av = (1 - downtime/uptime) * 100$

The mapping rules are defined by the provider using appropriate Domain Specific Languages (DSLs). These rules are used to compose, aggregate, or convert the low-level metrics to form the high-level SLA parameter including mappings at different complexity levels, e.g., $1 : n$ or $n : m$. The concept of detecting future SLA violation threats is designed by defining a more restrictive threshold than the SLA violation threshold known as threat threshold. Thus, calculated SLA values are compared with the predefined threat threshold in order to react before SLA violations happen. The generation of *threat thresholds* is far from trivial and is part of our ongoing work including sophisticated methods for the system state management as described in Section IV-A.

As described in [7] we implemented a highly scalable framework for mapping Low Level Resource Metrics to High Level SLA Parameters (LoM2HiS framework) facilitating the exchange of large numbers of messages. We designed and implemented a communication model based on the Java Messaging Service (JMS) API, which is a Java Message Oriented Middleware (MOM) API for sending messages between two or more clients. We use Apache ActiveMQ as a JMS provider that can manage the sessions and queues.

Once possible SLA violation threats are detected, reactive actions are taken in order to prevent real SLA violations. In the following we discuss the layered Cloud architecture followed by the discussion of the novel concept for the SLA violation threat propagation.

C. SLA-based Layered Cloud Infrastructures

In the following we present a unified service architecture that builds on three main areas [11]: agreement negotiation, brokering, and service deployment using virtualization. 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. The architectures' components are loosely coupled using SLAs between the components. Thus, in case of failures components can be exchanged easily by renegotiating with another instance, e.g. another broker.

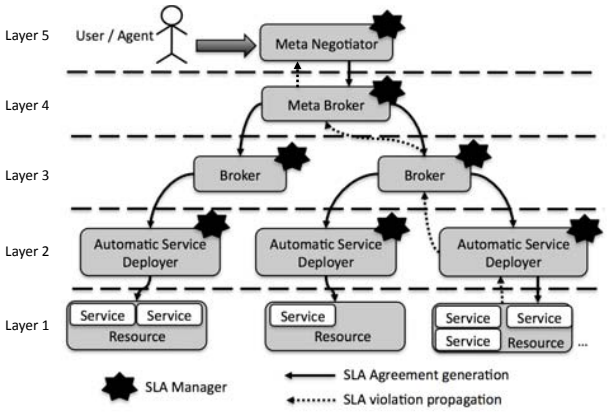

Figure 2. LAYSI infrastructure

Figure 2 shows our proposed general architecture. In the following we discuss the actors of the proposed architecture:

- **User:** A person, who wants to use a service, an agent or software application acting on behalf of a user.
- **Meta Negotiator:** A component that manages SLAs. It mediates between the user and the meta-broker, selects services, and resources considering prescribed protocols, negotiation strategies, and security restrictions as described in [5].
- **Meta Broker:** Its role is to select a broker that is capable of deploying a service with the specified user requirements as described in [12].
- **Broker:** It interacts with virtual or physical resources, and in case the required service needs to be deployed it interacts directly with the Automatic Service Deployer (ADS) [13].
- **Automatic Service Deployer:** It installs the required service on the selected resource on demand as described in [10].
- **Service:** The service that users want to deploy and/or execute is described using the concept of *virtual appliances*.
- **Resource:** Physical machines, network, or storage elements on which virtual machines can be deployed/installed.

The SLA negotiation is done as following: The *User* starts a negotiation for executing a service with certain QoS requirements. Then, the *Meta negotiator* asks the *Meta broker*, if it could execute the service with the specified requirements including required negotiation or security protocols. The *Meta broker* matches the requirements to the properties of the available *Brokers* and replies with an acceptance or a different offer for renegotiation. The aforementioned steps may continue for renegotiations until both sides agree on the terms (to be written to an SLA document) following the specific negotiation strategy or auction. Thereafter, the *User* calls the service with the *Service Description (SD)* and

the agreed *SLA*. SDs describe a master image by means of a self-contained software stack (OS, middleware, applications, data, and configuration) that fully captures the functionality of the component type. Moreover, the SD contains information and rules necessary to automatically create service instances from a single parametrized master.

Meta-negotiator passes the SD and the possibly transformed SLA (using a protocol the selected broker understands) to the Meta broker. The meta broker calls the selected Broker with the SLA and a possibly translated SD (to the language of the Broker). The Broker executes the service with respect to the terms of the SLA. The ASD monitors the states of the virtual resources and deploys services, as already stated in Figure 1. As shown in Figure 2 SLA generation is done top-down as already described. Management of the SLA threat violation is done bottom-up on behalf of the *SLA manager*, which is implemented by each component of the SLA layered architecture.

Table II shows the implementation choices for the layered Cloud architecture and the possible reactive actions each layer can perform. We use *Meta Negotiator* [4], *Meta Broker* [12], *GTBroker* [13] and *Automatic Service Deployer* [10] components, which have already been used and evaluated to build an SLA-based resource virtualization environment for on-demand service provision [11].

IV. LAYSI: A Layered Approach for SLA-Violation Propagation

In the following we present an architecture for the propagation of the sensed critical SLAs, which might be violated in the future. In particular we focus on two components: the *knowlege database* providing reactive action for possible detected SLA violation threats considering SLA threat thresholds and the current system status (Section IV-A), and the *SLA manager* propagating the sensed SLA violation threats to the appropriate layer of the infrastructure for preventive actions (Section IV-B).

A. Knowlegde DBs

For the decision making we use knowledge databases proposing the reactive actions by utilizing case based reasoning. *Case Based Reasoning (CBR)* is the process of solving problems based on past experience. It tries to solve a *case* (a formatted instance of a problem) by looking for similar cases from the past and reusing the solutions of these cases to solve the current one. In general, a typical CBR cycle consists of the following phases assuming that a new case has just been received: (i) retrieve the most similar case or cases to the new one, (ii) reuse the information and knowledge in the similar case(s) to solve the problem, (iii) revise the proposed solution, (iv) retain the parts of this experience likely to be useful for future problem solving.

As shown in Figure 3, a complete case consists of (a) the ID of the application being concerned (line 2, Figure 3); (b) the initial case measured by the monitoring component

```
1.  (
2.   (App,  1),
3.    (
4.    ((Incoming Bandwidth, 12.0),
5.     (Outgoing Bandwidth, 20.0),
6.     (Storage, 1200),
7.     (Availability, 99.5),
8.     (Running on PMs, 1)),
9.     (Physical Machines, 20)
10.   ),
11.  "Increase Incoming Bandwidth share by 5%",
12.  (
13.   ((Incoming Bandwidth, 12.6),
14.    (Outgoing Bandwidth, 20.1),
15.    (Storage, 1198),
16.    (Availability, 99.5),
17.    (Running on PMs, 1)),
18.    (Physical Machines, 20)
19.   ),
20.  0.002
21. )
```

Figure 3. CBR example

and mapped to the SLAs consisting of the SLA parameter values of the application and global Cloud information like number of running virtual machines (lines 4-10, Figure 3); (c) the executed action (line 11, Figure 3); (d) the resulting case measured some time interval later (lines 12-18, Figure 3) as in (b); and (e) the resulting utility (line 20, Figure 3).

We distinguish between two working modes of the knowledge DB: *active* and *passive*. In the active mode system states and SLA values are periodically stored into the DB. Thus, based on the observed violations and correlated systems states, cases are obtained as input for the knowledge DB. Furthermore, based on the utility functions, we evaluate the quality of the reactive actions and generate threat thresholds. In the passive mode notification are sent by the SLA manager (or LoM2HiS framework in case the layer=1) as described in Section III-B.

However, the output of the DB does not tell anything about how to react to the proposed actions as for example the suggested action *Increase Incoming Bandwidth share by 5%* depicted in Figure 3. An obvious reaction would be to increase the bandwidth share by the particular resource. However, if this is not possible due to resource restriction, current load, and services with competing priorities, the suggested action has to be propagated to the next layer. Then, in the next layer ASD could migrate the virtual appliance as specified in Table II (reactive actions of ASD: suspend, shut-down, and migrate VAs). This propagation can be continued until a specific layer is able to react to the particular suggested action.

In the following we discuss how the SLA manager can propagate the desired changes to the particular layer of the infrastructure, which can take appropriate actions.

B. SLA Manager

The SLA manager implements the component's self-management interfaces and invokes the self-management

Layer	Sample Implementation	Actions
Meta Negotiator	Meta Negotiator in Brandic et al. [4]	start new meta-negotiation
Meta Broker	Meta-Broker in Kertesz et al. [12]	allocate new broker
Broker	GTBroker in Kertesz et. al. [13]	start, stop, and suspend ASD instances
Automatic Service Deployment (ASD)	ASD in Kecskemeti et al. [10]	suspend, shut-down, and migrate virtual appliances (VAs)

Table II
IMPLEMENTATION CHOICES AND THE POSSIBLE REACTIVE ACTIONS OF THE PARTICULAR LAYER

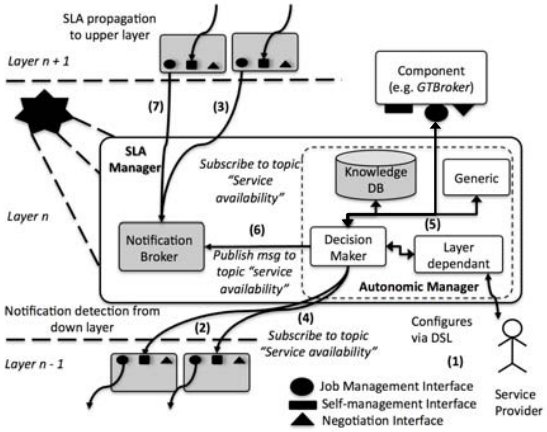

Figure 4. SLA Manager

interface of the upper layer in case the announced SLA violation threat cannot be solved by the layer's SLA manager. The SLA manager considers two main parts: the *Notification Broker* implemented using the WS-Notification mechanism and the *Autonomic Manager* managing the access to the knowledge DB, accessing the job management interfaces of the component, and making the decision whether the SLA violation threats can be handled by the layer or not.

Autonomic Manager: The Autonomic Manager receives notifications from the lower layer or from the *LoM2HiS framework* in case the layer=1. Thereafter, the knowledge DB is accessed in order to receive states, which should be achieved. The decision maker consists of two parts: the layer independent part managing the DB access and notifications, i.e., the *generic part* and the *layer dependent* part, which implements access to the components' interfaces e.g., in order to take reactive actions. The user can customize the autonomic manager, e.g., taking into account the job management interface of the component by modifying the component's dependent part using Domain Specific Languages (DSLs) (step 1, Figure 4). Customization could include for example utilization of the reactive actions of a component as shown in Table II. The notification mechanism for the propagation of SLA violation threats is explained next.

Notification Broker: The *SLA manager* employs the *WS-Notification* mechanism [21], which provides a set of standard interfaces to use the notification design pattern with services. WS-Notification is defined by three specifications: (i) *WS-Topics*; (ii) *WS-BaseNotification*; and (iii) *WS-BrokeredNotification*. The *WS-Topics* present a set of items of interest for subscription. Topics are very versatile and highly customizable. They even allow us to create topic trees, where a topic can have a set of child topics. *WS-BaseNotification* defines the standard interfaces used by the notification producer and consumer. The *WS-BrokeredNotification* delivers notification from the producer to the consumer through an intermediate entity (broker). The SLA manager is also equipped with a queueing network. The queues are used to temporarily store the notifications for processing. With the queueing networks, there is the possibility of selectively processing higher priority notifications against the lower priority ones.

Decision makers can subscribe different topics as shown in Figure 4, step 2 and 3. Once the SLA violation threat is detected the autonomic manager tries to find a reactive action by accessing the DB utilizing case based reasoning (step 5, Figure 4). The decision components decides whether the SLA violation threats can be deferred. If the SLA violation threats cannot be deferred at that certain layer, the SLA violation threats are propagated by publishing a message to the specific topic, e.g., to topic *service availability* as shown in step 6, Figure 4. Thereafter, all listeners (i.e., components of the layer $n+1$) are notified, step 7, Figure 4. The topics are organized in a hierarchical way considering the learning function of the CBR database. Based on the observed sensed violations, reactive actions, and the utility of the reactive action we define dependencies of the reactive actions which can be reflected in the topic hierarchy. The development of the advanced techniques for the automatic definition and utilizations of the topics hierarchies is subject of our ongoing work.

The LoM2HiS framework publishes monitored SLA parameters as a specific message of a WS-Topic. Thereafter, preventive actions of the SLA violation threats should be notified and handled at the ASD layer. In case the SLA violation threats can not be handled at layer n, the SLA manager publishes the problem to layer $n+1$. In the worst case, this propagation continues to the top level, i.e., the Meta Negotiator, which informs the user about the problem for a possible renegotiation or aborting the service execution.

V. Conclusion and Future Work

In this paper we presented how possible and costly SLA violations can be prevented by utilizing a layered SLA based Cloud infrastructure. Based on the novel approach for mapping low-level resource metrics to SLA parameters we can identify possible SLA violations. We devised a layered Cloud architecture for the bottom-up propagation of failures to the layer, which can react to sensed SLA violation threats. Moreover, we presented a communication model for the propagation of SLA violation threats to the appropriate layer of the Cloud infrastructure, which includes meta-negotiators, brokers, and automatic service deployer.

In the future we will integrate learning functions into the CBR databases in order to identify whether the propagation had a positive impact on the prevention of SLA violations. We plan to integrate trade-off analysis to examine how costly the interventions for the possible future SLA violations are instead of just reacting to occurred violations.

Acknowledgment

The work described in this paper was partially supported by the Vienna Science and Technology Fund (WWTF) under grant agreement ICT08-018 Foundations of Self-governing ICT Infrastructures (FoSII), and by the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

References

[1] M. Baker and G. Smith. *GridRM: A resource monitoring architecture for the grid. Lecture Notes in Computer Science*, Vol. 2536, pp. 268-273, 2002

[2] W. Fu and Q. Huang. *GridEye: A service-oriented grid monitoring system with improved forecasting algorithm.* GCCW'06, pp. 5-12, 2006.

[3] 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(6) pp. 599-616, June 2009.

[4] I. Brandic, D. Music, S. Dustdar. *Service Mediation and Negotiation Bootstrapping as First Achievements Towards Self-adaptable Grid and Cloud Services.* GMAC09 in conjunction with ICAC09, Barcelona, Spain, June 15-19, 2009.

[5] I. Brandic, S. Venugopal, M. Mattess, and R. Buyya.*Towards a Meta-Negotiation Architecture for SLA-Aware Grid Services*. SENOPT08, in conjunction with International Conference on High Performance Computing 2008 (HiPC 2008), Bangalore, India, December 17 - 20, 2008.

[6] Brein Project (Business objective driven reliable and intelligent Grids for real business), http://www.eu-brein.com 2009

[7] V. C. Emeakaroha, I.Brandic, M. Maurer, S. Dustdar. *Low Level Metrics to High Level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in Cloud environments.* The 2010 High Performance Computing and Simulation Conference (HPCS 2010) June 28-July 2, 2010, Caen, France.

[8] Foundations of Self-governing ICT Infrastructures (FoSII), http://www.infosys.tuwien.ac.at/linksites/FOSII

[9] H.M. Frutos and I. Kotsiopoulos. *BREIN: Business Objective Driven Reliable and Intelligent Grids for Real Business*. International Journal of Interoperability in Business Information Systems, 3(1) 2009.

[10] G. Kecskemeti, P. Kacsuk, G. Terstyanszky, T. Kiss, T. Delaitre. *Automatic Service Deployment Using Virtualisation*. 16th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2008), Toulouse, France, 13-15 February 2008.

[11] A. Kertesz, G. Kecskemeti, I. Brandic. *An SLA-based Resource Virtualization Approach for On-demand Service Provision.* VTDC 2009, In conjunction with ICAC09, Barcelona, Spain, June 15-19, 2009.

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

[13] A. Kertesz, G. Sipos, P. Kacsuk. *Multi-Grid Brokering with the P-GRADE Portal.* In Post-Proceedings of the Austrian Grid Symposium (AGS'06), pp. 166-178, OCG Verlag, Austria, 2007.

[14] B. Koller, L. Schubert. *Towards autonomous SLA management using a proxy-like approach.* Multiagent Grid Syst. Vol.3, 2007.

[15] K. Lee, R. Sakellariou, N. W. Paton, A. A. A. Fernandes. Workflow *Adaptation as an Autonomic Computing Problem*. WORKS'07 pages 29-34. in conjunction with HPDC 2007, Monterey, California, USA, 2007.

[16] G. Lin, G. Dasmalchi, J. Zhu. *Cloud Computing and IT as a Service: Opportunities and Challenges*. IEEE International Conference on Web Services (ICWS08), Beijing China, 23-26 Sept. 2008.

[17] M.L Massie, B.N Chun and D.E Culler. *Ganglia distributed monitoring system: design implementation, and experience.* Parallel Computing, Vol. 30 pp. 817-840, 2004.

[18] D. Nurmi, R. Wolski, Ch. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov. *The Eucalyptus Open-source Cloud-computing System*. Proceedings of Cloud Computing and Its Applications 2008, Chicago, Illinois, October 2008.

[19] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. B.-Y., W. Emmerich, F. Galan. *The RESERVOIR Model and Architecture for Open Federated Cloud Computing.*, IBM Journal of Research and Development, 53(4) (2009)

[20] R. Wolski, N.T. Spring and J. Hayes. *The network weather service: A distributed resource performance forecasting service for metacomputing.* In Journal of Future Generation Computing Systems, Vol. 15, pp. 757-768, 1999.

[21] WS-Notification, http://www.ibm.com/developerworks/webservices/library/specification/ws-notification