Placement and Monitoring of Orchestrated Cloud Services

Lars Larsson

Licentiate Thesis, January 2015
Department of Computing Science
Umeå University
Sweden

Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

larsson@cs.umu.se

ISBN 978-91-7601-205-5
ISSN 0348-0542
UMINF 15.02

Printed by Print & Media, Umeå University, 2015

Abstract

Cloud computing offers pay-per-use on-demand access to computer resources for hosting program execution environments for software service deployment. Management of cloud resources includes determining, based on current monitored resource availability, which part(s) of a computational infrastructure should host such program execution environments in a process called placement. Our work defines directives that lets consumers of cloud resources influence placement to express relationships between cloud services (orchestration) and deployment constraints to uphold for related service components, without surrendering the ultimate control over placement from the infrastructure owner. The infrastructure owner remains free to define their policies and placement optimization criteria, e.g., to consolidate work that needs to be done to as few physical host machines as possible for power savings reasons. We show how the placement process can be adjusted to take such influence into account and validate through simulations that the adjustments produce the correct result without too large computational impact on the placement process itself. Further, we present a technique for transferring large data files between cloud data centers that operate in (separate) cloud federations that avoids repeated transfers in a delegation chain between members of (different) cloud federations. Finally, we present a non-invasive method of extracting monitoring data from a service deployed in a cloud federation, and a framework for making monitoring information available and understandable in spite of technical differences between monitoring systems used in cloud federations.

Preface

This thesis contains an introduction to the field and the papers listed below (reprinted with permission from the individual publishers).

Paper I Erik Elmroth, **Lars Larsson**
Interfaces for Placement, Migration, and Monitoring of Virtual Machines in Federated Clouds. In the *Eighth International Conference on Grid and Cooperative Computing (GCC)*, pages 253–260. IEEE, 2009.

Paper II **Lars Larsson**, Daniel Henriksson, Erik Elmroth
Scheduling and Monitoring of Internally Structured Services in Cloud Federations. In the *IEEE Symposium on Computers and Communications (ISCC)*, pages 173–178. IEEE, 2011.

Paper III Daniel Espling, **Lars Larsson**, Wubin Li, Johan Tordsson, Erik Elmroth
Modeling and Placement of Cloud Services with Internal Structure. In *IEEE Transactions on Cloud Computing (TCC)*, to appear.

This work has been supported financially by the Swedish Research Council (VR) under contract 621-2005-3667 and for the Cloud Control project, the Swedish Government's strategic research project eSSENCE, and by the European Community's Seventh Framework Programme ([FP7/2001-2013]) under grant agreements no. 215605 (RESERVOIR) and no. 257115 (OPTIMIS).

Publications by the author not included in this thesis:

- Benny Rochwerger, Constantino Vázquez, David Breitgand, David Hadas, Massimo Villari, Philippe Massonet, Eliezer Levy, Alex Galis, Ignacio M. Llorente, Rubén S. Montero, Yaron Wolfsthal, Kenneth Nagin, **Lars Larsson**, and Fermín Galan. An Architecture for Federated Cloud Computing. In ed. Rajkumar Buyya, James Broberg, and Andrzej Goscinski *Cloud Computing Principles and Paradigms*, pages 393–412, Wiley 2001.

Acknowledgements

This thesis has been a long time coming. When I joined the research group in 2008 as a systems developer, cloud computing was still in its relative infancy. Since starting as a PhD student in 2009, I have both witnessed and been part of a lot of important development that has taken place in the cloud computing field, some of which has made it into papers and ultimately, this thesis. I am grateful to my supervisor **Erik Elmroth** who took me on and has let me walk this path, and to him and **Johan Tordsson** for the side path that we took in 2011 when they started Elastisys and I got to dive head-first into pure(-ish!) software development again in that context. I am in constant awe of how much you two manage to get done and still bring your full focus and attention to each individual meeting. Resource contention issues solved at its finest!

At Elastisys, **Peter Gardfjäll** always teaches me something new. You are by far the best software developer I know, and I feel like I grow every day I work with you. I am incredibly happy that I get to share an office with you, and that you are also laying down some serious roots in Umeå with your family. I hope that I get to be around you as a friend and colleague for a long time to come!

My dear former office-mate, colleague, friend, and fellow globetrotter **Daniel Espling**: you are amazing, and my life is richer for having you in it. Thank you for all the good times we have had, in- and outside of the office. I feel like we went through all of this together, and I thank you for being there with me for every step of the way, even when it got rough.

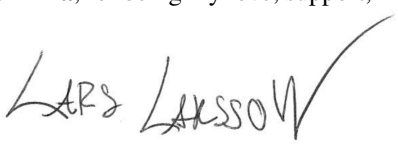
If it wasn't for **P-O Östberg**, I probably wouldn't even be a PhD student to begin with. I would likely have taken a job at Ericsson in southern Sweden in 2008 and only looked back when that whole division was laid off only two years later. Thank you for believing in me, encouraging me, and for your friendship: your love for research and enthusiasm is infectious (and also saved me from making a huge mistake)!

It is a pleasure to be part of our ever-expanding and always amazing group, that is full of so many wonderful people. Both the present members (**Ahmed**, **Amardeep**, **Cristian**, **Ewnetu**, **Francisco**, **Gonzalo**, **Jakub**, **Lennart**, **Luis**, **Mina**, **Olumuyiwa (Muyi)**, **Selome**, and **Tomas** apart from those already mentioned) and the ones that are gone but not forgotten (**Lei**, **Petter**, and **Wubin Li (Viali)**) fill my days with interesting discussions literally about anything and everything between heaven and earth. Some of it even work-related! Thank you for enriching my life with new perspectives, insights, and philosophies. You all rock!

On a personal note, I wish to thank my friends and family. You keep me grounded, sane, and happy. To my mother **Ursula** and sister **Lisa**, there are not words that can

sufficiently thank you for all the countless ways you have enriched and helped shape my life. **Johannes** and **Henrik**, my brothers from other mothers, you have been with me for most of my life, offering your support and love no matter the circumstances. And my more recent dear friends I have got to know since coming to Umeå: **Helena**, **Sofia**, **Peter**, **Mahsa**, and **Robin** — you fill this cold high north with warmth and love.

And finally, my most immediate family, where even more important developments have taken place during these years with the birth of my three children (**TINA** in 2009, and the twins **DAVID** and **SAMUEL** in 2013). I love you, and I am forever grateful for being able to spend time with you. Last, but in no way least, thank you to my beloved wife **Anna**, for being my love, support, my everything. I think I dreamed you into life.



Lars Larsson, Umeå, January 2015

Contents

CHAPTER 1
Introduction

Computers and their associated IT equipment such as networks and storage units are not only expensive to manufacture, but operating them requires expensive power and cooling, and when they break, they require maintenance by experts. While consumers are more than happy to periodically buy and briefly own computers used sparingly and mostly for leisure, businesses and researchers need to balance an ever-growing dependence on computational capabilities with keeping within a constrained budget. Hence, much effort has been devoted to letting some (third) party take over the costs of owning and operating large (distributed) computer infrastructures, while allowing the users of the infrastructures to pay in accordance with their usage. Seeing this need early on, John McCarthy in the 1960's envisioned delivery of computational power as a public utility [1, 2].

Several implementations of the idea of providing (remote) access to (distributed) computational infrastructures for payment based on usage of said infrastructure have been created over the years, each of which designed for a particular niche task and with the limitations of that time-period in mind. The first implementation is that of *mainframe computing* (and *clusters* of mainframes followed as soon as networks had increased sufficiently in performance), where users run processes on a particular machine, running a particular operating system. The first commercially successful such system was the IBM-360 in 1964 [3]. Each program typically had to be tailored for the particular execution environment, thus making mainframe systems rather difficult to work with. Time-sharing systems such as UNIX came from the 1960–1970's [4], and each user could be billed according to the amount of computer time they had consumed.

Fast network connectivity allowing transfers of large amounts of data made the *grid* emerge in the 1990's and early 2000's [5, 6, 7]. The main focus of grids is to allow for *batch processing* of typically large data sets, where access to results is not immediate, since submission of a time-limited *grid job* only specifies what work is to be done (possibly, however, with a deadline), and it is up to the grid system to find the best possible time slot to actually execute the job. The grid software (middleware) provides access to the computer, offering an *execution environment* for the grid job to be executed in. What libraries are available in this environment is dictated by the grid site's operators, which implies that the job process must be tailored to the particular environment in which it will run. In contrast, cloud computing offers *virtual machines* as the runtime environment, and the *cloud services* deployed are *not*

time-limited processes. Cloud services are expected to start within minutes of the user having requested access to the computational infrastructure, and run consistently and constantly until the service owner at their own discretion terminates the process and its execution environment. Chapter 4 takes a deeper look at services and how they are deployed in cloud computing, but for now, the simple definition of a service as a piece of software that offers its functionality over a network, e.g. streaming music files from a library of files, will suffice. Clouds and virtual machines are presented in more depth in Chapter 2, and management of cloud resources in Chapter 3.

Each utility computing implementation is niched toward a particular task or set of tasks, and designed accordingly. They are all still relevant today as the niches still remain, but with regard to current research effort, *cloud computing* is by far the largest today. Its niche, long-running general services, is also considerably broader than e.g. pure batch processing, which attracts researchers from many fields.

Autonomous computational infrastructures, whether they are grids or clouds, are often joined together in an attempt to pool resources and provide even more utility to their users. Such collaborations are known as *federations*, and their formation typically requires significant engineering effort and service contract formulation, as the infrastructures may not have been designed with collaboration as a goal in addition to being fully autonomous. There is therefore typically no central management functionality for the federation as a whole. The Atlas project at CERN [8], for instance, consumes what at the time of writing is *vast* amounts of computational resources, including processing power, network bandwidth, and storage, pooled from several universities. It produces about 1 Petabyte of data *per second*, and requires processing equivalent to about 50,000 modern PCs[1]. Since this amount of data processing greatly exceeds what one data center can handle, federations of computational infrastructures are required to deal with this type of Big Data [8]. Federated computational infrastructures offer great processing power, but also come with a set of unique challenges and trade-offs, including ones related to security, performance, trust, accountability, and a lack of control over remote sites, as each site is autonomic.

The work in this thesis addresses some of the problems faced in cloud federations. In particular, we: propose to deal with the lack of control by mechanisms for users to influence decisions that ultimately affect how their computational needs are handled, ensure accountability toward resource consumers that they get what they pay for through monitoring systems, and enable infrastructure owners to optimize their own internal processes according to their own criteria. The work is motivated by the dissatisfaction felt as a long-time cloud user/customer with how services are deployed in current clouds, and how little influence one is given in that situation today.

[1] According to: `http://atlas.ch/computing.html`

CHAPTER 2
Cloud Computing

Following its inception in the early 2000's, cloud computing has become an umbrella term coming to mean many different things that are not always even related to each other. In the broadest sense, cloud computing can mean anything where resources or software services are running at a remote site. This overly broad definition aids commercial marketing efforts, but hinders meaningful scientific discourse. We will only consider definitions that are in line with the utility computing vision, which is to allow for users to make use of computational resources (possibly owned by a third party), meter their usage, and (typically) then have them pay in accordance with their usage in some way that makes sense in the service context (e.g. either abstract credits from an allotment or actual currency). This is realized by virtual computing infrastructures, such as grids and clouds, that exist to provide an execution environment for computer programs. Execution environments are isolated from each other so that the resource demands and usage of each does not negatively affect the others [4, 9]. The type of program deployed dictates how the execution environments are designed and implemented, and how much the capacity consumer needs to worry about this this detail depends on the service model, i.e. how the computational resources are made available to its consumers. For cloud computing, these service models are [10, 11]:

- *Software-as-a-Service*. Scalable installations of a given software are offered to the consumer's users, e.g. Microsoft Outlook or SharePoint, and the consumer's users pay per use in some fashion that is natural for the software, typically per-licence.
- *Platform-as-a-Service*. A platform of useful surrounding services, e.g. application servers, preconfigured databases, message queues, mailing systems, etc., is offered to software developers, making it easy to focus only on developing the business logic of scalable applications. Use of surrounding services and of the business logic itself is metered and billed, according to some plan.
- *Infrastructure-as-a-Service*. Consumers provision custom execution environments known as virtual machines (VMs) directly from the cloud resource owner, and pay per use for the resources they consume. The VMs can be configured with any software of the consumer's choosing, and the consumer is responsible for setting up any required services and software stacks themselves.

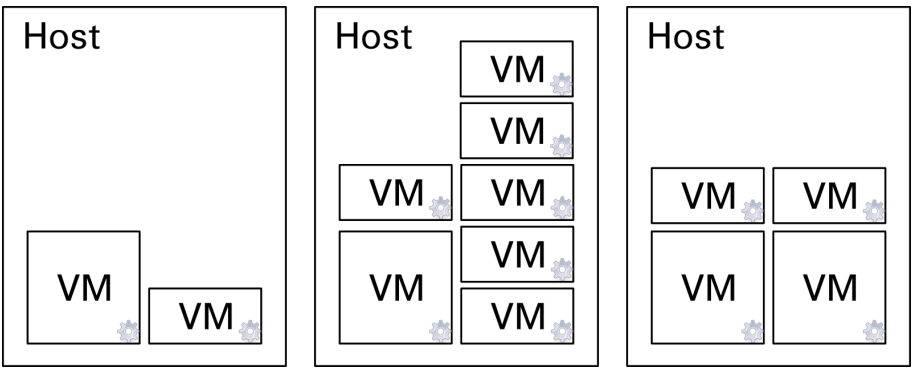


FIGURE 1: Hosting of several VMs within a number of hosts.

We will henceforth use the terminology *Infrastructure Provider* (IP) for cloud capacity producers, *Service Providers* (SP) for cloud capacity consumers, as this is both in line with the papers contained in this thesis, and with the well-cited definition by Vaquero et al. [10].

In this thesis, we focus exclusively on the Infrastructure-as-a-Service (IaaS) service model. The central unit in contemporary IaaS is the highly customizable execution environment implemented as VMs (but this is about to change, see Section 2.3 for a look toward the future with containers). A VM is an abstraction layer that acts as a partition of a large physical *host* machine, where each virtual machine can run its custom operating systems independently and in isolation from each other [12, 13, 14]. Access to hardware, such as network interface cards, is emulated or provided shared access to in a bridged fashion within the host operating system [14]. The software that coordinates virtual machines on the physical actual machine is called a *hypervisor*. Emulating an entire computer in this fashion in software alone is possible, but prohibitively slow. It was not until hardware-assisted virtualization became available and efficient [15, 16, 17, 18, 19, 20, 21] that virtualization and cloud computing was reasonably possible and desirable for paying customers. Figure 1 shows an overview of the concept of partitioning physical machines in differently sized VMs.

VMs are assumed to start provisioning the service within minutes after having been requested, and do not (typically) terminate until the paying user dictates it. A single physical host in a cloud site typically runs a (large) number of VMs, meaning that they are not isolated from each other temporally. Instead, *service-level agreements* (SLAs) [22,23] govern the quality of service experienced by the VMs, by defining a set of measurable *service-level objectives* (SLOs). Violations against SLOs are typically cause for financial compensation to the user, as outlined in the SLA.

Because VMs can be fully customized, they can support a large variety of use cases. One can deploy software in them to support long-running services (such as web services) or short-running tasks (as in the grid case). We focus solely on the long-running service aspect, as it opens up avenues for research and introduces addi-

tional complexity in e.g. the process of choosing *which* host should be used for VM provisioning. For short-running tasks, data locality is likely the primary driver in such processes, whereas longer-running services experience usage variations that can have significant implications on management processes (more about these in Chapter 3).

In closing of this brief introduction to cloud computing, we consider the definition of the U.S. National Institute of Standards and Technology (NIST) that lists the essential characteristics of cloud computing as follows [11] (with reworded explanations):

- *On-demand self-service*. As consumers of cloud resources (including virtual machines, networks, and storage), SPs should be able to provision resources with minimal IP effort, as demand dictates. Self-service implies that human interaction requirements should be minimized.

- *Broad network access*. Ubiquitous and standard network and transport protocols should be used to allow cloud resource consumers access from a (large) number of network locations.

- *Resource pooling*. SPs are presented with dynamically allocated resources from large pools of (heterogeneous) resources, but are typically kept unaware of precisely where these resources come from and with whom they are sharing these resources. This unawareness is of great importance for the contributions of this thesis, as we offer optimization possibilities from both the IP and SP aspects, hinged on the idea of allowing the IP to offer resources without surrendering neither information on nor control over management of these resources to SPs.

- *Rapid elasticity*. SPs should be able to rapidly provision more resources, or shed resources that are no longer needed, from a seemingly unlimited pool of resources. This is the topic of future work, see Chapter 6.

- *Measured service*. Use of various resources should be metered and billed according to usage. SPs typically pay per computational capacity and sizes of internal memory for virtual machines, network transfers, and storage size and use in terms of input/output operations separately, but on a single bill from the cloud provider.

2.1 Deployment Models

Cloud computing can be deployed according to a variety of models. The fundamental three models are, from most restrictive prospective users to least: *private clouds*, where a single organization owns and operates its own physical hardware and makes it available to their own users as cloud resources; *community clouds*, where collaborating organizations pool their resources together in a cloud spanning over the entire pool, but the resources are only offered to members of these partnering organizations; and *public clouds*, where a single or partnering organizations offer access to cloud resources to the public. Hybrid versions of these deployment models exist as well,

where the goal is to allow for *cloud bursting*, i.e. seamlessly making use of cloud resources from another cloud while still enforcing barriers between cloud entities, as appropriate. This typically requires compatible software to work [11].

The papers in this thesis all focus on *federated clouds* [24, 25], i.e. ones where some kind of collaboration takes place between cloud sites. We define a cloud site as an organizational entity within which the infrastructure is exposed to SPs as a unit, typically a data center. A single cloud infrastructure provider can own and maintain a number of such sites, in effect offering a federation of cloud sites. By this definition, Amazon Elastic Compute Cloud is a federation, as the various regions are completely separate, while availability zones within a single region are not to be regarded as units (hence, a region is not itself a federation). For our research, whether some of these are private, community, or public clouds does not matter, as long as there are no technical incompatibilities make a collaboration impossible.

For a toolkit aimed at making various cloud constellations easy to work with, see the OPTIMIS toolkit [26].

2.2 Cloud Service Deployment

Figure 2 shows the phases of service deployment onto a cloud and the tasks included in each phase, carried out by the SP and IP, respectively. We divide the provisioning process in three phases: *staging*, *deployment*, and *operation*. In the staging phase, the SP prepares a VM image and uploads it (Task 1) and any data required for processing to the IP for storage (Task 2). Note that clouds typically offer a catalog of pre-defined VM images. If this is the case, the SP may simply choose one from this catalog, and possibly pay licencing costs for doing so. A VM image is a file containing the contents of what is to become the (virtual) hard-drive contents of the VM. The response back from the IP is some kind of identifier to the stored image and data, which can later be referred to in the *service manifest* [25].

In the deployment phase, the SP prepares a service manifest for the service that is to be deployed (Task 3). This is a document containing orchestration information for one or more service components, and any additional information and rules concerning the placement of these service components. The cloud infrastructure performs admission control, i.e. determines if the service described in the service manifest can be accepted for deployment or not (Task 4). If the service is deemed admissible in accordance with the cloud's rules and resource availability, a suitable placement for the VMs each service component is to be deployed in is found (Task 5).

The operation phase starts as soon as a VM is started, and it contains the concurrent tasks of controlling (Task 7a) and monitoring (Task 7b), performed by the SP and IP, respectively. The SP can modify the state of the VM by issuing commands to pause/resume, stop/start, resize/duplicate, or terminate the VM. The IP continuously optimizes VM placement and monitors the resource usage by the VM to present the data to the SP, and for billing purposes.

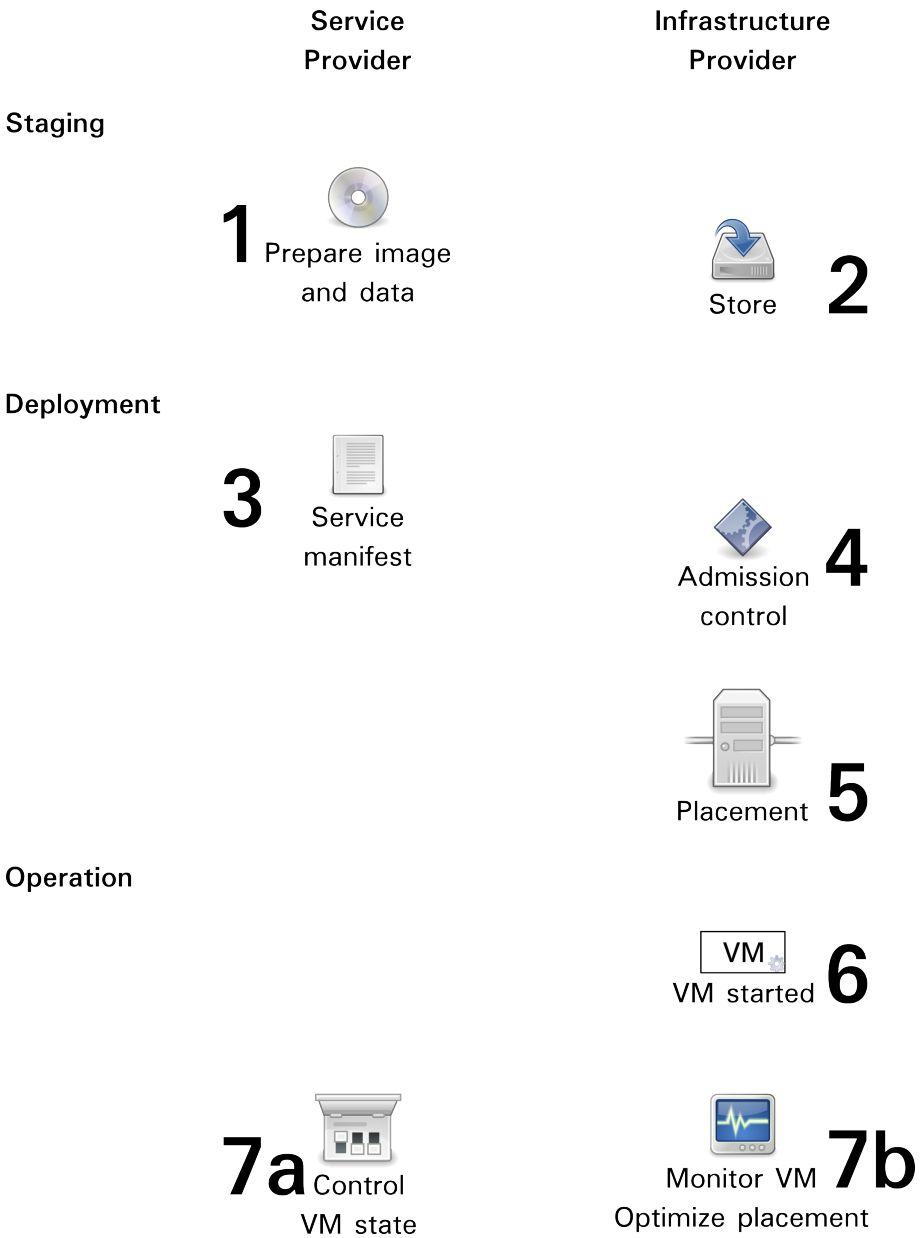


FIGURE 2: Service deployment phases and tasks for SP and IP.

2.3 A Look Ahead: Container Clouds

Providing access to fully customizable isolated program execution environments is one of the cornerstones of cloud computing. Currently, this isolation is provided by means of virtualization, for which a performance penalty must be paid as emulating an entire computer and its associated hardware is obviously more wasteful performance-wise than bare-metal access as offered by grids, even in spite of attempts to close the gap by e.g. hardware-assisted virtualization support in modern processors [15]. Virtualization is also wasteful in terms of storage space: most SPs base their VMs on some standard base VM (e.g. some well-supported version of Linux, such as Ubuntu or CentOS), but each individual VM still gets a full individual copy of the entire file system, although the difference between what is stored in each VM is typically orders of magnitude smaller than the overall allocated storage for each VM.

In recent years, *containers* rather than VMs have started to gain momentum as the customizable program execution environment of choice (in particular after recent crucial additions were made to the Linux kernel in version 3.8 [27]). Containers are a type of virtualization that does not rely on virtualizing an entire machine, as VMs do, but rather on isolating user space system instances within a single (shared) operating system kernel. In Linux, support for containers is granted by cgroups. Cgroups provides resource (CPU, memory, various I/O) and process namespace isolation, isolating applications from another with regard to process trees, network connections, user ids, and (notably) mounted file systems. Essentially, this means that containerized processes are free to define their own file systems, init systems, and background services but must share the host operating system's kernel. This is a limit to some applications (ones that rely on a custom kernel), but on the other hand, containerized processes are thereby provided with bare-metal access to the computer's hardware. This significantly reduces both performance and storage overheads [28]. Because containers offer merely a type of additional operating-system level isolation layers between processes, they start within seconds, rather than minutes, since they do not require lengthy provisioning and boot-up processes — essentially, running a program in a container is just like running a local program [29].

Containers are nothing new, as they have existed in various operating systems before (e.g. Solaris Zones [30], and BSD Jails [31]). Useful support for them in Linux, however, is. After the 3.8 Linux kernel release, and with the introduction of Docker [32], Linux containers and clouds based on them have gained significant momentum. This is in part because Docker makes it easy to package applications and all their dependencies together in an image that can then run *unmodified* on any other system that supports Docker — without paying the performance penalty that is associated with VMs. Additionally, the way that Docker images are constructed, and because they are mounted on a union file system, only unique data requires additional storage: data that can be shared between images is shared.

As a previously mentioned potential drawback, by construction, containerized processes must share kernel with the host machine. This may be a limit for certain services operating on a low enough level to warrant customized kernels, but the vast majority of cloud services do not require customized kernels. However, as containers

can define their own storage trees up to and including even entire Linux distributions, it is possible to run e.g. a mix of CentOS and Ubuntu containers on a single host, albeit with a kernel that may not be tuned for either distribution.

CHAPTER 3

Cloud Resource Management

Cloud infrastructures typically comprise several physical hosts, each capable of hosting a number of VMs. The SP should not (have to) be aware of how large the pool of resources actually is, but regard it as seemingly infinite. IPs can collaborate by forming cloud federations to make larger (possibly hybrid) cloud platforms that make use of capacity at remote sites, should the local resources be exhausted. The rapid, on-demand self-service that cloud infrastructures are supposed to offer requires capable management software, with core features including the following:

- *Admission control*. By examining the service manifest and current (and predicted) resource availability, the cloud management software must determine whether a given service can be accepted or not. Service deployment is a long-term commitment, and since terminating a service once it has been accepted typically violates the SLA, admission control must be done in a risk-conscious way [33].

- *Placement optimization*. Continuously finding the best possible physical host machine for the set of running VMs is termed placement optimization. Depending on optimization criteria, this process may lead to VMs being *migrated* from one host machine to another, to achieve e.g. consolidation to power off an unused host machine to save energy [34], or to minimize load differences among physical hosts for fault-tolerance reasons.

- *Monitoring*. The usage and condition of VMs must be monitored to ensure that no SLOs fail to be met, and to ensure that usage is correctly accounted and billed for. SLOs may state that no VM should suffer more than X% downtime over some time period, or that a certain performance characteristic is always guaranteed. Monitoring should transparently demonstrate any failures to meet these agreed-upon objectives.

- *Orchestration*. Services contained in VMs should be deployable in a deterministic, well-defined, and repeatable way. This process is known as orchestration, and while it has been the topic of much research [35,36,37,38,39,40] (including

our own), sophisticated multi-VM orchestration functionality has only recently been adopted and offered by commercial cloud vendors. Standardization efforts are also underway [41].

- *Storage*. Storage location and availability are important factors for VM performance and hence placement optimization, since VMs should ideally run close to where their backing storage (i.e. virtualized hard drives) are physically located, as this helps them perform better and decreases the internal network load [42]. In addition to the storage used directly by VMs, clouds typically need to offer a catalog of VM *images* that act as templates or starting points for making custom VMs, to avoid having to install every operating system from scratch, and some sort of *object storage*, that allows SPs to store various static data in a network-accessible way. Given the importance of the stored data, and the difficulty or cost associated with moving it to another cloud provider, it is often a cause of vendor lock-in [43]. Additionally, due to lack of insight and control over what entities have access to data stored in the cloud, potential cloud users are hesitant to fully use the cloud unless additional security measures are added [44, 45].

- *Networking*. VMs need to be network-accessible — not necessarily publicly, but without a network connection of some sort, they have no way of communicating outside of their hypervisor. Should the VM migrate, networks typically need to be reconfigured to avoid VM network connectivity loss (and the negative effects this may have on the software running inside). *Software Defined Networking* acts as a layer of additional abstraction that separates the software that decides where traffic is sent (control plane) from the underlying system that forwards the traffic to selected destinations (data plane) [46, 47]. In doing so, network connectivity and logical topology is made more dynamic in nature, allowing networks that best suit the services deployed in the cloud to be defined.

- *Accounting*. Since cloud resource consumption has to be paid for, cloud management services need to provide accounting features tied to the monitoring service. Accounting systems need to charge users in accordance with their usage, either in a pre- or post-paid fashion [48].

- *User management*. Cloud users must be authenticated and their actions authorized to ensure that they only perform actions they are allowed to. Some kind of user or identity management is therefore needed.

Large public cloud providers provide all of these services, in addition to various provider-specific ones. The research and open source communities have access to cloud platforms that also provide these services, such as OpenNebula [49], CloudStack, and OpenStack [50].

Of particular interest to this thesis are the following activities: placement, monitoring, orchestration, and, to a certain degree, storage. Paper I deals with placement and storage concerns as VMs are migrated to other clouds. Paper II presents solutions for problems in placement, orchestration, and monitoring. Finally, Paper III is focused on placement and orchestration, as an extension of the concepts introduced in

Paper II. Therefore, we devote the following sections to a more in-depth look at these management activities.

3.1 Placement Optimization

Placement is the process of determining which VMs should be provisioned on which hosts or partner clouds in a cloud federation [51]. We refer to the outcome of the process as a *mapping* between VMs and hosts. The algorithm driving the placement process typically optimizes some set of criteria. Typical such criteria include, e.g., using as few hosts as possible to make others available for other tasks or to power off unused ones [52, 53, 54] to reduce energy and operational costs, or distributing VMs evenly to ensure good performance [51, 55]. While VMs are isolated in theory, they suffer from the *noisy neighbors* problem, where large consumption of some resource (e.g. CPU processing power) in one VM negatively affects the amount of resources available to other VMs deployed on the same host, breaking isolation and possibly failing to meet SLOs. Industry experience and research has found that VM performance differs greatly depending on factors beyond the SP's control [56], something which is attributed to poor isolation between neighboring resource-intensive VMs (i.e. other VMs deployed on the same physical host or on the same network subnet). Placement, and the choice of optimization criteria, is therefore crucial to providing a reasonable cloud service.

The placement process is executed not just when the set of VMs change due to allocation or termination of VMs, but as a continuous process. This is necessary because the total execution time of VMs is not known up front, unlike in the grid job case. To optimize placement, and perform (re-)consolidation, should host machines have become unevenly loaded, VMs can be migrated (while running) from one host to another to improve an already established mapping if it is determined that the current mapping is suboptimal. This process is known as *(live) migration*, and its use to facilitate placement is a hot research topic [57, 58, 59, 60, 61, 62, 63, 64, 65, 66]. Live migration moves a VM from one host to another, and the capacity requirements for the VM is kept the same (i.e. it demands the same amount of RAM, CPU, storage, and network capacity from the new host as it did from the old).

Placement optimization (as opposed to greedily just finding *any* placement) is at its core an instance of the Generalized Assignment Problem, and is therefore NP-HARD to solve [67]. To further complicate matters, and increase potential income, cloud IPs may apply over-booking of the resources they actually have, while increasing the risk of failing to meet some performance SLOs, should simultaneous resource requirements actually exceed available capacity [68, 69, 70].

Most approaches so far have considered placement based on explicit resource requirements made by the SP up front in a service manifest, dealing with VMs as computational black boxes. However, looking in to these black boxes, and possibly modifying them slightly, has been shown to enable even further optimization methods. If the workload can be determined [71, 72, 73, 74, 75], so can the capacity requirements of the VMs, allowing for ahead of time adjustments to avoid the service being de-

ployed with too low capacity [76, 77, 78]. If a capacity demand profile for a VM can be determined, modifying the profile can be done in an effort to *re-pack* VMs to better provide them with the resources they actually require, so that they require more or less resources as the workload fluctuates [79]. Coupled with over-booking, this can offer IPs great possibilities to optimize without the SP suffering from poor performance.

Finally, in some cases, it is impossible to correctly predict or react to a change in workload. In an approach similar to supporting application checkpointing [80], wherein the program or service of interest is modified to allow for compensating for inadequate resources to carry out the task at hand, SPs can make certain computationally intensive sections of the service optional [81]. The reasoning is that it is better for end-users to get some kind of response, than none at all. Such a modified service may perhaps appear to be less dynamic due to failing over to only serve cached static responses than when resources are abundant.

3.2 Monitoring

Since cloud computing is defined as a pay-as-you-go service, monitoring resource usage is a crucial task. Monitoring occurs at three conceptual levels:

- *Infrastructure monitoring*. Core cloud management functionality such as the placement algorithm needs up to date information regarding the state of the infrastructure, including network utilization and the state of the physical hosts and the storage units attached to them. This data is typically not made public, apart from vague service health indicators. Problems that are detected need to be mitigated, possibly by migrating or restarting VMs away from an errant host machine.

- *Resource consumption monitoring*. SLAs stipulate the terms under which SPs obtain resources from IPs, and should the IP fail to deliver the agreed-upon capacity, some compensation is typically in order (although the potential loss in reputation and credibility may be higher and hard to quantify, hence the focus on avoiding failure to meet SLOs in the Placement section). VMs and their resource consumption are continually monitored, and the data should be transparently presented to the user. Transparency in this issue is a cornerstone of the trust between SPs and IPs.

- *Service monitoring*. A higher-level view on capacity requirements and resource consumption is to consider how well an service can handle its current workload. Rather than, as in resource consumption monitoring, which focuses on e.g. CPU instructions per second, service monitoring focuses on *key performance indicators* (KPIs), such as the number of currently logged in users or successfully served web requests. These measurements can typically be fed into the management functionality of the cloud, and be used as a basis for automatically adjusting the capacity allocation to better fit the current workload [78].

3.3 Orchestration

SPs submit a service manifest that describes the capacity they require from the IP. The IP then accepts or rejects the service after determining that there is sufficient capacity to host the service by invoking its placement algorithms. In the most trivial case, the SP requests each VM individually, in effect creating a service manifest for each. In non-trivial cases, these service manifests are documents that cover a large number of service components (VMs) and can include scaling directives, service-level objectives [26, 82], and explicit relationships between service components [83, 84].

In the research community, in particular the European line of research initiated by the RESERVOIR project [82] in 2008, rather complex and fully-featured service manifests are assumed to be available, since they allow both the SP and IP to reason about the service that is to be deployed as a whole, rather than in individual parts. Industry support for non-trivial service manifests and orchestration declarations have, however, been slow to emerge, but Amazon Web Services offers Cloud Formation [85] and recent OpenStack versions offer Heat [86], its Cloud Formation-compatible orchestration features. Orchestration standardization efforts are also underway in the shape of the OASIS Cloud Application Management for Platforms (CAMP) [87] and OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) [41, 88, 89].

3.4 Storage

While VMs can either be close-to-optimally placed from the start, or be reasonably easily migrated from one physical host to another, the data stores that VMs use for computation input or output may be orders of magnitude larger than the working memories of the VMs. While storage units to VMs typically reside on network-accessible nodes, network limitations and bandwidth considerations limit how VMs can be migrated to hosts typically on the same subnet or network switch as the storage node, as migration over the wide-area network (WAN) is technically possible but inefficient and warrants ongoing research [58, 90, 91]. Thus, storage management is key to ensuring good performance and ensuring data safety is of utmost importance. For fault-tolerance and performance reasons, storage is typically distributed [92, 93]. For cloud bursting or intra-federation migration, the data needs to be exported to the target cloud to ensure reasonable performance (WAN links are too slow to perform disk input and output). Doing so, without duplicating the transfers needlessly, is discussed in Paper I.

CHAPTER 4

Service-oriented Architectures in the Cloud

Service-oriented architectures (SOAs) are part of a design philosophy that helps shape and motivate cloud deployment of services, although the concepts have evolved since the 1990's and 2000's when SOAs were first introduced. Papers II and III in this thesis deal with orchestration of services in cloud environments, motivated by the particularities of services designed for cloud deployment. This chapter serves as a brief introduction to what type of software services are deployed in clouds, and how cloud computing itself is the evolution of software services.

Services are, in the context of cloud computing, a term for a composition of a number of *service components*, each of which contributes a well-defined functionality to the functionality of the service as a whole[1]. Not all components have to be deployed in the same cloud, or even in any cloud (i.e. a publicly facing web site could be deployed in a public cloud, but access information through a secured connection to an on-premise database with sensitive data). In SOA terms, services are network-accessible software programs that have a well-defined functionality, offer access over various transports and through various serialization formats, and they are regarded as always being available [94]. They should be technology neutral for interoperability, loosely coupled, and location transparent [94]. Their utility is the defining characteristic, and this utility is made available via some kind of Application Program Interface (API). Services (and the components thereof) and their APIs were in the SOA vision supposed to be automatically discoverable via cataloging services/registries, and the interaction between services facilitated almost to the point of automation by complex but well-defined document formats fully describing the operation signatures of the entire API of each service. For a number of reasons, most of which hardly technical, this is not how modern services are presented and used. Instead, Representational State Transfer (REST) [95, 96], which is focused on the data that a service represents and offers methods to query and modify the data using the standard HTTP methods (GET, PUT, POST, and DELETE), has emerged as the most common way of publishing an API that allows others to make use of a service.

[1] In a 2007 interview at the High Scalability Blog, the then CTO at Amazon stated that generating a single page involves about 100–150 backend service components.

4.1 Services Developed for Cloud Environments

Cloud computing is well-suited for hosting services, since each VM can be customized to suit a particular service component in terms of what software is installed and how much computational power the component needs. A large relational database may require a large amount of RAM, whereas a web frontend that coordinates several back-end services and presents a document to a web site visitor, might not. But clouds also has certain specific traits that shape how services are developed for being deployed in cloud environments.

Cloud computing offers rapid, on-demand elasticity by allowing SPs to commission and decommission VMs. This is called *horizontal scaling*, as it increases the number of VMs. Modifying the capacity of individual VMs is referred to as *vertical scaling*. The multi-core evolution [97] and how easy cloud computing makes it to obtain more VMs has had a profound effect on how software is designed: software services need to support running in a distributed manner, which adds significant complexity in terms of fault-tolerance, communication between components, and data consistency [98]. It is also desirable that the performance of the service increases (at least) linearly with each additional added instance.

A component that often becomes a performance bottleneck and has undergone such a change is the database. With the new scalability opportunities offered by cloud computing, we have over the last few years seen the emergence of new data storage options including NoSQL [99, 100] and NewSQL [101] data stores, to replace traditional relational SQL data stores. Their utility is sometimes questioned [102, 103], since they typically surrender some traits of SQL data stores (e.g. database-wide transaction consistency) to achieve their higher scalability and availability properties [104].

From experience, the ideal service component that benefits most from horizontal scalability is one designed to:

- perform a low number of quickly executed concise tasks, preferably just one;
- maintain no state information; and
- allow repeated idempotent calls.

If a service component is highly specialized and performs just a single, or a few related, tasks, increasing the number of deployed component instances gives a targeted boost to that particular function. Tasks should be concise, in that they do not solve several problems at once (this is a tenet of good object-oriented design, anyway). State information should be avoided, since it limits how well load can be distributed among instances: if only the instance I can serve requests related to a session it has already initiated, adding more instances cannot offload I. It is to prefer that new instances can connect to any instance, new or old, and get the same results. Similarly, if a service component instance is shut down in the middle of processing a request, it should be able to issue the same command to a different instance and get the sought response. This can only be guaranteed if idempotent calls are supported, meaning that several repetitions of a particular request does affect the system beyond what a single request would. Obviously, state has to be maintained somewhere (e.g. in a data store designed

for horizontal scalability, or a distributed caching system), but keeping components as free from it as possible is the goal of new frameworks that are emerging based on these ideas, e.g. the *Play! framework* [105] for web services.

Managing the deployment of all these instances of horizontally scalable service components is precisely what we set out to do in this thesis. Service components are obviously related to one another, whether they carry out the same or simply related tasks, and how they are deployed affects their performance and fault-tolerance. Without a language for the SP to express these relationships and how service components should be deployed, a suitable deployment cannot be guaranteed by the cloud IP.

CHAPTER 5
Thesis Contributions

This thesis contains contributions to cloud management and service deployment in cloud federations in three main areas: service orchestration, placement optimization, and monitoring. Challenges in these areas for federated cloud environments include dealing with a lack of control and trust across organizational domains. The work presented in the papers in this thesis highlight and present possible solutions to such problems, both from the point of view of the infrastructure provider and the service provider.

The main lessons learned during the work with this thesis are:

- **Influence rather than control**
  The IP and SP have conflicting goals and optimization criteria: the SP would optimally have the entire cloud site to itself, no network congestion, and each VM deployed on an individual dedicated physical host. The IP wants to make optimal use of the infrastructure, which means deploying as many services by different SPs on the infrastructure as possible. For this very simple reason, the SP cannot be granted control over how the infrastructure is used. And in contemporary (public) clouds, they are not. However, for some services, complete lack of any way to express placement restrictions is unacceptable. Offering influence to SPs, rather than control, is a reasonable trade-off.

- **Migration costs differ**
  The cost of migration, in terms of VM performance drop and network utilization (increases in congestion), needs to be taken into account when VMs are to be migrated. All VMs are not created equal in terms of migration cost: determining which VM is most suitable to move must take a large number of factors into account if we are to avoid picking the wrong VM. In particular, if we have granted the SP increased influence over placement, we might end up having to migrate a set of related VMs, should we choose one of them. Due to the impact on performance VM migration demonstrably has, heuristically determining which VM to migrate leads to more optimal use of resources and less performance degradation in the cloud site as a whole.

- **Monitoring systems can improve substantially**
  Monitoring, both the infrastructure itself on behalf of the IP and the deployed services on behalf of the SP, currently leaves much to be desired in terms of

compatibility, security, and ease of use. More work is required, and the field is rife with competing approaches, making one of the most obvious corner stones of any management process an exciting field for future developments.

5.1 Paper I

Paper I [106] deals with problems related to offloading (migrating) virtual machines from one cloud to another in an effort to provide the best possible placement, and monitoring of virtual machines, to ensure that they are given the agreed upon resources in the service-level agreement.

While migration is typically performed between physical hosts within a single cloud site (data center), it can be performed across wide area networks [58, 107], and thus within cloud federations, as well. The principle of location unawareness, as described in works by Hadas et al. [108], and a motivating design feature in the RESERVOIR project [82], suggests that the actual placement of a VM should be as transparent to the VM as possible.

Should a cloud A need to migrate away a VM to a partner cloud, that partner may in turn contact its partners (unbeknownst to A) to check if they can host the VM. If so, A's partner can accept responsibility for hosting the VM, but delegates it to one of its partners. And so forth. While operative commands to modify the VM's state need to be forwarded along such a chain, actually transferring the entire storage along the chain would be inefficient. In Paper I, we propose a more efficient alternative based on *transfer proxies*, which allows the originating cloud to transfer large data files to the final destination cloud directly, although it is unaware of which particular cloud that may be, due to being obscured by the chain of delegation.

A novel way of extracting monitoring data from VMs with the express goal of hiding complexity, loosening the coupling between the monitored service and the monitoring system, and requiring only minor modification to the service deployed in the VM is also presented.

5.2 Paper II

Paper II [83] expands on finding optimal placement and on monitoring of virtual machines, but does so also from a user perspective: as a user of a cloud infrastructure, one should be granted some level of *influence* (as opposed to outright *control*) over how virtual machines are placed. We propose adding *service structure* to service manifests, where SPs can define relationships between VMs and specify rules stipulating which components may or may not be deployed on the same host, cloud site, or geographical region as some other related VM. However, the SP is not granted any control over the placement, and cannot force the IP to use a certain set of hosts for the service deployment.

Service structure is expressed in terms of *affinity* and *anti-affinity* constraints on the host-, cloud-, and (geographical) region-level to *types* or *instances* of a given type.

Affinity requires co-placement on the defined level, whereas anti-affinity forbids it. The type and instance division makes it possible to express, for instance, that a single VM containing a database replica has a cloud-level affinity to all other database replicas (type), but a host-level anti-affinity to other database replicas (instance). We modeled relationships using directed acyclic graphs, an approach formalized and evaluated in Paper III.

A heuristic for determining the *migratability* of a (set of) VMs in a given placement mapping was also introduced, as part of determining which VMs would be easier to move, given that service structural constraints can imply that migration of one VM may require others need to be migrated, as well.

As a way to bridge the gap between incompatible monitoring systems in cloud federations, we suggested a novel monitoring system based on monitoring values with attached semantic meta-data.

5.3 Paper III

Paper III [84] expands further on the concept of service structure defined in Paper II. We provided a formal definition of the directed acyclic graphs that express the service structure and, as an illustration, how they can be translated into placement constraints matrices for input to an Integer Linear Programming (ILP) solver; presented a mathematical model for incorporating these constraint matrices in a placement engine based on ILP; and we also demonstrated through simulations that the proposed level of influence offers reasonable additional orchestration capabilities to users, while not making the management of the cloud infrastructure overly complex.

We simulated a cloud site of 80 hosts, and 100 VMs belonging to a single service to be placed upon these. Since service structure according to our definition does not apply to components in other services, i.e. components from a Service A cannot have any relationship to components from Service B (where A $\neq$ B, of course). Therefore, we simulated the presence of other services simply as a certain level of background load already deployed on each host. 15300 input permutations were performed, and the results show that there is a relationship between number of affinity constraints, background load, and placement algorithm execution time that matches intuition:

- a low amount of both constraints and background load makes placement quick and easy;
- more work is required if there are more complex affinity constraints at low background loads since the number of possible solutions increases; and
- placement of services with complex constraints on already loaded systems is also quick to compute, due to the low number of possible solutions.

Regardless, the computational impact on the placement engine is very low, and we argue that support for service structure should be added in future cloud orchestration tools.

CHAPTER 6
Future Work

The work presented in this thesis leaves a number of interesting topics to pursue, including (in addition to future work mentioned in the papers):

- **Modifying placement based on monitored application behavior**
  Monitoring of applications can provide hints towards the *fitness* (i.e. how good a placement mapping is) of the current placement mapping [109]. Large amounts of measured network traffic between two VMs can indicate that these should be placed close together, for instance. Large CPU consumption by a VM indicates that it should not be placed with other VMs that also consume large amounts of CPU. Coupled with predictive techniques, placement based on application insight can be made more intelligent and offer increased cloud infrastructure utilization. If the placement engine is already set up for dealing with service structures, modifications to placement can be done by inferring service structure, and taking this inferred service structure into account for upcoming placement optimization iterations.

- **Formal definition of migratability heuristic**
  In Paper II, we informally discuss the migratability heuristic. This should be formally defined, and continually improved upon by the research community as new possibilities for VM or container migration emerge.

- **Monitoring causally related events**
  As one of many improvements to contemporary monitoring systems, we would like to propose taking a lesson from distributed systems research in taking causal relationships between events into account for service monitoring. Isolating the cause of errors in services based on timestamped data is brittle due to the non-existence of perfectly synchronized clocks, and offers poor support for developers in determining the actual cause of an error condition. If causal relationships between events are part of the monitoring data stream, however, events in one causing errors in another can easily be identified. The main challenge is determining and distributing the information required for establishing causal relationships in an efficient manner.

- **Implementing service structure and evaluating exprimentally**
  With the recent interest in orchestration in the OpenStack community, imple-

menting service structure in the OpenStack placement engine (the Nova Scheduler) would be a worthwhile endeavor. Doing so would allow one to perform true experiments, rather than simulations, and get data and further insight on what utility service structure offers, and at what cost.

- **Investigate orchestration of services running in containers**
  Containers are very well suited for deploying services and constructing service-oriented architectures. Due to the cheap isolation that practically comes for free, and how platform-related software such as application servers are to be delivered as containers, dividing a service into components that each run in separate containers will be the norm in the containerized cloud. But this also creates a demand for automatically provisioning, orchestrating, and managing these containerized services in an automated fashion. The requirements and future challenges that this gives rise to need to be studied further.

Bibliography

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a new computing infrastructure*. Morgan Kaufmann, 2004.

[2] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*, pp. 1–10, IEEE, 2008.

[3] IBM, "System/360 announcement." 4 1964.

[4] D. Ritchie and K. Thompson, "The UNIX time-sharing system," *Communications of the ACM*, vol. 17, no. 7, pp. 365–375, 1974.

[5] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, *The Physiology of the Grid*, ch. 8, pp. 217–250. Wiley, 2003.

[6] I. Foster, "What is the grid? A three point checklist," *GRID today*, vol. 1, no. 6, pp. 32–36, 2002.

[7] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[8] G. Aad, E. Abat, J. Abdallah, A. Abdelalim, A. Abdesselam, O. Abdinov, B. Abi, M. Abolins, H. Abramowicz, E. Acerbi, *et al.*, "The ATLAS experiment at the CERN large hadron collider," *Journal of Instrumentation*, vol. 3, no. 08, p. S08003, 2008.

[9] L. A. Belady and C. J. Kuehner, "Dynamic space-sharing in computer systems," *Commun. ACM*, vol. 12, pp. 282–288, May 1969.

[10] L. M. Vaquero, L. Rodero-Merino, J. Cáceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2009.

[11] P. Mell and T. Grance, "The NIST definition of cloud computing," *NIST special publication*, vol. 800, p. 145, 2011.

[12] R. Adair, R. Bayles, L. Comeau, and R. Creasy, "A Virtual Machine System for the 360/40 - Cambridge scientific center report 320," tech. rep., IBM, May 1966.

[13] G. Popek and R. Goldberg, "Formal requirements for virtualizable third generation architectures," *Communications of the ACM*, vol. 17, no. 7, pp. 412–421, 1974.

[14] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 164–177, ACM, October 2003.

[15] Advanced Micro Devices, AMD Inc., "AMD virtualization codenamed "Pacifica" technology, secure virtual machine architecture reference manual." Reference manual, 5 2005.

[16] J. S. Bozman and G. P. Chen, "Optimizing hardware for x86 server virtualization." White Paper.

[17] K. Adams and O. Agesen, "A comparison of software and hardware techniques for x86 virtualization," in *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pp. 2–13, ACM, 2006.

[18] J. Walters, V. Chaudhary, M. Cha, S. Guercio Jr, and S. Gallo, "A comparison of virtualization technologies for HPC," in *22nd International Conference on Advanced Information Networking and Applications*, pp. 861–868, IEEE, 2008.

[19] P. Padala, X. Zhu, Z. Wang, S. Singhal, K. G. Shin, *et al.*, "Performance evaluation of virtualization technologies for server consolidation," *HP Labs Technical Report*, 2007.

[20] S. Crosby and D. Brown, "The virtualization reality," *ACM Queue*, pp. 34–41, December/January 2006-2007.

[21] L. Youseff, R. Wolski, B. Gorda, and C. Krintz, "Paravirtualization for HPC systems," in *Frontiers of High Performance Computing and Networking ISPA 2006 Workshops* (G. Min *et al.*, eds.), pp. 474–486, 2006.

[22] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck, "A service level agreement language for dynamic electronic services," *Electronic Commerce Research*, vol. 3, no. 1-2, pp. 43–59, 2003.

[23] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck, "Web service level agreement (WSLA) language specification," *IBM Corporation*, 2003.

[24] B. Rochwerger, C. Váquez, D. Breitgand, D. Hadas, M. Villari, P. Massonet, E. Levy, A. Galis, I. Llorente, R. Montero, Y. Wolfsthal, K. Nagin, L. Larsson, and F. Galán, "An architecture for federated cloud computing," *Cloud Computing*, 2010.

[25] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galán, "The RESERVOIR model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, 2009. Paper 4.

[26] A. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. 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," *Future Generation Computer Systems*, vol. 28, pp. 66 – 77, 2011.

[27] G. Kroah-Hartman, "Linux kernel 3.8.1 change log." Online resource: `https://www.kernel.org/pub/linux/kernel/v3.x/ChangeLog-3.8.1`, 2 2013.

[28] D. Strauss, "The future cloud is container, not virtual machines," *Linux Journal*, vol. 2013, Apr. 2013.

[29] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 275–287, Mar. 2007.

[30] D. Price and A. Tucker, "Solaris zones: Operating system support for consolidating commercial workloads.," in *LISA*, vol. 4, pp. 241–254, 2004.

[31] P.-H. Kamp and R. N. Watson, "Jails: Confining the omnipotent root," in *Proceedings of the 2nd International SANE Conference*, vol. 43, p. 116, 2000.

[32] Docker Inc., "What is Docker? an open platform for distributed apps." 11 2014.

[33] L. Tomas and J. Tordsson, "An autonomic approach to risk-aware data center overbooking," *IEEE Transactions on Cloud Computing*, vol. 2, pp. 292–305, July 2014.

[34] M. H. Ferdaus and M. Murshed, "Energy-aware virtual machine consolidation in IaaS cloud computing," in *Cloud Computing* (Z. Mahmood, ed.), Computer Communications and Networks, pp. 179–208, Springer International Publishing, 2014.

[35] C. Liu, B. T. Loo, and Y. Mao, "Declarative automated cloud resource orchestration," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, (New York, NY, USA), pp. 26:1–26:8, ACM, 2011.

[36] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *31st International Conference on Distributed Computing Systems (ICDCS), 2011*, pp. 559–570, IEEE, 2011.

[37] S. Dustdar, Y. Guo, R. Han, B. Satzger, and H.-L. Truong, "Programming directives for elastic computing," *IEEE Internet Computing*, vol. 16, no. 6, pp. 72–77, 2012.

[38] G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar, "Multi-level elasticity control of cloud services," in *Service-Oriented Computing* (S. Basu, C. Pautasso, L. Zhang, and X. Fu, eds.), vol. 8274 of *Lecture Notes in Computer Science*, pp. 429–436, Springer Berlin Heidelberg, 2013.

[39] G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar, "SYBL+MELA: Specifying, monitoring, and controlling elasticity of cloud services," in *Service-Oriented Computing* (S. Basu, C. Pautasso, L. Zhang, and X. Fu, eds.), vol. 8274 of *Lecture Notes in Computer Science*, pp. 679–682, Springer Berlin Heidelberg, 2013.

[40] R. Aiello and L. Sachs, *Configuration Management Best Practices: Practical Methods that Work in the Real World*. Addison-Wesley Professional, 1st ed., 2010.

[41] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "TOSCA: Portable automated deployment and management of cloud applications," in *Advanced Web Services* (A. Bouguettaya, Q. Z. Sheng, and F. Daniel, eds.), pp. 527–549, Springer New York, 2014.

[42] E. Kolodner, S. Tal, D. Kyriazis, D. Naor, M. Allalouf, L. Bonelli, P. Brand, A. Eckert, E. Elmroth, S. Gogouvitis, D. Harnik, F. Hernandez, M. Jaeger, E. Lakew, J. Lopez, M. Lorenz, A. Messina, A. Shulman-Peleg, R. Talyansky, A. Voulodimos, and Y. Wolfsthal, "A cloud environment for data-intensive storage services," in *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom) 2011*, pp. 357–366, Nov 2011.

[43] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "RACS: a case for cloud storage diversity," in *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 229–240, ACM, 2010.

[44] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: Outsourcing computation without outsourcing control," in *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, (New York, NY, USA), pp. 85–90, ACM, 2009.

[45] C. Wang, K. Ren, W. Lou, and J. Li, "Toward publicly auditable secure cloud data storage services," *Network, IEEE*, vol. 24, pp. 19–24, July 2010.

[46] S. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *Communications Magazine, IEEE*, vol. 51, pp. 136–141, February 2013.

[47] H. Kim and N. Feamster, "Improving network management with software defined networking," *Communications Magazine, IEEE*, vol. 51, pp. 114–119, February 2013.

[48] E. Elmroth, F. Galán, D. Henriksson, and D. Perales, "Accounting and billing for federated cloud infrastructures," in *GCC '09: Proceedings of the 2009 Eighth International Conference on Grid and Cooperative Computing*, (Washington, DC, USA), pp. 268–275, IEEE Computer Society, 2009.

[49] D. Milojicic, I. M. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," *IEEE Internet Computing*, vol. 15, no. 2, pp. 11–14, 2011.

[50] S. A. Baset, "Open source cloud technologies," in *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, (New York, NY, USA), pp. 28:1–28:2, ACM, 2012.

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

[52] S. Tesfatsion, E. Wadbro, and J. Tordsson, "A combined frequency scaling and application elasticity approach for energy-efficient cloud computing," *Sustainable Computing: Informatics and Systems*, 2014. To appear, preprint available online: `http://dx.doi.org/10.1016/j.suscom.2014.08.007`.

[53] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and migration cost aware application placement in virtualized systems," in *Middleware 2008* (V. Issarny and R. Schantz, eds.), vol. 5346 of *Lecture Notes in Computer Science*, pp. 243–264, Springer Berlin / Heidelberg, 2008.

[54] A. Verma, P. De, V. Mann, T. Nayak, A. Purohit, G. Dasgupta, and R. Kothari, "Brownmap: Enforcing power budget in shared data centers," in *Middleware 2010* (I. Gupta and C. Mascolo, eds.), vol. 6452 of *Lecture Notes in Computer Science*, pp. 42–63, Springer Berlin Heidelberg, 2010.

[55] W. Li, J. Tordsson, and E. Elmroth, "Virtual machine placement for predictable and time-constrained peak loads," in *Economics of Grids, Clouds, Systems, and Services*, pp. 120–134, Springer Berlin Heidelberg, 2012.

[56] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 358–367, 2012.

[57] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of delta compression techniques for efficient live migration of large virtual machines," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pp. 111–120, ACM, 2011.

[58] W. Voorsluys, J. Broberg, S. Venugopal, and R. Buyya, "Cost of virtual machine live migration in clouds: A performance evaluation," in *Cloud Computing* (M. Jaatun, G. Zhao, and C. Rong, eds.), vol. 5931 of *Lecture Notes in Computer Science*, pp. 254–265, Springer Berlin Heidelberg, 2009.

[59] H. Liu, H. Jin, C.-Z. Xu, and X. Liao, "Performance and energy modeling for live migration of virtual machines," *Cluster computing*, vol. 16, no. 2, pp. 249–264, 2013.

[60] Q. Huang, F. Gao, R. Wang, and Z. Qi, "Power consumption of virtual machine live migration in clouds," in *Communications and Mobile Computing (CMC), 2011 Third International Conference on*, pp. 122–125, IEEE, 2011.

[61] Y. Zu, T. Huang, and Y. Zhu, "An efficient power-aware resource scheduling strategy in virtualized datacenters," in *International Conference on Parallel and Distributed Systems (ICPADS), 2013)*, pp. 110–117, Dec 2013.

[62] S.-J. Yang, L.-C. Chen, H.-H. Tseng, H.-K. Chung, and H.-Y. Lin, "Designing automatic power saving on virtualization environment," in *12th IEEE International Conference on Communication Technology (ICCT), 2010)*, pp. 966–970, Nov 2010.

[63] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), 2010*, pp. 826–831, May 2010.

[64] A. Beloglazov and R. Buyya, "Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, pp. 1366–1379, July 2013.

[65] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi, "Reactive consolidation of virtual machines enabled by postcopy live migration," in *Proceedings of the 5th international workshop on Virtualization technologies in distributed computing*, pp. 11–18, ACM, 2011.

[66] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi, "Enabling instantaneous relocation of virtual machines with a lightweight VMM extension," in *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), 2010*, pp. 73–83, IEEE, 2010.

[67] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[68] D. Breitgand, Z. Dubitzky, A. Epstein, A. Glikson, and I. Shapira, "SLA-aware resource over-commit in an IaaS cloud," in *Proceedings of the 8th International Conference on Network and Service Management*, CNSM '12, (Laxenburg, Austria, Austria), pp. 73–81, International Federation for Information Processing, 2013.

[69] L. Tomás and J. Tordsson, "Improving cloud infrastructure utilization through overbooking," in *Proceedings of the ACM Cloud and Autonomic Computing Conference*, ACM, 2013. To appear.

[70] D. Breitgand and A. Epstein, "SLA-aware placement of multi-virtual machine elastic services in compute clouds," in *IFIP/IEEE International Symposium on Integrated Network Management (IM), 2011*, pp. 161–168, May 2011.

[71] R. Ghosh and V. Naik, "Biting off safely more than you can chew: Predictive analytics for resource over-commit in iaas cloud," in *IEEE 5th International Conference on Cloud Computing (CLOUD), 2012*, pp. 25–32, June 2012.

[72] A. Bahga, V. K. Madisetti, *et al.*, "Synthetic workload generation for cloud computing applications," *Journal of Software Engineering and Applications*, vol. 4, no. 07, p. 396, 2011.

[73] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *IEEE Network Operations and Management Symposium (NOMS), 2012*, pp. 1287–1294, April 2012.

[74] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth, "Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control," in *Proceedings of the 3rd workshop on Scientific Cloud Computing*, pp. 31–40, ACM, 2012.

[75] A. Eldin, A. Rezaie, A. Mehta, S. Razroev, S. Sjöstedt-de Luna, O. Seleznjev, J. Tordsson, and E. Elmroth, "How will your workload look like in 6 years? Analyzing wikimedia's workload," in *IEEE International Conference on Cloud Engineering (IC2E), 2014*, pp. 349–354, March 2014.

[76] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano, "Auto-scaling techniques for elastic applications in cloud environments," Research EHU-KAT-IK, Department of Computer Architecture and Technology, UPV/EHU, 2012.

[77] E. Caron, F. Desprez, and A. Muresan, "Forecasting for grid and cloud computing on-demand resources based on pattern matching," in *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), 2010*, pp. 456–463, Nov 2010.

[78] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pp. 204–212, April 2012.

[79] M. Sedaghat, F. Hernandez-Rodriguez, and E. Elmroth, "A virtual machine repacking approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, CAC '13, (New York, NY, USA), pp. 6:1–6:10, ACM, 2013.

[80] S. Garg, Y. Huang, C. Kintala, and K. S. Trivedi, "Minimizing completion time of a program by checkpointing and rejuvenation," *SIGMETRICS Perform. Eval. Rev.*, vol. 24, pp. 252–261, May 1996.

[81] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez, "Brownout: Building more robust cloud applications," in *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, (New York, NY, USA), pp. 700–711, ACM, 2014.

[82] M. B. Yehuda, O. Biran, D. Breitgand, K. Meth, B. Rochwerger, E. Salant, E. Silvera, S. Tal, Y. Wolfsthal, J. Cáceres, J. Hierro, W. Emmerich, A. Galis, L. Edblom, E. Elmroth, D. Henriksson, F. Hernández, J. Tordsson, A. Hohl, E. Levy, A. Sampaio, B. Scheuermann, M. Wusthoff, J. Latanicki, G. Lopez, J. Marin-Frisonroche, A. Dörr, F. Ferstl, S. Beco, F. Pacini, I. Llorente, R. Montero, E. Huedo, P. Massonet, S. Naqvi, G. Dallons, M. Pezzé, A. Puliato, C. Ragusa, M. Scarpa, and S. Muscella, "RESERVOIR — an ICT infrastructure for reliable and effective delivery of services as utilities," tech. rep., IBM Haifa Research Laboratory, 2008.

[83] L. Larsson, D. Henriksson, and E. Elmroth, "Scheduling and monitoring of internally structured services in cloud federations," in *Proceedings of IEEE Symposium on Computers and Communications 2011*, pp. 173–178, IEEE Computer Society, June 2011.

[84] D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth, "Modeling and placement of structured cloud services," *IEEE Transactions on Cloud Computing (TCC)*, vol. PP, October 2014. To appear.

[85] Amazon Web Services, Inc., "AWS CloudFormation – configuration management & cloud orchestration." 2014.

[86] OpenStack Foundation, "Heat — OpenStack." 2014.

[87] OASIS, "OASIS cloud application management for platforms (CAMP) TC." 2014.

[88] OASIS, "OASIS topology and orchestration specification for cloud applications (TOSCA) TC." 2014.

[89] T. Binz, G. Breiter, F. Leyman, and T. Spatzier, "Portable cloud services using TOSCA," *Internet Computing, IEEE*, vol. 16, pp. 80–85, May 2012.

[90] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe, "CloudNet: Dynamic pooling of cloud resources by live WAN migration of virtual machines," *SIGPLAN Not.*, vol. 46, pp. 121–132, Mar. 2011.

[91] T. Hirofuchi, H. Ogawa, H. Nakada, S. Itoh, and S. Sekiguchi, "A live storage migration mechanism over wan for relocatable virtual machine services on clouds," in *Proceedings of the 2009 9th IEEE/ACM International Symposium*

on Cluster Computing and the Grid, CCGRID '09, (Washington, DC, USA), pp. 460–465, IEEE Computer Society, 2009.

[92] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability and integrity layer for cloud storage," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, (New York, NY, USA), pp. 187–198, ACM, 2009.

[93] E. B. Lakew, F. Hernandez-Rodriguez, L. Xu, and E. Elmroth, "Management of distributed resource allocations in multi-cluster environments," in *IEEE 31st International Performance Computing and Communications Conference (IPCCC), 2012*, pp. 275–284, IEEE, 2012.

[94] M. Papazoglou, *Web services: principles and technology*. Pearson Education Limited, 2008.

[95] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, Irvine, California, 2000.

[96] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Trans. Internet Technol.*, vol. 2, pp. 115–150, May 2002.

[97] T. Peierls, B. Goetz, J. Bloch, J. Bowbeer, D. Lea, and D. Holmes, *Java Concurrency in Practice*. Pearson Education, 2006.

[98] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[99] K. Chodorow, *MongoDB: The Definitive Guide*. O'Reilly Media, 2013.

[100] K. Banker, *MongoDB in Action*. Greenwich, CT, USA: Manning Publications Co., 2011.

[101] M. Stonebraker and A. Weisberg, "The VoltDB main memory DBMS," *IEEE Data Eng. Bull.*, vol. 36, no. 2, pp. 21–27, 2013.

[102] N. Leavitt, "Will NoSQL databases live up to their promise?," *Computer*, vol. 43, pp. 12–14, Feb 2010.

[103] M. Stonebraker, "SQL databases v. NoSQL databases," *Commun. ACM*, vol. 53, pp. 10–11, Apr. 2010.

[104] R. Cattell, "Scalable SQL and NoSQL data stores," *SIGMOD Rec.*, vol. 39, pp. 12–27, May 2011.

[105] J. Richard-Foy, *Play Framework Essentials*. Packt Publishing, 2014.

[106] 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)*, pp. 253–260, IEEE Computer Society, August 2009.

[107] F. Travostino, "Seamless live migration of virtual machines over the MAN/WAN," in *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, p. 290, ACM, November 2006.

[108] D. Hadas, S. Guenender, and B. Rochwerger, "Virtual network services for federated cloud computing," Tech. Rep. H-0269, IBM Technical Reports, Nov. 2009.

[109] J. Mars and L. Tang, "Understanding application contentiousness and sensitivity on modern multicores," in *Advances in Computers*, vol. 91, pp. 59–85, New York, NY, USA: ACM, 2013.

I

Paper I

Interfaces for Placement, Migration, and Monitoring of Virtual Machines in Federated Clouds*

Erik Elmroth and Lars Larsson

Dept. Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden
{elmroth, larsson}@cs.umu.se
http://www.cloudresearch.org/

Abstract: Current cloud computing infrastructure offerings are lacking in interoperability, which is a hindrance to the advancement and adoption of the cloud computing paradigm. As clouds are made interoperable, federations of clouds may be formed. Such federations are from the point of view of the user not burdened by vendor lock-in, and opens for business possibilities where a market place of cloud computing infrastructure can be formed. Federated clouds require unified management interfaces regarding the virtual machines (VMs) that comprise the services running in the cloud federation. Standardization efforts for the required management interfaces have so far focused on definition of description formats regarding VMs, and the control of already deployed VMs. We propose technology neutral interfaces and architectural additions for handling placement, migration, and monitoring of VMs in federated cloud environments, the latter as an extension of current monitoring architectures used in grid computing. The interfaces presented adhere to the general requirements of scalability, efficiency, and security in addition to specific requirements related to the particular issues of interoperability and business relationships between competing cloud computing infrastructure providers. In addition, they may be used equally well locally and remotely, creating a layer of abstraction that simplifies management of virtualized service components.

Key words: cloud computing, federated cloud, migration, monitoring

* By permission of IEEE

Interfaces for Placement, Migration, and Monitoring of Virtual Machines in Federated Clouds

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

***Abstract*—Current cloud computing infrastructure offerings are lacking in interoperability, which is a hindrance to the advancement and adoption of the cloud computing paradigm. As clouds are made interoperable, federations of clouds may be formed. Such federations are from the point of view of the user not burdened by vendor lock-in, and opens for business possibilities where a market place of cloud computing infrastructure can be formed. Federated clouds require unified management interfaces regarding the virtual machines (VMs) that comprise the services running in the cloud federation. Standardization efforts for the required management interfaces have so far focused on definition of description formats regarding VMs, and the control of already deployed VMs. We propose technology-neutral interfaces and architectural additions for handling placement, migration, and monitoring of VMs in federated cloud environments, the latter as an extension of current monitoring architectures used in Grid computing. The interfaces presented adhere to the general requirements of scalability, efficiency, and security in addition to specific requirements related to the particular issues of interoperability and business relationships between competing cloud computing infrastructure providers. In addition, they may be used equally well locally and remotely, creating a layer of abstraction that simplifies management of virtualized service components.**

I. Introduction

Cloud computing is growing increasingly popular in the IT business, and industry leaders such as Bill Joy of Sun Microsystems fame estimated that utility and pervasive computing such as cloud computing may be a trillion dollar business (as quoted in [1]). Implementations and architectures vary widely, and the cloud computing offerings of one vendor are often not guaranteed to be compatible with those of some other vendor, thus creating vendor lock-in.

The US National Institute of Standards and Technology is currently working on a definition of cloud computing [2], where cloud computing is stated as having five key characteristics: (a) on-demand self-service; (b) ubiquitous network access; (c) location independent resource pooling; (d) rapid elasticity; and (e) pay per use. There are also three delivery models: (i) Software as a Service; (ii) Platform as a Service; and (iii) Infrastructure as a Service. These delivery models differ substantially in scope. Our focus is cloud infrastructure, which we use to denote the infrastructure required for hosting virtualized services, and thus Infrastructure as a Service (IaaS). In particular, the infrastructure provided should be flexible and adapt to the dynamics in demand for a deployed service in a cost-efficient way. This includes optimizing resource usage at the infrastructure provider's site locally (e.g. reducing the number of powered machines and consolidation of load), as well using bidirectional contracts with other infrastructure provider sites. A federated cloud is one where (competing) infrastructure providers can reach cross-site agreements of cooperation regarding the deployment of service components in a way similar to how electrical power providers provision capacity from each other to cope with variations in demand. Such collaboration increases location independence. To achieve this vision, the cloud sites in the federation must conform to common interfaces regarding virtualized service components and employ compatible virtualization platforms.

Cloud computing leverages technologies such as Grid computing and virtualization to provide the basic infrastructure and platform for the cloud computing stack. Services (or components thereof) are generally deployed in virtual machines (VMs) that can be defined in terms of the required virtual hardware resources and network connectivity. As demand dynamically fluctuates, the resources can be scaled up or down to allow the customer to pay only for the capacity needed and to reduce the costs for the cloud provider. Rules for this type of dynamic elasticity in resource demands are formulated as Service Level Agreements (SLAs), either in terms of virtual hardware resources and the utilization thereof, or as Key Performance Indicators (KPIs). KPIs include e.g. application-specific terms such as the number of jobs in a work queue.

The contribution of this article is two-fold. First, we analyze the current state of the art of (proposed) standards for VM management interfaces to find where enhancements are needed, and based on usage scenarios, determine what the requirements of such enhancements are. Second, we propose interfaces for supporting the additional required functionality, adding placement, migration, and monitoring interfaces to the interfaces already defined in (proposed) standards. We argue that placement is a special case of migration, and thus can be supported by the same homogeneous interface operations. We define these interface operations, and introduce a component called Transfer proxy that is used to carry out the file transfers. An algorithm that utilizes Transfer proxies for such transfer is presented. With regard to monitoring interfaces, we present

additions to the descriptors of VMs that configure a general monitoring system. In addition to discussing how monitoring data should be transferred in a general cloud computing platform and presenting our solution, we introduce a novel approach for supporting application-level measurements in virtualized cloud service components while requiring a minimal change to the application itself.

The remainder of the article is structured as follows. In Section II, we discuss the rationale and requirements regarding migration and monitoring interfaces. Section III presents a scalable and decentralized solution to handling migration of VMs that fulfills these requirements. In Section IV, we present additions to current standardized descriptors of VMs that relate to monitoring of both the infrastructure itself and of the applications that run in the VMs. These suggestions are later discussed in Section V. Related work in some currently active projects is presented in Section VI. Section VII contains a summary and conclusions.

II. Cross-site Management in Federated Cloud Environments

The previous section referred to the working definition of cloud computing by NIST, which states a number of key characteristics for cloud computing. As the cloud computing paradigm has become increasingly popular, various cloud infrastructure architectures have been developed. For the purpose of this work, we assume that a general cloud computing architecture requires functionality that may well be implemented by components divided into a number of layers with clear *separation of concern* and responsibilities. Although the number of layers vary among cloud computing implementations, we assume that a general architecture is divided in at least the following two conceptual layers: (a) the *Management layer*, responsible for overall management, such as admission control, decisions regarding where to deploy VEEs in the cloud (referred to as *placement* of VMs), control of resources, accounting for usage, etc; and (b) the *Implementation layer*, responsible for hosting the VMs, maintaining virtual networks across sites, etc.

To note the difference in responsibilities of the sites in the federation, we let *primary site* denote the site that has been contractually bound by the service provider (customer) to manage the VMs that comprise a service and other sites be denoted as *remote sites*.

The layered architecture clearly marks a separation of concern between the Management and Implementation layers. There is also a separation of concern between cloud sites, as sites may avoid disclosing exact information regarding the placement of VMs on physical host machines. We call this the *principle of location unawareness*. The principle is one of the pillars of the RESERVOIR project [3], motivated by use cases where cloud infrastructure providers do not wish to disclose information regarding the size or utilization of the infrastructure, due to the sensitivity of such information. Also, it is more convenient for a site to make use of the infrastructure offered as a service by the remote site, relying on SLA compliance rather than minute resource management of VMs running remotely. This also enables infrastructure-less resource brokers to act in the federated cloud.

We use the term *primary placement* for denoting the selection of, and transfer to, a site for future deployment of a VM that has not been run yet, or one that has been shut down after having run previously. A VM is regarded as *deployed* when all its files have been transferred to the host that will run it, it has been activated (or "booted up"), and is currently running. Transferring a VM from one VM host to another, possibly with either one or both hosts running at remote sites, is called *VM Migration*. Note that both local (across local hosts) and remote (cross-site) migration is included in this definition. The process of migration may be initiated reactively due to the submission of new VMs to deploy, or proactively to continuously optimize placement to avoid SLA violations, evenly distribute system load, consolidate VMs to few hosts for power savings, etc.

We claim that placement of a VM that has not yet been run, or one that has been shut down, may be regarded as a special case of cold migration, as the only conceptual difference is that there is no serialized runtime state to transfer. We therefore, for the purposes of this article, consider primary placement of a VM to be no different from migration. This lets us employ the same protocol in both contexts, and as a result, simplifies management operations.

We consider three general usage scenarios related to migration of VMs. In the first (Usage scenario 1), the primary site initiates the VM migration due to lack of local resources, whereas in the second (Usage scenario 2), a remote site currently running a VM on behalf of the primary site is unable to continue doing so and requires that the primary site finds new placement for the VM immediately. The third (Usage scenario 3) is the case when a VM running locally at a certain host must be migrated to another host, running at the same site. Such migration may be used for e.g. energy conservation (consolidation of VMs to a low number of hosts) or for load-balancing (maximizing distribution of VMs among local hosts).

Requests to deploy a VM on behalf of some other site are to be considered as offers of contracts. These contracts may contain SLAs that stipulate the requirements on both parties, and from the Infrastructure Provider's point of view, they could be costly to violate. Thus, in a migration scenario, the involved sites must be able to verify the identities of each other in a secure manner. If a site is unable to deploy a VM due to lack of resources, it should be able to delegate the responsibility to some partner site. Some of the aforementioned SLAs are expressed in terms of VM resource allocation (e.g. "5 Mb/second bandwidth 95% of the time over some time period"), whereas others are expressed in application-specific terms (e.g. "no more than 1000 concurrent database users at any time"). In order to avoid violating the SLAs, the Management layer is responsible for scaling the size and amount of the VMs allocated to the service up and down, as required. For reasons such as VM host maintenance, a site responsible for running a VM may require that a VM is migrated away from it. Such requests cannot be ignored by the site that has delegated responsibility to the site.

A. State of the Art Technologies and Standards

The state of the art technologies and standards include support for some of the features mentioned in the usage scenarios presented in Section II. In the following sections, we briefly consider the topics of two of these sets of standards: VM descriptors and VM control interfaces.

1) VM Descriptors : In order to be deployable in the federated cloud, each VM must be described in terms of required resources, its network membership and configuration, have a unique identifier, etc. One standard dealing with this issue is the Open Virtualization Format (OVF) [4] by the Distributed Management Task Force (DMTF). OVF is supported by leading technologies such as the open source Xen [5] and the proprietary VMware [6].

We use the term *VM Descriptor* to refer to the description of a single VM. Note that the OVF does not include a section related to monitoring of VMs, but rather the configuration of the virtual hardware and some basic configuration of this hardware, e.g. network addresses, etc.

2) VM Control Operations: Once deployed, VMs must be controlled by the management layer. A general Control interface for VMs is used to carry out two types of operations: (a) modifying the Descriptor of a VM, making it possible to alter the amount of resources allocated, or any other configurable parameter related to the VM; and (b) updating the runtime state of the VM, e.g. by shutting down a running VM or activating a hibernated one. The DMTF has proposed standards for these operations in [7] and [8], respectively. The states of VMs are described in [9].

B. Requirements

The usage scenarios in Section II require additional functionality not offered in the state of the art technologies described in Section II-A. We summarize these requirements below:

1) A cryptographically secure identification scheme must be used to authenticate the parties involved in all cross-site communication.
2) Monitoring of VM resource allocation must be provided.
3) Monitoring of application-specific values must be provided.
4) Migration of VMs must be supported.
5) Migration must be possible to delegate.
6) A remote site must be able to make a request to the site from which the delegation of responsibility came to initiate immediate migration of a VM away from the remote site. Such requests must not be ignored.

The following sections contain an approach for extending upon the state of the art and meeting these requirements.

III. Migration

Migration of a VM requires that the source and destination host machines coordinate the transfer of the VM's definition and state files. We wish to set up such a transfer without losing the separation of concern between the Management layer and the Implementation layer or, similarly, the cross-site location-unawareness, while still adhering to the overall requirements of efficiency, security, and scalability.

To this end, we propose that each site maintains at least one *Transfer proxy* component in the Management layer. A Transfer proxy is a component that provides a layer of abstraction and information hiding, while at the same time associates an upcoming transfer with a transfer negotiation process. For scalability reasons, a site may wish to deploy several Transfer proxies. On behalf of the site, the Transfer proxies maintain a mapping between a VM identifier and where (at the Implementation level) its files are to be placed or read from. This mapping is called the *Transfer token*. The Transfer token is a unique identifier whose meaning is only known within the originating Transfer proxy. In this way, only the address of the Transfer proxy at a site is disclosed, not the address (or any other internal information) of the VM host at the Implementation level.

Using an approach similar to the File Transfer Protocol (FTP [10, Section 2.3]), the system uses separate control and transfer channels (out-of-band). The control channel is entirely contained within the Migration management protocol, maintaining the separation between the Management and Implementation layers.

A. Roles

Migration of a VM can be initiated for several reasons, and by several parties. In all cases, we can identify the following roles:

- *Controller*. The site responsible for setting up the transfer of the VM is denoted the Controller. It needs not awareness of the actual current placement of the VM, and is unaware of whether it communicates directly with the Source or Destination sites or indirectly via any number of Intermediary sites.
- *Source*. The Source is where the VM currently resides.
- *Destination*. The Destination is where the VM is to be deployed.
- *Intermediary*. Between the Controller and the Source/Destination sites, there may be any number of Intermediaries. On an intuitive level, the Intermediary acts as Controller on behalf of another Controller. This offers a layer of abstraction regarding the actual placement of the VM.

A site may, depending on context, take on several of the roles, e.g. act as both Controller and Source in the case of migration of a VM from the primary site that was also initiated by the primary site. Also, it should be noted that any site already involved in the management of a VM due to a prior delegation of responsibility can take on the role of Controller if needed.

Figure 1 shows an overview of the migration process. Because management of a VM can be delegated, the Controller is unaware of the actual placement of the VM on which site or host a VM is placed. Thus, the Controller only keeps track of what Intermediary site to contact in order to control the VM.

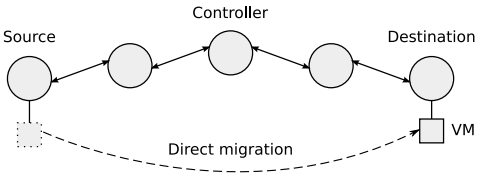


Figure 1: Migration process overview: solid lines denote control messages that are sent indirectly from the Controller to the Source site (where the VM is currently running) to the Destination site (where the VM should be migrated to), whereas dashed lines are used for direct network communication between sites. The transfer of the VM is carried out directly between the Source and Destination for efficiency reasons.

B. Operations

The operations of the protocol are as follows:

- *Migration request* (from the Controller and Intermediaries to possible Destinations). Migration requests are sent to possible future Destinations, to verify if the prospect site can accept a VM migration. The VM Descriptor and VM identifier are sent as input to the operation. Note that, depending on context and infrastructural/architectural policies, the remote site may in turn forward the request to another site and thus delegating the responsibility of managing the VM. The receiver of a Migration request may either accept or reject, depending on local site heuristics that take business rules and infrastructure capabilities into account. The return value contains a list of the information necessary to later issue the "Initiate transfer" operation presented below. This information includes the Transfer proxy's URI and the Transfer token.
- *Forced migration* (from either the Controller to a Source, or from the Source to the site that delegated the VM to the Source). A Forced migration request may be sent to indicate that a VM must immediately be prepared for migration away from its current placement. This call may be initiated by either a remote or the primary site, as a result of e.g. the Source site shutting down a host machine for maintenance (operation initiated remotely) or the primary site wishing to avoid SLA violations that are to be expected at the remote site (operation initiated by the primary site). Note that in addition to the Controller and Source sites, any Intermediary site involved with the VM may initiate a Forced migration as they are also stakeholders and may wish to optimize the placement of the VM in accordance with some site criteria.
- *Initiate transfer* (from the Controller to the Destination, via Intermediaries). The Initiate transfer operation is used to trigger the previously negotiated migration of the VM. The operation parameters contain the URIs and tokens of both the Source's and Destination's Transfer proxies.
- *Transfer verification* (from the Controller to the Source, via Intermediaries). This operation is issued by the Controller to the Source to verify that a given transfer was completed successfully.

C. Migration algorithm

In this section, we present the algorithms for the Controller, Destination, and Source sites. Intermediaries should only forward the requests along the path to the intended destination, and thus do not warrant an algorithm description.

1) At the Controller: The Controller migration algorithm is initiated either by the Controller being in the process of accepting a new service or VM, performing periodic placement optimization, or by a remote site requesting that a given VM should be migrated away invoking the Forced migration operation on the previous Intermediary in the chain of Intermediaries. Such an invocation may be regarded as an optional Step 0 for the Controller in the following algorithm. Note that *any* site along the path of responsibility may act as a Controller.

Name: Main migration algorithm.
Runs at: Controller.
Input: An event causing replacement has occurred.
Result: VM migrated to new location.

1) Let D be an empty list.
2) Let P denote the set of possible Destination sites, including the local site. For each $p \in P$ (performed in parallel):
   a) Call Migration request on p. The return value is a list. Add each returned tuple of $\langle p_{uri}, p_{tok} \rangle$ containing the returned Transfer proxy URI and Transfer token to the list D.
3) Sort D according to greatest benefit for the site and choose $d \in D$ to be the Destination.
4) Let s denote the Source and Invoke Forced migration on s. Store returned Transfer proxy URI as s_{uri} and Transfer token as s_{tok}.
5) Invoke Initiate migration on d, passing the tuple $\langle s_{uri}, s_{tok}, d_{uri}, d_{tok} \rangle$ as parameter. Store result as d_{stat}.
6) Invoke Transfer verification on s passing $\langle s_{uri}, s_{tok} \rangle$ as parameter. Store result as s_{stat}.
7) Unless $d_{stat} = s_{stat}$, go back to Step 5.

2) At the Destination: The Destination can be called by the Controller for two reasons, to handle migration requests and to react to the transfer initiation operation call.
Name: Migration request handler.
Runs at: Destination.
Input: VM identifier, VM descriptor.
Result: List of possible placement options.

1) Let D be an empty list.
2) If placement is possible locally, then for each possible local host h:
   a) Let T denote the set of Transfer proxies. Choose $t \in T$.
   b) From t, obtain Transfer token t_{tok} by supplying h and the VM identifier.

c) Add the tuple $\langle t_{uri}, t_{tok} \rangle$ containing the URI of the Transfer proxy and the Transfer token to D.

3) If delegation is allowed according to site policy:
   a) Act as the Controller does in Step 2. Add returned possible destinations to D.
4) Limit D to include only Destinations that should be exposed as possible Destinations, according to site policies (e.g. "only local" or "only preferred partner sites").
5) Return D.

Name: Initiate transfer handler.
Runs at: Destination.
Input: $\langle s_{uri}, s_{tok}, d_{uri}, d_{tok} \rangle$.
Result: "success" or "failure" of the transfer.

1) Forward the tuple of $\langle s_{uri}, s_{tok}, d_{uri}, d_{tok} \rangle$ to the Transfer proxy at d_{uri}.
2) At the Transfer proxy:
   a) Connect to s_{uri}, and supply s_{tok}.
   b) Begin copying VM-related files over a secure channel (e.g. scp).
   c) Return either "success" or "failure", depending on the status of the transfer.
3) Forward the return value from the Transfer proxy.

3) At the Source: The Source will be called upon to prepare the VM for migration using the Forced migration call, and to verify the transfer afterward. These operations are carried out as follows.

Name: Forced migration handler.
Runs at: Source.
Input: VM identifier.
Result: $\langle t_{uri}, t_{tok} \rangle$.

1) Let T denote the set of Transfer proxies. Choose $t \in T$.
2) From t, obtain Transfer token t_{tok} by supplying h and the VM identifier.
3) Return the tuple $\langle t_{uri}, t_{tok} \rangle$ containing the URI of the Transfer proxy and the Transfer token.

Name: Transfer verification handler.
Runs at: Source.
Input: $\langle s_{uri}, s_{tok} \rangle$.
Result: "success" or "failure" of the transfer.

1) Connect to the Transfer proxy at s_{uri}, supplying s_{tok} as parameter and ask for file transfer status.
2) Forward the return value from the Transfer proxy.

D. Remarks

The algorithm does not dictate how the Controller or Destination should obtain a list of possible destinations. In some contexts, it may be most appropriate to have a central Directory Service of all sites known in the cloud and for the site looking for a partner site to opportunistically connect to them all. In others, sites may have preferred partner sites, due to existing contracts or similar. We do not specify which alternative should or must be used, since they are equally applicable.

Note that the direct transfer from Source Transfer proxy to Destination Transfer proxy (Step 2b in the Main migration algorithm) could also be relayed through the Transfer proxies of the Intermediaries to provide complete location unawareness. However, this is more costly in terms of network traffic, so the suggested approach is to allow such out-of-band transfers to occur. This trade-off is however not dictated by design, but rather by infrastructural policies that govern the use of the system.

All requests sent are signed with the private key of the sender. This makes it possible for the receivers to verify the origin of the requests. This applies not only to the endpoints (Controller and Source/Destination), but to all Intermediaries as well.

Note that the local site is included in the set of possible Destinations in Step 2 of the Main migration algorithm. Therefore, the algorithm needs no modification to be used between hosts within the local site, as eligible hosts will be found in Step 2 of the Migration request handler algorithm.

IV. Monitoring

In a cloud environment, monitoring is performed for two main reasons: (a) to ensure that VMs get the capacity stipulated in the SLAs; and (b) to collect data for system-wide accounting of the resources that have been in use on behalf of the service providers, which is required for billing. There are two complimentary types of monitoring that must be carried out by the system: (a) infrastructure measurements, including low-level measurements of the resources used by the VM, such as the amount of RAM or the network bandwidth; and (b) KPIs specific to the application, e.g. the amount of currently logged in users at a server. Both types of values may be of a sensitive nature, and they must be appropriately secured.

Grid computing forms the basis for the infrastructure of cloud computing, and thus, we shall briefly consider monitoring in Grid environments. The Global Grid Forum has defined a Grid Monitoring Architecture (GMA) [11]. In the architecture, a central Directory Service allows a Producer of monitoring events to register itself, so that Consumers can look it up and start subscribing to such events. Consumers can also register, so that Producers may perform a lookup regarding Consumers. All components can be replicated as needed, and the system is inherently scalable. Many implementations of Grid monitoring are based on GMA, notably including R-GMA [12], which is being used in large projects such as the European DataGrid project [13] and gLite [14].

There are two sides to monitoring that require consideration: *what* to monitor, and *how* to monitor it. Using the terminology of the GMA, *what* to monitor and at what frequency data is generated is determined by the configuration of the Producers, whereas *how* to monitor the data and the frequency of such measurements is determined by the configuration of the Consumers. In the general two-layer cloud architecture discussed previously, the Implementation layer is the Producer, and the Management layer is the Consumer. If more than one site is involved in the management of a VM, the Management layers of many sites may be regarded as Consumers.

Monitoring of resources is conceptually performed either continuously or at discrete times. Continuous measuring is a special case of discrete measuring, where the interval between measurements is very small and the data is delivered like a stream. The measured data can be delivered according to any of the following schemes: (a) as soon as possible; (b) at regular intervals; or (c) when a predicate evaluates to true (such as when the measured quantity falls below some threshold value). Similar to continuous vs. discrete measurements, scheme (a) may be regarded as a special case of scheme (b).

The data can be returned in *raw* or *processed* form. In raw form, all data measured during the interval is returned. In processed form, some mathematical function has been applied to the data set, e.g. maximum, minimum, mean, or median. Processing the information at the Implementation level at the Source lowers the amount of required network bandwidth, as less data needs to be sent back to the Management layer at the primary site.

All VMs require monitoring — at the very least for accounting purposes. Also, both the Implementation and Management layers require full information to configure the monitoring Producers and Consumers correctly. Thus, it is appropriate to specify the monitoring frequency, delivery scheme, and required format in the VM Descriptor. Should changes be required during the lifetime of the VM, they can be made through the Control interface, where other VM Descriptor changes are made possible.

We therefore suggest that the VM Descriptor is augmented with the following additional configuration parameters regarding monitoring:

- A set of monitoring specifiers, one or more for each type of data point that can be monitored, including the following information:
  - The monitoring interval length in milliseconds. A value of zero indicates that monitoring should be performed as frequently as possible.
  - The delivery interval length in milliseconds. A value of zero indicates that monitoring data should be sent as soon as it is available.
  - The delivery format, i.e. raw or processed. If the data is to be processed, it should also be specified what mathematical function to process the data with (min, max, mean, or median).
- The public key of the primary site, used to enable optional encryption of monitoring data (see Section IV-B).

A. Obtaining measurements

Measurements on the infrastructure or implementation level are straight-forward to perform, as most (if not all) virtualization technologies offer monitoring support. Thus, we simply note that obtaining such values is possible using existing hypervisor-specific APIs and rather focus on obtaining KPI measurements.

Ideally, application software should not have to be altered for cloud deployment. However, without such alterations, it is impossible to allow the running service applications to report KPI values to the cloud monitoring system. Our approach is to require only that the application performs standard file I/O operations on a particular file system partition that is added to the VM. Using File System in User Space (FUSE) [15], the application is unaware that the data being written to a file is actually passed to a program, running in user space. We let this program be a wrapper for a database, where the data is actually stored. The VM host can then register triggers within the database to fire when new data arrives (alternatively, if database triggers are not supported, poll the database at the required interval), and the data is guaranteed by the database to adhere to the ACID (Atomicity, Consistency, Isolation, Durability) requirements [16]. Thus, the only modifications to the system running in the VM required for this approach is: (a) that FUSE or similar is enabled; and (b) that the application writes KPI monitoring data to a regular file on the FUSE file system partition. Using this approach, the VM host at the Implementation layer is able to extract both types of monitoring measurements, and may publish the data to the consumers at the configured intervals.

A more detailed description of the FUSE-based database-backed file system is as follows. The file system partition is created with two directories, *encrypted* and *unencrypted*. Files stored in the encrypted directory are assumed to be encrypted in their entirety, and therefore cannot be parsed by the Implementation layer. On the other hand, files stored in the unencrypted directory may be parsed by the Implementation layer, which makes it possible to process the data. Whenever a file is stored in the file system, regardless of whether it is encrypted or not, an entry is made in the database back end where the table name equals the file name. In addition to the data written to the file, a timestamp is added that marks the time at which the data was saved in the file system. Unencrypted files should be stored as comma-separated value (CSV) files, and be either one or two lines long. For a two-line file, the first line is assumed to contain the column names that shall be used in the database. These names can also be referenced in the SLAs (e.g. `current_users`). Columns without labels are given a standard name (such as "column$X$", where $X \in \{1, 2, \ldots\}$). A single-line file is regarded as a two-line file where the first line is empty, and thus the columns are all named according to the naming scheme.

If the service application is not modifiable (closed source application), but has some other type of monitoring facility, e.g. a protocol for querying a server via the network, a custom program can be written that obtains such values and writes them to the file system can be deployed alongside the main application.

B. Security

Monitoring data is sensitive, as it can be used to disclose information about the running application. We therefore suggest that there should be support for securing the data before it is transmitted over a network, although using such measures should be optional. Infrastructure monitoring data can be secured by encrypting it with the primary site's public key (as

the ultimate destination for the monitoring data is the primary site). Application-specific KPI values may on the other hand be too sensitive to disclose even at the Implementation level, since the VM may run at a cloud infrastructure provider that the customer has no trust relationship with. In that case, we suggest that it is encrypted using the primary site's public key before it is written by the application itself.

V. Discussion

The proposed protocols and interfaces conform to the requirements gathered from the Usage Scenarios defined in Section II. Usage Scenarios 1 and 3 are supported directly by the migration algorithm as it is presented, whereas Usage Scenario 2 additionally requires the optional Step 0 as described in Section III-C. The operations are carried out in a cryptographically secure manner, and the VMs can be monitored using the monitoring proposal of Section IV.

The work presented in this article has been developed in accordance with the principle of location-unawareness, as defined in Section II. The principle adds some complexity, as it requires control messages to be sent through a chain of Intermediaries rather than directly between the two endpoint sites. We argue that this is acceptable overhead, as the gains made by adhering to the principle are greater: (a) the system is distributed to a higher degree, which benefits scalability as the reliance upon a single point of failure decreases; (b) there is a more clear separation of concern as the Intermediaries may act as Controllers for a VM should their site policies dictate that placement should be altered; (c) there is less information exchange between sites, and thus less information to keep current using concurrency control schemes; and (d) adherence to the principle guarantees a more general system that can adapt to new environments and use cases, rather than requiring that all placement decisions are made at a central site. The less general case, where a VM must be placed at a site of the primary site's choosing, is merely a restriction on the approaches and architecture presented in this article — the algorithms need only be modified to disallow delegation of responsibility to Intermediaries. Such restrictions are best to make at deploy time, rather than at design time, since the generality of a design increases its applicability.

Transfer proxies, as presented in this article, make a logical chain of Intermediaries between the primary site and the Source where a given VM is being deployed. Should a site become unavailable due to e.g. a network error, such chains may be broken. Let us first note that policies of a site may prohibit it from ever acting as an Intermediary, thus, for the site, circumventing the problem of broken trust chains altogether. It is also reasonable to assume that a site may be configured to only consider acting as an Intermediary for another site that is within the same administrative domain. Since every Intermediary adheres to the terms in the SLA regarding the VM, and thus are at risk of paying for possible SLA violations, reasonable sites will not delegate responsibility to sites that cannot be trusted. Thus, these chains of Intermediaries will be only as short as the trust relationship between sites permits. The risk of a broken chain of trust must be weighed against the benefits offered by the possibility to delegate responsibility over VMs within the federated cloud. We argue that a distributed system design should be open-ended and *enabling*, rather than *restricting*. This allows the design to remain relevant as it can accommodate for more use cases and be adapted to more situations.

Monitoring data must reach all Intermediaries involved with a given VM, to ensure the sites that the VM is still running correctly — if it is not, placement has to be re-evaluated to avoid SLA violation penalties. The data may either be passed repeatedly through each Intermediary site from the Source to the primary site, or it may be placed in a common Enterprise Service Bus (ESB). Neither of these break the principle of location unawareness.

The overall design goals of any system are scalability, efficiency, and security. Let us now evaluate the suggested design from these perspectives. From a combined scalability and efficiency point of view, the suggested migration and monitoring interfaces are inherently scalable. If monitoring data is passed along path of Intermediaries, each site acts as both a Consumer and as a Producer of monitoring data. R-GMA has been developed with this type of data forwarding in mind [12], as it increases the scalability and efficiency of the system. ESB-style solutions are also inherently scalable and suitable for this type of application. The migration architecture, including the interface, is also very efficient and scalable, as it gives a high degree of autonomy to each site while at the same time requires very little network traffic overhead to perform migration. A high level of autonomy is important for scalability, as sites are more self-contained and thus the amount of information that has to be exchanged is reduced. Security is built in to both the migration and the monitoring interfaces and architectures, and the use of asymmetric encryption offers the confidentiality and integrity required.

Future work includes evaluation of the proposed solutions. Since the amount of control messages exchanged by the sites in the Transfer Proxy-supported migration is low, and the sizes of such messages is much lower than the transfer of the image files (which will be measured in gigabytes, rather than kilobytes for the messages), the added overhead in terms of network traffic must be assumed to be very low. To evaluate the monitoring proposal, a prototype is being developed as a proof of concept and its performance and usability will be tested.

VI. Related Work

Several standardization projects are in the early stages of developing interoperable cloud interfaces, such as the OCCI [17] working group at the Open Grid Forum. However, the topics KPI-aware monitoring and federation of clouds are deemed out of scope for the project. The authors of this work will be involved with developing extensions as contributions to OCCI to address these matters.

OpenNebula [18] is a open-source virtualization management software. It leverages various existing virtualization technologies and enables system administrators to administer a cluster

of hosts. It allows resources to be added dynamically, and advanced management functions such as workload distribution control and cluster partitioning. Migration within a single site is already supported, as is deploying VMs to Amazon EC2. Currently, the OpenNebula project is developed to support cross-site management functionality and cross-site migration of VMs.

Related Grid computing interfaces include WS-GRAM [19] and OGSA-BES [20]. The former is related to submission of jobs to a Grid, whereas the latter defines a state model, an informational model, and Web Service port types for management of Grid jobs. It also includes a proposed distributed monitoring of the resources where the jobs are running. The OGSA-BES allows for migration of Grid jobs, but does not specify how such migration should be implemented.

Resource scheduling and Grid brokering may be viewed as a theoretical basis for how to perform VM placement in a cloud computing environment. Relevant work in this field includes [21], [22]. The aforementioned research can be leveraged in Destination selection process in the Transfer Proxy-supported migration.

VM migration has been studied extensively in works such as [23], [24]. However, these works focus on the technical aspect of performing migration, rather than defining the interfaces for initiating and managing the migration process itself.

VII. Summary and Conclusions

We have presented two novel interface and architectural contributions, facilitating for cloud computing software to make use of inter- and intra-site VM migration and improved inter- and intra-site monitoring of VM resources, both on an infrastructural and on an application-specific level. Existing monitoring architectures may be leveraged, as the proposed monitoring solution is compatible with the Grid Monitoring Architecture [11], although it is proposed that a more highly distributed solution is used instead. The additions presented in the article adhere to a principle of location-unawareness, which increases scalability, decreases the degree of coupling between sites in the federated cloud environment, and makes a clear separation of concern between sites. The proposed additions expose a high level of generality, and are thus adaptable and usable in many scenarios, without being impractical to implement or standardize.

Acknowledgment

The authors are grateful to Daniel Henriksson and Johan Tordsson for their contributions to the foundation upon which the work was based. This work has been partly funded by the EU-IST-FP7-215605 (RESERVOIR) project.

References

[1] R. Buyya, C. Yeo, S. Venugopal, M. Ltd, and A. Melbourne, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08, IEEE CS Press, Los Alamitos, CA, USA)*, 2008.

[2] National Institute of Standards and Technology, Systems and Network Security Group, "Draft NIST Working Definition of Cloud Computing," 2009. [Online]. Available: http://csrc.ncsl.nist.gov/groups/SNS/cloud-computing/cloud-def-v12.doc

[3] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, L. 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 Systems Journal*, 2009, to appear.

[4] Distributed Management Task Force, Inc., "Open Virtualization Format Specification," DMTF 0243 (Standard), Feb. 2009. [Online]. Available: http://www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf

[5] Xen community, "Xen Hypervisor Web page," 2003. [Online]. Available: http://www.xen.org/

[6] VMware Inc., "VMware Virtualization Technology Web page," Visisted March 30, 2009, 1999. [Online]. Available: http://www.vmware.com/

[7] Distributed Management Task Force, Inc., "System Virtualization Profile," DMTF 1042 (Preliminary Standard), Aug. 2007. [Online]. Available: http://www.dmtf.org/standards/published_documents/DSP1042.pdf

[8] ——, "System Virtualization Profile," DMTF 1057 (Preliminary Standard), May 2007. [Online]. Available: http://www.dmtf.org/standards/published_documents/DSP1057.pdf

[9] ——, "CIM System Virtualization White Paper," DMTF 2013 (Informational), Nov. 2007. [Online]. Available: http://www.dmtf.org/standards/published_documents/DSP2013_1.0.0.pdf

[10] J. Postel and J. Reynolds, "File Transfer Protocol," RFC 959 (Standard), Oct. 1985, updated by RFCs 2228, 2640, 2773, 3659. [Online]. Available: http://www.ietf.org/rfc/rfc959.txt

[11] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, and M. Swany, "A Grid Monitoring Architecture," GWD-I (Informational), Aug. 2002. [Online]. Available: http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-3.pdf

[12] A. C. et al., "The Relational Grid Monitoring Architecture: Mediating Information about the Grid," *Journal of Grid Computing*, vol. 2, pp. 323–339, 2004.

[13] B. Segal, L. Robertson, F. Gagliardi, and F. Carminati, "Grid computing: The European data grid project," Lyon, pp. 15–20, 2000.

[14] E. Laure, S. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White *et al.*, "Programming the Grid with gLite," pp. 33–45, 2006.

[15] M. Szeredi, "Filesystem in userspace," 2004. [Online]. Available: http://fuse.sourceforge.net/

[16] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Computing Surveys*, vol. 15, no. 4, pp. 287–317, 1983.

[17] Open Grid Forum OCCI-WG, "Open Cloud Computing Interface," 2009. [Online]. Available: http://forge.ogf.org/sf/go/projects.occi-wg

[18] B. Sotomayor, R. Montero, I. Llorente, and I. Foster, "Capacity Leasing in Cloud Systems using the OpenNebula Engine," *Cloud Computing and Applications*, vol. 2008, 2008. [Online]. Available: http://www.cca08.org/papers/Paper20-Sotomayor.pdf

[19] Globus Project by the University of Chicago, "GRAM4," 2008. [Online]. Available: http://www.globus.org/toolkit/docs/4.2/4.2.1/execution/gram4/

[20] Open Grid Forum, "OGSA Basic Execution Services WG," 2005. [Online]. Available: https://forge.gridforum.org/sf/projects/ogsa-bes-wg

[21] E. Elmroth and J. Tordsson, "Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions," *Future Generation Computer Systems. The International Journal of Grid Computing: Theory, Methods and Applications*, vol. 24, no. 6, pp. 585–593, 2008.

[22] ——, "An interoperable, standards-based Grid resource broker and job submission service," in *First International Conference on e-Science and Grid Computing*, H. Stockinger *et al.*, Eds. IEEE CS Press, 2005, pp. 212–220.

[23] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, "Xen and the Art of Virtualization," in *Proceedings of the ACM Symposium on Operating Systems Principles*, 2003, pp. 164–177.

[24] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2 table of contents*. USENIX Association Berkeley, CA, USA, 2005, pp. 273–286.

II

Paper II

Scheduling and Monitoring of Internally Structured Services in Cloud Federations*

Lars Larsson, Daniel Henriksson, and Erik Elmroth

Dept. Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden {larsson, danielh, elmroth}@cs.umu.se
http://www.cloudresearch.org/

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.

Key words: cloud computing, directed graphs, formal specification, scheduling, virtual machines

* By permission of IEEE

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.

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

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.

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

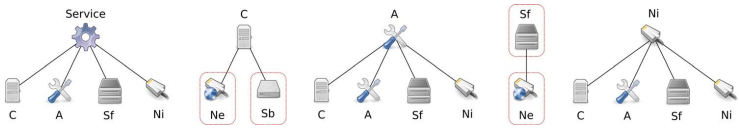


Figure 1. Rules defining valid inter-node relationships for service definition DAGs. C denotes Compute resources, A denotes *AA*-constraints, S_f and S_b denote file and block storage, respectively, and N_i and N_e denote internal and external networks. Valid terminal nodes are marked with a border.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

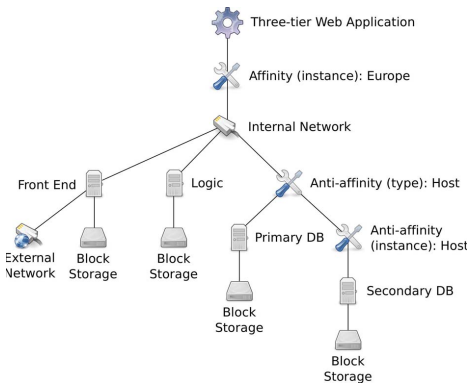


Figure 2. Example of a three-tier Web application service represented using a DAG which includes *AA*-constraints and network setup. Node types are shown in Figure 1, and labels have been added for clarity.

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.

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

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

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

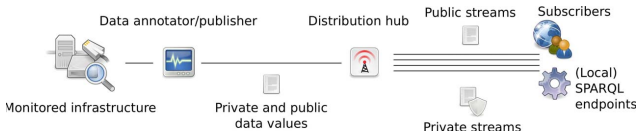


Figure 3. Overview of the MEDICI monitoring data distribution architecture.

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

978-1-4577-0681-3/11/$26.00 ©2011 IEEE

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

III

Paper III

Modeling and Placement of Cloud Services with Internal Structure*

Daniel Espling, Lars Larsson, Wubin Li
Johan Tordsson, and Erik Elmroth

Dept. Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden {espling, larsson, viali, tordsson, elmroth}@cs.umu.se
http://www.cloudresearch.org/

Abstract: Virtual machine placement is the process of mapping virtual machines to available physical hosts within a datacenter or on a remote datacenter in a cloud federation. Normally, service owners cannot influence the placement of service components beyond choosing datacenter provider and deployment zone at that provider. For some services, however, this lack of influence is a hindrance to cloud adoption. For example, services that require specific geographical deployment (due e.g. to legislation), or require redundancy by avoiding co-location placement of critical components. We present an approach for service owners to influence placement of their service components by explicitly specifying service structure, component relationships, and placement constraints between components. We show how the structure and constraints can be expressed and subsequently formulated as constraints that can be used in placement of virtual machines in the cloud. We use an integer linear programming scheduling approach to illustrate the approach, show the corresponding mathematical formulation of the model, and evaluate it using a large set of simulated input. Our experimental evaluation confirms the feasibility of the model and shows how varying amounts of placement constraints and data center background load affects the possibility for a solver to find a solution satisfying all constraints within a certain time-frame. Our experiments indicate that the number of constraints affects the ability of finding a solution to a higher degree than background load, and that for a high number of hosts with low capacity, component affinity is the dominating factor affecting the possibility to find a solution.

Key words: service management, service structure, placement, affinity, collocation

* By permission of IEEE

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2362120, IEEE Transactions on Cloud Computing

Modeling and Placement of Cloud Services with Internal Structure

Daniel Espling, Lars Larsson, Wubin Li, Johan Tordsson, and Erik Elmroth
Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden

***Abstract*—Virtual machine placement is the process of mapping virtual machines to available physical hosts within a datacenter or on a remote datacenter in a cloud federation. Normally, service owners cannot influence the placement of service components beyond choosing datacenter provider and deployment zone at that provider. For some services, however, this lack of influence is a hindrance to cloud adoption. For example, services that require specific geographical deployment (due e.g. to legislation), or require redundancy by avoiding co-location placement of critical components. We present an approach for service owners to influence placement of their service components by explicitly specifying service structure, component relationships, and placement constraints between components. We show how the structure and constraints can be expressed and subsequently formulated as constraints that can be used in placement of virtual machines in the cloud. We use an integer linear programming scheduling approach to illustrate the approach, show the corresponding mathematical formulation of the model, and evaluate it using a large set of simulated input. Our experimental evaluation confirms the feasibility of the model and shows how varying amounts of placement constraints and data center background load affects the possibility for a solver to find a solution satisfying all constraints within a certain timeframe. Our experiments indicate that the number of constraints affects the ability of finding a solution to a higher degree than background load, and that for a high number of hosts with low capacity, component affinity is the dominating factor affecting the possibility to find a solution.**

***Index Terms*—service management, service structure, placement, affinity, collocation**

I. Introduction

In cloud computing, infrastructure providers offer rapidly provisioned hosting of *services* (applications). Software providers provide and own the services and are the consumers of the infrastructure providers' resources. A service may be comprised of several components, each of a specific type. This can be, for example, a database server, a front-end, and a logic tier in a typical three-tier Web application.

This paper addresses the issue of application owners not being able to impact where service components are being placed for execution. Although it is often beneficial not having to deal with such resource allocation issues, there are situations where it would be beneficial to guide the infrastructure provider's decision making in this respect. This may be that the application owners want some data to reside within a certain country for legislative reasons, that a secondary database server deployed for redundancy purposes should run on a different host or even a different datacenter than the primary database, or that software components with excessive inter-communication should be placed on the same datacenter or even on the same physical host.

Before continuing our exposition, we clarify some nomenclature used throughout the paper. A *type* corresponds loosely to *launch configurations* used in Amazon EC2 and *server templates* used by RightScale. Each *instance* of a type shares a type-specific base virtual machine (VM) image containing the startup state (operating system and installed applications) and configuration. The total amount of capacity of a service can be adjusted by changing the number of running instances of each type. In this paper, we use the term VM to denote VM instance, and explicitly state when we refer to a VM type.

An infrastructure provider may collaborate with other remote providers on workload sharing and resource subcontracting to easier cope with spikes in resource consumption or other unexpected events that affects hosting of services. Currently, such collaborations are most natural among datacenters all belonging to the same large-scale global company. In the future, we also expect this to be an important characteristic of a global cloud resource market, e.g., with small providers with strong local presence partnering with large global providers for meeting pure capacity demands, e.g., processing data subject to legal constraints locally and bulk computations, requiring more compute power than is available locally, on large-scale international providers. Cloud collaborations may also become central to providing resources closer to end users or to take advantage of cheap capacity under dynamic pricing arrangements, e.g., through load sharing during nights in different time zones to take advantage of lower electricity prices. There are several different collaboration models [1], [2] and different levels of collaboration between different sites [3]. Each collaboration scenario has its own set of challenges, but in all cases the general problem of performing placement (mapping resources to VMs) locally is extended to also include resources offered by collaborating sites.

In a collaborative cloud setting, the service owner cannot normally affect on which site in the collaboration the different instances comprising a service will be hosted. Instead, the responsibility for placing the service components is delegated to the infrastructure provider, and in some cases the infrastructure provider may outsource components to a partner provider [1], [2]. Many services can function well despite this lack or influence, but the lack of control may have a negative effect on services that need to be hosted in a specific fashion. For example, some services are not allowed to be hosted in or

Manuscript submitted January 27, 2014.
Corresponding author: D. Espling (espling@cs.umu.se).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2362120, IEEE Transactions on Cloud Computing

outside specific regions either for legislative reasons [4] or to ensure that they are located close to end users. Furthermore, fault-tolerance can be greatly improved by enforcing that replicas of the same service component are not deployed on the same physical hardware. Conversely, host-level co-location of certain components may be essential to achieve low-latency. These scenarios are the main motivations behind our work.

In our previous contribution [5], we presented early work on representing the structure of services explicitly, making it possible for placement algorithms and procedures to take the structure and internal placement constraints (such as explicit co-hosting) into consideration when performing service placement. In this paper we extend on our previous work by (I) showing how the hierarchical graph structure can be converted into formalized placement constraints; (II) presenting a mathematical model for placement optimization with constraints that can be used to extend existing placement procedures with support for detailed and service owner-controlled placement directives; and (III) demonstrating the feasibility of this model and its performance through a set of experiments. Notably, development of new core placement algorithms or deployment systems are outside the scope of this paper. Instead, our ambition is to provide the support to extend current such algorithms and systems with the ability to take service structure constraints into account as a natural part of their normal operation. Consequently, practical experiments are performed to illustrate the practical usage of such constraints and to highlight some performance aspects rather than to fully re-evaluate the performance and capabilities of previously published solvers.

The remainder of the paper is organized as follows. Section II presents background information and related work. Section III elaborates on placement constraints and describes formalized syntactical and semantic representations used in our model. In Section IV, we present a model for placement of VMs that takes placement constraints into account for local and remote placement of VMs. The results of experiments using structured services are shown in Section V before the paper is concluded with comments and a proposal for future work in Section VI.

II. Background and Related work

This section has been divided into two subsections: background and related work material concerning service placement, and the same concerning inter-component affinities. This division is due to the large body of research that has been performed in service placement, but without concern for inter-component affinities, and the relatively small body of research that focuses mostly on the latter. Our work is positioned in the middle of these two fields, as it leverages service placement research and extends it to include not merely affinity but a more holistic view on service structuring.

Notably, our aim in this paper is not to design another placement algorithm to tackle scalability issues in service placement, but rather to define a mechanism that makes it possible to model dependencies between components of a service, including hierarchies, affinities, and anti-affinities. Our work should be seen as a complement to existing work on cloud placement algorithms, as presented by e.g., Lampe et al. [6] and Genez et al. [7]. We also remark that in our work, we use the Integer Linear Programming (ILP) approach to illustrate how our service structure mechanisms can be used as a set of constraints in cloud VM placement (as presented in the following sections). However, our mechanism is not limited to ILP formulations of the placement problem. Moreover, service reallocation scenarios are not in the scope. A multi-cloud scenario close to this topic is studied in our previous work [8], which presents a linear integer programming model for dynamic cloud VM placement with expressibility of service reallocation overhead.

A. Service Placement

The problem of optimizing placement of virtual machines in cloud environments has lately attracted research both from academia and industry [9], [10], [11], [12], [13]. However, a potential problem from the perspective of service providers that so far has received little attention is the loss of control over *how* their services are deployed.

The need for the SP to impact how the service is deployed is actualized by federations and other multi-site cloud deployments as demonstrated by the RESERVOIR [1] and OPTIMIS [2] projects. Data and computation provisioning in multi-site clouds raises concerns regarding locality, both from a performance, fault-tolerance, and a legislative point of view [14], [15]. Currently, public clouds at best offer coarse-grained mechanisms of specifying where application components should be placed (e.g., choosing in which continent components should be deployed [16]), but this functionality does not extend to a finer level of detail and control, and is furthermore not enforceable if clouds are part of a collaboration, e.g., includes multiple datacenters but no means to specify which of them to use for the deployment.

Split Service Deployment: Emerging technology in cloud service placement supports automatically splitting a service into several smaller sub-services, in order to spread the service across different infrastructures. Although not yet reflected in the literature, OPTIMIS [2] is one of the projects with early results on splitting of services. Our ongoing work in this context includes permutation-pack based optimization for service deployment in multi-cloud environments [17].

We foresee that split service deployment could benefit greatly from the service structure presented and discussed within this paper, as the inherit graph structure can be used as a good starting point for educated decomposition of a service manifest (description) into smaller parts, while still retaining critical relations between the different components making up the service.

Mathematically, the service placement problem in cloud environments can be formulated as a variant of the class constrained multiple-knapsack problem that is known to be NP hard. Approximation algorithms are proposed to tackle the scalability issue and, e.g., Breitgand et al. [18] propose an integer linear program formulation for policy-driven service placement optimization in federated clouds, and a 2-approximation algorithm based on a rounding of a linear relaxation of the problem. Li et al. [17] have also suggested

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2362120, IEEE Transactions on Cloud Computing

a general approach to automatic service deployment in cloud environments. An in-depth analysis of scalability of ILP solvers is out of scope for this paper, but has been studied extensively in the operations research community, e.g., by Atamtürk et. al. [19], and Koch et. al. [20].

B. Inter-component Affinities

Brandic et al. proposed the concept of *affinity* (forced co-placement of components) in [14]. Their work focused on expressing inter-component affinity relations between grid jobs in grid workflows, and the work presented in this paper uses similar inter-component relationships for cloud service components. In the management software provided by the RESERVOIR project, host-level anti-affinity is supported [1]. Breitgand et al. [18] present a model and placement algorithm framework with support for both anti-affinity and cross-federation capabilities. They model the scheduling problem using integer linear program formulations for placement strategies, and focus on presenting a complete objective function to be optimized.

In our previous work [5] we presented a model that allows service providers to specify the structure and deployment directives for a service using a directed acyclic graph structure with nodes representing either service components or placement constraints. This structure allows modeling of the cloud services inherent structure, making it possible to preserve conditions and relations throughout the lifecycle of the service.

Since the publication of our previous paper on this topic [5], Jayasinghe et al. have published an alternative approach [21] to solve a similar problem. Their work aims to solve three related problems: (I) communication-aware VM clustering, (II) mapping of VM groups to server racks, and (III) VM to physical host machine mapping. Our work focuses on Problem III, since we do not expressly take into account the communication delays, but rather assume that a service with tight communication delay bounds will use placement constraints to ensure suitable hosting (VM group to server rack mapping is in the cited work used to ensure this co-location). The work published in this paper presents an approach to extract and represent placement constraints in a mathematical model solvable using integer-linear programming.

In theory, either the work earlier published by us [5] or by Jayasinghe [21] can be modeled as an ILP. However, in the latter, the solution to the problem is found using an explicit divide-and-conquer methodology while we investigate an integer-linear programming approach that hopefully presents a stronger starting point for future work including costs for replacement. Our previous work [5] touched upon this subject, but did not investigate it in detail. More recent work by Hermenier et al. [22] formulates the rescheduling problem (finding a new VM to physical host mapping that takes a prior mapping into account) and present a constraints programming-based solution and implementation based on the Entropy [23] autonomous VM manager.

Alicherry and Lakshman [24], [25] have studied optimizing the placement of VMs to minimize data access latencies between pairs of processing VMs and data nodes, and between sets of VMs, respectively. In the former case [24], they present algorithms based on linear assignment algorithms for deploying processing VMs close to data nodes with the objective to minimize the total access time. In the later case [25], a new algorithm based on 2-approximation is used with the goal to minimize the traffic between VMs in a set (and hence also between datacentres). Compared to our work, Alicherry and Lakshman provide interesting solutions to find the optimal solutions for these problems (subject to heuristics), whereas our work strives for a more generally applicable approach considering many kinds of constraints and scenarios, but where the fully optimal solution might be harder to find.

In this work we extend upon previous work by supporting several levels of constraints (both for affinity and anti-affinity), by showing how they may be specified prior to deployment using service structure graphs, and by showing how they can be extracted and modeled as constraints in a placement optimization algorithm. The constraints model developed in this work and the comprehensive utility function from Breitgand et. al. [18] are complementary. We also provide an evaluation of the constraints model using simulations.

C. Structured Services

As this paper extends on the work on our previous contribution [5], this section only briefly presents concepts from that work that forms the foundation for the work presented in the upcoming sections of this paper. We also revisit the example given in that work and use it in the upcoming sections as input to our placement optimization model that takes placement constraints into account.

Where both affinity and anti-affinity are applicable we use the term *AA-constraints*, and each term alone if something applies only to either affinity or anti-affinity. *AA*-constraints, as illustrated in Figure 1, are used to express either the affinity or anti-affinity between two types, or between a type and a specific placement, and allows the SP to instruct the IP how (but not exactly where) each part of the service should be placed. We consider three levels of *AA*-constraints, namely *host*, (cloud) *site*, and geographical *region* due to clear real-world semantics and implied relationships between these levels and due to prior work in this area ([1], [14]). Hosts belong to a site and sites reside in a region, thus, there is a clear hierarchical relation between these levels. These levels are specifications of a more general grouping mechanism for virtual machines: by extending this work, arbitrary groupings can be supported.

As outlined in the previous work, for an affinity level l, if VM types A and B are in the relation, all instances of these types must be placed so that placement restrictions are adhered to. Affinity is used to express that several service components must be co-placed at a given level. Conversely, anti-affinity requires that VM instances may *not* be placed on the same level. 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 site, and may not be placed in a certain region.

Service Example: An example of a service represented using this model is presented in Figure 1. In this three-tier

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2362120, IEEE Transactions on Cloud Computing

Web application, immediately below the service root node an affinity constraint states that all descendants of all resource types must be located within the EU. 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. An 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 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.

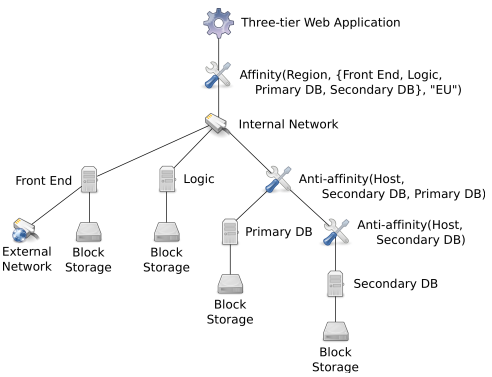


Figure 1. A three-tier Web application service [5]. The uppermost affinity constraint is expressed in a more compact set notation to improve readability (cf. Section III).

III. Placement Constraints

Extending the previous work, we present a more formal definition of *AA*-constraints. They are specified using rules of the following form:

$$\text{Affinity}(L, A, B) \quad (1)$$
$$\text{Affinity}(L, A) \quad (2)$$
$$\text{Affinity}(L, A, l) \quad (3)$$
$$\text{AntiAffinity}(L, A, B) \quad (4)$$
$$\text{AntiAffinity}(L, A) \quad (5)$$
$$\text{AntiAffinity}(L, A, l) \quad (6)$$

Where $L \in \{\text{Region}, \text{Site}, \text{Host}\}$, A and B are types of VMs, and l is a specific region, site, or host (as appropriate, considering the value of L). The semantics are as follows. Equation (1) states that for the level L, an instance of type A must be placed at the same location as at least one instance of type B. Note that there is no such relation from B to A unless explicitly stated, i.e., specifying that instances of type B need (or need not) be placed at the same location as an instance of type A. Equation (2) states that all instances of type A must be co-placed at the given level L. Note that there is a semantic difference between applying Equation (1) to the same type (i.e., $\text{Affinity}(L, A, A)$), compared to using Equation (2). The former would enforce a pair-wise deployment of VM instances of type A, while the latter would cluster all available VM instances of type A. Equation (3) states that all instances of type A must be placed at the named location l. Similar interpretations hold for anti-affinity (expressed in Equation (4) – (6)), with the difference that they prevent placement rather than require it.

We conclude this section with an example. The following four rules specify that VMs of type A cannot be placed in Sweden, an instance of type A must be placed at the same site as some instance of type B, that all instances of B must be placed at the same site, and that instances of type A must be placed at distinct hosts.

1) $\text{AntiAffinity}(\text{Region}, A, \text{Sweden})$
2) $\text{Affinity}(\text{Site}, A, B)$
3) $\text{Affinity}(\text{Site}, B)$
4) $\text{AntiAffinity}(\text{Host}, A)$

Thus, a placement optimization engine has to further infer that: all instances of type A must be placed at the same site as all instances of type B (Rule 2 and 3) and; the (single) site on which all instances of type A and B are placed may not be located in Sweden (Rule 1). Rule 4 does not allow the placement engine to infer any new information, but does specify rules that must be taken into consideration when placement is performed.

In the upcoming section, we show how these *AA*-constraints can be expressed prior to deployment using a simple to understand graph structure.

IV. Structure-Aware Service Placement

Using the structure of a service it is possible to formulate and subsequently enforce constraints and conditions to be considered when placing service components across collaborating infrastructures. This is effectively a two step process where the first step is to extract information from the service structure and convert this into a suitable format, and the second step is to utilize the structured data when performing service placement.

A. Structure Representation

Service structure conceptually constitutes a directed acyclic graph of nodes, representing both types and constraints. Current popular choices for representing cloud service definitions are based on either XML or JSON formats, both of which are hierarchical (tree-based, rather than graph-based) in nature. This slight mismatch can easily be overcome, however, using element identifiers and identifier references. An extension to, e.g., the XML-based Open Virtualization Format [26] can be constructed in the following way:

- Introduce a Structure element, which is the parent that holds all structure-related information.
- As a child of Structure, introduce a Types element, which in turn lists a set of Type elements that contain unique element identifiers and human-readable names (such as “Primary Database”).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2362120, IEEE Transactions on Cloud Computing

Table I
HOST-LEVEL VM TYPE CONSTRAINTS EXTRACTED FROM FIGURE 1.

	FE	LO	PDB	SDB
FE	0	0	0	0
LO	0	0	0	0
PDB	0	0	0	0
SDB	0	0	-1	-1

Table II
EXTRACTED REGION-LEVEL AFFINITY RELATIONS FROM FIGURE 1.

	US-E	US-W	EU	Asia-S	Asia-T
FE	0	0	1	0	0
LO	0	0	1	0	0
PDB	0	0	1	0	0
SDB	0	0	1	0	0

- As a child of Structure, introduce a Constraints element, which in turn lists a set of elements that are of subtypes of a Constraint element, representing the various constraint types that are listed in Section III, e.g. AntiAffinity-Constraint. Such Constraint elements all have mandatory attributes stating their direction (from/to) between types using references to their corresponding element identifiers.

It is evident that such a representation can easily be both generated and parsed and that the resulting data structure can easily be converted into something equivalent. We do not present a full representation XML Schema here for space reasons, rather just note that the step between Figure 1 and the matrices that follow is not as long as it may seem upon a first glance.

B. Placement Constraint Extraction

Placement constraints between different VM types and those between VM types and specific named locations can be extracted from the service structure graph. Table I shows a representation of host-level *AA*-constraints for the types of VMs in the example of Figure 1. The table illustrates the relations between four different VM types: Front End (FE), Logic (LO), Primary DB (PDB), and Secondary DB (SDB). The relations shown are extracted from the service structure and the values in the matrix show in Table I are either 1 for affinity, -1 for anti-affinity or 0 to denote that no specific constraints are present. This notation lends itself well to ILP-based solutions (illustrated later), and is a more compact alternative to having two separate binary matrices (one for affinity and one for anti-affinity). This approach can also be extended in the future to support soft constraints with higher or lower values to indicate preference or to be more easily integrated with other approaches not based on ILP.

The second set of relations are those between VM types and named elements of the three levels of *AA*-constraints, e.g. to a specific region. These constraints are represented using the same values and semantics as before. Table II shows the region-level affinity relations extracted from the example in Figure 1. It shows that the service has an affinity to the EU, and therefore may not be placed in any of the other regions.

Another example service is shown in Figure 2. This illustrates how a cloud based ”Split & Merge” video encoding service

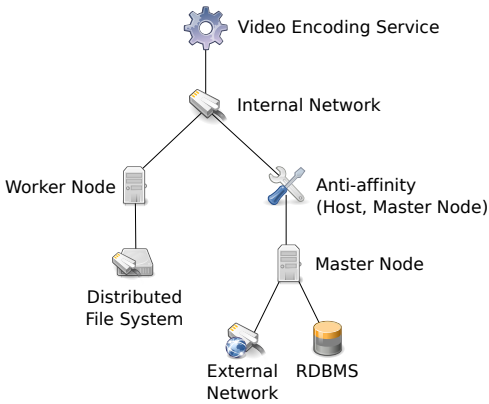


Figure 2. Represesentation of a video encoding service by Pereira et al. [27].

Table III
EXTRACTED HOST-LEVEL AFFINITY RELATIONS FROM FIGURE 2.

	Worker Node	Master Node
Worker Node	0	0
Master Node	0	-1

presented by Pereira et al. [27] would be modelled using our approach. The service comprises one or more Master nodes that receive encoding requests from the external network and performs the encoding on a set of worker nodes communicated with over an internal network. To avoid a single point of failure of Master nodes, Pereira et al. employs a failover approach with the critical state stored in a relational database. This has been expressed in our model with the precense of the Relational Database component, and with an Anti-affinity constraint to ensure that multiple instances of the Master node are not placed on the same physical host. The resulting affinity matrix is illustrated in Table III.

C. Constraint Model

Placement constraints extracted from the service structure can be enforced by a placement engine with ability to handle various constraints [28], [12], making it structure-aware. In this section, we present as an example a typical binary integer programming formulation of the placement problem that takes placement constraints into consideration. Without loss of generality, we have chosen to provide a simple model and straight-forward objective function to more clearly focus on the important aspects of the formulation. Most notably, we represent capacity by a single one-dimensional value, rather than a multidimensional one (e.g. separate storage, network, CPU, memory requirements).

The model does not permit omission of any VM in the set of VMs that are to be placed. In effect, it is not allowed to avoid placing any parts of the service. Such decisions are on a higher administrative level, and we focus on finding a placement plan that places all VMs.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2362120, IEEE Transactions on Cloud Computing

Given: V = set of VMs
T = set of VM types
H = set of hosts
I_h = index of h $\in H$
v_t = type of $v, v \in V, v_t \in T$
c_h = cost per capacity unit at $h \in H$
cap_h = capacity of host $h \in H$
req_v = requirements of $v \in V$
$\text{type}_{v,t} = 1$ if $v_t = t, 0$ otherwise
$\text{vcons}_{t,u}$ = host-level constraints between t and u where $t, u \in V$
$\text{hcons}_{t,h}$ = host-level constraints between t and h where $t \in T, h \in H$
Variable: $m_{v,h}$ = mapping of v to $h, v \in V, h \in H$
1 if v is mapped to $h, 0$ otherwise

Minimize:
$$\sum_{v \in V, h \in H} (m_{v,h} * \text{req}_v * c_h) \quad (7)$$

Subject to:

$$\forall v \in V : \sum_{h \in H} m_{v,h} = 1 \quad (8)$$

$$\forall h \in H : \sum_{v \in V} m_{v,h} * \text{req}_v \leq \text{cap}_h \quad (9)$$

$$\forall v, w \in V, \forall t, u \in T, \forall h \in H : \text{vcons}_{t,u} = -1 \wedge v_t = t \wedge w_t = u \implies m_{v,h} + m_{w,h} \leq 1 \quad (10)$$

$$\forall v \in V, \forall t, u \in T, t \neq u : \text{vcons}_{t,u} = 1 \wedge v_t = t \implies \exists w \in V, w_t = u, w \neq v : \forall h \in H : m_{v,h} = m_{w,h} \quad (11)$$

$$\forall v, w \in V, \forall t \in T, \forall h \in H : \text{vcons}_{t,t} = 1 \wedge v_t = t \wedge w_t = t \implies m_{v,h} = m_{w,h} \quad (12)$$

$$\forall v \in V, \forall t \in T, \forall h \in H : \text{hcons}_{t,h} = -1 \wedge v_t = t \implies m_{v,h} = 0 \quad (13)$$

$$\forall v \in V, \forall t \in T, \forall h \in H : \text{hcons}_{t,h} = 1 \wedge v_t = t \implies m_{v,h} = 1 \quad (14)$$

Equation (7) is the objective function, minimizing the total cost of hosting a set of VMs by placing them across a set of hosts with different associated costs. In multi-site clouds, the inherent differences in hosting cost may have a great impact on the final placement solution. In single clouds, the cost function may instead focus on consolidation to lower power consumption. In this work we focus on a simple objective function representing the multi-site case, as this is the scenario where placement constraints have the largest impact. Future work involves developing and modeling a more complex objective function, also incorporating local VM consolidation.

Constraints derived from the service structure are modeled in equations (8 - 14) and can be interpreted as follows:

(8) each VM has to be placed at exactly one host;
(9) the total required capacity for all VMs placed at a host may not exceed the total capacity of the host;
(10) if there is a host-level anti-affinity constraint from one VM type to another, there may not exist a mapping such that a VM instance of the first type and a VM instance of the second type are placed on the same host;
(11) if a VM instance is of a certain type, and there is a host-level affinity constraint from that type to another, there must exist a mapping such that the VM is placed on a host along side a VM instance of the other type;
(12) if there is a host-level affinity relation within a VM type (c.f. Equation (2)) and two VM instances both are of that type, then both instances must be placed at the same host;
(13) if there is a host-level anti-affinity constraint from a VM type to a host, instances of that VM type may not be placed at that host; and
(14) if there is a host-level affinity constraint from a VM type to a host, instances of that VM type must be placed at that host.

For clarity we have presented only the constraints and equations for host-level constraints. The constraints for the other levels are very similar: expressions similar to (Equations (10) – (14)) have to be added to account for sites and regions to support these levels of *AA*-constraints. We have chosen to omit these here because the extracted data and mathematical model for host-level are sufficiently similar to the other levels so presenting only host-level constraints makes the model easier to read. Furthermore, we again note that the type of relationship here (host-level placement constraints) is merely a specification of a more general arbitrary grouping of VMs for placement constraints.

Also note that anti-affinity is expressed using one rule (Equation (10)) while affinity is expressed using two (Equations (11) and (12)). This is because an anti-affinity rule that prevents co-placement with *at least one* other instance implies a rule preventing co-placement with *all* other instances. Concerning affinity, these cases are not equivalent and hence two separate rules are required.

As previously discussed, one of the specific challenges related to cloud collaborations is that a local site has no control over (and rarely has access to information about) host-level specifics of remote sites: how many hosts are available, what their performance capabilities are, etc. Because the ultimate goal of a placement engine is to map VMs to specific hosts,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2362120, IEEE Transactions on Cloud Computing

creating a placement mapping that works without access to this information can be done in two ways:

1) Model the remote site as single (local) host with sufficient capacity to host all $v \in V$. *AA*-constraints between VM types hosted at this host are assumed to be enforced by the remote site and can therefore be disregarded.
2) Model the remote site as a set of (local) hosts with sufficient capacity and create a valid mapping assuming these hosts exist at the remote site.

The difference between these two alternatives is twofold: where the placement mapping is constructed (either at both sites or just the first), and that the second option does not make placement at remote sites a special case where the placement host-level *AA*-constraints are to be deferred to the remote site. For clarity we have chosen to model according to the second option as it avoids special cases. Investigating which option is best suited for implementation in production systems remains as future work. Notably, these abstractions are used for modelling reasons, but the actual agreement for resource exchange between two sites would normally be stipulated using Service Level Agreements (SLAs).

D. Scalability of ILP-based Scheduling

In general, scheduling of VMs onto resources is an NP-hard problem [25], [29] and heuristics are frequently used to mitigate the effects of scale on the solvability and solution time. In this work we illustrate how to include the affinity and anti-affinity constraints in pure ILP solvers that provide the optimal solution without use of heuristics. For practical use on larger systems, scalability enhancements should be employed, just as it successfully have been done with very good results for scheduling problems without such constraints [30].

Our experiments are performed with scheduling on a per-service level, which is one common strategy to be able to manage larger systems. For each scheduling iteration, the background load on hosts is comprised of the capacity requirements of previously deployed services.

V. Experimental Evaluation

To assess the applicability of the proposed approach, we have conducted an extensive evaluation. The overall goal of the experiments is to investigate the impact of affinity and anti-affinity on service placement algorithms in terms of feasibility, time-outs, and execution time. As there are no traces from real-world systems available with dependencies among components, we use synthetic data. The main focus is not to evaluate the performance of the proposed approach, but rather to investigate the impact of various factors involved, and thus we believe that synthetic data can fit this goal.

The experimental setup is a scenario consistent with the example presented in Figure 1, where a service comprised of four different types of VMs is analyzed. The number of VM instances of each type and their capacity requirements are shown in Table IVa. As shown in Table V, hardware discretization metrics are adopted to categorize VMs with different computation capacities in a similar approach as used by Amazon EC2. In the evaluation, we strive to place the entire service across a set of hosts with discrete hardware capabilities corresponding to the capacity requirements of the VM types. To add another dimension, we have also assigned different costs to local and remote hosts and the objective function is to reduce the total cost of hosting the service. The host configuration parameters are shown in Table IVb. The host set is assumed to only contain a subset of hosts from the infrastructure (see Section IV-D), and only include local and remote hosts which are eligible for placement in this scenario, i.e., located within the EU.

Table V
Hardware metrics for instance types.

Instance Size	Small	Medium	Large	XLarge	XXLarge
CPU (# cores)	1	1	2	4	8
CPU (GHz/core)	1	2	2	2	2
Memory (GB)	1.7	3.5	7.5	15	30
Capacity	**2**	**4**	**8**	**16**	**32**

The evaluation is carried out by generating a large set of cases with varying amounts of *AA*-constraints and background load on the hosts. The *AA*-constraints are generated by assigning constraint values to random coordinates in a 4×4 host-level constraint matrix corresponding to the one illustrated in Table I. The dataset is generated with the following properties:

1) Background load in the range of [0%, 10%, ..., 90%] of the total host capacity (load is randomly distributed across the set of hosts).
2) Affinity-constraints ranging between 0 and 16 elements in the constraints matrix (randomly placed).
3) Anti-affinity-constraints ranging between 0 and 16 elements in the constraints matrix (randomly placed).
4) Cases where the number of elements needed for affinity and anti-affinity combined exceeds the size of the constraint-matrix (in effect, requiring 17 or more elements) are ignored to improve simulation time.
5) Conflicting distributions (i.e., cases with conflicting *AA*-constraints) are avoided by regenerating the input until a valid distribution can be found.
6) N iterations of the above distributions, where $N = 10$ for these tests.

The dataset is thus comprised of 15300 input permutations, each one encoded using the AMPL [31] modeling language and solved with the Gurobi [32] solver. All experiments are performed on a workstation with 2.70 GHz quad-core CPU and 8 GB of memory. The problem set size (100 VM instances to be placed across 80 hosts) is, to the best of our knowledge, considerably larger than the amount of VMs required to host a typical three-tier Web application. We forsee that *AA*-constraints as a concept is more interesting for owners of large and complex services than for those running services with fewer components. Large services with *AA*-constraints are also more difficult to place compared to smaller services, and therefore provide a more interesting case for testing. To avoid introducing unreasonably long delays in the placement process, we specify a 30 seconds execution time limit for each problem case. Cases that can not be solved within 30 seconds count as timeouts.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2362120, IEEE Transactions on Cloud Computing

Table IV
EXPERIMENTAL CONFIGURATION

(a) VM configuration.

Instance Type	FE	LO	PDB	SDB
Number of instances	40	20	20	20
Capacity req. / instance	2	4	8	8

(b) Host configuration.

Host Type	Local	Remote
Number of hosts	60	20
Capacity / host	32	32
Cost / capacity unit	10	15

Table VI
OVERALL TENDENCY AS IMPACT FACTORS INCREASE.

Observation \ Factor	Load	Affinity	Anti-Affinity
Feasibility	↘	↘	↘
Time-outs	↘	↘	↗
Execution Time	↘	↗	↗

A. Results and Discussion

The results of the evaluation as such are highly dependent on a number of factors, e.g. quality of the solver, number of VM instances, requirements of VM types, random distribution of background load, and randomly allocated *AA*-constraints. Therefore, the discussion instead focus on how certain factors such as affinity and anti-affinity affect the overall scheduling process with respect to solvability, execution time, etc.

As previously mentioned we have elected to focus on three main parameters; background load, affinity, and anti-affinity while keeping the other factors constant. We have analyzed these parameters in terms of how they affect the feasibility, the amount of time-outs, and the execution time of the solver. A summary of the results is presented in Table VI, and further analysis of the factors follows.

1) Impact of Background Load: As the background load of the hosts increases, less residual capacity can be used to schedule the current service, which also means that there are fewer plausible placement options for the solver. In this evaluation, the total capacity requirements for the service is $40 \times 2 + 20 \times 4 + 20 \times 8 + 20 \times 8 = 480$ units. At 32 capacity units per host, the total available capacity is 2560 units. This means that the service requires at least $480/2560 = 18.75\%$ of the total host capacity, and thus the service can only be placed at all if the hosts are running at 81.25% capacity or less. Figures 3 and 5 confirm that feasible solutions are only found when the background load is 80% or less.

2) Impact of Affinity Constraints: In our experimental setting, affinity turns out to be the most dominating factor with regards to feasibility. Figure 3 shows a varying background load at different amounts of affinity (anti-affinity is set to zero). As illustrated in the figure, the background load is dominated by affinity, and only has a noticeable impact on the results when it reaches very high numbers (80-90%).

Recall that affinity can be interpreted in two different ways; affinity between two different types means that each instance has to be co-hosted with *at least one* instance of the other type, while affinity within the same type means that *all* instances of that type needs to be co-hosted. Due to the large scale of the service used in this evaluation, no single host has the capacity to do such co-hosting as the maximum capacity per host (32 units) is not sufficient to host all instances of any type (with a combined required capacity of 80 or 160 units). In effect, this means that if a random affinity distribution contains a constraint on the diagonal (within the same type), then that particular distribution cannot be solved using the range of available hosts. This behavior is further discussed in the evaluation summary.

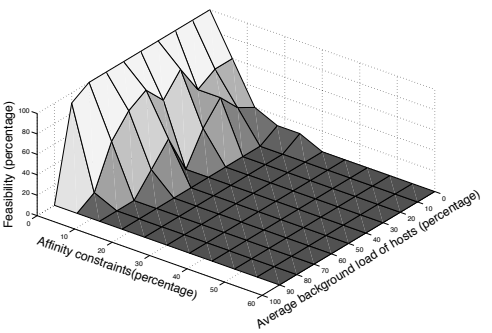


Figure 3. Feasibility depending on affinity constraints and background load.

Figure 4 shows affinity when combined with anti-affinity (at a constant background load of zero). As is evident when comparing Figure 3 and 4, the results are very similar and affinity is the dominating factor also in this case. The major difference is when anti-affinity reaches over 30%, at which point the anti-affinity has a considerably stronger impact than affinity.

3) Impact of Anti-Affinity Constraints: Another analysis was performed to compare the impact between anti-affinity and background load (illustrated in Figure 5). Based on this, we can conclude that the large number of available hosts (80) compared to the number of VM instances in the service (100) is able to sustain a higher percent of anti-affinity constraints (compared to affinity constraints) before the ability to successfully place the service is affected.

4) Timeouts and Execution Time: Figure 6 summarizes the impact of affinity on timeouts and execution time. In this figure, the execution time line is the average of all cases that could be solved within 30 seconds, either by finding an optimal solution or concluding that no solution is possible. The line marked timeout shows the percent of experiments that could not be solved within 30 seconds.

We can examine the data in three different segments:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2362120, IEEE Transactions on Cloud Computing

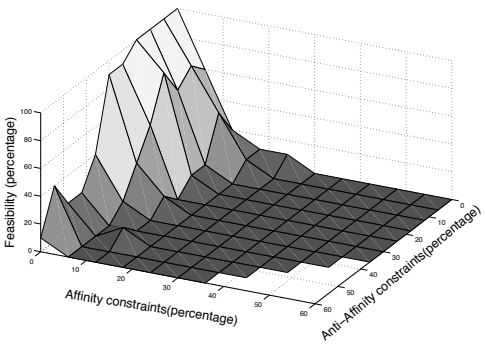


Figure 4. Feasibility depending on affinity- and anti-affinity constraints.

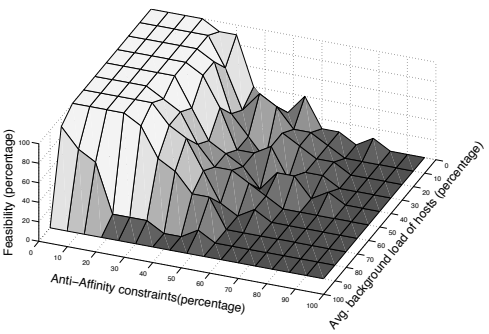


Figure 5. Feasibility depending on anti-affinity constraints and average background load.

- At zero affinity, the execution time and number of timeouts are both very low. In this case, finding the optimal solution is trivial for the solver as it can simply maximize the use of the cheapest available resources.
- When affinity increases, the number of candidate optimal solutions increases rapidly and the solver needs to evaluate many more alternatives before concluding which placement is optimal. As affinity increases to the 30% range, the amount of feasible solutions (as shown in Figure 3) decreases rapidly, which results in fewer timeouts.
- Above 30% affinity the execution time increases linearly with regards to the amount of affinity (and hence the number of constraints in the model). At the same time, the solver can more accurately determine that no solution will be found and the number of timeouts decreases.

5) Evaluation Summary: This evaluation has served to illustrate how *AA*-constraints under varying background load affect the placement of VM instances across as set of hosts. As illustrated in Table VI, the feasibility of placing a service decreases as the background load and number of *AA*-constraints

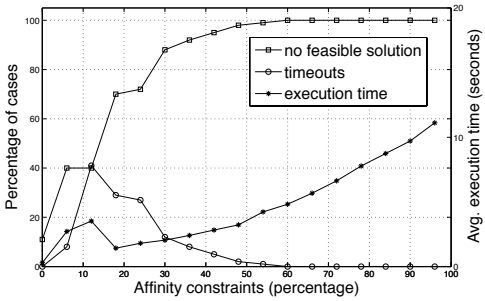


Figure 6. Timeouts and execution time vs. affinity constraints

increase. This is expected, as any service is easier to place without any restrictions and with a lower background load resulting in more available resources. The amount of time outs increases with a higher number of anti-affinity constraints, while it decreases as the number of affinity constraints and the percentage of background load increase. Increased background load and increased amounts of affinity constraints reduce the amount of possible solutions and thereby also the amount of time outs, while a higher number of anti-affinity constraints will generate a large set of cases that the solver can optimize. This is consistent with the results discussed in Section V-A4. Finally, the execution time increases with a higher number of *AA*-constraints (although not linearly, as shown in Section V-A4), while it decreases with a higher percentage of background load as the search space of candidate hosts is smaller. This suggests that in case of using, e.g., heuristic-based solvers providing non-optimal solutions, it would make sense to investigate methods that first strive to fulfill AA-constraints before solving the complete problem.

We have elected to count any case that cannot be solved within 30 seconds as a time out failure. In realistic scenarios, it is likely that the best solution found within the time-frame will be used, even if it is suboptimal. The only way to determine how far from optimality such suboptimal solutions are is to let the solver run (possibly indefinitely) until an optimal solution has been found, and compare the two according to the objective function. Further experiments of this kind using our model would be interesting as part of future work.

The relative impact of background load and *AA*-constraints in these tests indicate that affinity is the most restrictive factor followed by anti-affinity, and that placement feasibility is only marginally affected by background load. It is very likely that anti-affinity would instead be the dominating factor in a test environment with fewer hosts but with a higher capacity per host, as that would allow more instances to be co-placed at the same host while making it harder to achieve anti-affinity with fewer physical hosts. This observation can be turned into a model used to quantify the ability of an existing infrastructure to cope with *AA*-constraints by analyzing the number and capacity of available hosts. Creating and evaluating such a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2362120, IEEE Transactions on Cloud Computing

model is also a part of future work.

VI. Conclusions and Future Work

This work is motivated by the current lack of influence offered to service providers regarding placement of their service components in clouds. This limitation makes cloud hosting inappropriate for several service categories depending on, e.g., certain legislation, geographical proximity, and fault-tolerance.

Based on previous work on structured services, we have in this paper (I) showed how hierarchical graph structures can be converted into placement constraints, modeled as matrices; (II) presented a mathematical model for service owner-controlled placement directives, and; (III) demonstrated the feasibility of this model using a large set of simulated cases with varying amounts of background load and *AA*-constraints. Together, the contributions of this paper enables infrastructure providers to extend their placement engines and algorithms to offer service providers influence over how services are placed without giving up control over their own infrastructure. This enables cloud adoption also for services with the aforementioned requirements.

We have identified several interesting subjects for future work, including support for arbitrary groupings and level divisions for *AA*-constraints; to consider also inter-service relations; studying how to best overcome the uncertainty of not having access to complete information from collaborating remote sites; and support for soft constraints (e.g. preferences) as exemplified for network distances by Alicherry and Lakshman [24], [25]. We would also like to compare using suboptimal results (the best found within a certain amount of time) to using the optimal results obtained by allowing the solver to run uninterrupted.

There are also interesting tasks regarding the adoption of *AA*-constraints into cloud infrastructure offerings. The added complexity of supporting service owner-controlled placement directives needs to be economically compensated for, and devising such compensation models is a necessary step toward adoption.

Acknowledgments

We thank Eddie Wadbro for sharing his knowledge on linear programming solvers. The research that led to these results has been partially supported by the European Commission's Seventh Framework Programme under grant agreements no. 215605 (RESERVOIR), no. 257115 (OPTIMIS), and is further support the Swedish Government's strategic research project eSSENCE and the Swedish Research Council under contract no. C0590801 (Cloud Control).

References

[1] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galán, "The RESERVOIR model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, 2009, paper 4.

[2] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, 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," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66–77, 2012.

[3] R. Buyya, J. Broberg, and A. Gościński, Eds., *Cloud Computing: Principles and Paradigms*, ser. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons, 2011.

[4] P. Massonet, S. Naqvi, C. Ponsard, J. Latanicki, B. Rochwerger, and M. Villari, "A monitoring and audit logging architecture for data location compliance in federated cloud infrastructures," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*. IEEE, 2011, pp. 1510–1517.

[5] L. Larsson, D. Henriksson, and E. Elmroth, "Scheduling and Monitoring of Internally Structured Services in Cloud Federations," in *Proceedings of IEEE Symposium on Computers and Communications*, 2011, pp. 173–178.

[6] U. Lampe, M. Siebenhaar, R. Hans, D. Schuller, and R. Steinmetz, "Let the clouds compute: cost-efficient workload distribution in infrastructure clouds," in *Proceedings of the 9th international conference on Economics of Grids, Clouds, Systems, and Services*, ser. GECON'12. Springer-Verlag, 2012, pp. 91–101.

[7] T. A. Genez, L. F. Bittencourt, and E. R. Madeira, "Workflow scheduling for SaaS/PaaS cloud providers considering two SLA levels," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 906–912.

[8] W. Li, J. Tordsson, and E. Elmroth, "Modeling for Dynamic Cloud Scheduling via Migration of Virtual Machines," in *Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2011)*, 2011, pp. 163–171.

[9] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic Placement of Virtual Machines for Managing SLA Violations," in *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management*. IEEE, 2007, pp. 119–128.

[10] S. Chaisiri, B. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC)*. IEEE, 2009, pp. 103–110.

[11] F. Machida, M. Kawato, and Y. Maeno, "Redundant virtual machine placement for fault-tolerant consolidated server clusters," in *Proceedings of the IEEE Symposium on Network Operations and Management (NOMS)*. IEEE, 2010, pp. 32–39.

[12] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proceedings of the IEEE INFOCOM*. IEEE, 2010, pp. 1–9.

[13] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 331–340.

[14] I. Brandic, S. Pllana, and S. Benkner, "High-level composition of QoS-aware Grid workflows: an approach that considers location affinity," in *Proceedings of the workshop on Workflows in Support of Large-Scale Science, in conjunction with the 15th IEEE International Symposium on High Performance Distributed Computing*, 2006, pp. 1–10.

[15] 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.

[16] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing Public Cloud Providers," in *Proceedings of the 10th ACM SIGCOMM conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 1–14.

[17] W. Li, P. Svärd, J. Tordsson, and E. Elmroth, "A general approach to service deployment in cloud environments," in *Proceedings of the 2nd IEEE International Conference on Cloud and Green Computing*. IEEE, 2012, pp. 17–24.

[18] D. Breitgand, A. Marashini, and J. Tordsson, "Policy-Driven Service Placement Optimization in Federated Clouds," IBM Research, Tech. Rep. H-0299, 2011.

[19] A. Atamtürk and M. Savelsbergh, "Integer-Programming Software Systems," *Annals of Operations Research*, vol. 140, no. 1, pp. 67–124, 2005.

[20] T. Koch, A. Martin, and M. Pfetsch, "Progress in Academic Computational Integer Programming," in *Facets of Combinatorial Optimization*. Springer Berlin Heidelberg, 2013, pp. 483–506.

[21] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, "Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement," in *Proceedings of the IEEE International Conference on Services Computing (SCC)*. IEEE, 2011, pp. 72–79.

[22] F. Hermenier, S. Demassey, and X. Lorca, "Bin repacking scheduling in virtualized datacenters," *Principles and Practice of Constraint Programming–CP*, pp. 27–41, 2011.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2014.2362120, IEEE Transactions on Cloud Computing

[23] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, L., "Entropy: a consolidation manager for clusters," in *VEE '09: Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 2009, pp. 41–50.
[24] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 963–971.
[25] ——, "Optimizing data access latencies in cloud systems by intelligent virtual machine placement," in *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 647–655.
[26] Distributed Management Task Force, Inc., "Open Virtualization Format Specification version 2.0.0c," work in progress, http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.0.0c.pdf. [Online]. Available: http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.0.0c.pdf
[27] R. Pereira, M. Azambuja, K. Breitman, and M. Endler, "An architecture for distributed high performance video processing in the cloud," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010, pp. 482–489.
[28] W. Li, J. Tordsson, and E. Elmroth, "Modeling for dynamic cloud scheduling via migration of virtual machines," in *Proceedings of the Third IEEE International Conference on Cloud Computing Technology and Science (Cloudcom)*. IEEE Computer Society, 2011, pp. 357–366.
[29] A. Roytman, A. Kansal, S. Govindan, J. Liu, and S. Nath, "PACMan: Performance Aware Virtual Machine Consolidation," in *Proceedings of the 10th International Conference on Autonomic Computing*. USENIX, 2013, pp. 83–94.
[30] S. Chaudhuri, R. A. Walker, and J. E. Mitchell, "Analyzing and exploiting the structure of the constraints in the ilp approach to the scheduling problem," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 4, pp. 456–471, 1994.
[31] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, Nov. 2002.
[32] Gurobi, gurobi Optimization, http://www.gurobi.com, visited January 2013.

Daniel Espling holds a Ph.D. from the Department of Computing Science, Umeå University, Sweden. He received his master's degree in Computing Science in 2007, and his Ph.D. degree in 2013. He conducts research on management topics in multi-grid and multi-cloud computing environments. He has contributed to the European Commission's RESERVOIR and OPTIMIS projects and his research topics include monitoring of hardware and software based measurements, accounting and billing, and scheduling/placement of grid job and cloud services.

Lars Larsson received his MSc. in computing science at Umeå University in 2008 with a 4.9 of 5 grade average. Mixing software development with post-graduate studies that started in 2009, his research interest is cloud computing infrastructure management. He has worked on the European Commission's RESERVOIR project, has taught the advanced course at Distributed Systems at the Department of Computing Science at Umeå University, and is currently working with research and development of a commercial auto-scaling software.

Wubin Li holds a Ph.D. from the Department of Computing Science, Umeå University, Sweden. He received his Ph.L. (licentiate) degree in 2012 and his Ph.D. degree in 2014. Before that, he received his master's degree in 2005 from Institute of Computing Technology, Chinese Academy of Sciences. He also worked as a research assistant and software engineer in Tencent, Inc during 2008 and 2009. He has contributed to the European Commission's OPTIMIS project and his research topics focus on resource placement and scheduling in cloud environments.

Johan Tordsson is Assistant Professor at the Department of Computing Science, Umeå University, from he also received his Ph.D. in 2009. After a period as visiting postdoc researcher at Universidad Complutense de Madrid, he worked for several years in the RESERVOIR, VISION Cloud, and OPTIMIS European projects, in the latter as Lead Architect and Scientific Coordinator. Tordsson's research interestests include autonomic management problems for cloud and grid computing infrastructures as well as enabling technologies such as virtualization.

Erik Elmroth is Professor at the Department of Computing Science, Umeå University, Sweden. His background in eScience includes virtual computing infrastructures (Clouds and Grids), parallel computing, algorithms for managing memory hierarchies, linear algebra library software, and ill-posed eigenvalue problems. He received the Nordea Scientific Award 2011 and he was co-recipient of the SIAM Linear Algebra Prize 2000, for the most outstanding linear algebra publication world-wide during the preceding three-year period. He currently leads the Distributed Systems computing research at Umeå University, focusing on infrastructure and application tools for Cloud computing. International experiences include a year at NERSC, Lawrence Berkeley National Laboratory, University of California, Berkeley, and one semester at the Massachusetts Institute of Technology (MIT), Cambridge, MA. Erik is Chairman of the Swedish National Infrastructure for Computing (SNIC), has been member of the Swedish Research Council's Committee for Research Infrastructures (KFI), and Chairman of its expert group on eScience. He has been appointed the Scientific Secretary for producing two e-science strategies for the Nordic Council of Ministers.