Scheduling and Monitoring of Internally Structured Services in Cloud Federations

Lars Larsson, Daniel Henriksson, Erik Elmroth
Department of Computing Science and HPC2N
Umeå University, Umeå, Sweden
Email: {larsson, danielh, elmroth}@cs.umu.se

***Abstract*—Cloud infrastructure providers may form Cloud federations to cope with peaks in resource demand and to make large-scale service management simpler for service providers. To realize Cloud federations, a number of technical and managerial difficulties need to be solved. We present ongoing work addressing three related key management topics, namely, specification, scheduling, and monitoring of services. Service providers need to be able to influence how their resources are placed in Cloud federations, as federations may cross national borders or include companies in direct competition with the service provider. Based on related work in the RESERVOIR project, we propose a way to define service structure and placement restrictions using hierarchical directed acyclic graphs. We define a model for scheduling in Cloud federations that abides by the specified placement constraints and minimizes the risk of violating Service-Level Agreements. We present a heuristic that helps the model determine which virtual machines (VMs) are suitable candidates for migration. To aid the scheduler, and to provide unified data to service providers, we also propose a monitoring data distribution architecture that introduces cross-site compatibility by means of semantic metadata annotations.**

I. Introduction

Cloud computing has the potential to offer cost-efficient and seemingly unlimited computational capacity to resource consumers, and more importantly, to deal seamlessly with unexpected spikes in resource consumption that would be unmanageable for in-house hosting alternatives. The problem of maintaining sufficient resources is transferred from the resource consumers to Cloud *Infrastructure Providers* (IPs). We refer to the consumers of Cloud infrastructure as *Service Providers* (SPs), which typically are companies who in turn offer services to end users. *Service-Level Agreements* (SLAs) specify the terms under which the SP provisions resources from the IP and at what cost, and define economical penalties if the IP fails to deliver accordingly.

IPs can collaborate on workload sharing and resource subcontracting to easier cope with spikes in resource consumption or other unexpected events that affects hosting of services. Such collaboration may exploit pricing differences at Cloud IPs which can yield savings, even for a low amount of requested resources [1]. We use the same definition for *Cloud federations* and *framework agreements* as in [2], namely that Cloud federations allow IPs to subcontract resources at remote Cloud sites when local resources are running low, as governed by bilateral framework agreements. The SP needs not be aware of such subcontracting and only interacts with the original IP. *Cloud bursting* can be seen as a special case of federation where resources are only provisioned by one party from the other, usually by a private Cloud from a public provider. Alternatively, an SP may directly host a service across several IPs. As in [3], we refer to this as a *multi-provider hosting* and consider it to be separate from Cloud federations. In multi-provider hosting, management and service orchestration across several sites is managed by the SP. In Cloud federations, the IP manages provisioning and monitoring of remote resources on behalf of the SP. IP-level management of e.g. elasticity and SLAs in Cloud federations [4] or federation/multi-hosting hybrids [3] is currently under research.

In this paper, we present ongoing work related to solving core management issues that arise specifically in Cloud federations. Specifying service structure and placement constraints affords the SP a sufficient amount of control over service deployment in Cloud federations. Schedulers must take this information into account when determining placement for each service component, and may use migration as a tool to optimize placement according to some management objective. Once a component has been placed and is executing, its state must be monitored to make placement optimization possible. Our contributions are the following:

- we define a hierarchical graph structure for service representation and intra-service rule specification which impacts scheduling within the Cloud federation,
- we present a scheduling model and heuristic that optimizes VM placement via local and remote migration, and
- we present a semantic monitoring data distribution architecture, which provides interoperability between different Cloud infrastructure monitoring systems.

The remainder of the paper is organized as follows. Section II briefly describes the design principles and the features that motivate our work. Section III presents how a graph may be used to represent structured services with rules concerning component placement and includes an example thereof. In Section IV, we present a model and heuristic for a scheduler that takes placement constraints into account for local and remote placement of VMs in the Cloud federation. Section V introduces an architecture of a system aimed to provide compatibility for disparate monitoring systems via employing semantic metadata to bridge the differences. The paper is concluded in Section VII.

II. Design Principles and Motivating Features

In this section, we briefly describe the design principles and features that motivate our work. We formulate the principle of *location unawareness* based on [5] and [6] such that it states that *neither* the management system *nor* the VMs should be needlessly aware of current VM placement. From the management point of view, this means that e.g. the scheduler is perfectly aware of whether a given VM is placed at a local host or at a remote site R, but it does not know *which particular host* at R hosts the VM (and it cannot request to change this placement). The VM may even have been delegated to another partner site by R without notifying the original IP.

From the VM point of view, location unawareness implies that the VM is not aware of its current hosting within the Cloud federation, including its location in the network. Thus, virtualized overlay networks must span across sites and allow VMs to keep all private and public IP addresses, even during migration from one site to another. Offering such networking functionality is the topic of ongoing research [6] and currently not offered by any commercial vendors.

Data and computation provisioning in federated Clouds raises concerns regarding locality, both from a performance and a legislative point of view [7], [8]. To ensure that resources are provisioned satisfactorily while retaining location unawareness, *affinity* and *anti-affinity* rules may be specified. We use the same definition of affinity as [9] i.e. to denote a set of placement constraining relationships between sets of related VMs. We use the term *AA-constraints* where both affinity and anti-affinity are applicable, and each term alone if something applies only to either affinity or anti-affinity.

Without loss of generality, we consider three levels of *AA*-constraints, namely host, (Cloud) site, and geographical region. For an affinity level L, if VM types A and B are in the relation, a scheduler must place all instances of these types so that placement restrictions are adhered to, e.g. instances must be placed on the same host machine or at the same site if this is the specified affinity relation. Conversely, anti-affinity requires that instances of VM types may *not* be placed on the same level, e.g. on the same host or at the same site. Using several *AA*-constraints, it is possible to restrict placement such that, e.g., all VMs must be placed on different hosts, avoid a certain competitor site, and may never be migrated to or placed in a region where certain legislation applies.

III. Service Representation

Some model is required to allow the SP to specify both the structure of the service and *AA*-constraints. We propose that hierarchical directed acyclic graphs (DAGs) are suitable service representations. The reasons are twofold: (a) there is an implied or explicitly stated structure between resources, e.g. between attached storage units and computational resources (parent-child relationship); and (b) *AA*-constraints may apply only to certain related service subsets (sibling relationship). In our formulation, trees are insufficient since a node may require more than one parent, for example if a VM is part of two otherwise disjoint internal networks.

Table I
Node types used to define the structure of a service.

Node type	Abbr.	Description
Service Root		Common ancestor for all service components.
Compute Resource	C	Compute resource, which can be connected to networks and storage units.
AA-constraint	A	Metadata for use within a scheduler to determine placement according to affinity and anti-affinity rules. *Scope* may either be *type* or *instance* and must be specified.
Block Storage	S_b	A mountable data storage for a Compute resource. Cf. Amazon EBS.
File Storage	S_f	Data storage which may be accessed by multiple Compute resources simultaneously. Cf. Amazon S3.
Internal Network	N_i	Internal network for all underlying Compute resources and File storages.
External Network	N_e	External network connection (IP address) for the parent Compute or File storage resource.

Special meaning is reserved for the words *type* and *instance* when used to describe resources: types act as templates for instances, and one-to-many instances can be instantiated of each type. Table I lists node types with description and abbreviation. Nodes of type *AA*-constraints (A) only affect Compute (C) and File storage (S_f) nodes. The other resources, networks and block storage, implicitly or explicitly belong to instances of either C or S_f, and thus are covered by the same *AA*-constraints as the node to which they belong.

Figure 1 shows examples of structures which can be composed into valid hierarchical DAGs. The relationship marked with edges create parent-child relationships. Instances of child nodes are attached to each instance of their parent. Both A and internal network (N_i) nodes may be nested to arbitrary depth. Nodes of type A stipulate constraints for all descendants as described above. For nested N_i nodes, C and S_f nodes require a virtual network interface for each ancestor of type N_i and each descendant of external network (N_e) nodes to connect them to each of these network instances.

An *AA*-constraint affects all descending C and S_f nodes but may have different scope, either type or instance, as specified as an attribute of the constraint. An *AA*-constraint with *type scope* affects how instances of a type can be placed in relation to instances of *other* types, but not instances of the same type. An *AA*-constraint with *instance scope* affects all descending instances *regardless of type*, and therefore also affects instances of the same type. For example, consider an *AA*-constraint A_1 specifying ”not same host” with two underlying compute node types C_1 and C_2:

1) If the scope of A_1 is *type scope*, no instance of type C_1 may be placed at the same host as an instance of type C_2. (However, two instances of C_1 may be placed at the same host.)
2) If the scope of A_1 is *instance scope*, no pair of instances of either type (C_1 or C_2) may be placed at the same host.

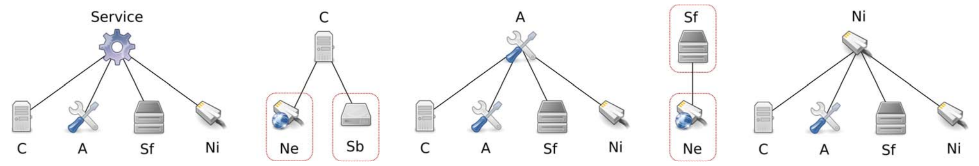

Figure 1. Rules defining valid inter-node relationships for service definition DAGs. C denotes Compute resources, A denotes *AA*-constraints, S_f and S_b denote file and block storage, respectively, and N_i and N_e denote internal and external networks. Valid terminal nodes are marked with a border. Further node description can be found in Table I.

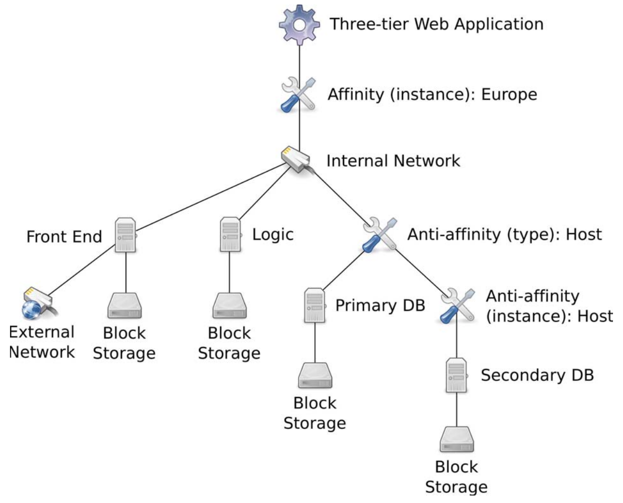

Figure 2. Example of a three-tier Web application service represented using a DAG which includes *AA*-constraints and network setup. Node types are shown in Figure 1, and labels have been added for clarity.

A. Service Definition Example

We exemplify this structure by describing a typical three-tier Web application in Figure 2 as a DAG. Immediately below the service root node, an *AA*-constraint states that all descendants of all resource types must be located in Europe. Thus, a scheduler may choose freely among Cloud federation partner sites located in Europe, but not elsewhere. An internal network resource node specifies that all its descendants are connected to a single local network instance. In addition, instances of the front end compute resource type are accessible via per-instance individual external IP addresses. A type scope anti-affinity constraint forbids placement of instances of the primary and secondary database servers at the same physical host. For the secondary database servers, an instance scope anti-affinity constraint explicitly forbids placement of instances at the same host, for fault-tolerance reasons. An individual block storage is attached to each compute node instance.

IV. Model for Scheduling in Federated Clouds

Scheduling is the process by which a VM management system decides on which physical host machine or partner site within a Cloud federation a VM should be placed. The general problem is to create a placement mapping between VMs and physical hosts such that placement fulfills certain management objectives [3], e.g. to maximize profit, avoiding loss of reputation, maximizing resource usage, etc. Mappings are evaluated using a number of factors, e.g. power consumption of physical host machines, economical penalties stipulated in pertinent SLAs, etc.

We present fundamental ongoing work for scheduling based on a model that takes *AA*-constraints, e.g. the ones shown in Figure 2, into account. The model assumes that migration can be used to optimize placement, but avoids unnecessary or risky (in terms of SLA violation risk) migrations.

The model regards remote sites as logical local hosts with different service-level characteristics, e.g. network capacity. Thus, management is simplified while still representing the performance and SLA-related differences between the local and the remote site(s).

Our model is formally described as follows. Let V be the set of VMs that need placement and H be the set of hosts to our disposal (including remote sites as logical members of H). M denotes a set of mappings $m_{v,h} \in M$ of VM v to host h stating that VM v is placed on h. Time is discretized and each interval has one active mapping. We wish to determine a new mapping M_n based on an old mapping M_{n-1} such that net profit is maximized. Net profit is expressed as the difference between a benefit function $B(V)$, a cost function $C(M)$ (models e.g. power usage due to the current host utilization), and estimated SLA-related costs due the inherent risk of performance loss associated with migration $S(M_{n-1}, M_n)$ in modifying the old mapping into the new one. We express this in Equation 1.

$$maximize \left(B(V) - \sum_{h=1}^{H} \sum_{v=1}^{V} C(M_n) - S(M_{n-1}, M_n) \right) \tag{1}$$

Note that if a mapping M makes use of remote resources in the Cloud federation, this will likely incur a larger cost $C(M)$ but (hopefully) also reduce the expenses if an SLA is violated, since the remote site also must provide compensations in that case. For sufficiently large problem instances, investigation of all possible new mappings to determine which gives sufficiently small values for $S(M_{n-1}, M_n)$ is too computationally intensive to be feasible. To that end, we define a heuristic to avoid wasting time investigating migrations that have a high risk of resulting in SLA violations.

A. Migratability heuristic

We define a *migratability* function $Mig(v, M)$ of a VM v given a current mapping M, where low migratability value implies that migration of v from its mapping in M is less desirable. The scheduler uses this heuristic in an attempt of minimizing $S(M_{n-1}, M_n)$ from Equation 1, while still being open to performing migrations to optimize placement.

Due to affinity relationships, it is not sufficient to consider the migratability of a single VM in isolation. Rather, for a given proposed migration of a VM v from one host or site to another location, let O denote the set of other VMs that must also be migrated due to affinity constraints. We then define $Mig(O, M)$ as the migratability function for all $o \in O$, relative to the mapping M. Obviously, if the selected new location for a VM is a remote site, the scheduler uses site and geographical level affinity to determine eligibility since actual host deployment is not known at remote sites due to location unawareness. The remote site must abide by affinity rules or reject the request to run the VMs if unable to do so. Also, due to anti-affinity constraints, the set O may be limited in which host machines may be used for placement of the VMs. The value of $Mig(O, M)$ depends on the migratability value of each individual VM $o \in O$.

For a single VM v, the factors that determine $Mig(v, M)$ relate to the cost and risk of violating pertinent SLAs. The risk calculation is based on:

- Long-term high-level monitoring data collected on the usage patterns of the VM and the service it belongs to. For instance, this helps determine if the service usually peaks in usage at some regular intervals, e.g. the end of the month.
- Short-term low-level monitoring data from the hypervisor internals regarding the memory usage of the VM. As the number of dirtied memory pages per time unit increases, estimated migration time for the VM increases [10].
- Sizes of storage and volatile memory that have to be transferred to the new destination and current network utilization, as well as other currently active migrations. If shared storage is used, typically only volatile memory must be transferred. If, however, the VM is to be migrated to another Cloud, it may be required to transfer the regular storage as well.

The migratability heuristic prunes the search space and helps the scheduler concentrate only on potentially fruitful mappings. The heuristic identifies and confirms the intuition that the easiest VMs to migrate are the ones that have few affinity (and to lesser extent, anti-affinity) relations to other VMs, are not currently (or in the near foreseeable future) highly active, and such that decreased performance due to migration will not be costly in terms of SLA violations.

As summarized in [11], even in research VM management projects, schedulers are quite rudimentary: by default, only various subsets of greedy, round-robin, and explicit (manual) scheduling are supported. Most schedulers will also avoid performing migration of a VM once it has found its initial placement, which leads to sub-optimal performance and possibly higher energy costs than necessary. Although research has been made on this topic [2], there is to our knowledge currently no scheduler software that takes *AA*-constraints into account that is open to the research community.

V. Monitoring Data Distribution in Cloud Federations

All Cloud sites offer monitoring of virtual resources, however, there are many different and incompatible monitoring systems in current use and this causes integration problems. We present our ongoing MEDICI project, a monitoring data distribution architecture that collects data from various existing monitoring systems, marks it up with semantic metadata, and publishes it to subscribers, one of which is a semantic database. The database allows complex queries on the semantic self-describing data, and the result can be transformed into a desired output format.

The MEDICI architecture is designed to leverage existing software for its core operation in a scalable way. The components of the architecture shown in Figure 3 are as follows:

- *Monitored infrastructure*. A virtual Cloud infrastructure that is monitored continuously, e.g. computational resources, storage entities, and interconnecting networks.
- *Data annotator/publisher*. Data annotators and publishers are the core of the MEDICI system, providing:
  - Canonicalization and semantic annotation of monitoring values by plugins. The annotations conform to OWL (Web Ontology Language) ontologies, facilitating parsing and conversion at the consumer level.
  - Preparation of annotated monitoring data which is then published to the distribution hub.
- *Distribution hub*. Distribution hubs distribute semantically annotated monitoring data to a set of subscribers.
- *Subscribers*. Any consumer implementing the hub's protocol may be a subscriber, enabling e.g. external components, the SPs, and other Clouds in the federation to gain access to the data using a single hub. As shown in Figure 3, the hub may distribute both public and private streams of data. This distinction makes it possible to prevent inappropriate disclosure of data to different parties.
- *SPARQL endpoints*. SPARQL [12] endpoints are databases that act as subscribers and are deployed either locally or remotely. They make it possible to aggregate data from the federation and make SPARQL queries on the data.

The architectural components in MEDICI are designed to expose remotely invokable interfaces and the number of instances of each component may due to loose coupling be independently increased to handle scalability gracefully.

The raw monitoring values and basic metadata (interval length, information source, and monitoring system identifier for future parsing by plugins) are transferred from the monitored infrastructure to the data annotator/publisher using light-weight REST methodology by system-specific plugins. The data may be extracted using e.g. libvirt [13], which is compatible with several underlying hypervisor technologies. Plugins may also be developed for other monitoring systems, e.g. collectd and Nagios. Higher-level service-specific data, e.g. "number of currently logged in users", can also be distributed by the system.

The data annotator/publisher maintains a separate set of plugins for handling various input of raw monitoring values. Upon data arrival, the appropriate plugin creates a semantically

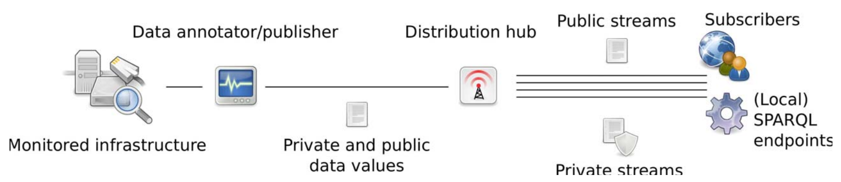

Figure 3. Overview of the MEDICI monitoring data distribution architecture.

annotated transformation from the raw data format in MEDICI canonical form. The data is then transferred to a distribution hub, which handles delivery to the subscribers.

The MEDICI canonical form for infrastructure data is based on the data set provided by libvirt. This choice was made for two reasons: (a) libvirt is compatible with most popular hypervisors; and (b) libvirt provides a reasonable subset of infrastructure-related measurements. However, note that since MEDICI uses extensible OWL ontologies, specific plugins can be developed for any input format. This allows service-specific data to be distributed.

The SPARQL endpoint acts as a subscriber to the hub and exposes its data via a rich semantic query language. This may be used for complicated queries, including queries for inclusion or inspection of remote monitoring data and accounting in a federated Cloud setting. It is i.e. possible to make queries that transform the remotely published data into data using the same measurement intervals as done locally, making it easier to apply the same mathematical functions for accounting and SLA violation detection purposes.

The distribution hub conforms to the PubSubHubbub [14] protocol, which uses the Atom format for data transport. We consider Atom suitable for this purpose for several reasons: (a) it is simple, incurs relatively low overhead, and is well-defined; (b) it is easily viewable in a Web browser or feed aggregator, requiring very little special software for a large variety of use cases; and (c) as an XML format it is easy to translate into other formats, and can transport other (semantic) XML data well, in addition to being platform independent. PubSubHubbub enables close to real-time updates of information in a scalable way, and by design of the PubSubHubbub protocol, the functionality of the hub is transparently hidden from consumers.

The strengths of this approach are that (a) plugins can be developed for specific monitoring already in use at Cloud sites; (b) plugins should not have a large negative performance impact on monitoring systems; and (c) publishing data to a database upon which semantic queries can be invoked, the data from a remote site can be queried and transformed into a format that is compatible with the monitoring system on the local site.

The architecture enables location unawareness from the management point of view, since it aids in bridging the gap between the management systems used at different Cloud sites, making monitoring data from one Cloud site easily integratable with the other.

VI. RELATED WORK

Service structure does currently not have wide-spread support. APIs such as Amazon EC2 or Open Cloud Computing Interface (OCCI) allow the SP to specify parent-child relationships (e.g. storage unit s is the child of VM v), but do not support sibling relationships such as the anti-affinity in Figure 2. As for *AA*-constraints, large public clouds such as Amazon EC2 and Microsoft Azure allow the SP to choose a coarse-grained geographical location, but not on finer levels such as site or host. To our knowledge, this functionality is currently only also available in [9], [3].

Verma et al. [15] present a power- and migration-cost aware scheduler (pMapper) upon which we have based our contribution. There are a number of differences between their work and ours: (a) our scheduling model may also be applied in a federated rather than an isolated Cloud; (b) the scheduling model presented here has the notion of *AA*-constraints between VMs; and (c) since our model is also usable for federations, it takes other costs than power and migration into account.

Breitgand et al. [2] present a scheduler with support for both affinity and cross-federation capabilities. They have developed Integer Linear Program formulations for placement strategies, and use the COIN-OR solver to obtain solutions. Our approach is different in that it provides a heuristic to determine which VMs should be easiest to migrate, making it suitable for local search algorithms.

Li et al. [16] extend upon the work in [1] by adding support for dynamic rescheduling and using migration to optimize placement of VMs across a multi-provider hosting scenario. Their broker acts on the behalf of a single SP, rather than at the IP level. The impact of using different instance templates (e.g. different VM sizes in Amazon EC2) as Cloud offerings may differ is studied. Since the broker acts on behalf of the SP, it does not have to avoid violating SLAs but instead attempts to minimize service downtime due to cold migrations. Since their model includes the possibility to assign per-VM penalties for migration, the migratability heuristic can be adapted for use within this system.

Existing approaches for monitoring in Clouds are presented in, e.g., [17], [18]. Both present relevant ways of extracting and managing data, but do not employ semantic metadata to achieve cross-Cloud compatibility. Said et al. [19] present a system and algorithm for automatically adding extensible semantic metadata by inferring structure from Globus Grid monitoring data. In addition to architectural differences, the key conceptual difference between that and our approach is that we believe that monitoring system-specific plugins produce richer semantic metadata than a generic algorithm could. Their algorithm infers a structure and annotates the data accordingly, but does not handle input from non-Globus systems and does not aim at making monitoring systems cross-compatible.

Passant et al. [20] use PubSubHubbub to provide close to real-time updates of data sets matching SPARQL queries. The result is turned into Atom feeds, which in turn are published using PubSubHubbub. This approach gives real-time updated streams of specific data, which could be used in conjunction with the MEDICI system to provide access to only relevant subsets of the information.

VII. CONCLUSIONS

This work describes ongoing work on fundamental service management tasks key to federated Cloud environments. We present a hierarchical graph structure representing a service and any placement restrictions placed upon the service components, such as site-level affinity, usable in Cloud federations. This way of structuring a service and defining *AA*-constraints offers a certain amount of control to the SP, which is then enforced by the IP. This facilitates management considerably for an SP compared to multi-provider hosting scenarios.

We define a model for scheduling in Cloud federations that abides by SP-specified *AA*-constraints. We present a heuristic that helps the model determine which VMs are suitable candidates for migration. The model is designed for optimizing placement both within a single site and in a Cloud federation. The heuristic is based on the intuition that the VMs that are most potentially costly in terms of SLA violations are those which are highly active, have *AA*-constraints that require further migrations, and where most data needs to be transferred.

All management of services in Cloud federations, including scheduling, requires cross-site compatible monitoring systems. Current monitoring systems are incompatible in both data format and semantics of what the data represents. To help overcome these issues, we present MEDICI, a monitoring data distribution architecture that annotates data with semantic metadata. Interaction with the data is made simple and flexible e.g. by publishing it to a semantic database upon which SPARQL queries can be made.

ACKNOWLEDGMENTS

The research that led to these results is partially supported by the European Community's Seventh Framework Programme (FP7/2001-2013) under grant agreements no. 215605 (RESERVOIR) and no. 257115 (OPTIMIS) and the Swedish Government's strategic research project eSSENCE. We thank the anonymous referees for their valuable feedback.

REFERENCES

[1] J. Tordsson, R. Montero, R. Vozmediano, and I. Llorente, "Optimized placement of virtual machines across multiple clouds," 2010, submitted for journal publication.

[2] D. Breitgand, A. Marashini, and J. Tordsson, "Policy-driven service placement optimization in federated clouds," IBM Research Report, Tech. Rep. H-0299, 2011.

[3] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forgó, T. Sharif, and C. Sheridan, "OPTIMIS: a holistic approach to cloud service provisioning," in *First IEEE International Conference on Utility and Cloud Computing (UCC 2010)*, December 2010, accepted.

[4] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 4:1 –4:11, July 2009. [Online]. Available: http://dx.doi.org/10.1147/JRD.2009.5429058

[5] E. Elmroth and L. Larsson, "Interfaces for Placement, Migration, and Monitoring of Virtual Machines in Federated Clouds," in *Eighth International Conference on Grid and Cooperative Computing (GCC 2009)*. Los Alamitos, CA, USA: IEEE Computer Society, August 2009, pp. 253–260. [Online]. Available: http://dx.doi.org/10.1109/GCC.2009.36

[6] D. Hadas, S. Guenender, and B. Rochwerger, "Virtual Network Services For Federated Cloud Computing," IBM Technical Reports, Tech. Rep. H-0269, Nov. 2009. [Online]. Available: http://domino.watson.ibm.com/library/cyberdig.nsf/papers/3ADF4AD46CBB0E6B852576770056B848

[7] K. Jeffery and B. Neidecker-Lutz, Eds., *The Future Of Cloud Computing, Opportunities for European Cloud Computing Beyond 2010*. European Commission, Information Society and Media, January 2010. [Online]. Available: http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf

[8] I. Brandic, S. Pllana, and S. Benkner, "High-level composition of QoS-aware Grid workflows: an approach that considers location affinity," in *Workshop on Workflows in Support of Large-Scale Science. In conjunction with the 15th IEEE International Symposium on High Performance Distributed Computing, Paris, France*, 2006.

[9] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L. M. Vaquero, "Service Specification in Cloud Environments Based on Extensions to Open Standards," in *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, ser. COMSWARE '09. New York, NY, USA: ACM, 2009, pp. 19:1–19:12. [Online]. Available: http://doi.acm.org/10.1145/1621890.1621915

[10] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines," in *VEE '11: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2011)*. ACM, March 2011, accepted for publication.

[11] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, pp. 14–22, 2009.

[12] E. Prud'hommeaux and A. Seaborne, "SPARQL query language for RDF," W3C, Tech. Rep., January 2008. [Online]. Available: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

[13] libvirt development team, "libvirt: The virtualization api," December 2005. [Online]. Available: http://libvirt.org/

[14] B. Fitzpatrick, B. Slatkin, and M. Atkins, "PubSubHubbub Core 0.3," February 2010. [Online]. Available: http://pubsubhubbub.googlecode.com/svn/trunk/pubsubhubbub-core-0.3.html

[15] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems," in *Middleware 2008*, ser. Lecture Notes in Computer Science, V. Issarny and R. Schantz, Eds. Springer Berlin / Heidelberg, 2008, vol. 5346, pp. 243–264. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89856-6_13

[16] W. Li, J. Tordsson, and E. Elmroth, "Modelling for dynamic cloud scheduling via migration of virtual machines," 2011, to appear.

[17] S. Clayman, A. Galis, C. Chapman, G. Toffetti, L. Rodero-Merino, L. Vaquero, K. Nagin, and B. Rochwerger, "Monitoring Service Clouds in the Future Internet," in *Towards the Future Internet - Emerging Trends from European Research*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2010, pp. 115–126.

[18] G. Katsaros, G. Kousiouris, S. Gogouvitis, D. Kyriazis, and T. Varvarigou, "A service oriented monitoring framework for soft real-time applications," in *Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on*. IEEE, pp. 1–4.

[19] M. Said and I. Kojima, "S-MDS: Semantic Monitoring and Discovery System for the Grid," *Journal of Grid Computing*, vol. 7, pp. 205–224, 2009, 10.1007/s10723-008-9111-2. [Online]. Available: http://dx.doi.org/10.1007/s10723-008-9111-2

[20] A. Passant and P. Mendes, "sparqlPuSH: Proactive notification of data updates in RDF stores using PubSubHubbub," in *Scripting for the Semantic Web Workshop (SFSW2010) at ESWC2010*, 2010. [Online]. Available: http://www.semanticscripting.org/SFSW2010/papers/sfsw2010_submission_6.pdf