Algorithms and Systems for Virtual Machine Scheduling in Cloud Infrastructures

Wubin Li

李务斌

PhD Thesis, April 2014
Department of Computing Science
Umeå University
Sweden

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

wubin.li@cs.umu.se

ISBN 978-91-7601-019-8
ISSN 0348-0542
UMINF 14.06

Printed by Print & Media, Umeå University, 2014

Abstract

With the emergence of cloud computing, computing resources (i.e., networks, servers, storage, applications, etc.) are provisioned as metered on-demand services over networks, and can be rapidly allocated and released with minimal management effort. In the cloud computing paradigm, the virtual machine (VM) is one of the most commonly used resource units in which business services are encapsulated. VM scheduling optimization, i.e., finding optimal placement schemes for VMs and reconfigurations according to the changing conditions, becomes challenging issues for cloud infrastructure providers and their customers.

The thesis investigates the VM scheduling problem in two scenarios: (i) single-cloud environments where VMs are scheduled within a cloud aiming at improving criteria such as load balancing, carbon footprint, utilization, and revenue, and (ii) multi-cloud scenarios where a cloud user (which could be the owner of the VMs or a cloud infrastructure provider) schedules VMs across multiple cloud providers, targeting optimization for investment cost, service availability, etc. For single-cloud scenarios, taking load balancing as the objective, an approach to optimal VM placement for predictable and time-constrained peak loads is presented. In addition, we also present a set of heuristic methods based on fundamental management actions (namely, suspend and resume physical machines, VM migration, and suspend and resume VMs), continuously optimizing the profit for the cloud infrastructure provider regardless of the predictability of the workload. For multi-cloud scenarios, we identify key requirements for service deployment in a range of common cloud scenarios (including private clouds, bursted clouds, federated clouds, multi-clouds, and cloud brokering), and present a general architecture to meet these requirements. Based on this architecture, a set of placement algorithms tuned for cost optimization under dynamic pricing schemes are evaluated. By explicitly specifying service structure, component relationships, and placement constraints, a mechanism is introduced to enable service owners the ability to influence placement. In addition, we also study how dynamic cloud scheduling using VM migration can be modeled using a linear integer programming approach.

The primary contribution of this thesis is the development and evaluation of algorithms (ranging from combinatorial optimization formulations to simple heuristic algorithms) for VM scheduling in cloud infrastructures. In addition to scientific publications, this work also contributes software tools (in the OPTIMIS project funded by the European Commissions Seventh Framework Programme) that demonstrate the feasibility and characteristics of the approaches presented.

Sammanfattning

I datormoln tillhandahålls datorresurser (dvs., nätverk, servrar, lagring, applikationer, etc.) som tjänster åtkomliga via Internet. Resurserna, som t.ex. virtuella maskiner (VMs), kan snabbt och enkelt allokeras och frigöras alltefter behov. De potentiellt snabba förändringarna i hur många och hur stora VMs som behövs leder till utmanade schedulerings- och konfigureringsproblem. Scheduleringsproblemen uppstår både för infrastrukturleverantörer som behöver välja vilka servrar olika VMs ska placeras på inom ett moln och deras kunder som behöver välja vilka moln VMs ska placeras på.

Avhandlingen fokuserar på VM-scheduleringsproblem i dessa två scenarier, dvs (i) enskilda moln där VMs ska scheduleras för att optimera lastbalans, energiåtgång, resursnyttjande och ekonomi och (ii) situationer där en molnanvändare ska välja ett eller flera moln för att placera VMs för att optimera t.ex. kostnad, prestanda och tillgänglighet för den applikation som nyttjar resurserna. För det förstnämnda scenariot presenterar avhandlingen en scheduleringsmetod som utifrån förutsägbara belastningsvariationer optimerar lastbalansen mellan de fysiska datorresurserna. Därtill presenteras en uppsättning heuristiska metoder, baserade på fundamentala resurshanteringsåtgärder, för att kontinuerligt optimera den ekonomiska vinsten för en molnleverantör, utan krav på lastvariationernas förutsägbarhet.

För fallet med flera moln identifierar vi viktiga krav för hur resurshanteringstjänster ska konstrueras för att fungera väl i en rad konceptuellt olika fler-moln-scenarier. Utifrån dessa krav definierar vi också en generell arkitektur som kan anpassas till dessa scenarier. Baserat på vår arkitektur utvecklar och utvärderar vi en uppsättning algoritmer för VM-schedulering avsedda att minimera kostnader för användning av molninfrastruktur med dynamisk prissättning. Användaren ges genom ny funktionalitet möjlighet att explicit specificera relationer mellan de VMs som allokeras och andra bivillkor för hur de ska placeras. Vi demonstrerar också hur linjär heltalsprogrammering kan användas för att optimera detta scheduleringsproblem.

Avhandlingens främsta bidrag är utveckling och utvärdering av nya metoder för VM-schedulering i datormoln, med lösningar som inkluderar såväl kombinatorisk optimering som heuristiska metoder. Utöver vetenskapliga publikationer bidrar arbetet även med programvaror för VM-schedulering, utvecklade inom ramen för projektet OPTIMIS som finansierats av EU-kommissionens sjunde ramprogram.

Preface

This thesis consists of an introduction to cloud computing and virtual machine scheduling in cloud environments, and the below listed papers.

Paper I	**Wubin Li**, Johan Tordsson, and Erik 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)*, pp. 163–171, 2011.
Paper II	**Wubin Li**, Johan Tordsson, and Erik Elmroth. Virtual Machine Placement for Predictable and Time-Constrained Peak Loads. In *Proceedings of the 8th international conference on Economics of grids, clouds, systems, and services (GECON 2011)*, Lecture Notes in Computer Science, Vol. 7150, Springer-Verlag, pp. 120–134, 2011.
Paper III	**Wubin Li**, Petter Svärd, Johan Tordsson, and Erik Elmroth. A General Approach to Service Deployment in Cloud Environments. In *Proceedings of the 2nd IEEE International Conference on Cloud and Green Computing (CGC 2012)*, pp. 17-24, 2012.
Paper IV	**Wubin Li**, Petter Svärd, Johan Tordsson, and Erik Elmroth. Cost-Optimal Cloud Service Placement under Dynamic Pricing Schemes. In *Proceedings of the 6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2013)*, pp. 187-194, 2013.
Paper V	Daniel Espling, Lars Larsson, **Wubin Li**, Johan Tordsson, and Erik Elmroth. Modeling and Placement of Cloud Services with Internal Structure. Submitted, 2014.
Paper VI	Petter Svärd, **Wubin Li**, Eddie Wadbro, Johan Tordsson, and Erik Elmroth. Continuous Datacenter Consolidation. Technical Report UMINF-14.08. Department of Computing Science, Umeå University, March, 2014.

This research was conducted using the resources of the High Performance Computing Center North (HPC2N) and the UMIT research lab. Financial support has been provided by the European Community's Seventh Framework Programme ([FP7/2001-2013]) under grant agreements no. 257115 (OPTIMIS) and no. 257019 (VISION Cloud), and the Swedish Government's strategic effort eSSENCE.

Additional publications by the author not included in the thesis:

- Erik Elmroth, Johan Tordsson, Francisco Hernandez, Ahmed Ali-Eldin, Petter Svärd, Mina Sedaghat, and **Wubin Li**: Self-Management Challenges for Multi-Cloud Architectures. *ServiceWave 2011, Lecture Notes in Computer Science*, Vol. 6994, Springer-Verlag, pp. 38-49, 2011.
- **Wubin Li**, Johan Tordsson, and Erik Elmroth An Aspect-Oriented Approach to Consistency-Preserving Caching and Compression of Web Service Response Messages. In *Proceedings of the 8th IEEE International Conference on Web Services (ICWS 10)*, pp. 526-533.
- **Wubin Li** and Petter Svärd: REST-Based SOA Application in the Cloud: A Text Correction Service Case Study. In *Proceedings of IEEE 6th World Congress on Services (SERVICES 2010)*, pp. 84-90.

Acknowledgments

How time flies. It has been 4+ years since I moved to Sweden. During these years, lots of people have contributed to this work and deserve acknowledgments. Without their support, patience and guidance, this thesis would not have been completed. In particular, I would like to express my sincere and deepest gratitude to:

Erik Elmroth, my supervisor, for inviting me to Sweden and providing an excellent research environment, for your timely meetings, discussions and emails, for your enthusiasm and invaluable suggestions.

Johan Tordsson, my co-supervisor, for working over time with me for paper deadlines during the weekend(s), for sharing your knowledge and experience, for the inspiring discussions, for the outstanding job you have done on proofreading my papers.

Eddie Wadbro, for your guidance and interesting ideas during discussions, and your effort on our joint work.

Petter Svärd, for the joint work and your nice personality which always makes me feel easy and relaxed when working and traveling with you.

Daniel Espling and **Lars Larsson**, for sharing your ideas and experience, for answering me tons of questions (not just work related) and helping me with various tools and systems.

P-O Östberg, for sharing your cabin and making my first ski-trip in Sweden a reality, and for the excellent lectures presented in the SOA course.

Mina Sedaghat, for bringing so much fun and interesting moments, and for sending me a lot of information on job opportunities.

Other group members (in no particular order), **Gonzalo Rodrigo Alvarez**, **Peter Gardfjäll**, **Luis Tomás**, **Ahmed Ali-Eldin**, **Amardeep Mehta**, **Ewnetu Bayuh Lakew**, **Kosten Selome Tesfatsion**, **Olumuyiwa Ibidunmoye**, **Francisco Hernández**, **Lennart Edblom**, **Lei Xu**, **Cristian Klein**, and **Jakub Krzywda** for creating a fantastic atmosphere and all fruitful discussions.

All technical support staff, especially **Tomas Forsman**, **Mats Johansson**, and **Bertil Lindkvist**, for their skillful and continued support which eases the burden on us and helps us focus on our own tasks.

All floorball players, for creating the exciting and passionate moments every Tuesday at IKSU. I will never forget the moment when I scored the first goal for our white team in my first game! I hope you guys can find enough players to continue this great tradition which is more than 20 years old.

Chinese friends I have made since I moved to Sweden, especially **Da Wang**, **Meiyue Shao**, **Yafeng Song**, **Xueen Jia**, **Guangzhi Hu**, **Jia Wang**, **Shi Tang**, **Zhi-**

han Lv, for the time we spent together (for discussion, barbecue, fishing, basketball, and running, etc.).

Xiao-yu Huang and **Bing Liang**, for your helpful suggestions, timely help, and kindly encouragement since I graduated with my Bachelor degree.

Lastly, I would like to thank my family for all their love and encouragement. Particularly, I am extremely grateful to my mother-in-law **Zhengyun Yang** and my wife **Shijun Wang** for undertaking household chores and taking care of little **Hannah**, enabling me to focus on my work. Without their unconditional support, this work would not have been accomplished.

Thank you!

Umeå, March 2014
Wubin Li

Contents

Chapter 1

Introduction

By provision of shared resources as metered on-demand services over networks, Cloud Computing is emerging as a promising paradigm for providing configurable computing resources (i.e., networks, servers, storage, applications, and services) that can be rapidly allocated and released with minimal management effort. Cloud end-users (e.g., service consumers and developers of cloud services) can access various services from cloud providers such as Amazon, Google and SalesForce. They are relieved from the burden of IT maintenance and administration and their total IT cost is expected to decrease. From the perspective of a cloud provider or an agent, however, resource allocation and scheduling become challenging issues. This may be due to the scale of resources to manage, and the dynamic nature of service behavior (with rapid demands for capacity variations and resource mobility), as well as the heterogeneity of cloud systems. As such, finding optimal placement schemes for resources, and making resource reconfigurations in response to the changes of the environment are difficult [21].

There are a multitude of parameters and considerations (e.g., performance, cost, locality, reliability and availability) in the decision of where, when and how to place and reallocate virtualized resources in cloud environments. Some of the considerations are aligned with one another while others may be contradictory. This work investigates challenges involved in the problem of VM scheduling in cloud environments, and tackles these challenges using approaches ranging from combinatorial optimization techniques and mathematical modeling to simple heuristic methods. Note that the term *scheduling* in the context of this thesis is referred to as the initial placement of VMs and the readjustment of placement over time.

Scientific contributions of this thesis include modeling for dynamic cloud scheduling via VM migration in multi-cloud environments, cost-optimal VM placement across multiple clouds under dynamic pricing schemes, modeling and placement of cloud services with internal structure, as well as to optimize VM placement within data centers for predicable and time-constrained load peaks, and continuous VM scheduling aiming at maximizing the profit for a cloud infrastructure provider. In addition, the feasibility and characteristics of the proposed solutions are demonstrated by a set of software tools contributed in the EU-funded project OPTIMIS [26].

The rest of this thesis is organized as follows. Chapter 2 provides a brief introduction to Cloud Computing. Chapter 3 describes virtual machine scheduling in cloud environments. Chapter 4 summarizes the contributions of the thesis and presents the papers. Finally, conclusions and future work are given in Chapter 5 followed by a list of references and the papers.

Chapter 2

Cloud Computing

Cloud Computing provides a paradigm shift following the shift from mainframe to client-server architecture in the early 1980s [33, 92]. It is a new paradigm in which computing is delivered as a service rather than a product, whereby shared resources, software, and information are provided to consumers as a utility over networks.

The vision of this paradigm can be traced back to 1969 when Leonard Kleinrock [47, 48], one of the chief scientists of the original Advanced Research Projects Agency Network (ARPANET) project, which preceded the Internet, stated at the time of ARPANET's development:

"*as of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of 'computer utilities' which, like present electric and telephone utilities, will service individual homes and offices across the country.*"

Over the past decades, new computing paradigms (e.g., Grid Computing [45], P2P Computing [72], and Cloud Computing [6]) promising to deliver this vision of computing utilities have been proposed and adopted. Of all these paradigms, the two most frequently mentioned ones with differing areas of focus are Grid Computing and Cloud Computing [12]. Grids are designed to support sharing of pooled resources, usually used for solving problems that may require thousands of processor cores or hundreds of terabytes of storage, while cloud technologies are driven by economies of scale, focusing on integrating resource capacities to the public in the form of a utility and enabling access to leased resources (e.g., computation power, storage capacity, and software services) at prices comparable to in-house hosting [24, 27]. The distinctions between these two paradigms are sometimes not clear as they share the same vision [85]. An in-depth comparison between girds and clouds is beyond the scope of this thesis, but for details there are a number of valuable works available, e.g., by Foster et al. [27], Mei et al. [63], Zhang et al. [95], EGEE [8], and Sadashiv et al. [77].

One of the main advantages and motivations behind Cloud Computing is reducing the CAPEX (capital expenditures) of systems from the perspective of cloud users and providers. By renting resources from cloud providers in a pay-per-use manner [85], cloud customers benefit from lowered initial investments and relief of IT maintenance. On the other hand, taking advantage of virtualization technologies, cloud providers are enabled to increase the energy-efficiency of the infrastructures and scale the costs of the offered virtualized resources. The paradigm has been proved to be suitable for a wide range of applications, e.g., for hosting websites [66] and social networks applications [14], scientific workflows [36], Customer Relationship Management [78, 91], and high performance computing [20].

2.1 Virtualization

Virtualization is a technology that separates computing functions and implementations from physical hardware. Early related research dates back to 1960s and the joint work of IBM TJ Watson and MIT on the M44/44X Project [38]. Now virtualization has become the foundation of Cloud Computing [93], since it enables isolation between hardware and software, between users, and between processes and resources. These isolation problems have not been well solved by traditional operating systems. With virtualization, software capable of execution on the raw hardware can be run in a virtual environment. Depending on the layer where the virtualization occurs, two major categories of virtualization can be identified (as illustrated in Figure 1):

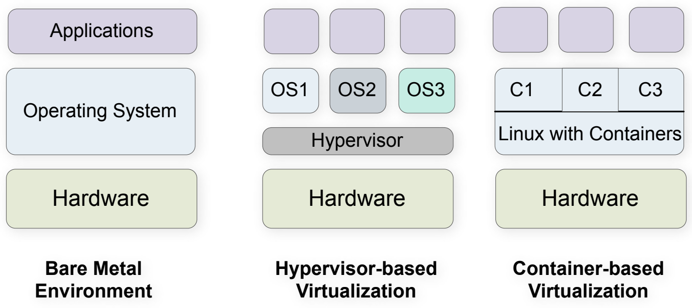

Figure 1: A bare-metal environment (left), compared to two major categories of virtualization (center and right). Illustration from MontaVista [68].

Hypervisor-based Virtualization. This technology is based on a layer of software (i.e., the hypervisor) that manages the resources of physical hosts and provides the necessary services for the VMs to run. Instead of direct access to the underlying hardware layer, all VMs request resources from the hypervisor that is in charge of resource allocation and scheduling for VMs. There are two major types of implementations of this kind of virtualization, briefly described as follows.

I. *Full virtualization* [87], fully emulates system hardware, and thus does not require changes to the operating system (OS) or applications. Virtualization is done transparently at the hardware level of the system. Well known implementations include Microsoft Virtual PC [37], VMware Workstation [88], VirtualBox [90], and KVM [46].

II. *Paravirtualization* [87], requires changes to the OS and possibly the applications to take full advantage of optimizations of the virtualized hardware layer, and thus achieves better performance than Full Virtualization. As a well established example, Xen [7] offers a Paravirtualization solution.

In environments with hypervisor-based virtualization, Cloud services can be encapsulated in virtual appliances (VAs) [44], and deployed by instantiating virtual machines with their virtual appliances [43]. Moreover, since the underlying hardware is emulated, multiple different operating systems (see OS1, OS2 and OS3 in Figure 1) are usually allowed to run in virtual machines atop the hypervisor. This new type of service deployment provides a direct route for traditional on-premise applications to be rapidly redeployed in a Software as a Service (SaaS) manner for SPs. By decoupling the infrastructure provider possessing hardware (and usually operating system) from the application stack provider, virtual appliances allow economies of scale which is a great attraction for IT industries. This thesis work is based on hypervisor-based virtualization. Throughout the thesis, unless otherwise specified, the term virtualization refers to this category.

Container-based Virtualization. This technology is also known as operating system virtualization [18, 79, 86], a light-weight virtualization which is not aimed to emulate an entire hardware environment, as traditional virtual machines do. Relying on the recent underlying improvements that enable the Linux kernel manage isolation between applications, an operating system-level virtualization method can run multiple isolated LXC (LinuX Containers) on a single control host. Rather than providing virtualization via a virtual machine managed by a specific hypervisor, LXC provides a virtual environment that has its own process and network space. Systems such as Docker [18], Linux-VServer [57] and OpenVZ [71] are implementation examples of this kind. This category of virtualization is more efficient than traditional virtualization technologies since the virtualization is at the OS API level. There are, however, some drawbacks to containers, e.g., they are not as flexible as other virtualization approaches because it is infeasible to host a guest OS different from the host OS, or a different guest kernel. As a consequence, workload migration is more complex than that in an environment supporting hypervisor-based virtualization.

2.2 The XaaS Service Models

Commonly associated with cloud computing are the following service models, differing in the service offered to the customers:

I. **Software as a Service (SaaS)**
In the SaaS model, software applications are delivered as services that execute on infrastructure managed by the SaaS vendor itself or a third-party infrastructure provider. Consumers are enabled to access services over various clients such as web browsers and programming interfaces, and are typically charged on a subscription basis [64]. The implementation and the underlying cloud infrastructure where the service is hosted are transparent to consumers.

II. **Platform as a Service (PaaS)**
In the PaaS model, cloud providers deliver a computing platform and/or solution stack typically including operating system, programming language execution environment, database, and web server. Application developers can develop and run their software on a cloud platform without having to manage or control the underlying hardware and software layers, including network, servers, operating systems, or storage, but maintain the control over the deployed applications and possibly configuration settings for the application-hosting environment [64].

III. **Infrastructure as a Service (IaaS)**
In the IaaS model, computing resources such as storage, network, and computation resources are provisioned as services. Consumers are able to deploy and run arbitrary software, which can include operating systems and applications. Consumers do not manage or control the underlying physical infrastructure but have to control their own virtual infrastructures typically constructed by virtual machines hosted by the IaaS vendor. This thesis work is mainly focusing on the IaaS model, although it may be generalized also to apply to the other models.

2.3 Cloud Computing Scenarios and Roles

Based on the classification of cloud services into SaaS, PaaS, and IaaS, three main stakeholders in a cloud provisioning scenario can be identified:

I. **Infrastructure Providers (IPs)** provision infrastructure resources such as virtual instances, networks, and storage to consumers usually by utilizing hardware virtualization technologies. In the IaaS model, a consumer rents resources from an infrastructure provider or multiple infrastructure providers, and establishes its own virtualized infrastructure, instead of maintaining an infrastructure with dedicated hardware. There are numerous infrastructure providers on the market, such as Amazon Elastic

Compute Cloud (EC2) [3], GoGrid [29], and Rackspace [75]. To simplify the application delivery for consumers, some infrastructure providers go a step further with the PaaS model, i.e., in addition to supporting application hosting environments, these infrastructure providers also provide development infrastructure including programming environment, tools, configuration management, etc. [17]. Some notable providers of this type include Google App Engine [30], Salesforce.com [78], and AppFog [5]. In academia, some ongoing projects such as ConPaaS [74] and 4CaaSt [28] are developing new PaaS frameworks that enable flexible deployment and management of cloud-based services and applications.

II. **Service Providers (SPs)** use either their own resources (taking both the SP and IP roles) or resources leased from one or multiple IPs to deliver end-user services to their consumers. It can be a telco service provider, an internet service provider (e.g., LinkedIn [56]), etc. These services can be potentially developed using PaaS tools as mentioned previously. In particular, when cloud resources are leased from external IPs, SPs are not in charge of maintaining the underlying hardware infrastructures. Without having direct control over the low-level hardware resources, SPs can use performance metrics (e.g., response time) to optimize their applications by scaling their rented resources from IPs, providing required Quality of Service (QoS) to the end users.

III. **Cloud End Users** who are the consumers of the services offered by SPs and usually have no concerns on where and how the services are hosted.

As identified by M. Ahronovitz et al. [1], differing from deployment models, four main types of cloud scenarios can be listed as follows.

I. **Private Cloud**.

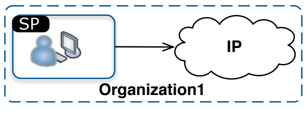

Figure 2: Private cloud scenario.

An organization provisions services using internal infrastructure, and thus plays the roles of both a SP and an IP. Private clouds can circumvent many of the security and privacy concerns related to hosting sensitive information in public clouds. They may also offer stronger guarantees on control and performance as the whole infrastructure can be administered within the same domain.

II. **Cloud Bursting**.

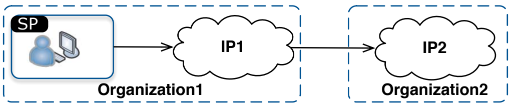

Figure 3: Cloud bursting scenario.

Private clouds may offload capacity to other IPs under periods of high workload, or for other reasons, e.g., planned maintenance of the internal servers. In this scenario, the providers form a hybrid architecture commonly referred to as a *cloud bursting* as seen in Figure 3. Typically, less sensitive tasks are executed in the public cloud while tasks that require higher levels of security stay in the private infrastructure.

III. **Federated Cloud**.

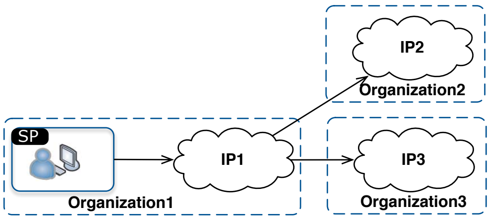

Figure 4: Cloud federation scenario.

Federated clouds are IPs collaborating on a basis of joint load-sharing agreements enabling them to offload capacity to each other [76] in a manner similar to how electricity providers exchange capacity. The federation takes place at the IP level in a transparent manner. In other words, a SP that deploys services to one of the IPs in a federation is not notified if its service is off-loaded to another IP within the federation. However, the SP may be able to steer in which IPs the service may be provisioned, e.g., by specifying location constraints in the service manifest. Figure 4 illustrates a federation between three IPs.

IV. **Multi-Cloud**.

In multi-cloud scenarios, the SP is responsible for handling the additional complexity of coordinating the service across multiple external IPs, i.e., planning, initiating and monitoring the execution of services.

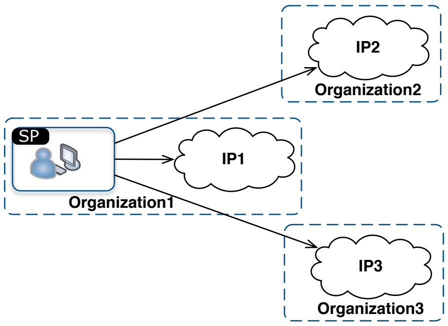

Figure 5: Multi-cloud scenario.

It should be remarked that the multi-cloud and federated cloud scenarios are commonly considered only in the special case where Organization 1 does not possess an internal infrastructure, corresponding to removing IP1 from Figures 4 and 5.

Chapter 3

Virtual Machine Scheduling

Given a set of admitted services and the availability of local and possibly remote resources, there are a number of scheduling problems to be solved to determine where to store data and where to execute and reallocate VMs. We categorize these problems into two classes, namely single-cloud environments and multi-cloud environments. The following sections describe these two scenarios respectively, as well as the challenges and the state of the art of VM scheduling.

3.1 Scheduling in Single-cloud Scenarios

In this thesis, VM scheduling in single-cloud environments is referred to as scenarios where VMs are scheduled within an infrastructure provider that can have multiple data centers geographically distributed. This is consistent with the Private Cloud scenario described in Chapter 2, while cases where the private infrastructure outsources (part of) its workload to external infrastructure provider(s) belong to another class of scenarios discussed in the following section. In single-cloud scenarios, resource characteristics, including the real-time state of the whole infrastructure, the revenue model, and the schedule policies, are usually exposed to the scheduling optimization process. A scheduling algorithm can thus take full advantage of the information potentially available.

A well-known case is when a cloud provider strives to lower the carbon footprint of operating the infrastructures and scale the costs of the offered virtualized resources. This is very appealing to IT industries and also has significant impact on the global environment, as more than 1% of the global electricity consumption is consumed by data centers [49]. Energy cost per year can exceed $105,000 for a single rack of servers [73], while according to a study by IDC and IBM in 2008, most test servers run at 10% utilization. Furthermore, 30% of all defects are caused by wrongly configured servers and 85% of computing sites are idle [39]. To improve the energy-efficiency of infrastructures that rely on virtualization technologies, VMs running in the cloud need to be properly configured and scheduled, ensuring high energy-efficiency of the cloud systems [58]. As another key aspect, from a profit perspective, Service-Level Agreement (SLA) compliance is also crucial as violations in SLA can result in significant revenue loss to both the customer and the provider. This may also require accurate and efficient SLA compliance monitoring [80].

3.2 Scheduling in Multi-cloud Scenarios

Multi-cloud scenarios include (i) one cloud infrastructure that offloads its workload to another infrastructure, for example in order to lower the operating costs while maintaining customer satisfaction, and (ii) a cloud user who deploys and manages VMs across multiple cloud infrastructures gaining advantage of avoidance of vendor lock-in problem, improving service availability and fault-tolerance, etc. This is consistent with the cases of cloud bursting, cloud federation, and multi-cloud mentioned in Chapter 2. In such cases, decision making is usually focused on selecting which cloud to run in, not which server. The detailed states of the infrastructures are commonly opaque to the cloud user or the cloud infrastructure that initiates the non-local actions. Conversely, the remote cloud infrastructures usually only expose business-related info such as VM instance types, pricing schemes, locality of the infrastructures, and legal information to the optimization process. VM scheduling in these scenarios is also complicated by obstacles in integrating resources from various cloud providers which usually have their own characteristics of resources, protocols and APIs.

3.3 Objectives and Considerations

There are a multitude of parameters and considerations involved in the decision on where and when to place or reallocate data objects and computations in cloud environments. An automated scheduling mechanism should take the considerations and tradeoffs into account, and allocate resources in a manner that benefits the stakeholder for which it operates (SP or IP). For both of these, this often leads to the problem of optimizing cost or performance subject to a set of constraints. Among the main considerations are:

- **Performance**. In order to improve the utilization of physical resources, data centers are increasingly employing virtualization and consolidation as a means to support a large number of disparate applications running simultaneously on server platforms. With different VM scheduling strategies, the achieved performance may differ significantly [83]. In scenarios where multiple cloud providers are involved, the performance is of additional concern, as preserving performance of systems constructed by integrating resources from heterogeneous infrastructures is a challenge with high complexity.

- **Energy-efficiency**. In line with the interest in eco-efficiency technologies, increasing overall efficiency of cloud infrastructures in terms of power, cost, and utilization has naturally become a major concern. However, this is usually conflicting with other concerns, e.g., performance.
- **Costs**. The price model was dominated by fixed prices in the early phase of cloud adoption. However, cloud market trends show that dynamic pricing schemes utilization is increasing [60]. Deployment costs decrease by dynamically placing services among clouds or by dynamically reconfiguring services (e.g., resizing VM sizes without harming service performance) become possible. In addition, internal implicit costs for VM scheduling, e.g., interference and overhead that one VM causes on other concurrently running VMs on the same physical host, should also be taken into account.
- **Locality**. In general, for considerations of usability and accessibility, VMs should be located close to users (which could be other services or VMs). However, due to e.g., legal issues and security reasons, locality may become a constraint for optimal scheduling. This may apply to both cloud providers with geographically distributed data centers and service providers utilizing resources from multiple cloud providers.
- **Reliability and continuous availability**. Part of the central goals for VM scheduling is service reliability and availability. To achieve this, VMs may be replicated across multiple (at least two) geographical zones. During this procedure, factors such as the importance of the data and/or service encapsulated in VMs, its expected usage frequency, and the reliability of the different data centers, must be taken into account. As such, scheduling VMs within a single-cloud environment may also cause service degradation, e.g., by introducing additional delays due to VM migration, or by co-locating to many VMs with competing demands on a single physical server.

3.4 Main Challenges

Given the variety of VM scheduling scenarios, the wide range of relevant parameters, and the set of constraints and objective functions of potential interest, there are a number of challenges in the development of broadly applicable scheduling methods, some of which are presented below.

I. There exists no generic model to represent various scenarios of VM scheduling, especially when users' requirements are vague and hard to encode through modeling languages. In particular, mapping QoS requirements (e.g., latency, consistency, and reliability) of applications to fine-grained resource-level attributes is difficult [22]. Applications have various business-level requirements for QoS based on different metrics such as response time, throughput, and transaction rate. Such requirements depend on the type of the applications and how they are being used. Modeling and quantifying these requirements, especially non-functional requirements such as high availability, is challenging.

II. Model parameterization, i.e., finding suitable values for parameters in a proposed model, is a tedious task when the problem size is large. For example, for a multi-cloud scenario that includes n cloud providers and m VMs, $m * n^2$ parameter assignments are needed in principle to express the VM migration overheads ignoring possible changes of VM sizes. Therefore, mechanisms that can help to automatically capture those values are required.

III. The initial VM placement problem is typically formulated as a variant of the class constrained multiple-knapsack problem that is known to be NP hard [15]. Thus, to solve large-scale problem instances, tradeoffs between quality of solution and execution time must be taken into account. This is a very important issue given the size of real life data centers, e.g., by 2011, Amazon EC2 [3], the leading cloud provider, has approximately 40,000 servers and schedules 80,000 VMs every day [23]. These numbers may be even larger today as the cloud market is much bigger than three years ago. Finding usable solutions for such large-sized data centers in an acceptably short time, resulting high scalability of the solution, is known to be difficult [11].

IV. Conflicting objectives. On energy-efficient scheduling, existing work [9, 19, 51, 94] focuses on certain aspects of QoS, however they commonly overlook the energy-efficiency aspect that may conflict with the other QoS requirements. For example, the migration of a given VM from one data-center to another may have a positive impact on reducing the carbon footprint. However it may also cause service degradation by introducing additional delays, or even reduce the availability if two redundant VMs are co-located on the same physical server that forms a single point of failure. Besides, harmonizing the incompatibility between conflicting objectives becomes even more challenging when the importance of these objectives is hard to quantify accurately.

V. Continuous optimization. Given the dynamic nature of clouds, resource allocations need to be renewed regularly for performance reasons, failure, etc., for example when a SLA violation is detected, or when the cloud resources are not efficiently utilized. It is challenging to efficiently decide when and how to reconfigure the cloud in order to dynamically adapt to the changes. Such a challenge has been identified as a MAPE-K (Monitoring, Analysis, Planning, Execution, and Knowledge) [40] control loop by IBM, resulting in the concept of *Autonomic Computing*. In the context of the MAPE-K reference model, information such as resource usage, and workload demands is collected from managed entities, aggregated, filtered and reported by *Monitoring* mechanisms. An adaptation *Plan* is produced and *Executed* based on the *Analysis* of the collected information. *Knowledge* collected or derived is shared among all parties involved, possibly providing solid support to decision making.

3.5 State of the Art

Virtual machine scheduling in distributed environments has been extensively studied in the context of cloud computing. Such approaches address separate problems, such as initial placement, consolidation, or tradeoffs between honoring SLAs and constraining provider operating costs, etc. [67]. Studied scenarios are usually encoded as assignment or packing problems in mathematical models and are finally solved either by approximations, e.g., greedy packing and heuristic methods, or by existing mathematical programming solvers such as Gurobi [32], CPLEX [41] and GLPK [31]. As before, related work can be separated into two sets: (i) VM scheduling in single-cloud environments and (ii) VM scheduling in multi-cloud environments.

In the single-cloud scenario, given a set of physical machines and a set of services (encapsulated within VMs) with dynamically changing demands, to decide how many instances to run for each service and where to put and execute them, while observing resource constraints, is an NP hard problem [15]. Tradeoff between quality of solution and computation cost is a challenge. To address this issue, various approximation approaches are applied, e.g., Tang et al. [15] propose an algorithm that can produce high-quality solutions for hard placement problems with thousands of machines and thousands of VMs within 30 seconds. This approximation algorithm strives to maximize the total satisfied application demand, to minimize the number of application starts and stops, and to balance the load across machines. Hermenier et al. [34] present the Entropy resource manager for homogeneous clusters, which performs dynamic consolidation based on constraint programming and takes migration overhead into account. Entropy chooses migrations that can be implemented efficiently, incurring a low performance overhead. The CHOCO constraint programming solver [42], with optimizations e.g., identifying lower and upper bounds that are close to the optimal value, is employed to solve the problem. To reduce electricity cost in high performance computing clouds that operate multiple geographically distributed data centers, Le et al. [50] study the impact of VM placement policies on cooling and maximum data center temperatures. They develop a model of data center cooling for a realistic data center and cooling system, and design VM distribution policies that intelligently place and migrate VMs across the data centers to take advantage of time-based differences in electricity prices and temperatures. Targeting the energy efficiency and SLA compliance, Borgetto et al. [11] present an integrated management framework for governing Cloud Computing infrastructures based on three management actions, namely, VM migration and reconfiguration, and power management on physical machines.

Incorporating an autonomic management loop optimized using a wide variety of heuristics ranging from rules over random methods, the proposed approach can save energy up to 61.6% while keeping SLA violations acceptably low.

For VM scheduling across multiple IPs, information about the number of physical machines, the load of these physical machines, and the state of resource distribution inside the IPs' side is normally hidden from the SP and hence is not part of the parameters that can be used for placement decisions. Only provision-related information such as types of VM instance and price schemes, is exposed to SP. Therefore, most work on VM scheduling across multi-cloud environments is focusing on cost aspects. Chaisiri et al. [13] propose an stochastic integer programming (SIP) based algorithm that can minimize the cost spending in each placement plan for hosting virtual machines in a multiple cloud provider environment under future demand and price uncertainty. Van den Bossche et al. [16] examine the workload outsourcing problem in a multi-cloud setting with deadline-constrained workloads, and present a cost-optimal optimization method to maximize the utilization of the internal data center and minimize the cost of running the outsourced tasks in the cloud, while fulfilling the QoS constraints for applications. Tordsson et al. [84], propose a cloud brokering mechanism for optimized placement of VMs to obtain optimal cost-performance tradeoffs across multiple cloud providers. Similarly, Vozmediano et al. [69, 70] explore the multi-cloud scenario to deploy a compute cluster on top of a multi-cloud infrastructure, for provisioning loosely-coupled Many-Task Computing (MTC) applications. In this way, the cluster nodes can be provisioned with resources from different clouds to improve the cost-effectiveness of the deployment, or to implement high-availability strategies.

Chapter 4

Summary of Contributions

4.1 Paper I

In non-local scenarios, cloud users may want to distribute the VMs across multiple providers for various purposes, e.g., in order to construct a user's cloud environment and prevent potential vendor lock-in problems by means of migrating applications and data between data centers and cloud providers. Most likely, the decision on VMs distribution among cloud providers is not a one-time event. Conversely, it needs to be adjusted according to the changes exposed. In Paper I [55], we investigate dynamic cloud scheduling in scenarios where conditions are continuously changed, and propose a linear programming model to dynamically reschedule VMs (including modeling of VM migration overhead) upon new conditions such as price changes and service demand variation. Our model can be applied in various scenarios through selections of corresponding objectives and constraints, and offers the flexibility to express different levels of migration overhead when restructuring an existing virtual infrastructure, i.e., VM layout.

In scenarios where new instance types are introduced, the proposed mechanisms can accurately determine the break-off point when the improved performance resulting from migration outweighs the migration overhead. It is also demonstrated that our cloud mechanism can cope with scenarios where prices change over time. Performance changes, as well as transformation of VM distribution across cloud providers as a consequence of price changes, can be precisely calculated. In addition, the ability of the proposed mechanism to handle the tradeoff between vertical (resizing VMs) and horizontal elasticity (adding VMs), as well as to improve decision making in complex scale-up scenarios with multiple options for service reconfiguration, e.g., to decide how many new VMs to deploy, and how many and which VMs to migrate, is also evaluated in scenarios based on commercial cloud providers' offerings.

4.2 Paper II

In Paper II [54], the VM placement problem for load balancing of predictable and time-constrained peak workloads is studied for placement of a set of VMs within a single datacenter. We formulate the problem as a Min-Max optimization problem and present an algorithm based on binary integer programming, along with three approximations for tradeoffs in scalability and performance. Based on the observation that two VM sets (i.e., VMs provisioned to fulfill service demands) may use the same physical resources if they do not overlap in runtime, we define an approximation based on discrete time slots to generate all possible overlap sets. A time-bound knapsack algorithm is derived to compute the maximum load of machines in each overlap set after placing all VMs that run in that set. Upper bound based optimizations are used to shorten the time required to compute a final solution, enabling larger problems to be solved. An evaluation based on synthetic workload traces suggests that our algorithms are feasible, and that these can be combined to achieve desired tradeoffs between quality of solution and execution time.

4.3 Paper III

The cloud computing landscape has developed into a spectrum of cloud architectures, resulting in a broad range of management tools for similar operations but specialized for certain deployment scenarios. This not only hinders the efficient reuse of algorithmic innovations for performing the management operations, but also increases the heterogeneity between different cloud management systems. A overarching goal is to overcome these problems by developing tools general enough to support the range of popular architectures. In Paper III [52], we analyze commonalities in multiple different cloud models (private clouds, multi-clouds, bursted clouds, federated clouds, etc.) and demonstrate how a key management functionality - service deployment - can be uniformly performed in all of these by a carefully designed system. The design of our service deployment solution is validated through demonstration of how it can be used to deploy services, perform bursting and brokering, as well as mediate a cloud federation in the context of the OPTIMIS Cloud toolkit.

4.4 Paper IV

At the early stage of the cloud era, most cloud providers used fixed pricing schemes to offer capacity to customers. Under these schemes, the price of a compute unit was usually set regardless of the available capacity at the provider. For example, GoGrid [29] and Rackspace [75] offer capacity on hourly, monthly, semi-annual, and annual base, without considering the real-time state of the backend infrastructures. As a consequence, most research on cloud service placement has focused on static pricing scenarios. However, the concept of dynamic resource pricing is becoming popular and has garnered a lot of attention recently. One promising advantage of this concept is that it enables cloud providers the ability to attract more customers by offering lower price if they have excess capacity. Amazon for example has introduced spot instances [2], enabling users to bid for spare Amazon EC2 instances and run them whenever the bid exceeds the current spot price, which is set by Amazon and varies in real-time based on supply and demand for the spot instance capacity.

From the cloud customer's perspective, such pricing models complement static pricing, potentially providing the most cost-effective option for obtaining compute capacity. To investigate cloud service placement under dynamic pricing schemes, we in Paper IV [53] study a set of algorithms to find cost-optimal deployment of services across multiple cloud providers. The algorithms range from simple heuristics to combinatorial optimization solutions. By deploying nearly 3000 synthetically constructed services with varying amounts of service components and VM instance types on three simulated cloud providers using the service deployment toolkit presented in Paper III [52], the studied algorithms are evaluated in terms of execution time, ratio of successfully solved deployment cases, and the quality of the solution. The results suggest that exhaustive search based approach is (as expected) good at finding optimal solutions for service placement under dynamic pricing schemes, but execution time is usually very long. In contrast, greedy approaches perform surprisingly well with fast execution times and acceptable solutions. More specifically, the very fast greedy algorithm finds optimal solutions in more than half of all cases, and for 90% of the rest of cases, the quality of solution is within 25% from optimal. As such, it can be a suitable compromise considering the tradeoffs between quality of solution and execution time. We believe that results from this paper can be helpful in the design of scheduling algorithms and mechanisms in cloud environments with dynamic pricing schemes.

4.5 Paper V

A cloud service might be compromised of multiple components. For example, a three-tier web application may consist of a database component (e.g., Oracle DB), an application component (e.g., JBoss application server), and a presentation layer component (Apache web server). There are multiple advantages in terms of e.g., fault tolerance, redundancy, and legislation compliance of taking the internal structure of the service into consideration when placing components in cloud infrastructures. For example, due to legislative reasons [61], some services might not be allowed to be provisioned in specific regions. Furthermore, some services might require redundancy by avoiding collocation of critical components in the same host.

In Paper V [25], we present an approach that formalizes hierarchical graph structures for inter-dependencies among components in a service, enabling service owners to influence placement of their service components by explicitly specifying the service structure, component relationships, and constraints among components. We also demonstrate how these can be converted into placement constraints. In particular, we use *affinity* constraints to express restrictions on service components that must be co-located, and *anti-affinity* constraints to express the requirements on component instances that may not be placed on the same host or cloud level. An integer linear programming formulation is defined to illustrate how scheduling may be performed using the presented approach. The feasibility of the model is confirmed by experimental evaluation on 15300 randomly constructed services with varying amounts of background load, affinity constraints and anti-affinity constraints. Our experimental results indicate that (i) with respect to the ability of finding a solution, the impact introduced by the number of affinity and anti-affinity constraints is higher than that by background loads, and (ii) component affinity is the dominating factor affecting the possibility to find a solution in a setting with a large number of hosts with low capacity.

4.6 Paper VI

To continuously optimize the mapping of VMs to physical servers is crucial in cloud infrastructures, as the initial mapping might become suboptimal upon changes introduced by for example workload variations, failures, energy-management actions such as power-off and frequency-scaling, and the availability of resources. In Paper VI [82], we present a continuous VM consolidation approach that aims to maximize cloud provider revenue over time. A combination of management actions, including suspend and resume physical servers, suspend and resume VMs, and VM migration, is used to achieve this goal. Based on these actions, we define a set of heuristic algorithms to continuously optimize the revenue for the cloud infrastructure without limitation on the predictability of the workload. The performance of the proposed algorithms are confirmed by experimental evaluation on synthetic workloads. To verify that the proposed ideas are applicable in real-world scenarios, we also design and implement a proof-of-concept software to manage a small-sized datacenter, showing the feasibility of our approach.

Chapter 5

Future Work

The thesis focuses on cloud services placement and scheduling in cloud infrastructures from the perspective of an infrastructure provider and a service provider, respectively. This chapter presents potential directions for future investigations.

Constraint Relaxation and Improve Algorithms

Future directions for this work include approximation algorithms based on problem relaxations and heuristic approaches such as greedy formulation for considerations of tradeoff between quality of solution and execution time. It would also be interesting to allow cloud users to specify hard constraints and soft constraints when demanding resource provisions. A hard constraint is a condition that has to be satisfied when deploying services, i.e., it is mandatory. In contrast, soft constraints (also called preferences) are optional. An optimal placement solution with soft constraints satisfied is preferable over other solutions. The hard and soft constraints can, e.g., be used to specify co-location or avoidance of co-location of certain VMs. We also plan to investigate how to apply multi-objective optimization techniques to this scenario.

Modeling Inter-VM Relationships

Despite the attempt in Paper V, we believe that modeling the interconnection requirements that can precisely express the relationships between VMs is far from complete. For example, extending the affinity mechanism to support more internal network properties is one direction. There exists no specification or standard to semantically describe the relationships between VMs in the context of cloud computing. Given the variety of existing applications, it is unlikely to find a general approach that can be applied to any domain. We also foresee that the relationships between VMs are not necessarily static. They may change over time. For example, a VM responsible for secondary storage might take the place of a VM for primary storage on hardware failure. The inter-VM relationships should be updated (reconstructed) accordingly. VM scheduling in cloud infrastructures with compliance to such a dynamic relationship is challenging. In addition, by inferring from the VM relationships, scheduling algorithms would possibly benefit from knowing the interference and overhead that one VM causes on other concurrently running VMs on the same physical machine.

Network and Storage Considerations

So far, most research on VM scheduling in cloud environments has focused on aspects of computational resource management. Scheduling network resources and storage in combination with computation of VMs remains largely unexplored. The distribution of VMs in cloud infrastructures might affect the network traffic and scalability of the infrastructures. For example, by localizing large chunks of traffic and thus reducing load at high-level switches, a traffic-aware VM scheduling approach can have a significant performance improvement compared with existing generic methods that do not take advantage of traffic patterns and data center network characteristics [65]. On the other hand, the interconnection network of cloud infrastructures has a significant impact on VM scheduling strategies and system utilization. Network topology knowledge is important for efficient path selection for VM migration. Higher workload density in combination with network bandwidth intensive migrations can lead to network contention [81].

Scheduling for Container-based Virtualization Platforms

As discussed in Section 2.1, this thesis work is based on traditional virtualization techniques, i.e., hypervisor-based virtualization. Compared with hypervisor-based virtualization, despite being less mature and providing less isolation [35, 62], Linux Containers offer several advantages. For example, reduced overhead can be obtained by using normal system call interface instead of introducing a hypervisor layer as a intermediate to support hardware emulation, and by maintaining operating systems with the same kernel rather than maintaining multiple different operating systems in a hypervisor-based virtualization environment which can be a heavy task [68]. Moreover, as a lightweight virtualization mechanism [10, 86], containers are usually smaller than conventional virtual machines, meaning that given the same physical machine, it is possible to run more containers on it than conventional virtual machines.

To the best of our knowledge, however, there exists very few work on cloud services (encapsulated in containers) scheduling on platforms based on container-based virtualization. In particular, there is no literature on this topic in the context of multi-cloud scenarios (if this is even feasible). Most of the existing works on this topic are focusing on building clouds atop of container-based virtualization, e.g., by Anwer et al. [4] who present the design and implementation of a fast, virtualized data center with OpenVZ [71] as the virtualization support and with NetFPGA [59, 89] as the hardware layer. An interesting direction would be to investigate VM scheduling problems on container-based virtualization infrastructures. Intuitively, new constraints would be introduced to express the corresponding restrictions, e.g., with the existing technology, a container is not allowed to be migrated to another host with different kernel. Moreover, VM migration overhead and the interference introduced by co-location should be modeled and profiled in a different way from hypervisor-based virtualization platforms.

Bibliography

[1] M. Ahronovitz, D. Amrhein, P. Anderson, A. de Andrade, J. Armstrong, B. Arasan, J. Bartlett, R. Bruklis, K. Cameron, and M. Carlson. Cloud Computing Use Cases White Paper, v4.0. `http://www.cloudusecases.org`, visited March 2014.

[2] Amazon. Amazon EC2 Spot Instance. `http://aws.amazon.com/ec2/spot-instances/`, visited March 2014.

[3] Amazon.com, Inc. Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`, visited March 2014.

[4] M. B. Anwer and N. Feamster. Building a Fast, Virtualized Data Plane with Programmable Hardware. *ACM SIGCOMM Computer Communication Review*, 40(1):75–82, 2010.

[5] AppFog, Inc. AppFog PaaS Cloud. `https://www.appfog.com`, visited March 2014.

[6] M. Baker, G. C. Fox, and H. W. Yau. Cluster Computing Review. 1995. `http://surface.syr.edu/npac/33`, Northeast Parallel Architecture Center. Paper 33.

[7] 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. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[8] M.-E. Bégin, B. Jones, J. Casey, E. Laure, F. Grey, C. Loomis, and R. Kubli. An EGEE Comparative Study: Grids and Clouds - Evolution or Revolution. *EGEE III Project Report*, 30, 2008.

[9] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing. *Future Generation Computer Systems*, 28(5):755 – 768, 2012. Special Section: Energy Efficiency in Large-scale Distributed Systems.

[10] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford. Trellis: A Platform

for Building Flexible, Fast Virtual Networks on Commodity Hardware. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 72:1–72:6. ACM, 2008.

[11] D. Borgetto, M. Maurer, G. Da-Costa, J.-M. Pierson, and I. Brandic. Energy-efficient and SLA-aware Management of IaaS Clouds. In *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, page 25. ACM, 2012.

[12] R. Buyya, C. S. Yeo, and S. Venugopal. 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)*, pages 5–13. IEEE, 2008.

[13] S. Chaisiri, B.-S. Lee, and D. Niyato. Optimal Virtual Machine Placement across Multiple Cloud Providers. In *Proceedings of the 4th IEEE Asia-Pacific Services Computing Conference*, pages 103–110.

[14] K. Chard, S. Caton, O. Rana, and K. Bubendorfer. Social Cloud: Cloud Computing in Social Networks. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, pages 99–106. IEEE, 2010.

[15] T. Chunqiang, S. Malgorzata, S. Michael, and P. Giovanni. A Scalable Application Placement Controller for Enterprise Data Centers. In *Proceedings of the 16th International Conference on World Wide Web*, WWW'07, pages 331–340. ACM, 2007.

[16] R. V. den Bossche, K. Vanmechelen, and J. Broeckhove. Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads. In *Proceedings of the 2010 IEEE International Conference on Cloud Computing*, pages 228–235. IEEE Computer Society, 2010.

[17] T. Dillon, C. Wu, and E. Chang. Cloud Computing: Issues and Challenges. In *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 27–33. IEEE, 2010.

[18] Docker Inc. Docker: The Linux Container Engine. `http://www.docker.io`, visited March 2014.

[19] C. Dupont, G. Giuliani, F. Hermenier, T. Schulze, and A. Somov. An Energy Aware Framework for Virtual Machine Placement in Cloud Federated Data Centres. In *Proceedings of 2012 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy)*, pages 1–10. IEEE, 2012.

[20] J. Ekanayake and G. Fox. High Performance Parallel Computing with Clouds and Cloud Technologies. In *Cloud Computing*, pages 20–38. Springer, 2010.

[21] E. Elmroth, J. Tordsson, F. Hernández, A. Ali-Eldin, P. Svärd, M. Sedaghat, and W. Li. Self-management Challenges for Multi-cloud Architectures. In W. Abramowicz, I. Llorente, M. Surridge, A. Zisman, and J. Vayssière, editors, *Towards a Service-Based Internet*, volume 6994 of *Lecture Notes in Computer Science*, pages 38–49. Springer Berlin/Heidelberg, 2011.

[22] V. C. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar. Low level Metrics to High level SLAs-LoM2HiS Framework: Bridging the Gap between Monitored Metrics and SLA Parameters in Cloud Environments. In *Proceedings of the 2010 International Conference on High Performance Computing and Simulation (HPCS)*, pages 48–54, 2010.

[23] D. Erickson, B. Heller, S. Yang, J. Chu, J. D. Ellithorpe, S. Whyte, S. Stuart, N. McKeown, G. M. Parulkar, and M. Rosenblum. Optimizing a Virtualized Data Center. In *Proceedings of the 2011 ACM SIGCOMM Conference (SIGCOMM'11)*, pages 478–479, 2011.

[24] D. Espling. Enabling Technologies for Management of Distributed Computing Infrastructures. Doctoral Thesis, Faculty of Science and Technology, Department of Computing Science, Umeå University, October, 2013.

[25] D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth. Modeling and Placement of Cloud Services with Internal Structure. Submitted, 2014.

[26] A. J. Ferrer, F. Hernádez, 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*, 28(1):66–77, 2012.

[27] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-degree Compared. In *Proceedings of Grid Computing Environments Workshop (GCE'08)*, pages 1–10. IEEE, 2008.

[28] S. Garcia-Gomez, M. Escriche-Vicente, P. Arozarena-Llopis, F. Lelli, Y. Taher, C. Momm, A. Spriestersbach, J. Vogel, A. Giessmann, F. Junker, et al. 4CaaSt: Comprehensive Management of Cloud Services through a PaaS. In *Proceedings of the IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 494–499. IEEE, 2012.

[29] GoGrid. GoGrid Cloud. `http://www.gogrid.com`, visited March 2014.

[30] Google Inc. Google App Engine. `https://developers.google.com/appengine`, visited March 2014.

[31] GUN. GNU Linear Programming Kit, `http://www.gnu.org/s/glpk/`, visited March 2014.

[32] Gurobi Optimization, Inc. Gurobi Optimization Solver, `http://www.gurobi.com`, visited March 2014.

[33] B. Hayes. Cloud Computing. *Communications of the ACM*, 51(7):9–11, July 2008.

[34] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall. Entropy: a Consolidation Manager for Clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, pages 41–50. ACM, 2009.

[35] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 2011.

[36] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good. On the Use of Cloud Computing for Scientific Workflows. In *Proceedings of the IEEE Fourth International Conference on eScience*, pages 640–645. IEEE, 2008.

[37] J. Honeycutt. Microsoft Virtual PC 2007 Technical Overview. *Microsoft, February*, 2007. `http://goo.gl/HsjCeK`, visited March 2014.

[38] J.-Y. Hwang, S.-B. Suh, S.-K. Heo, C.-J. Park, J.-M. Ryu, S.-Y. Park, and C.-R. Kim. Xen on ARM: System Virtualization using Xen Hypervisor for ARM-based Secure Mobile Phones. In *Proceedings of the 5th IEEE Consumer Communications and Networking Conference*, CCNC 2008, pages 257–261. IEEE, 2008.

[39] IBM. Industry Developments and Models – Global Testing Services: Coming of Age. IDC, 2008 and IBM Internal Reports.

[40] IBM Corp. An Architectural Blueprint for Autonomic Computing, Oct. 2004. Tech. Rep.

[41] IBM Corporation. IBM ILOG CPLEX Optimizer, `http://www.ibm.com/software/integration/optimization/cplex-optimizer/`, visited March 2014.

[42] N. Jussien, G. Rochart, and X. Lorca. The CHOCO Constraint Programming Solver. In *Proceedings of the CPAIOR'08 Workshop on OpenSource Software for Integer and Contraint Programming (OSSICP'08)*, 2008.

[43] G. Kecskemeti, P. Kacsuk, T. Delaitre, and G. Terstyanszky. Virtual Appliances: A Way to Provide Automatic Service Deployment. In F. Davoli, N. Meyer, R. Pugliese, and S. Zappatore, editors, *Remote Instrumentation and Virtual Laboratories*, pages 67–77. Springer US, 2010.

[44] G. Kecskemeti, G. Terstyanszky, P. Kacsuk, and Z. Neméth. An Approach for Virtual Appliance Distribution for Service Deployment. *Future Generation Computer Systems*, 27(3):280–289, March 2011.

[45] C. Kesselman and I. Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Nov. 1998.

[46] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: the Linux Virtual Machine Monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[47] L. Kleinrock. UCLA to be First Station in Nationwide Computer Network. UCLA Press Release, July 1969.

[48] L. Kleinrock. A Vision for the Internet. *ST Journal of Research*, 2(1):4–5, 2005.

[49] J. Koomey. Growth in Data Center Electricity Use 2005 to 2010. *The New York Times*, 49(3), 2011.

[50] K. Le, R. Bianchini, J. Zhang, Y. Jaluria, J. Meng, and T. D. Nguyen. Reducing Electricity Cost Through Virtual Machine Placement in High Performance Computing Clouds. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 22:1–22:12. ACM, 2011.

[51] Y. C. Lee and A. Y. Zomaya. Energy Efficient Utilization of Resources in Cloud Computing Systems. *The Journal of Supercomputing*, 60(2):268–280, 2012.

[52] W. Li, P. Svärd, J. Tordsson, and E. Elmroth. A General Approach to Service Deployment in Cloud Environments. In *Proceedings of the 2nd International Conference on Cloud and Green Computing (CGC 2012)*, pages 17–24. IEEE Computer Society, 2012.

[53] W. Li, P. Svärd, J. Tordsson, and E. Elmroth. Cost-Optimal Cloud Service Placement under Dynamic Pricing Schemes. In *Proceedings of the 6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2013)*, pages 187–194. IEEE Computer Society, 2013.

[54] W. Li, J. Tordsson, and E. Elmroth. Virtual Machine Placement for Predictable and Time-Constrained Peak Loads. In *Proceedings of the 8th International Conference on Economics of Grids, Clouds, Systems, and Services (GECON'11).* Lecture Notes in Computer Science, Vol. 7150, Springer-Verlag, pp. 120-134, 2011.

[55] 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)*, pages 163–171, 2011.

[56] LinkedIn Corporation. LinkedIn. `http://www.linkedin.com`, visited March 2014.

[57] Linux-VServer Project. Linux-VServer. `http://linux-vserver.org`, visited March 2014.

[58] L. Liu, H. Wang, X. Liu, X. Jin, W. B. He, Q. B. Wang, and Y. Chen. GreenCloud: A New Architecture for Green Data Center. In *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session*, ICAC-INDST'09, pages 29–38. ACM, 2009.

[59] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. NetFPGA-an Open Platform for Gigabit-rate Network Switching and Routing. In *Proceedings of the IEEE International Conference on Microelectronic Systems Education*, pages 160–161. IEEE, 2007.

[60] J. Lucas Simarro, R. Moreno-Vozmediano, R. Montero, and I. Llorente. Dynamic Placement of Virtual Machines for Cost Optimization in Multi-Cloud Environments. In *Proceedings of the 2011 International Conference on High Performance Computing and Simulation (HPCS)*, pages 1 –7, July 2011.

[61] 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 (IPDPSW)*, pages 1510–1517. IEEE, 2011.

[62] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens. Quantifying the Performance Isolation Properties of Virtualization Systems. In *Proceedings of the 2007 Workshop on Experimental Computer Science*, page 6. ACM, 2007.

[63] L. Mei, W. K. Chan, and T. Tse. A Tale of Clouds: Paradigm Comparisons and some Thoughts on Research Issues. In *Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC'08)*, pages 464–469. IEEE, 2008.

[64] P. Mell and T. Grance. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology (NIST)*, 2011.

[65] 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*, pages 1–9. IEEE, 2010.

[66] M. Miller. *Cloud Computing: Web-based Applications that Change the Way You Work and Collaborate Online.* Que publishing, 2008.

[67] K. Mills, J. Filliben, and C. Dabrowski. Comparing vm-placement algorithms for on-demand clouds. In *Proceedings of the 2011 IEEE 3rd International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, pages 91–98. IEEE Computer Society, 2011.

[68] MontaVista. Beyond Virtualization: The MontaVista Approach to Multi-core SoC Resource Allocation and Control. `http://mvista.com/download/Whitepaper-Beyond-Virtualization.pdf`, visited March 2014.

[69] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Elastic Management of Web Server Clusters on Distributed Virtual Infrastructures. *Concurrency and Computation: Practice and Experience*, 23(13):1474–1490, 2011.

[70] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Multicloud Deployment of Computing Clusters for Loosely Coupled MTC Applications. *IEEE Transactions on Parallel and Distributed Systems*, 22:924–930, 2011.

[71] OpenVZ Project. OpenVZ Linux Containers. `http://openvz.org`, visited March 2014.

[72] A. Oram. *Peer-to-peer: Harnessing the Benefits of a Disruptive Technologies.* O'Reilly Media, Inc., 2001.

[73] R. Paquet and N. Drakos. Technology Trends You Can't Afford to Ignore, 2009. Gartner Report. `http://my.gartner.com/it/content/992600/992612/technology_trends_you_cant_afford_to_ignore.pdf`.

[74] G. Pierre and C. Stratan. ConPaaS: A Platform for Hosting Elastic Cloud Applications. *Internet Computing, IEEE*, 16(5):88–92, 2012.

[75] Rackspace, US Inc. Rackspace Cloud. `http://www.rackspace.com`, visited March 2014.

[76] 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. Galan. The RESERVOIR Model and Architecture for Open Federated Cloud Computing. *IBM Journal of Research and Development*, 53(4):1–11, 2009.

[77] N. Sadashiv and S. D. Kumar. Cluster, Grid and Cloud Computing: a Detailed Comparison. In *Proceedings of the 6th International Conference on Computer Science and Education (ICCSE)*, pages 477–482. IEEE, 2011.

[78] Salesforce.com, Inc. Salesforce.com. `http://www.salesforce.com`, visited March 2014.

[79] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 275–287. ACM, 2007.

[80] J. Sommers, P. Barford, N. Duffield, and A. Ron. Accurate and Efficient SLA Compliance Monitoring. *ACM SIGCOMM Computer Communication Review*, 37(4):109–120, 2007.

[81] A. Stage and T. Setzer. Network-aware Migration Control and Scheduling of Differentiated Virtual Machine Workloads. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, pages 9–14. IEEE Computer Society, 2009.

[82] P. Svärd, W. Li, E. Wadbro, J. Tordsson, and E. Elmroth. Continuous Datacenter Consolidation. Technical Report UMINF-14.08. Department of Computing Science, Umeå University, March, 2014.

[83] O. Tickoo, R. Iyer, R. Illikkal, and D. Newell. Modeling Virtual Machine Performance: Challenges and Approaches. *SIGMETRICS Perform. Eval. Rev.*, 37(3):55–60, Jan. 2010.

[84] J. Tordsson, R. Montero, R. Moreno-Vozmediano, and I. Llorente. Cloud Brokering Mechanisms for Optimized Placement of Virtual Machines across Multiple Providers. *Future Generation Computer Systems*, 28(2):358 – 367, 2012.

[85] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A Break in the Clouds: towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[86] S. J. Vaughan-Nichols. New Approach to Virtualization is a Lightweight. *Computer*, 39(11):12–14, 2006.

[87] VMware. Understanding Full Virtualization, Paravirtualization, and Hardware Assist. `http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf`.

[88] B. Ward. *The Book of VMware: the Complete Guide to VMware Workstation.* No Starch Press, 2002.

[89] G. Watson, N. McKeown, and M. Casado. NetFPGA: A Tool for Network Research and Education. In *Proceedings of the 2nd Workshop on Architectural Research using FPGA Platforms (WARFP)*, volume 3, 2006.

[90] J. Watson. Virtualbox: Bits and Bytes Masquerading as Machines. *Linux Journal*, 2008(166):1, 2008.

[91] J. Yang and Z. Chen. Cloud Computing Research and Security Issues. In *Proceedings of the 2010 International Conference on Computational Intelligence and Software Engineering (CiSE)*, pages 1–3. IEEE, 2010.

[92] P.-C. Yang, J.-H. Chiang, J.-C. Liu, Y.-L. Wen, and K.-Y. Chuang. An Efficient Cloud for Wellness Self-management Devices and Services. In *Proceedings of the Fourth International Conference on Genetic and Evolutionary Computing (ICGEC)*, pages 767 –770, Dec. 2010.

[93] Q. Zhang, L. Cheng, and R. Boutaba. Cloud Computing: State-of-the-art and Research Challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.

[94] Q. Zhang, M. F. Zhani, S. Zhang, Q. Zhu, R. Boutaba, and J. L. Hellerstein. Dynamic Energy-aware Capacity Provisioning for Cloud Computing Environments. In *Proceedings of the 9th International Conference on Autonomic Computing*, pages 145–154. ACM, 2012.

[95] S. Zhang, X. Chen, S. Zhang, and X. Huo. The Comparison between Cloud Computing and Grid Computing. In *Proceedings of the 2010 International Conference on Computer Application and System Modeling (ICCASM)*, volume 11, pages 72–75. IEEE, 2010.

I

Paper I

Modeling for Dynamic Cloud Scheduling via Migration of Virtual Machines*

Wubin Li, Johan Tordsson, and Erik Elmroth

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

Abstract: Cloud brokerage mechanisms are fundamental to reduce the complexity of using multiple cloud infrastructures to achieve optimal placement of virtual machines and avoid the potential vendor lock-in problems. However, current approaches are restricted to static scenarios, where changes in characteristics such as pricing schemes, virtual machine types, and service performance throughout the service life-cycle are ignored. In this paper, we investigate dynamic cloud scheduling use cases where these parameters are continuously changed, and propose a linear integer programming model for dynamic cloud scheduling. Our model can be applied in various scenarios through selections of corresponding objectives and constraints, and offers the flexibility to express different levels of migration overhead when restructuring an existing infrastructure. Finally, our approach is evaluated using commercial clouds parameters in selected simulations for the studied scenarios. Experimental results demonstrate that, with proper parametrizations, our approach is feasible.

Key words: cloud computing; dynamic scheduling; virtual machine placement; migration overhead; linear integer programming;

[*] By permission of IEEE Computer Society Press.

2011 Third IEEE International Conference on Coud Computing Technology and Science

Modeling for Dynamic Cloud Scheduling via Migration of Virtual Machines

Wubin Li, Johan Tordsson, and Erik Elmroth
Department of Computing Science and HPC2N
Umeå University, Sweden
{wubin.li, tordsson, elmroth}@cs.umu.se

***Abstract*—Cloud brokerage mechanisms are fundamental to reduce the complexity of using multiple cloud infrastructures to achieve optimal placement of virtual machines and avoid the potential vendor lock-in problems. However, current approaches are restricted to static scenarios, where changes in characteristics such as pricing schemes, virtual machine types, and service performance throughout the service life-cycle are ignored. In this paper, we investigate dynamic cloud scheduling use cases where these parameters are continuously changed, and propose a linear integer programming model for dynamic cloud scheduling. Our model can be applied in various scenarios through selections of corresponding objectives and constraints, and offers the flexibility to express different levels of migration overhead when restructuring an existing infrastructure. Finally, our approach is evaluated using commercial clouds parameters in selected simulations for the studied scenarios. Experimental results demonstrate that, with proper parametrizations, our approach is feasible.**

***Index Terms*—cloud computing, dynamic scheduling, virtual machine placement, migration overhead, linear integer programming**

I. Introduction

As the use of cloud computing grows and usage models [1] become more complex, cloud users are confronted with obstacles in integrating resources from various cloud providers. In this context, the use of efficient cloud brokering mechanisms are essential to negotiate the relationships between cloud service consumers and providers, including integrating cloud services to make up a user's cloud environment. A cloud broker also helps users prevent potential vendor lock-in problems by means of migrating applications and data between data centres and different cloud providers.

However, current brokering approaches are limited to static scenarios, where changes in characteristics such as pricing schemes, virtual machine (VM) types, and service performance throughout the service life-cycle are ignored. Conversely in dynamic scenarios, it is arguable that either the offers of the cloud providers or the requirements of the service owner change over time. When conditions change, it is necessary to analyse how to optimally reconfigure the service to adapt it to new situations. For example, if a vendor retreats from the market, cloud users may be forced to migrate some resources from one cloud provider to another so as to guarantee the service availability. Similarly, when a price reduction occurs, the current configuration may become suboptimal, and it may be possible to obtain a better placement of resources by restructuring the virtual infrastructure.

In this paper, we focus on modeling for dynamic scheduling in the context of cloud brokerage where cloud users employ multiple cloud infrastructures to execute their VMs in which business services are encapsulated. In dynamic scheduling scenarios, the ability to efficiently migrate VMs between servers or data centres is crucial for the efficient and dynamic resource management. VM migration is essential to increase the flexibility in VM provisioning, avoid vendor lock-in problems, and guarantee the service availability, etc. One of the key issues for dynamic cloud scheduling is finding a suitable metric for VM migration overhead, a metric that captures the distance between two infrastructures in order to estimate the feasibility of restructuring an existing infrastructure. Possible infrastructure distance metrics include number of VMs to migrate, number of VMs to migrate weighted with VM size, and total migration downtime, etc. Another issue is how to express and embody the migration overhead metric in an objective function that can equivalently represent the user's goal. To tackle these problems, we investigate and classify multiple dynamic scenarios and propose a linear integer programming model. With proper parametrization and selections of objective functions and constraints, our model can be used in a wide range of scenarios. The optimization problem is finally encoded in a mathematical modeling language and solved using a linear programming solver.

In summary, our contributions are the following. We investigate dynamic cloud scheduling use cases and propose a linear integer programming model for dynamic cloud scheduling via VM migration across multiple clouds. Evaluations based on characteristics of current commercial cloud offerings demonstrate that our model provides the flexibility of expressing different levels of migration overhead when restructuring an existing infrastructure. By proper parametrizations, our approach can be used to accurately decide optimal VM migration strategies for elasticity scenarios, as well as handling changes in provider offers and prices.

The remainder of the paper is organized as follows: Section II describes background about cloud brokerage, placement optimization for cloud resources, and VM migration. Section III introduces cloud brokering mechanisms for optimized placement of VMs across multiple providers, describes the proposed model, and elaborates its flexibility for expressing different levels of migration overhead for restructuring an existing infrastructure. Section IV presents experimental evaluations against commercial clouds offerings. Finally, some conclusions are presented in Section V followed by a presentation of future work, acknowledgements, and a list of references.

II. Background and Related Work

A. Cloud Brokerage

Cloud brokerage aims to bridge the gap between the cloud service consumer and the provider. Gartner Research divides the responsibility of cloud brokers into three main categories: cloud service intermediation, aggregation and cloud service arbitrage [2]. On-going research on cloud brokerage has caught substantial attention, including efforts that target cloud management middleware (e.g., Emotive Cloud [3] and OpenNebula [4]), virtualization APIs (e.g., libvirt [5]), and cloud interoperability and standardization.

Grivas et al. propose a central Cloud Broker component responsible for the management and the governance of the cloud environment [6]. However, this approach is mainly focusing on business process management. It should be remarked that this approach is still in the phase of comprehensive architecture design. A cloud broker with an optimal VM placement algorithm is presented by Chaisiri et al. [7]. This algorithm can minimize the cost for hosting VMs in a multi-provider environment. This work is however limited to static scenarios where the number of required virtual resources is constant, and the cloud provider conditions (resource prices, resource availability, etc.) do not change throughout the service life-cycle.

B. VM Placement Optimization for Clouds

Virtual machine placement in distributed environments has been studied in the context of cloud computing extensively, e.g., by Bobroff et al. [8] who present a management algorithm for dynamic placement of VMs to physical servers, which provides substantial improvement over static server consolidation in reducing the amount of required capacity and the rate of Service Level Agreement (SLA) violations. Their algorithm pro-actively adapts to demand changes and migrates VMs between physical hosts thus providing probabilistic SLA guarantees. Another SLA-driven dynamic VM placement optimization approach is proposed by Iqbal et al. [9], who describe the problem of bottleneck detection and resolution of multi-tier Web applications hosted on a cloud. They present a solution to minimize the probability that tiers (hosted on VMs) become bottlenecks by optimizing the placement of VMs.

For VM placement optimization in a single cloud, Andreolini et al. [10] present a management algorithm to re-allocate the placement of VMs for better performance and resource utilization by considering the load profile of hosts and the load trend behaviour of the guest instead of thresholds, instantaneous or average measures that are typically used in literature. VM placement optimization for multi-cloud scenarios is studied e.g., by Chaisiri et al. [7], Moreno-Vozmediano et al. [11] [12] and Tordsson et al. [13]. However, so far, most of efforts that target VM placement optimization for clouds have focused on either scenarios of one single cloud provider or static scenarios in multi-cloud environments. VM placement issues for dynamic scenarios across multiple cloud providers remain largely unexplored.

C. Virtual Machine Migration

Leveraging the ability of VM migration, cloud users are able to switch data and services between different physical machines in a cloud or even different clouds. In this paper, we consider a VM to be migrated if either it is moved from one cloud to another, or its hardware configuration is changed. VM migration is inevitable when reconstructing virtual infrastructure for cloud users in cloud brokerage scenarios.

Heterogeneous live migration of virtual machines is studied by Liu et al. [14]. Their work demonstrates that due to high variances of memory page dirtying rate, it is possible to get very slow migrations (result in long downtime) although a VM uses only 156MB of memory. Another comprehensive study of VM migration research, as well as an evaluation of methods for efficient live migration of VMs is presented by Svärd et al. [15], who also demonstrates how live migration of large VMs or VMs with heavy load can be done with shortened migration time, migration downtime, and reduced risk of service interruption.

While VM migration research has currently focused on single-cloud scenarios where data and services are located within the same cloud infrastructure, we expect that VM migration across different cloud providers will become a reality in a near future. In our work, the time and cost for VM migration are approximated by looking at the time required to shut down a VM in one cloud provider and start a new VM with the same configurations in another provider.

III. System Model and Problem Definition

A. Cloud Brokerage and Modeling

Figure 1 illustrates three roles in the studied cloud brokerage scenario: the *User*, the *Cloud Providers*, and the *Cloud Broker*. The user requests a virtual infrastructure by submitting a service description, which contains a manifest of required resources (e.g., number of VMs, size of storage, etc.), optimization criteria, and a set of constraints to the cloud broker.

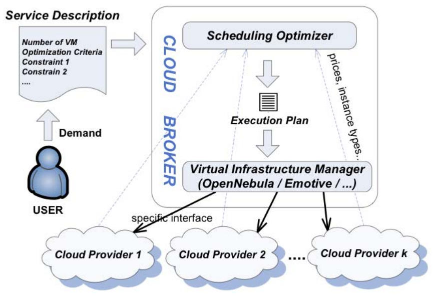

Fig. 1. Architecture overview for cloud brokerage scenario.

The *Scheduling Optimizer* component of the broker generates an *Execution Plan* based on requirement criteria provided by the user, the offerings of the available cloud providers, and the change of situation (e.g., service performance scales up or down, cloud providers' offers change and so forth). The *Execution Plan* includes either a list of VM templates that can equivalently represent the implementation of the user's abstract infrastructure request, or a description that represents an adjustment of an existing infrastructure. Finally, the Execution Plan is enacted by the *Virtual Infrastructure Manager* component that is built on a cloud management middleware such as Emotive [3] and OpenNebula [4]. We remark that in this paper we focus on problem formulations and modeling, while difficulties and challenges involving cloud interoperability, robustness of migration, and similar practical matters, although important for a full implementation, are out of scope.

Each cloud provider supports several VM configurations, often referred to as instance types. An instance type is defined in terms of hardware metrics such as main memory, CPU (number of cores and clock frequency), the available storage space, and price per hour. Our model has no limitations on the number of instance types. While we currently use five instance types, i.e., micro, small, medium, large, and xlarge (see Table I in Section IV) for the evaluation in this paper, it is straight forward to extend or decrease the number of instance types.

More formally, in a static scheduling scenario, our goal is to deploy n VMs, $v_1, v_2, \ldots, v_n$, across m available clouds, $c_1, c_2, \ldots, c_m$, which provide l possible instance types, $IT_1, IT_2, \ldots, IT_l$, to improve criteria such as cost, performance, or load balance. The hourly computational capability of a given instance type is denoted $C_j, 1 \leq j \leq l$. The hourly price for running a VM of instance type IT_j in cloud c_k is denoted by P_{jk}. One of the most common used objective function is to *maximize* the Total Infrastructure Capacity (TIC) of the deployed VMs given by:

$$TIC = H * \sum_{j=1}^{l} C_j (\sum_{i=1}^{n} \sum_{k=1}^{m} x_{ijk}), \tag{1}$$

where H specifies the expected lifetime of the infrastructure, i.e., how long the virtual infrastructure is to be deployed, x_{ijk} is a decision variable that satisfies $x_{ijk} = 1$ if v_i is of type IT_j and placed at cloud c_k, and 0 otherwise. Users can specify the following types of deployment restriction constraints:

- *Budget constraints* - Let $Budget$ denote the maximum investment that can be used. Now, the deployment is restricted to solutions where the total infrastructure price (TIP) satisfies

$$TIP = H * \sum_{j=1}^{l} \sum_{k=1}^{m} P_{jk} (\sum_{i=1}^{n} x_{ijk}) \leq Budget. \tag{2}$$

- *Placement constraints* - Each VM has to be of exactly one instance type and placed in exactly one cloud:

$$\forall i \in [1..n] : \sum_{j=1}^{l} \sum_{k=1}^{m} x_{ijk} = 1. \tag{3}$$

- *Load balancing constraints* - can be encoded as:

$$\forall k \in [1..m] : LOC_{min} \leq (\sum_{i=1}^{n} \sum_{j=1}^{l} x_{ijk})/n \leq LOC_{max}, \tag{4}$$

where LOC_{min} and LOC_{max} are the minimum and maximum percent of all VMs to be located in each cloud.

Note that additional constraints, such as for example network resource requirements, and data locality restrictions can also be added to the model.

As studied by Tordsson et al. [13], the static cloud scheduling problem on performance goals can be formulated as a linear programming model with objective function (1) and constraints (2), (3), and (4). In static scenarios, parameters $(n, l, m, P_{jk}(1 \leq j \leq l, 1 \leq k \leq m), and\ Budget)$ are constant throughout the service life-cycle where placement decisions can be taken off-line, once only, and in advance to service deployment.

B. Dynamic Cloud Scheduling

In dynamic scenarios, any of the previously discussed parameters may change. We identify two main categories of dynamic scenarios of cloud scheduling, which respectively reside in cloud providers and service providers: varying cloud providers offers, and service performance elasticity.

- Examples in the first category include varying offers:
  - A new provider appears or withdraws from the offer space. For example, Heroku [16] proclaimed the availability of the commercial version of its new cloud hosting and deployment service on 2009-04-24.
  - Price changes, e.g., in form of new agreements, spot prices, special discount during night time, etc.
  - New instance type offers are introduced, e.g., Amazon announced Micro Instances for EC2 on 2010-09-09 [17].
- Examples in the second category (service elasticity): In this case, the service owner wants to scale up or down the performance while optimizing the cost.
  - The service owner adjusts the number of VMs, e.g., removes a mail server from a current infrastructure.
  - The service owner increases or decreases the budget investment, e.g., budgetary reduction during recession.

In some scenarios, e.g., price reduction, the cloud user is offered an opportunity to obtain a better placement of VMs, while in other scenarios, e.g., an in-use cloud vendor withdraws from the market, the cloud user is forced to reconstruct the current infrastructure, striving to guarantee the service availability. Therefore, possible objectives can be identified as follows:

I. ***Maximize*** the possible new TIC with consideration of VM migration overhead under new situations.

II. ***Minimize*** the possible new TIP while obtaining a new TIC that can satisfy new performance demands.

III. ***Minimize*** the overhead of reconstruction a current configuration. The rationale behind this is service continuity. The more VMs the broker has to start and/or shut down, the more the service is impacted by the change.

To model dynamic scenarios, we introduce several notations:

- MC - denotes the overhead of changing the current placement to a new service layout. It can be defined in terms of system downtime, the number of VMs migrated, etc.
- β_i - denotes the service downtime penalty per time unit, which can be defined either in terms of capacity loss or monetary loss.

MC is given by:

$$MC = \sum_{i=1}^{n}(\beta_i * \Delta_i), \tag{5}$$

where Δ_i denotes the overhead of migrating VM v_i. For VM v_i, Δ_i depends on its previous instance type, its new instance type, the previous cloud it is placed in, and in which cloud it is about to be located. To calculate Δ_i, we introduce M, where $M_{i,j',j,k',k}$ denotes the overhead for migrating VM v_i of instance type $IT_{j'}$ in cloud $c_{k'}$ to cloud c_k with instance type IT_j. Let $x'_{ij'k'}$ denote the current placement status for VM v_i. Notably, here $x'_{ij'k'}$ is a constant that denotes the current placement status for VM v_i while x_{ijk} is a decision variable for the new model. Now we get:

$$\Delta_i = \sum_{j'=1}^{l}\sum_{j=1}^{l}\sum_{k'=1}^{m}\sum_{k=1}^{m}(x_{ijk} * x'_{ij'k'} * M_{i,j',j,k',k}). \tag{6}$$

We remark that both $x'_{ij'k'}$ and x_{ijk} are sparse 0-1 matrices that satisfy $\sum_{j=1}^{l}\sum_{k=1}^{m} x_{ijk} = 1$ and $\sum_{j'=1}^{l}\sum_{k'=1}^{m} x_{ij'k'} = 1$ for each $i, 1 \leq i \leq n$. Consequently, the expression for Δ_i is neat and fast to compute although the formulation in equation (6) is in the form of four-layer nested accumulated operation. Now, Objective III can be modelled and formulated using equations (5) and (6). Objective I can be expressed as *maximize* the TIC that is given by:

$$\begin{aligned} TIC &= H * \sum_{j=1}^{l} C_j(\sum_{i=1}^{n}\sum_{k=1}^{m} x_{ijk}) - MC \\ &= H * \sum_{j=1}^{l} C_j(\sum_{i=1}^{n}\sum_{k=1}^{m} x_{ijk}) - \sum_{i=1}^{n}(\beta_i * \Delta_i). \end{aligned} \tag{7}$$

Hence, Objective I is formulated using equations (6) and (7). We remark that the TIC can also be a constraint and TIP can be an objective function in the dynamic model. For example, a new model can be formulated as:

$$Minimize: \quad TIP = H * \sum_{j=1}^{l}\sum_{k=1}^{m} P_{jk}(\sum_{i=1}^{n} x_{ijk}). \tag{8}$$

$$Subject\ to:$$

$$TIC = H * \sum_{j=1}^{l} C_j(\sum_{i=1}^{n}\sum_{k=1}^{m} x_{ijk}) \geq Threshold \tag{9}$$

where the user wants to minimize the TIP while maintaining the TIC in a certain level. To conclude, three forms of the model are identified:

- Model I: ***maximize*** objective function (7), with equation (2), (3), and (4) as constraints. A cloud broker employs this model to obtain an optimal infrastructure capacity that also takes migration overhead into account.
- Model II: ***minimize*** objective function TIP (2), with equation (3) and (4) as constraints. The goal of this model is minimization of the price of the new infrastructure, while keeping the capacity above than a certain threshold.
- Model III: ***minimize*** objective function (5), with equation (2), (3), and (4) as constraints. This model is used when the cloud broker minimizes the migration overhead, and meanwhile fulfils the constraints for budget, placement, and load balancing.

C. Model Parametrization and Application

In our model, β_i signifies the weight of migrating VM v_i. We argue that the overhead for migrating different VMs differs, e.g., the overhead of migrating a backup server is lower than that of migrating a server running an ERP system.

By assigning suitable values to β_i ($1 \leq i \leq n$), and the matrix M, it is possible to express the migration overhead for various scenarios, e.g., a number of VMs to migrate metric can be concisely expressed as:

$$M_{i,j',j,k',k} = \begin{cases} 1 & \text{if } j' \neq j \text{ or } k' \neq k; \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

Infeasible migration can be modelled through ∞-assignment, e.g., assignment $M_{i,j',j,k',k} = \infty$ (or $\beta_i = \infty$) specifies that it is impossible to migrate VM v_i of instance type $IT_{j'}$ placed in cloud $c_{k'}$ to cloud c_k and of instance type IT_j. In practical applications, β_i and M can be estimated based on practical experience of the used cloud platforms and data collection to learn the behaviour of the migrated applications. To use the proposed approach, we sketch an overall algorithm as follows.

Algorithm 1 Cloud re-scheduling

```
Input:
    Environment changes, e.g., updated prices, VM numbers,
    budget, cloud provider configurations, etc.
Output:
    New placement after reconfiguration;
1: Select optimization criteria (including objective selection
   and constraints selection);
2: Determine parameter $\beta_i$;
3: Determine parameter values in matrix $M$;
4: Solve problem for criteria selected in step 1;
5: Migrate VMs if the solution is feasible;
```

This VM placement problem is known to be NP hard. Existing approximation and heuristic algorithms can scale to at most a few hundred machines, and may produce solutions that are far from optimal when system resources are tight [18]. An in-depth study of integer programming scalability is given by Alper et al. [19]. Instead of proposing new approximation algorithms, we encode our model using a mathematical modeling language and use state-of-the-art linear programming solvers. Optimizations for improving the scalability problem and complexity investigation are left to future works.

IV. Evaluation and Discussion

We evaluate our approach using imaginary service scenarios based on performance figures from contemporary clouds offerings. Notably, our goal is not to evaluate the various providers but rather to investigate how well our proposed cloud brokering approach can adapt to changes in realistic scenarios. Two commercial cloud providers are used to model in our experiments. The first one is GoGrid [20], and the second is Amazon EC2 [21]. EC2 offers two separate clouds, one is in the USA, the other in Europe. These three clouds are henceforth referred to as EC2-US, EC2-EU and GoGrid. To solve the optimization problem, we use AMPL [22] as the modeling language and Gurobi [23] as the binary integer programming problem solver.

A. Experimental Setting and Parameter Estimation

We consider five different VM instance types, their hardware characteristics and prices are listed in Table I.

TABLE I
Hardware metrics and prices for instance types.

Instance Type	micro	small	medium	large	xlarge
CPU (# cores)	1	1	1	2	4
CPU (GHz/core)	1	1	2	2	2
Memory (GB)	0.613	1.7	3.5	7.5	15
Storage (GB)	50	160	300	850	1700
Computing Capacity	1	2	4	8	16
Provider Instance type prices ($/h)					
EC2-US	0.02	0.1	N/A	0.4	0.8
EC2-EU	0.025	0.11	N/A	0.44	0.88
GOGRID	0.1	0.19	0.76	1.52	3.04

In the new instance type scenario (see Section IV-B) and the price change scenario (see Section IV-C), we set $LOC_{min} = 30\%$ and $H = 1hour$. These two scenarios are evaluated for one hour, and each cloud should host at least 30% of the VMs. To estimate parameter β_i and the values in matrix M, we use the service downtime statistics (see Table II) presented by Iosup et al. in [24] and [25] to calculate the computation capacity losses of the infrastructure.

TABLE II
Statistics for resource allocation/release time (seconds).

Instance Type		micro	small	medium	large	xlarge
EC2-US	Allocation	71	82	N/A	90	64
	Release	20	21	N/A	20	25
EC2-EU	Allocation	71	82	N/A	90	64
	Release	20	21	N/A	20	25
GOGRID	Allocation	260	540	1290	2300	3012
	Release	158	210	192	200	240

More specifically, $\beta_i = C_{j'}$, $M_{i,j',j,k',k} = Downtime\ of\ VM_i$, where *Downtime of* VM_i is the sum of Release Time of v_i of instance type $IT_{j'}$ placed in $c_{k'}$ and Allocation time of v_i of instance type IT_j placed in c_k. Notably, $M_{i,j',j,k',k}$ can be ignored if H is large enough.

In the following, three dynamic cloud scheduling scenarios are selected to evaluate our proposed model. In all experiments, the number of VMs (n) to be deployed is 32.

B. Scenario I: New instance type offers

In this case, we consider a service owner who has a limited budget, $5 per hour to run 32 VMs. At first, there are only four instance types available - small, medium, large and xlarge, and then we simulate the event that the micro instance type is introduced [17].

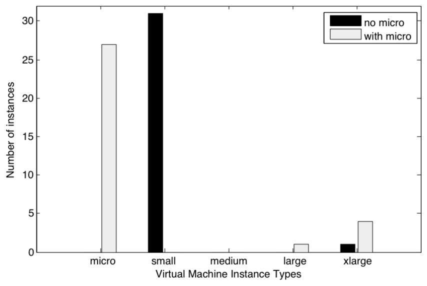

Fig. 2. VM placement with and without the micro instance type.

In our experiment, the user obtains an optimal total infrastructure capacity (TIC) of 78 by placing 31 VMs of small instance type and 1 VM of xlarge (at EC2-US) instance type. This situation changes when the micro instance type is announced. As Figure 2 illustrates, the virtual infrastructure is reconstructed accordingly. The number of micro instances is increased from 0 to 27, while the number of small instances is decreased from 31 to zero. Since the micro instance type is offered at a very low price (see Table I), the system now can improve the investment proportion for other instance types with larger computing capacity: the number of large instances is increased from 0 to 1, and the number xlarge instances is increased from 1 to 4. As a result, the TIC is increased by 22% to 95.2 with no need to increase the budget.

Figure 3 shows how the performance of the infrastructure changes in the first 800 seconds. In this figure, there are two obvious inflection points (encircled in the figure) which indicates the significant growth of capacity for the infrastructure with micro instances. The first inflection point is after 90 seconds before only one VM is running in the infrastructure of micro instance type; afterwards, the cloud broker completes migration processes for 11 VMs and restarts them. The second inflection point is after around 280 seconds, when 20 more VMs are rebooted after migrations. After 610 seconds, the performance of the infrastructure with micro instance surpasses the one without micro instances, and the difference expands increasingly as time elapses as illustrated in Figure 4. In this case, we can conclude that, it is worthy to perform migration if the infrastructure is to run for more than 10 minutes. This evaluation demonstrates that our cloud

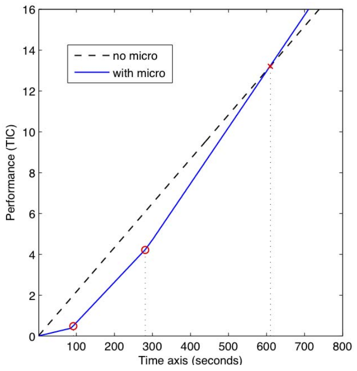

Fig. 3. Performance improvement with and without the micro instance type.

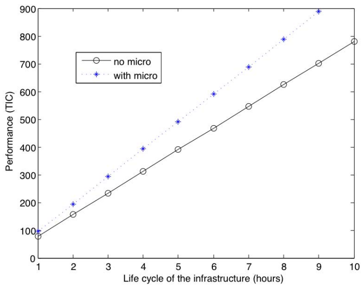

Fig. 4. Performance improvement with and without the micro instance type.

brokering mechanisms can handle the scenarios with new instance types. Interestingly, the proposed mechanisms can accurately determine the break-off point when the improved performance resulting from migration outweighs the migration overhead.

C. Scenario II: Prices change

In this second experiment, we first simulate an imaginary scenario where cloud providers offer a price discount of 20% during the night time due to less energy consumption. To study the effect of this, we increase the budget from $\$5$ per hour to $\$60$ per hour in 55 steps. We then calculate the TIC values under three different scenarios: static placement with old prices, static placement with new prices ignoring migration overhead, and dynamic placement with new prices and consideration of migration overhead.

We observe in Figure 5 that, for lower budgets, the performance improvement due to price discounts is more significant. The performance difference between two price offers (i.e., original prices and prices after discount) is notable, and despite the consideration of migration overhead, the new optimal TICs are very closed to the values in the static scenario,

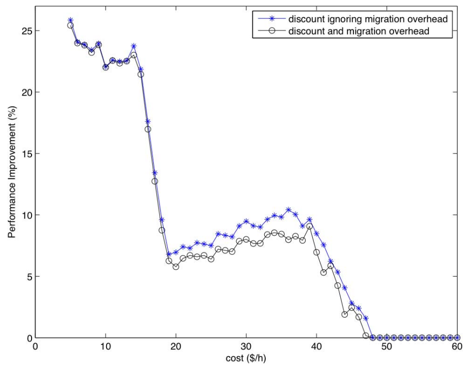

Fig. 5. VM placement with and without price discount.

especially when the budget is lower than $\$20$ per hour. However, when the budget is higher than $\$48$ per hour, there is no difference among the three scenarios. This is because the budget is excessive compared to the VMs to be deployed and the price offers, and hence, the broker does not migrate any VM even if the prices are lowered. To use all the budget, the broker may suggest the service owner to deploy more VMs (as discussed in the next scenario), so that the performance can be improved further.

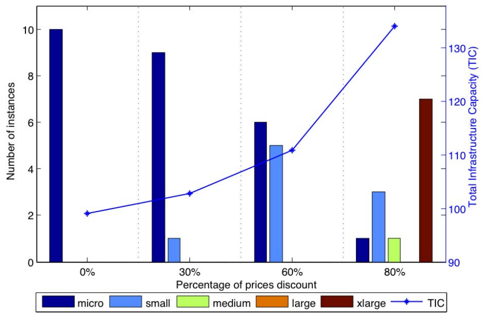

Fig. 6. VM placement with varying prices discount by GOGRID.

We also explore the behaviour of our model under the condition that only one of the cloud provider (i.e., GOGRID) offers price discounts. We set $Budget = \$5$ per hour. Due to the load balancing constraint (4), each cloud hosts at least 30% of the VMs (notably, $32 * 30\% \approx 10$) and thus at most 12 VMs. Since GOGRID is the most expensive cloud provider, to fulfil the minimum requirement for loading balancing, the cloud broker assigns only 10 VMs (of small instance types) to it, and obtains a $TIC = 99$ (see Figure 6).

As illustrated in Figure 6, the cloud broker manages to obtain higher TICs as the discount offered by GOGRID increases. The number of VMs hosted in GOGRID is increased from 10 to 12. The cloud broker first tries to increase the number of VMs of larger instance types, e.g., when the price discount is 30%, the number of small instances increases from 0 to 1, while the total number of VMs located in GOGRID does not change. When the discount is larger (i.e., $\geq 60\%$), the number of VMs of small instance types is scaled up to 5 and the total number of VMs located in GOGRID increases to 11.

Resources allocation for instance type medium, large and xlarge in GOGRID cloud is comparatively time-consuming (see Table II), and therefore the cloud broker does not assign any medium, large or xlarge instance in GOGRID even when the prices discount increases to 60%. However, 7 xlarge instances and 1 medium instance are employed when the discount comes to 80% which means that the cost for hosting more VMs or upgrading VMs with more computing power in GOGRID is inexpensive enough and the benefit from it suppresses the overhead arises from VM migrations.

In these experiments, we do not consider the overhead of re-migrating the infrastructure when the day time returns at the end of the discount period. One way of incorporating this could be to simply multiply MC by 2 (migration to and from new infrastructure), but this is a simplification as the previous infrastructure needs not be optimal, unless we know that we after the discount period will re-migrate the infrastructure to the original layout.

To summarize, this evaluation demonstrates that our cloud mechanism can cope with scenarios with changes in price. Performance change, as well as transformation of VM distribution across cloud providers evolved with prices change can be precisely calculated through the proposed approach.

D. Scenario III: Service performance elasticity

In this scenario, the service owner needs to increase the infrastructure capacity due to business growth. Before the expansion, \$5 is invested per hour, and the service owner obtains TICs of 115, 108, 102 and 99 per hour under load balancing (LB) constraints 0%, 10%, 20%, and 30% respectively. To fulfil the new business demands, the service owner needs to increase the budget so as to obtain a new TIC of 230 per hour. This goal can be done either through adding certain amount of new VMs without migrating any running VMs, or by migrating some running VMs and meanwhile adding some new VMs.

Figures 7, 8, 9 and 10 illustrate how the minimum budget and infrastructure reconfiguration overhead (IRO) evolve with the number of new VMs added for these two options. In this experiment, we define the IRO the sum of resource release time for VMs shut down weighted with VM size and resource allocation time for VM booted weighted with VM size, and it is given by:

$$IRO = \sum_{V_i\ is\ shut\ down} (RT_i * ComputingCapacity_i) + \sum_{V_i\ is\ booted} (AT_i * ComputingCapacity_i),$$

where RT denotes recourse release time of shutting down a VM, AT denotes resource allocation time of booting a VM, and the computing capacity of VM depends on its instance type, i.e., $ComputingCapacity_i = C_j$ if VM_i is placed with instance type j. IRO indicates the capacity loss when re-constructing an existing infrastructure. Notably, IRO is a dynamic form of MC mentioned in Section III-B, and it can also be expressed through assigning $\beta_i = 1$ and $M_{i,j',j,k',k}$ as follows:

$$M_{i,j',j,k',k} = RT_{j'k'} * C_{j'} + AT_{jk} * C_j \quad (11)$$

where values for RT and AT can be found in Table II, and $RT_{j'k'} = 0$ if a VM is newly added.

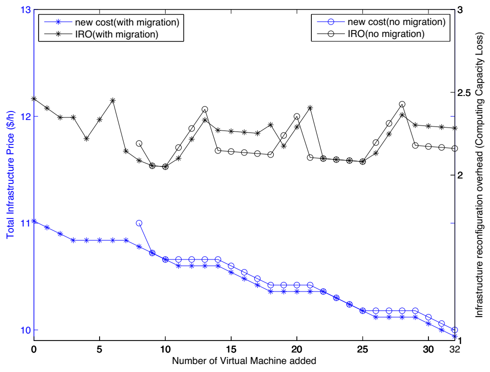

Fig. 7. Illustration of performance scale-up (LB constraint: 0%).

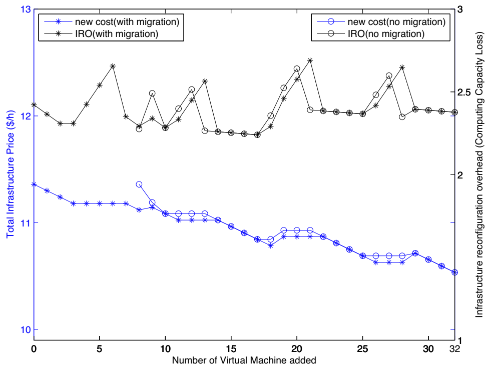

Fig. 8. Illustration of performance scale-up (LB constraint: 10%).

Figure 7 illustrates that, without load balancing constraints, the performance can be doubled to 230 per hour by replicating the number of VMs (i.e., adding 32 VMs) without any VM migration using twice the budget (\$10 per hour).

In cases where no migration is performed, it is not possible to achieve a solution until 8 (or 9, if LB constraint is 30%)

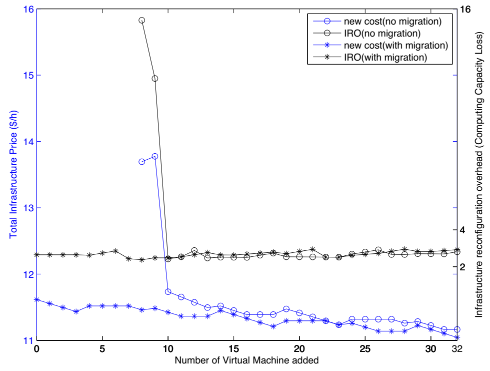

Fig. 9. Illustration of performance scale-up (LB constraint: 20%).

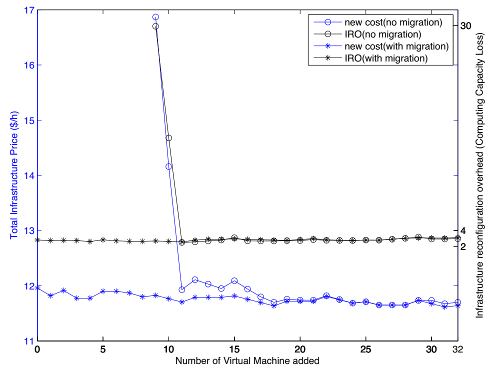

Fig. 10. Illustration of performance scale-up (LB constraint: 30%).

new VMs are added. Another interesting finding is that, in some cases, IROs with migration are higher than IROs without migration, whereas the opposite is true in other cases. The rationale behind this is the fact that, according to the statistics in Table II, it is possible that in some cases, the time for shutting down a VM and booting a new one is shorter than the time for only booting a new VM of some other type. For example, increasing the TIC (to be higher than 7) of an infrastructure with 1 VM of small instance type in EC2-US can be implemented by shutting down the small instance and booting an xlarge instance, which takes 85 seconds (21 seconds for shut-down, and 64 seconds for booting), or only starting a large instance using 90 seconds.

We can also observe from Figure 9 and Figure 10 that load balancing (LB) constraints impose a significant impact on infrastructure cost and IRO when migration is prohibited and few VMs (less then 11) are allowed to assign. Compared with Figure 7 and Figure 8, when the LB constraint is as 20%, to fulfil the minimum performance requirement, and meanwhile comply with the LB constraint, the broker has to place some VMs with large size in the least cost-efficient provider (i.e., GOGRID), which is harmful for the infrastructure cost and IRO. However, as the number of VMs that are added increases, the distances between solutions with migration and solutions without migration are narrowed down again, since the broker is able to place VMs of small size (instead of larger size) in GOGRID in order to comply with the LB constraint and performance constraint.

This experiment demonstrates the ability of the cloud brokering mechanism to handle the tradeoff between vertical (resizing VMs) and horizontal elasticity (adding VMs), as well as to improve decision making in complex scale-up scenarios with multiple options for service reconfiguration, e.g., to decide how many new VMs to deploy, and how many and which VMs to migrate.

Through the evaluations above, it is demonstrated that our model can support a wide range of dynamic scenarios, and by proper parametrizations, many interesting behaviours can be achieved. Finally, we point out that values in matrix M in real world applications are normally much higher than they are in Section IV-A. This is because VM migration across cloud providers located in different regions is a tedious task due to the fact that establishing a high-speed network tunnels to transfer VM images (that usually consist of Gigabytes of data) is time-consuming and costly.

V. Conclusions and Future Work

With the emergence of cloud computing as a paradigm, users can buy computing capacity from public cloud providers to minimize investment cost rather than purchasing physical servers. However, users are faced with the complexity of integrating various cloud services as the cloud computing market grows and the number of cloud providers increases. Despite the existence of a large number of efforts targeting cloud brokerage mechanisms, dynamic cloud scheduling issue remains largely unexplored. We present a linear integer programming model for dynamic cloud scheduling via migration of VMs across multiple clouds, which offers the flexibility of expressing different levels of migration overhead when restructuring an existing infrastructure. By proper parametrization, this model can be applied to handle changes both in infrastructure (new providers, prices, etc.) and services (elasticity in terms of sizes and/or number of VMs in a service). The proposed model is evaluated against commercial clouds offering settings, and it is demonstrated that our model is applicable in dynamic cloud scheduling cases aiming at cost-efficiency and performance-efficiency solutions.

Future directions for our work include investigation of mechanisms for model parametrization for dynamic cloud scheduling use cases, i.e., finding suitable values for parameters in the proposed model for different scenarios. Additionally, SLA violation compensation for users has not been taken into account in our model. Another interesting topic would be to apply our model to real world services.

Acknowledgement

We thank Rubén Montero, Rafael Moreno-Vozmediano, and Ignacio Llorente for their valuable contributions to this work, as well as for providing the foundation [13] of this paper. We are also grateful to the anonymous reviewers for their constructive comments. Financial support has in part been provided by the European Community's Seventh Framework Programme ([FP7/2010-2013]) under grant agreement no. 257115 (OPTIMIS [26]) and the Swedish Government's strategic effort eSSENCE.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009.

[2] "Gartner research: Cloud consumers need brokerages to unlock the potential of cloud services," http://www.gartner.com/it/page.jsp?id=1064712, visited Oct. 2011.

[3] EMOTIVE Cloud, http://www.emotivecloud.net, visited October 2011.

[4] OpenNebula, http://www.opennebula.org, visited October 2011.

[5] Libvirt, http://libvirt.org, visited October 2011.

[6] S. Grivas, T. Uttam Kumar, and H. Wache, "Cloud broker: Bringing intelligence into the cloud," in *Proceedings of IEEE 3rd International Conference on Cloud Computing (CLOUD 2010)*, pp. 544 –545.

[7] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Proceedings of the 4th IEEE Asia-Pacific Services Computing Conference*, pp. 103–110.

[8] N. Bobroff, A. Kochut, and K. A. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in *Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*, pp. 119 –128.

[9] W. Iqbal, M. N. Dailey, and D. Carrera, "SLA-driven dynamic resource management for multi-tier web applications in a cloud," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010)*, pp. 832 –837.

[10] M. Andreolini, S. Casolari, M. Colajanni, and M. Messori, "Dynamic load management of virtual machines in a cloud architectures," in *Proceedings of the 1st International Conf. on Cloud Computing*, 2009.

[11] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Multicloud deployment of computing clusters for loosely coupled mtc applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, pp. 924–930, 2011.

[12] ——, "Elastic management of web server clusters on distributed virtual infrastructures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 13, pp. 1474–1490, 2011.

[13] J. Tordsson, R. Montero, R. Moreno-Vozmediano, and I. 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.

[14] P. Liu, Z. Yang, X. Song, Y. Zhou, H. Chen, and B. Zang, "Heterogeneous live migration of virtual machines," in *In International Workshop on Virtualization Technology (IWVT08)*, 2008.

[15] 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*, ser. VEE '11. New York, NY, USA: ACM, 2011, pp. 111–120.

[16] Heroku, http://heroku.com/, visited October 2011.

[17] Amazon Announced Micro Instances for EC2, http://aws.amazon.com/about-aws/whats-new/2010/09/09/announcing-micro-instances-for-amazon-ec2, visited October 2011.

[18] 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*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 331–340.

[19] A. Atamtürk and M. Savelsbergh, "Integer-programming software systems," *Annals of Operations Research*, vol. 140, pp. 67–124, 2005.

[20] GoGrid, http://www.gogrid.com, visited October 2011.

[21] Amazon EC2, http://aws.amazon.com/ec2, visited October 2011.

[22] R. Fourer, D. M. Gay, and B. W. Kernighan, "Ampl: A modeling language for mathematical programming," *Management Science*, vol. 36, pp. 519–554, May 1990, http://www.ampl.com.

[23] Gurobi Optimization, http://www.gurobi.com, visited October 2011.

[24] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, pp. 931–945, 2011.

[25] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of EC2 cloud computing services for scientific computing," in *Proceedings of the First International Conference on Cloud Computing*. Springer, 2010, pp. 115–131.

[26] A. J. Ferrer, F. Hernádez, 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.

II

Paper II

Virtual Machine Placement for Predictable and Time-Constrained Peak Loads*

Wubin Li, Johan Tordsson, and Erik Elmroth

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

Abstract: We present an approach to optimal virtual machine placement within datacenters for predicable and time-constrained load peaks. A method for optimal load balancing is developed, based on binary integer programming. For tradeoffs between quality of solution and computation time, we also introduce methods to pre-process the optimization problem before solving it. Upper bound based optimizations are used to reduce the time required to compute a final solution, enabling larger problems to be solved. For further scalability, we also present three approximation algorithms, based on heuristics and/or greedy formulations. The proposed algorithms are evaluated through simulations based on synthetic data sets. The evaluation suggests that our algorithms are feasible, and that these can be combined to achieve desired tradeoffs between quality of solution and execution time.

Key words: Cloud Computing; Virtual Machine Placement; Binary Integer Programming; Off-line Scheduling; Load Balancing;

* By permission of Springer Verlag.

Virtual Machine Placement for Predictable and Time-Constrained Peak Loads

Wubin Li, Johan Tordsson, and Erik Elmroth

Department of Computing Science and HPC2N,
Umeå University, SE-901 87 Umeå, Sweden
{wubin.li,tordsson,elmroth}@cs.umu.se
http://www.cloudresearch.se

Abstract. We present an approach to optimal virtual machine placement within datacenters for predicable and time-constrained load peaks. A method for optimal load balancing is developed, based on binary integer programming. For tradeoffs between quality of solution and computation time, we also introduce methods to pre-process the optimization problem before solving it. Upper bound based optimizations are used to reduce the time required to compute a final solution, enabling larger problems to be solved. For further scalability, we also present three approximation algorithms, based on heuristics and/or greedy formulations. The proposed algorithms are evaluated through simulations based on synthetic data sets. The evaluation suggests that our algorithms are feasible, and that these can be combined to achieve desired tradeoffs between quality of solution and execution time.

Keywords: Cloud Computing, Virtual Machine Placement, Binary Integer Programming, Off-line Scheduling, Load Balancing.

1 Introduction

Building on technologies such as distributed systems, autonomic computing, and virtualization, cloud computing emerges as a promising computing paradigm for providing configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [13]. A key feature of future cloud infrastructures is elasticity [2], i.e., the ability of the cloud to automatically and rapidly scale up or down the resources allocated to a service according to the workload demand while enforcing the Service Level Agreements (SLAs) specified.

In this paper, we focus on elasticity scenarios where workloads are predictable and to be deployed and scaled-out quickly through the rapid provisioning of Virtual Machines (VMs). Predictable workload scenarios are frequently occurring, e.g., online banking has regular peaks once a month, streaming video is consumed mostly during evenings, and video gaming workloads exhibit predictable daily and weekly changes [6], etc. Both the service and the cloud infrastructure can benefit from the predictability of the workloads, since placement schemes for VMs are possible to be pre-calculated and resources can be set up in advance.

To fulfil the service demand, the cloud infrastructure usually produces VM placement solutions improving criteria such as cost, performance, resource utilization, etc. However, from a cloud infrastructure perspective, physical machines usually have non-uniform capacities. Their respective utilizations may have high variances. Users of a service may suffer from high latency due to high utilizations of some physical servers. In other words, certain types of applications could benefit from keeping the utilization of individual machines as close as possible to the utilization of the entire system [15]. To tackle this problem, the worst case of individual physical machine utilization should be minimized and load balancing in the whole system should thus be optimized.

The VM placement problem can be generally formulated as a variant of the class constrained multiple-knapsack problem that is known to be NP hard [14]. Existing approximation algorithms can scale to at most a few hundred machines, and may produce placement solutions that are far from optimal when system resources are scarce [15]. In this paper, we focus on properties of the load balancing problem itself instead of proposing new generic approximation algorithms. We analyse how the studied problem differs from general VM placement problems, and present a linear programming formulation of the optimization problem along with some approximations. An evaluation based on synthetic workloads is used to investigate the feasibility of the algorithms.

The remainder of the paper is organized as follows. Section 2 briefly describes the problem and motivates our work. Section 3 presents the problem formulation, defines an optimal algorithm, as well as describes three problem-specific approximations. Section 4 presents an evaluation of our approach. Section 5 discusses related work. Finally, conclusions and future work are given in Section 6.

2 Problem Description

The studied scenario is illustrated in Figure 1. A set of physical machines with diverse capacities are used to execute VMs of different sizes. The VMs are grouped by VM sets, i.e., prepared bundles of, e.g., application servers, front ends, and data base replicas for managing peak loads of certain applications. These VM sets are to be deployed across the physical machines, i.e., PM_1, PM_2, ..., PM_m, which may have different background loads and non-uniform capacities. Each VM set is comprised of multiple VMs with various capacity requirements. The durations and sizes of VMs are known in advance. This life cycles of VM sets may be different, e.g., some may be provisioned longer than others, some may start to run earlier than others, etc.

The most significant aspect that could distinguish the VM placement for predictable peak loads from general placement problems is that the peak loads are time-constrained. After a certain period, the additional VMs are removed from the cloud infrastructure. During this period, multiple placement requests

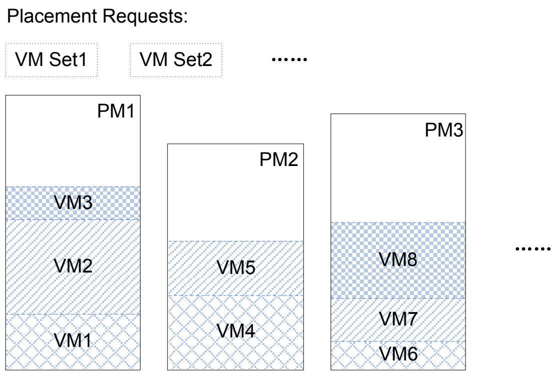

Fig. 1. Studied scenario illustration

may start or terminate. In this paper, the placement objective is load balancing, i.e., to minimize the highest utilization of any individual physical machine during this period.

3 Problem Analysis and Formulation

We use a quadruple $r =< id, s, e, VMSet >$ to uniquely identify a placement request, where s indicates when the request starts and e specifies the end-time. A placement request set can thus be represented by an array of quadruples temporally ordered by s. The *VMSet* is a collection of VMs, each of which may have different computation capacities.

Table 1. Hardware metrics for instance types

Instance Type	micro	small	medium	large	xlarge	...
CPU (# cores)	1	1	1	2	4	
CPU (GHz/core)	1	1	2	2	2	
Memory (GB)	0.613	1.7	3.5	7.5	15	...
Storage (GB)	50	160	300	850	1700	
Computing Capacity	**1**	**2**	**4**	**8**	**16**	**...**

To distinguish VMs with different computation capacities, we use the hardware discretization approach, used e.g., by Amazon EC2 as shown in Table 1. An example of placement request is <23, 2011-05-30 18:30, 2011-06-02 12:00, {4, 2, 1, 4, 16}>. This request has id 23, starts at 2011-05-30 18:30, ends at 2011-06-02 12:00 and demands 5 VMs with capacities 4, 2, 1, 4, and 16 respectively. All VMs in a request are to start and terminate at the same time.

Table 2. Symbols used in this paper

H	Time period that includes placement requests.
N	Number of placement requests.
N_i	Size of the VMSet in the ith request.
C_{ij}	Capacity requirement of the jth VM in the ith request.
M	Number of physical machines.
w_k	Existing load of the kth physical machine.
W_k	Total capacity of the kth physical machine.
x_{ijk}	The placement decision variable. $x_{ijk} = 1$ iff the jth VM in request i is placed on physical machine k, and 0 otherwise.
G	Number of overlap sets generated.
y_{ig}	$y_{ig} = 1$ if the ith placement request is in the gth overlap set, and 0 otherwise.

Table 2 contains an overview of the symbols used to formulate the load minimization placement problem. Now, for a given VM set in a request set R and a set of M physical machines, the highest utilization of any individual physical machine can be described by

$$Load(R) = \underset{k \in [1..M]}{\text{Max}} \frac{\sum_{i=1}^{N} \sum_{j=1}^{N_i} (x_{ijk} * C_{ij}) + \omega_k}{W_k}, \tag{1}$$

where x_{ijk} is the decision variables for placement, C_{ij} the VM capacity, and w_k and W_k the existing load and total capacity of the physical machines. For any allocation of VMs to physical machines, the following constraints apply:

$$\forall i \in [1..N], j \in [1..N_i] :$$
$$\sum_{k=1}^{M} x_{ijk} = 1 \tag{2}$$
$$\forall k \in [1..M]$$
$$\sum_{i=1}^{N} \sum_{j=1}^{N_i} (x_{ijk} * C_{ij}) + \omega_k \leq W_k. \tag{3}$$

Constraint (2) specifies that each VM in every placement request has to be assigned to exactly one physical machine, and constraint (3) describes how the total capacity of each physical machine cannot be exceeded.

There are multiple possible approaches to the placement request allocation problem for load minimization. Our first and simplest algorithm is a greedy formulation that for each VM in each VM set (in order by request start time) finds the placement that keeps the average load at a minimum. This is done by finding the physical machines that provide the worst-fit for each VM, i.e., leaves the maximum residual capacity. Of course, before placing a certain request, previous requests that have terminated can be excluded and the physical machines reused. This algorithm, *Greedy Worst-Fit*, is defined in Algorithm 1.

Algorithm 1. Greedy Worst-Fit(R)

```
   Input: Placement request set $R = \{r_1, r_2, \ldots, r_n\}$.
   Output: Placement Scheme for $R$.
1  Sort all the requests by start-time $s$;
2  for $1 \leq i \leq n$ do
3      for $1 \leq j < i$ do
4          if $r_j$ is expired but still being provisioned then
5              exclude $r_j$ and release capacities of the physical machines that host
               the VMs of $r_j$;
6          end
7      end
8      foreach $vm$ in $VMSet_i$ do
9          $pm_k \leftarrow$ the least loaded physical machine with highest residual capacity;
10         if $vm$ can fit in $pm_k$ then
11             assign $vm$ to $pm_k$;
12         end
13         else
14             no feasible solution;
15             return;
16         end
17     end
18 end
```

Although the greedy formulation is fast to compute, it does not provide an optimal solution to the VM placement problem (with respect to load balancing), as VM placement is a version of the general assignment problem [7]. Our second algorithm operates in a similar manner to the Greedy Worst-Fit one in that it considers the placement requests sequentially in order of start time. However, instead of performing a greedy allocation, the second algorithm finds, for each point in time when a VM set is about to start, the allocation of all running VM sets, including the new one, that minimizes the average utilization. This method, (*Sequential*) is described in more detail in Algorithm 2 and the mathematical expressions for load minimization is given by equations (1), (2), and (3). Note that, in Algorithm 2, the minimization in each iteration (see line 8) treats only the active request sets.

As the complete set of VM requests are known in advance, we can, at the expense of additional complexity, solve the load balancing optimization problem not only for the currently running VMs, but for all VMs. This algorithm is a knapsack formulation, and is defined in Algorithm 3 (*Knapsack*).

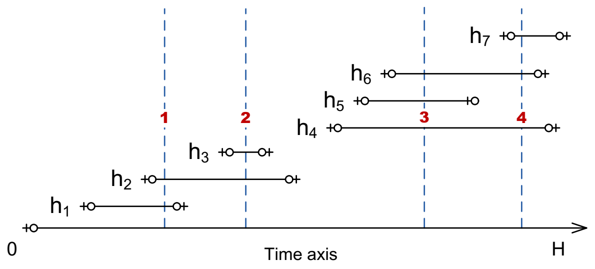

Fig. 2. Illustration for coexistence of placement requests

One key observation is that two VM sets may use the exact same physical resources if they do not overlap in runtime. More formally, two placement requests are coexistent if and only if their lifetimes overlap, i.e., placement request r_1 and r_2 are coexistent if and only if $s_2 \leq s_1 < e_2$ or $s_1 \leq s_2 < e_1$. Figure 2 shows an illustration of coexistence. In this figure, there are 7 placement requests whose start-times are h_i $(1 \leq i \leq 7)$ for request r_i, respectively. For a given placement request set R, we introduce the notion of *OverlapSets* to define a subset of R where any two requests in the subset are coexistent. Furthermore, there exists no request in R that is not in *OverlapSet* that is coexistent with every request in *OverlapSet*. For the example in Figure 2, we get $OverlapSets = \{\{r_1, r_2\}, \{r_2, r_3\}, \{r_4, r_5, r_6\}, \{r_4, r_6, r_7\}\}$.

In principle, to calculate the highest utilization of any individual physical machine during the whole period H, we must generate all *OverlapSets*, and compute the maximum load of machines in each *OverlapSet* after placing all VMs that run in that set. From the definition of the overlap sets, a straightforward recursive algorithm to generate the sets can be derived. However, this recursion results in an exponential runtime complexity. It is thus a very time-consuming task to complete generating all *OverlapSet*s when the number of placement requests is large. For example, in our experiments, the time required to generate all *OverlapSet*s varied from 0.01 second to 45 minutes.

Algorithm 2. *SequentialPlacement*(R)

```
  Input: Placement request set $R = \{r_1, r_2, \dots, r_n\}$.
  Output: Placement Scheme for $R$.
1 Sort all the requests by start-time $s$;
2 for $1 \leq i \leq n$ do
3     for $1 \leq j < i$ do
4         if $r_j$ is expired but still being provisioned then
5             exclude $r_j$ and release capacities of the physical machines that host
              the VMs of $r_j$;
6         end
7     end
8     Minimize (1) with (2) and (3) as constraints;
9 end
```

Algorithm 3. *Knapsack*(R)

```
Input: Placement request set $R = \{r_1, r_2, \ldots, r_n\}$.
Output: Placement Scheme for $R$.
1 Minimize (1) with (2) and (3) as constraints;
```

We instead use an approximation based on discrete time slots: The time from the earliest request start-time to the latest end-time is divided into T time slots. Every time slot is examined and placement requests in this slot are collected and considered as a potential element of *OverlapSet*s (see line 7 in Algorithm (4)). If a potential element is not a subset of some element in *OverlapSet*s, it is finally added to *OverlapSet*s after its subsets (if non-empty) are removed from *OverlapSet*s (lines $9-15$ in Algorithm (4)). Obviously, the quality of solution generated by this algorithm depends on T. If T is large enough, the solution is close to the one generated by the exact recursive method. Since the time complexity of Algorithm (4) is polynomial ($\Theta(T * n)$), it is much faster than the recursive formulation even when T and n are large. Through experiments, we note that it takes around 2 seconds to complete the generation process when $T = 10000$, $H = 24$ hours, and $n = 1000$, whereas with the recursive method, this problem size would take a day or more.

Algorithm 4. *GenerateOverlapSets*(R, T)

```
Input: Placement request set $R = \{r_1, r_2, \ldots, r_n\}$, the number of time slots $T$.
Output: The OverlapSets of $R$.
1  $OverlapSets \leftarrow \{\}$;
2  Sort all the requests by start-time $s$;
3  $S = \underset{i \in [1..n]}{Min}\{s_i\}, E = \underset{i \in [1..n]}{Max}\{e_i\}$, $interval = (E - S)/T$;
4  for $1 \leq i \leq T$ do
5      $ts \leftarrow S + (i - 1) * interval$;
6      $te \leftarrow S + i * interval$;
7      $currentSet = \{r \in R \mid r$ starts in $[ts, te]\}$;
8      $should\_add \leftarrow true$;
9      foreach $P$ in $OverlapSets$ do
10         if $P \subset currentSet$ then
11             $OverlapSets \leftarrow OverlapSets \setminus P$;
12         end
13         if $currentSet \subseteq P$ then
14             $should\_add \leftarrow false$;
15         end
16     end
17     if $should\_add$ then
18         $OverlapSets \leftarrow OverlapSets \cup currentSet$;
19     end
20 end
```

Incorporating the concept of overlap sets, our knapsack algorithm can now be reformulated as:

$$\text{Minimize} \quad \{ \underset{R' \in GenerateOverlapSets(R)}{\text{Maximize}} \ \text{Load}(R') \ \}$$

Subject to

$$\forall i \in [1..N], j \in [1..N_i]:$$

$$\sum_{k=1}^{M} x_{ijk} = 1 \tag{4}$$

$$\forall k \in [1..M], g \in [1..G]:$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N_i}(x_{ijk} * C_{ij} * y_{ig}) + \omega_k \leq W_k. \tag{5}$$

Here, y_{ig} is a decision variable for coexistence used to determine if two VMs can use the same physical resources i.e., if they do not overlap in time. Constraint (4) is the same as constraint (2) and specifies that each VM in every placement request has to be placed in exactly one physical machine. Constraint (5) is the capacity constraint for each physical machine, with the coexistence as an additional feature. This is a Min-Max optimization problem, which is non-linear. To transform this problem to a linear programming problem, we add μ to the list of unrestricted variables subject to the constraints

$$\forall R' \in GenerateOverlapSets(R): \quad \text{Load}(R') \leq \mu \tag{6}$$

and try to minimize μ.

Two steps are required to solve the problem: generation of *OverlapSet*s from placement requests, and solving the model using the *OverlapSet*s as inputs. In principle, the solver must enumerate each possible placement scheme, check whether it is viable, and compare the μ to the minimum found so far. There are multiple potential optimizations to reduce the computation cost for generating *OverlapSet*s and solving this model. To reduce the search space, we can significantly improve the performance of the solver by identifying upper bounds that are easy to compute. Since Greedy Worst-Fit is polynomial and fast to complete, we use the approximated load calculated through Greedy Worst-Fit as an upper bound as shown in Equation (7):

$$\mu \leq \gamma, \tag{7}$$

where γ is the highest utilization of any individual physical machine as calculated by Greedy Worst-Fit algorithm. This optimization tends to reduce the time required to compute a solution drastically, thus improving scalability. We refer to this approach that combines upper bound optimizations and overlap sets as *Time-bound Knapsack*, as described in Algorithm 5. In this algorithm, Line 1 calculates the approximative placement using Greedy Worst-Fit algorithm. Lines 2-12 determine the upper bound value for the approximative placement, by finding the highest load for any physical machine that follows the greedy placement scheme. Line 13 generates the overlap sets, and Line 14 minimizes the maximum load.

Algorithm 5. Time-bound Knapsack(R, T)

```
   Input: Placement request set $R = \{r_1, r_2, \ldots, r_n\}$, the number of time slots $T$.
   Output: Placement Scheme for $R$.
1  Execute Greedy Worst-Fit algorithm to initialize variables $x_{ijk}$;
2  $\gamma \leftarrow 0$;
3  for $1 \leq i \leq n$ do
4      for $1 \leq j < i$ do
5          if $r_j$ is expired but still being provisioned then
6              exclude $r_j$ and release capacities of the physical machines that host
               the VMs of $r_j$;
7          end
8      end
9      if $\gamma < Load(r_i)$ then
10         $\gamma \leftarrow Load(r_i)$;
11     end
12 end
13 $OverlapSets \leftarrow GenerateOverlapSets(R, T)$;
14 Minimize $\mu$ with (4), (5), (6) and (7) as constraints;
```

4 Evaluation and Discussion

In this section, the four proposed algorithms are studied from three perspectives: how good they are at finding solutions to the placement problems, the quality of the found solutions, and the computational complexity. The experimental setup is a scenario with a cloud provider with 100 physical machines and 32 placement requests, each with between 1 and 8 VMs (uniformly distributed). As outlined in Table 3, VM capacity is uniformly distributed between micro (computing capacity 1) and xxlarge (computing capacity 32). The background load for each physical machine is uniformly distributed between 20% and 50%. The placement problems are encoded using the AMPL [9] modelling language and solved with the Gurobi [1] solver. All experiments are performed on a workstation with a 2.67 GHz quad-core CPU and 4 GB of memory.

To evaluate the performance of our approach with respect to quality of solution, we first perform 1000 experiments with groups of placement requests. We specify a one minute execution time limit for all algorithms. Even for very short term peak loads, e.g., hourly spikes, this one minute limit should be short enough to calculate a placement solution and configure the system accordingly.

Table 4 summarizes the 1000 experiments. We note that Sequential is able to solve most problems (994), followed by Time-bound Knapsack (923), Greedy Worst-Fit (870), and Knapsack trailing with 732 successfully solved problems. Looking closer at the unsuccessfully solved problems, we note that Time-bound

Table 3. Experiment Setup

H (experiment duration)	48 hours
Number of physical machines	100
Existing load for each physical machine	Uniform(20%, 50%)
Capacity for each physical machine	$2^{\text{Uniform}(0,7)}$
Number of placement requests	32
Number of VMs in a request	Uniform(1, 8)
VM capacity demands	$2^{\text{Uniform}(0,5)}$
Life-cycles of placement requests	Uniform(1,H)

Table 4. 1000 groups experiment with 1 minute execution time limitation

Algorithms	Feasible Solutions	No Solution	Time-out
Time-bound Knapsack	923	0	77
Knapsack	732	30	238
Sequential	994	4	2
Greedy Worst-Fit	870	130	0

Knapsack encounters no infeasible placements, whereas this happens 4 times for Sequential, 30 for Knapsack and 130 for Greedy. Considering the problem instances that could not be solved within feasible time (here selected as one minute), we note that Greedy always completes within this time, but Sequential fails in 2 cases, Time-bound Knapsack in 77, and Knapsack in 238 cases. When combining the two reasons for failing to solve the placement problems, Time-bound Knapsack and Sequential appear to be the most promising approaches.

Looking further into quality of solutions, we exclude, for each algorithm, the experiments that could not be solved successfully (or within a minute). The left part of Figure 3 shows the average load balance (i.e., the maximum load for any machine during the experiments), including Standard Deviation (SD), for the successfully solved instances for each algorithm. Here we note that Time-bound Knapsack result in the best load balance, 71.9% ± 6.1%, whereas the three other algorithms all result in loads above 80%, with Sequential the second best at 80.5% ± 7.6%. The right part of Figure 3 shows the average execution time, including deviation, for the successfully solved problems. As expected, the polynomial Greedy algorithm is the fastest with average execution time less than 0.5 seconds, as compared to 8 seconds for Time-bound Knapsack, 11 seconds for Sequential, and 13 seconds for Knapsack. For the last three algorithms, there are large deviations in execution time for successfully solved problems, also after excluding the experiments that failed to due exceeding the one minute threshold.

To understand the behaviour of the algorithms more in-depth, we focus on how the maximum load of any physical machine (the load balance) varies over the 48 hours experiment duration for one of the 1000 experiments. As illustrated in Figure 4, the Greedy algorithm results in volatile loads with large deviations over time, whereas and Sequential is more stable but still experiences fluctuations. In contrast, both Knapsack and Time-bound Knapsack are very stable, and keep the

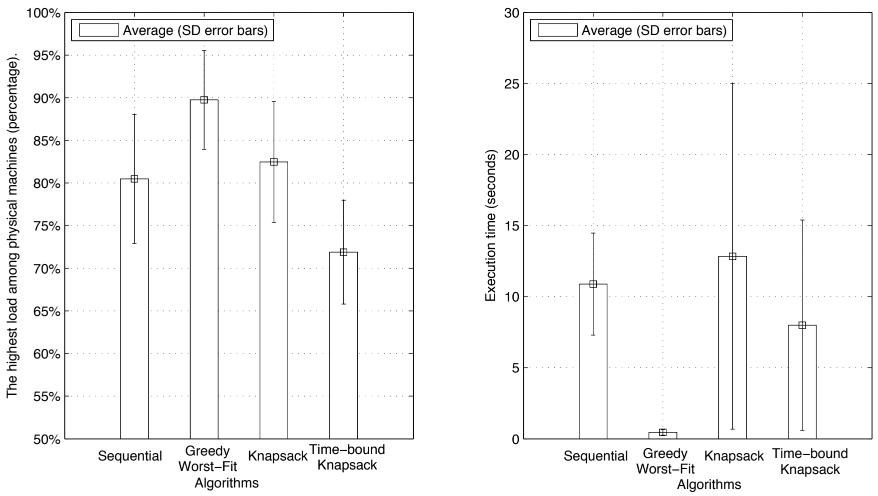

Fig. 3. Performance and execution time comparison for 1000 tests

maximum load constant for almost the full duration of the experiment. The low load very early and very late for all algorithms is due to there being few running VM sets at these points in time. Figure 4 also gives some insight into how often the algorithms cannot find feasible solutions. For complicated problems with many VMs, Greedy, Knapsack, and sometimes also Sequential may fail due to capacity constraints of the physical machines, whereas Time-bound Knapsack is more likely to find a solution.

To study the computational complexity (execution times) of the algorithms further, we perform a second experiment with 100 groups of placement requests where the execution time was unlimited. Here, we focus on the experiments where the placement took longer than one minute to solve. Table 5 presents the number of failures (experiments that ran for more than one minute) and their execution time deviations in the evaluated 100 tests. Here, we observe that Knapsack exceeds the time limit in 20% of all tests, Time-bound Knapsack in 4% of the tests, and Sequential in a single test, whereas Greedy always completes well within one minute. Looking at the average execution times for these tests, we note that Sequential requires 2.6 minutes, Time-bound Knapsack 95 ± 129 minutes, and Knapsack 346 ± 788 minutes, i.e., there are a few cases where the latter two algorithms required several hours to complete. A comparison of the required execution time and the percentage of problems successfully solved is shown in Figure 5. This figure illustrates that although the Knapsack and Time-bound Knapsack algorithms in a few specific cases can be very slow, they most often generate solutions within a few seconds, and allowing these to execute a couple of minutes improves the probability of finding a solution substantially.

To summarise these experiments, the Time-bound Knapsack algorithm generates the best solutions, i.e., finds the placement with the lowest average load, and is also able to find valid placements in complicated cases where the other algorithms fail. However, it can at times be very slow to execute. Conversely, the

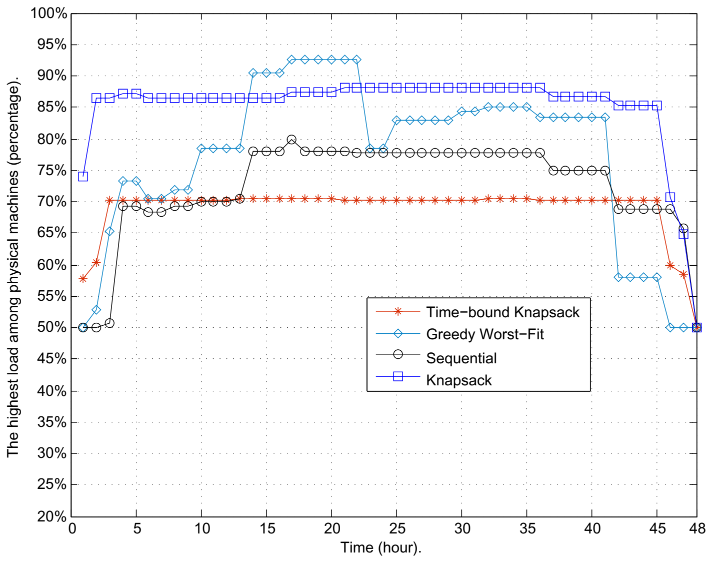

Fig. 4. Illustration of highest loads of the infrastructure evolve with time

Table 5. Number of failures (slow executions) and execution times for 100 tests

Algorithms	Failure	Failure execution time (minutes)
Greedy Worst-Fit	0	
Sequential	1	2.6 ± 0
Time-bound Knapsack	4	94.7 ± 128.6
Knapsack	20	345.5 ± 787.5

Greedy algorithm is very fast to compute and should scale well also for larger problem sizes due to its polynomial complexity. However, it generates placements with worse load balance and fails to find feasible solutions in some high workload scenarios. In comparison with these two algorithms, Knapsack performs worse in overall. Notably, Sequential can be a suitable compromise between quality of solution and execution time, although it does not excel in either.

5 Related Work

Virtual machine placement across physical servers has recently gained a lot of attraction. Our previous contributions within this area include integer programming methods to obtain optimal cost-performance tradeoffs in deploying VMs across multiple clouds [17] and methods to dynamically reschedule VMs (including modeling of VM migration overhead) upon changed conditions [12].

Other contributions to VM placement include a binary integer program formulation for cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads is proposed by den Bossche et al. [4]. It is demonstrated that this

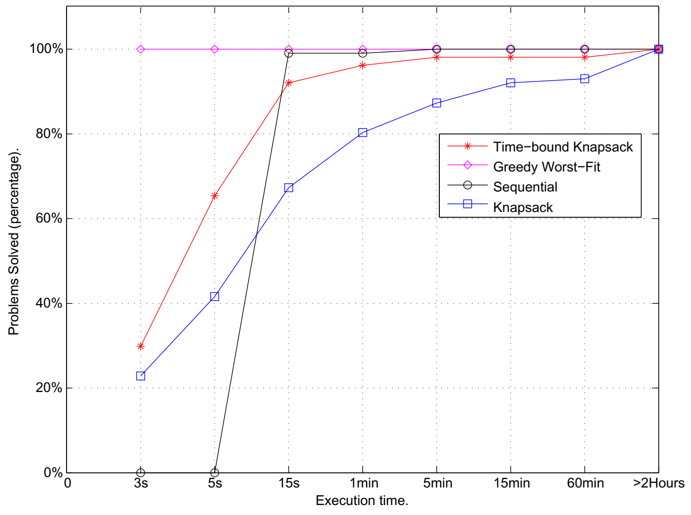

Fig. 5. Execution time vs. percentage of problems solved for 100 tests

approach results in a tractable solution for scheduling applications in a public cloud, but that the same method becomes much less feasible in a hybrid cloud setting due to sometimes having long solving time. Compared to our work, their approach also considers the life-cycles of workloads, but mainly focuses on cost-effective scheduling of applications in a hybrid cloud setting. Load balancing issues are not considered.

Bobroff et al. present a dynamic server migration and consolidation algorithm to minimize the number of working physical machines without violating SLAs [3]. This work takes only CPU demands into account and uses classification of workload signatures to identify the servers that benefit most from dynamic migration. Using adaptable forecasting techniques well suited for the classification, substantial improvement over static VM placement is shown, reducing the amount of required capacity and the rate of SLA violations.

A scalable application placement controller for enterprise data centres is proposed by Tang et al. [15]. The objective of this controller is to maximize the total satisfied application demand, to minimize the number of application starts and stops, and to balance the load across machines. Compared to existing state-of-the-art algorithms, this controller can produce within 30 seconds high-quality solutions for hard placement problems with thousands of machines and thousands of applications. This work is later extended to a binary search based framework striving to limit the worst case of each individual server's utilization by Tian et al. [16]. The system cost, defined as the weighted combination of both placement change and inter-application communication cost, can be also maintained at a low level. However, life-cycles of workloads remain unexplored.

Breitgand et al. [5] propose a multiple-choice multidimensional knapsack problem formulation for policy-driven service placement optimization in federated clouds, and a 2-approximation algorithm based on a greedy rounding of a linear relaxation of the problem. The proposed placement algorithms aims at maximizing provider profit while protecting Quality of Service (QoS) as specified in SLAs of the workloads, and can be used to optimize power saving or load balancing internally in a cloud, as well as to minimize the cost for outsourcing workloads to external cloud providers. Breitgand et al. encode load balancing as the standard deviation of the residual capacity, which is a non-linear function. A binary search-based heuristic is used to minimize that function, and thus an optimal solution is not guaranteed.

6 Conclusions and Future Work

We study the VM placement problem for load balancing of predictable and time-constrained peak workloads. We formulate the problem as a Min-Max optimization one and present an algorithm based on binary integer programming, along with three approximations for tradeoffs in scalability and performance. An experimental study compares the proposed methods with respect to ratio of problems successfully solved, quality of solution, and computational complexity.

Future directions for our work include studies of other load balancing metrics, e.g., looking at how to minimize the average load over time instead of the maximum load. Another topic is how to refine the models and replace the one-dimensional computing capacity performance metric, e.g., with CPU, memory, disk, etc. as suggested by Breitgand et al. [5] and to incorporate inter-VM resources such as network bandwidth, as demonstrated by Lampe et al. [11]. Additionally, one interesting feature to consider in optimization is the grouping of VMs to hosts based on the interference and overhead that one VM causes on the other concurrently running VMs on the same physical host, as discussed by Kousiouris et al. [10].

Acknowledgments. We are grateful to the anonymous reviewers for their constructive and valuable feedback, improving the quality of this work. Financial support has in part been provided by the European Community's Seventh Framework Programme ([FP7/2010-2013]) under grant agreement no. 257115 (OPTIMIS [8]) and the Swedish Government's strategic effort eSSENCE.

References

1. Gurobi Optimization (2010), `http://www.gurobi.com` (visited October 2011)
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Communications of the ACM 53, 50–58 (2010)
3. Bobroff, N., Kochut, A., Beaty, K.A.: Dynamic placement of virtual machines for managing sla violations. In: Proceedings of the 10th IFIP/IEEE International Symposium on Integrated Network Management, IM 2007, pp. 119–128 (2007)
4. den Bossche, R.V., Vanmechelen, K., Broeckhove, J.: Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In: Proceedings of the 3rd IEEE International Conference on Cloud Computing, pp. 228–235 (2010)

5. Breitgand, D., Marashini, A., Tordsson, J.: Policy-driven service placement optimization in federated clouds. Tech. rep., IBM Haifa Labs (2011)
6. Chambers, C., Feng, W., Sahu, S., Saha, D.: Measurement-based characterization of a collection of on-line games. In: Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement, IMC 2005, pp. 1–14 (2005)
7. Chen, C., Tsai, K.: The server reassignment problem for load balancing in structured p2p systems. IEEE Transactions on Parallel and Distributed Processing 19(2), 234–246 (2008)
8. Ferrer, A.J., Hernádez, F., Tordsson, J., Elmroth, E., Ali-Eldin, A., Zsigri, C., Sirvent, R., Guitart, J., Badia, R.M., Djemame, K., Ziegler, W., Dimitrakos, T., Nair, S.K., Kousiouris, G., Konstanteli, K., Varvarigou, T., Hudzia, B., Kipp, A., Wesner, S., Corrales, M., Forgó, N., Sharif, T., Sheridan, C.: OPTIMIS: A holistic approach to cloud service provisioning. Future Generation Computer Systems 28(1), 66–77 (2012)
9. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming. Duxbury Press (November 2002)
10. Kousiouris, G., Cucinotta, T., Varvarigou, T.: The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. Journal of Systems and Software 84(8), 1270–1291 (2011)
11. Lampe, U., Siebenhaar, M., Schuller, D., Steinmetz, R.: A cloud-oriented broker for cost-minimal software service distribution. In: Proceedings of the 2nd ServiceWave Workshop on Optimizing Cloud Services (2011)
12. Li, W., Tordsson, J., Elmroth, E.: 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), pp. 163–171 (2011)
13. Mell, P., Grance, T.: The NIST definition of cloud computing. National Institute of Standards and Technology, NIST (2011)
14. Shachnai, H., Tamir, T.: On two class-constrained versions of the multiple knapsack problem. Algorithmica 29(3), 442–467 (2001)
15. Tang, C., Steinder, M., Spreitzer, M., Pacifici, G.: A scalable application placement controller for enterprise data centers. In: Proceedings of the 16th International Conference on World Wide Web, WWW 2007, pp. 331–340. ACM (2007)
16. Tian, C., Jiang, H., Iyengar, A., Liu, X., Wu, Z., Chen, J., Liu, W., Wang, C.: Improving application placement for cluster-based web applications. IEEE Transactions on Network and Service Management 8(2), 104–115 (2011)
17. Tordsson, J., Montero, R., Moreno-Vozmediano, R., Llorente, I.: Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Future Generation Computer Systems 28(2), 358–367 (2012)

III

Paper III

A General Approach to Service Deployment in Cloud Environments[*]

Wubin Li, Petter Svärd, Johan Tordsson, and Erik Elmroth

Department of Computing Science, Umeå University
SE-901 87 Umeå, Sweden
{*wubin.li, petters, tordsson, elmroth*}*@cs.umu.se*
http://www.cloudresearch.org

Abstract: The cloud computing landscape has recently developed into a spectrum of cloud architectures, leading to a broad range of management tools for similar operations but specialized for certain deployment scenarios. This both hinders the efficient reuse of algorithmic innovations within cloud management operations and increases the heterogeneity between different management systems. Our overarching goal is to overcome these problems by developing tools general enough to support the full range of popular architectures. In this contribution, we analyze commonalities in recently proposed cloud models (private clouds, multi-clouds, bursted clouds, federated clouds, etc.), and demonstrate how a key management functionality - service deployment - can be uniformly performed in all of these by a carefully designed system. The design of our service deployment solution is validated through demonstration of how it can be used to deploy services, perform bursting and brokering, as well as mediate a cloud federation in the context of the OPTIMIS Cloud toolkit.

Key words: Cloud Computing; Service Deployment;

[*] By permission of IEEE Computer Society Press.

A General Approach to Service Deployment in Cloud Environments

Wubin Li, Petter Svärd, Johan Tordsson, and Erik Elmroth
Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden
Email: {wubin.li, petters, tordsson, elmroth}@cs.umu.se

***Abstract*—The cloud computing landscape has recently developed into a spectrum of cloud architectures, leading to a broad range of management tools for similar operations but specialized for certain deployment scenarios. This both hinders the efficient reuse of algorithmic innovations within cloud management operations and increases the heterogeneity between different management systems. Our overarching goal is to overcome these problems by developing tools general enough to support the full range of popular architectures. In this contribution, we analyze commonalities in recently proposed cloud models (private clouds, multi-clouds, bursted clouds, federated clouds, etc.), and demonstrate how a key management functionality - service deployment - can be uniformly performed in all of these by a carefully designed system. The design of our service deployment framework is validated through a demonstration of how it can be used to deploy services, perform bursting and brokering, as well as mediate a cloud federation in the context of the OPTIMIS Toolkit.**

***Index Terms*—Cloud Computing; Cloud Architecture; Service Deployment**

I. Introduction

In the context of cloud computing, deployable services are encapsulated in virtual machines (*VMs*), and deployment is performed by instantiating VMs on top of a virtualized infrastructure. This new way of service deployment enables a traditional on-premises application to be rapidly redeployed as Software as a Service.

In this paper, we present a novel approach to service deployment, general enough to meet the requirements of a range of common cloud scenarios, including private clouds, bursted clouds, federated clouds, multi-clouds, and cloud brokering. We identify key requirements for service deployment in these scenarios and present the architecture for a service deployment tool to meet these requirements. Our proposed tool interacts with components for data management, service contextualization, and service management in its orchestration of the service deployment process.

Our approach is validated by implementation and integration in a private cloud, a bursted cloud, and a brokered multi-cloud scenario using tools from the OPTIMIS Toolkit [15] providing the required complementing functionalities. The verification study is performed in a cross-European testbed consisting of cloud resources at Atos (Spain), BT (UK), Flexiant (UK), and Umeå University (Sweden).

The remainder of the paper is organized as follows. Section II describes fundamental cloud concepts and outlines the service lifecycle. Section III discusses the studied cloud deployment scenarios. Core requirements for service deployment in cloud environments are described in Section IV. Section V describes the service deployment process. Section VI presents the design of our service deployment solution. Section VII contains a validation study of our approach in the context of OPTIMIS toolkit. Related work within service deployment is described in Section VIII. Finally, our conclusions are presented in Section IX, followed by a presentation of future work, acknowledgments, and a list of references.

II. Cloud Service Concepts

Cloud services can be categorized into Software as a Service, Platform as a Service, and Infrastructure as a service, or SaaS, PaaS, and IaaS for short [31]. In a cloud service deployment scenario the two stakeholders are the Infrastructure Provider (*IP*) and the Service Provider (*SP*). An IP offers infrastructure resources such as VMs, networks, and storage which can be used by SPs to deliver SaaS solutions to their customers. The SPs can also use PaaS tools to develop their services, or offer this functionality to their customers who may want to construct and deploy custom services. Without loss of generality, we concentrate in this contribution on the cases where a SP or IP deploys services to an IP providing IaaS.

A. Deployable Services

IaaS is based on virtualization technology which means that a deployable cloud service is in fact a VM or a collection of VMs. We refer to a VM of a certain type as a *component* and note that a service can consist of multiple components. For example, a three-tier web application service may consist of a database component (e.g., MySQL), an application component (e.g., Weblogic server [8]), and a presentation layer component (e.g., Apache server).

The information about which components the service is composed of along with functional and non-functional requirements for a deployment target is described in a document, the *service manifest*. The service manifest can also define elasticity bounds for the service, i.e., upper and lower limits for how many instances of a component that may be provisioned at any time. These bounds are commonly associated with elasticity rules for when to scale up or down the number of instances of a component, and such rules can range from simple *condition-action* statements to complex expressions that reason about statistical properties of the service workload. In addition, a service manifest typically contains various constraints such as desired geographical location, and data protection requirements.

B. *The Service Lifecycle*

The lifecycle of a cloud service can be summarized as construction, deployment, operation, and undeployment. In the construction phase, the service applications (Virtual Appliances) are implemented and packaged into a set of VMs. The construction of the above discussed service manifest ends the service construction phase. The service deployment includes identification of a suitable deployment target, installation of the service VMs in the selected provider, and initialization of these VMs by the provider, i.e., VMs are booted, configured, and start to deliver the service. In the operation phase, the IP, and potentially also the SP, perform a set of management actions to ensure efficient and robust provisioning of the service. Once the service is no longer needed, it can be undeployed by the SP, upon which the IP shuts down the running VMs and removes any assets of the service. Notably, multiple instances of the same service can be created from a single service manifest and these instances can be shutdown or restarted as needed.

III. Cloud Deployment Scenarios

Cloud environments can be set up differently depending on the types of interaction between the collaborating providers. The main differences between the scenarios are the number of involved actors and which actor is in control during the deployment process. The scenarios described in this section have been proposed and discussed in previous research [10], [15], [28], [31], [34], albeit typically in isolation and they have not been compared in the context of service deployment.

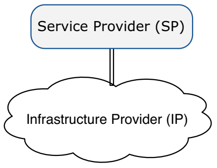

Fig. 1. Public cloud.

In the original and today most common cloud scenario, the *public cloud*, a SP deploys services to an IP offering infrastructure resources to the general public. In this case, all services are deployed to the same IP. Notably, if the IP has reached its maximum capacity, it rejects requests to deploy additional services. The public cloud scenario is illustrated in Figure 1.

Despite the rapid adoption of public clouds, there are several security and privacy concerns associated with service deployment to public IPs. To address these issues, a SP can set up a cloud infrastructure for its own internal use, commonly referred to as a *private cloud*, which is illustrated in Figure 2. Private clouds can circumvent many of the security and privacy concerns related to hosting sensitive information in public clouds, and may also offer stronger guarantees on control and performance as the service as well as the whole infrastructure is administered from within the same domain.

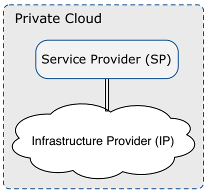

Fig. 2. Private cloud.

Private clouds may offload capacity to other IPs under periods of high workload, or for other reasons, e.g., planned maintenance of the internal servers. In this scenario, the providers form a hybrid architecture commonly referred to as a *bursted cloud* as seen in Figure 3. Typically, less sensitive tasks are executed in the public cloud instead while tasks that requiring higher levels of security are provisioned the private infrastructure.

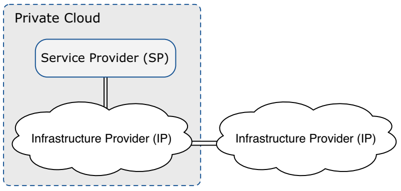

Fig. 3. Bursted (private) cloud.

Federated clouds are IPs collaborating on a basis of joint load-sharing agreements enabling them to offload capacity to each other [28] in a manner similar to how electricity providers exchange capacity. The federation takes place at the IP level in a transparent manner. In other words, a SP that deploys services to one of the IPs in a federation is not notified if its service is off-loaded to another IP within the federation. However, the SP is able to steer in which IPs the service may be provisioned, e.g., by specifying location constraints in the service manifest. Figure 4 illustrates a federation between three IPs.

If the SP itself is involved in selecting which IP a service should be deployed or re-deployed to the scenario is known as a *multi-cloud*. In multi-cloud deployments, such as in Figure 5,

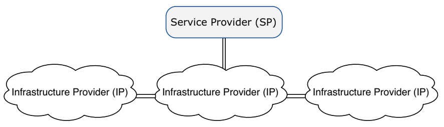
Service Provider (SP)
Infrastructure Provider (IP)
Infrastructure Provider (IP)
Infrastructure Provider (IP)

Fig. 4. Cloud federation.

the SP is responsible for planning, initiating and monitoring the execution of services. Notably, we are implicitly considering split deployment scenarios, i.e., when the components of the service are deployment across multiple IPs.

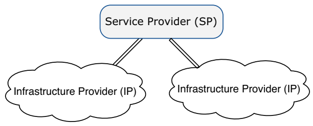

Fig. 5. Multi-cloud scenario.

A related scenario is that when a *cloud broker* [34] handles the complexity of prioritization and selection of IPs, and may also offer value-added services to IPs and SPs. In this case, the broker may have pre-arranged agreements with a number of IPs and selects the best match for a service based on the SP's desired criteria. The broker operates between the SP and the IPs, offering an IP-like interface to SPs and a SP-like interface to IPs, as illustrated in Figure 6.

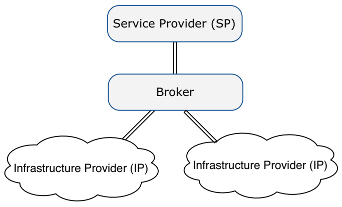

Fig. 6. Cloud brokering.

IV. Requirements for Service Deployment

A general approach to service deployment should work transparently in all types of clouds. Based on the deployment scenarios discussed in Section III, we identify a number of requirements for service deployment. Notably, these requirements for service deployment have significant similarities with the tasks identified in the overall process for resource selection (scheduling) in Grid computing environments [30]. In the below, the SP is normally the actor who negotiates terms with the IPs, initializes the service deployment, and performs any associated tasks. However, in the cases of cloud federation and bursting, these interactions occur between two IPs, with the IP initializing the deployment process acting as the SP.

- *Discovery of IPs*. In order to deploy a service, the deploying actor must identify the IPs that are available for deployment. IPs can be discovered by looking them up in a registry or by using auto-discovery mechanisms. We remark that discovery (along with the later filtering and selection) of an IP is trivial in the private cloud case, as only a single IP is available.
- *Filtering of available IPs*. In order not to add overhead by negotiating deployment with IPs that fail to fulfill fundamental requirements for the particle service to be deployed, an initial filtering of the list of IPs retrieved during IP discovery must be possible. Criteria for filtering include both functional aspects, e.g., support for certain hypervisors and VM image formats, as well as non-functional criteria such as constraints based on the country in which the IP is based (for legal and/or data-protection reasons).
- *Service Manifest construction*. Each service must include a service manifest that describes the functional and non-functional parameters of the service. A service manifest is an abstract definition of the service, which is used to negotiate with IPs and later becomes part of the service agreement with the IP. Data specified in the service manifest, i.e. VM disk images, must also be prepared. A set of utilities for creation, modification, etc. of service manifests would greatly simplify this procedure.
- *Negotiation and deployment optimization*. A SP must be able to negotiate with available IPs for service hosting offers. Note that it is not always desirable to deploy the whole service to the same provider. For reasons such as security, performance, and fault tolerance it can be preferable to split the service between several IPs which means that a negotiation can be for part of a service. Based on the results of these negotiations and data such as reputation statistics, that could be gathered and evaluated by third-party entities, the SP must decide where to deploy the service.
- *Service contextualization*. When a service component's disk images are generated all required information is not known. Data such as locations of shared network resources and security credentials depends on the context in which the service is deployed. In order to launch the service successfully, this information must be propagated to the VM. A possible mechanism for this *contextualization* process is to embed various scripts in the VM images that dynamically retrieves information upon boot, enabling VM instances to self-contextualize.
- *Service data transfer*. The contextualized VM images, along with any other data required by the service, must be transferred to the IP. To guarantee properties such as confidentiality and data integrity during this transfer, reliable security mechanisms are required.
- *Service Level Agreement creation*. To ensure that the service operates according to the SP's expectations, it must be possible to establish a Service Level Agreement (*SLA*), that governs the relationship between the SP and

IP for the provisioning of the service. SLA's for service provisioning commonly include segments that address service definition, performance measurements, problem management, customer duties, warranties, disaster recovery, as well as conditions for termination of the agreement [1]. Penalties may be agreed upon in the case of non-compliance with the terms in the SLA.

V. The Service Deployment Process

Based on the requirements study in the previous sections, we derive a general sequence for service deployment. This process, consisting of tasks to be performed, is illustrated by the sequence diagram in Figure 7. Notably, the complexity and details of the service deployment process may vary with the deployment scenario, but the process remains similar.

Step 1, service construction, is identical in all scenarios. The SP constructs (implements, packages, etc.) a service in the same way no matter how it will be deployed. Step 2, discovery and filtering of IPs to find out what IPs are available for the deployment, is trivial in the private cloud case as the IP is already known. It is also relatively simple in cloud brokering scenarios as only the broker needs to be known. However in federation, public, and multi-cloud cases this step can be quite complex, involving auto-discovery mechanisms and/or IP registries.

Most of the algorithmic complexity in service deployment is associated within the related tasks of SLA negotiation and IP assessment (Steps 3 and 4 in Figure 7). In scenarios where the SP interacts with a single provider, these tasks are simplified. Conversely, for federation, bursting, and multi-cloud deployments, interaction with more than one IP complicates the process. The richness of the negotiation protocol can range from simple versions with primitives such as offer, accept, and refuse, to more complicated versions with counter-offers, and approaches based on auctions. An in-depth analysis of negotiation protocols is beyond the scope of this paper and further details on this topic are given, e.g., by Sarangan et al. [29] and Jennings et al. [16]. Similarly, for IP assessment, the complexity of estimating the utility associated with deploying the service in each potential provider can differ significantly based on the modeling method used. Algorithms proposed for optimizing provider selection include scheduling-inspired combinatorial optimization approaches such as integer programming, which are commonly suggested [13], [34], but tend to scale very poorly with the number of IPs. Other approaches include heuristic solutions [21] that trade optimality for faster decision-making.

Once the most suitable provider (or potentially, set of providers) is identified, the SP performs contextualization (Step 5 in the sequence diagram) to prepare the service VM images with any dynamic information that is needed for these to boot and configure themselves properly. This step is more complicated if the service is split among several IPs, as an external rendezvous mechanism is typically required in order to initialize cross-provider networking for the VMs of the service.

After the VM images are properly configured, they are uploaded to the selected provider(s) as illustrated in Step 6 of Figure 7. As VM images typically are very large, significant performance gains can be achieved by proper tuning of network parameters. In private clouds where a network file system may connect the SP and the IP, image transfer is much less of an issue. Alternatively, if an IP does not support upload of SP-defined VM images, a custom service image must be pre-created (based on templates from the provider) and stored at that IP. In such a case, contextualization abilities are significantly reduced.

When the contextualized VM images are stored in the IP's repository, the SP confirms the offer negotiated in Step 3 and a SLA is created between the SP and the IP for the operation of the service, as illustrated in Step 7. Once again, this step becomes more complex if the SP needs to aggregate multiple SLAs from different IPs.

Finally, Step 8 in Figure 7 illustrates that once the service is deployed, the SP stores information about the deployment in a registry to enable subsequent service monitoring, management, and undeployment.

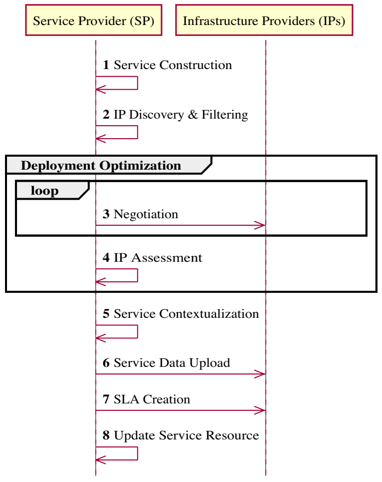

Fig. 7. Sequence diagram for service deployment.

VI. Proposed Service Deployment Architecture

To meet the requirements of service deployment, we propose a service deployment architecture. The purpose of architecture is two-fold - it is responsible for generating optimal deployment solutions for a service, and for coordinating the deployment process in order to provision a service according to the deployment plan. In order to separate the placement optimization from the deployment coordination functionality, our proposed software, referred to as the Service Deployment

Optimizer (*SDO*), is divided into two components, the *Service Deployer* (SD), and the *Deployment Optimizer* (DO), both illustrated in Figure 8. The DO is a decision-making component and the SD is a module that orchestrates the DO and various utility functionalities in order to perform the deployment sequence described in Figure 7. Notably, to provide a complete solution for cloud deployment, the SD and DO interacts with external components for service contextualization, data management, service management, and IP assessment, all illustrated in Figure 8 and further discussion in Section IV. We remark that these external components may need customization and/or replacement depending on, e.g., the protocols and data formats used by the IPs.

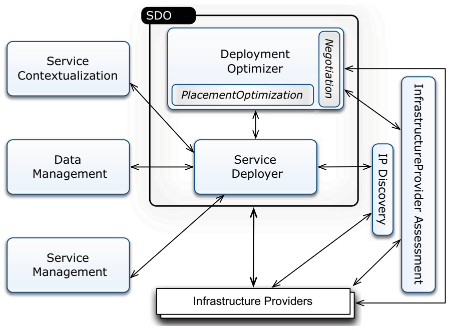

Fig. 8. Overview of the SDO architecture.

We outline the main design rationale for the SD and DO components below, as well as discuss how they interact with each other and related utility functionalities for data transfer, etc.

A. Service Deployer

The purpose of the SD is to coordinate the deployment and interact with the other involved parties in a deployment. The SD takes a service deployment request, contacts the IP discovery service to obtain which providers are available and performs filtering (see steps 1-2 in Figure 7). To retrieve an optimal placement scheme, SD contacts the DO who performs calculation for placement optimization. Once an optimal placement solution is returned, the SD deploys a whole service following steps 5-8 in Figure 7 with the support of external components. *Service Contextualization* is in charge of contextualizing VM images, *Data Management* is responsible for data transfer from the SP side to the IP side, *Service Management* creates service resource and updates resource accordingly, and *SLA Management* handles the IP side creation of agreement.

B. Deployment Optimizer

The DO's role in a deployment is to perform placement optimization based on the inputs from the SD, including a service manifest, the optimization objective, and available IP info, etc. Based on this information, the DO generates an optimal placement scheme for the service. In order to achieve an optimal placement objective, the DO may split services that contains more than one component into several sub-services, and map them to different IPs. This is provided it can do so without breaking affinity constraints specified in the service description, During the calculation, the DO negotiates with IPs and the IP assessment tools, see steps 3-4 in Figure 7. Optimization techniques such as combinatorial optimization, problem relaxations and heuristic approaches such as greedy formulation can be applied in this component.

VII. Validation Study

In order to verify that our service deployment architecture is suitable for the envisioned cloud architectures, we perform a validation study. The study is carried out in the context of the OPTIMIS Toolkit [15], which includes a set of independent components that can be adopted, either in full or in part, by IPs that provide infrastructure resources, and by SPs that use these capacities to deliver services. The study comprises three cloud service deployment scenarios: private cloud, cloud brokerage, and cloud bursting.

In these three scenarios, the service we use for validation is a composite service for gene detection presented in [33]. This service contains five components. First, there are four functionality components which contribute to the overall gene detection process: translation of the input genomic database to a given format (component GA); obtention of a list of amino acid sequences which are similar to a reference input sequence (component GB); search of the relevant regions of the genomic database (component GC) and execution of the GeneWise [12] algorithm on them (component GD). Additionally, there is one component for coordination (component GP). Each component is encapsulated in a VM sized approximately 9.8 GB. To avoid repetitive data transfer, only one VM image is transferred from the SP to the IP in case multiple components are deployed to the IP. This way, multiple VM instances can be started from the same image by associating each instance with different contextualization data.

For the validation, we use a distributed testbed with four IPs located across Europe: Atos [2] (Spain), BT [3] (UK), Flexiant [4] (UK) and Umeå University (Sweden). Each IP site hosts selected parts of the OPTIMIS Toolkit, as well as fundamental management software such as Xen [5] and Nagios [6]. The role of the IPs in the different scenarios is summarized in Table I. Notably, our goal is not to evaluate the various providers but rather to investigate how well our proposed approach adapt to real scenarios.

TABLE I
USE CASE CONFIGURATIONS.

	Atos	BT	Umeå University	Flexiant
Private cloud	✓			
Cloud brokerage	✓	✓	✓	✓
Cloud bursting	✓			✓

A. SDO in OPTIMIS

We below outline how the SDO is integrated with selected components of the OPTIMIS Toolkit, i.e., how the architecture outlined in Figure 8 is achieved in an OPTIMIS cloud scenario.

- *IP Discovery:* In OPTIMIS, IP information is registered in a simple on-line registry accessed through a REST interface. In this registry, information such as IP identifier, IP name, and endpoints for negotiation, etc., are stored.
- *VM Contextualization*: The OPTIMIS VM Contextualization component provides an interface for constructing service context data, such as security certificates, VPN hostnames, VPN DNS and Gateway IP addresses, mount points for network data stores, monitoring manager hostnames, off-line software license tokens, as well as list of software dependencies [11].
- *Data Manager*: In OPTIMIS, a Hadoop-based [7] Data Management service enriched with RESTful APIs is combined with a series of tools that aim to extend Hadoop's functionality beyond its well known scope - heavy data processing [19]. In all, these tools provide data management functionalities to cloud services as well as the capabilities needed to deploy VM images to IPs.
- *SLA Management*: A service and client based on WS-Agreement protocol [9] is used in OPTIMIS for negotiating and creating Service Level Agreements between IPs and SPs [20].
- *Infrastructure Provider Assessment*: for OPTIMIS, we implement a two-step IP assessment strategy, where in a first step, IPs that do not fulfill functional requirements or have unsuitable data protection levels, etc. are filtered out. In a second step, the DO negotiates deployment terms (cost, etc.) with the remaining IPs and ranked these on basis of assessment of four non-functional properties: trust, risk, eco-efficiency, and cost [18], [27].

B. Scenarios Descriptions and Statistics

- Private cloud:

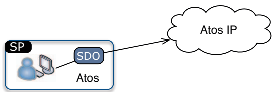

Fig. 9. Private scenario.

In the private cloud scenario, the SP (also located in the Atos cloud) submits the gene detection service deployment request to the Atos cloud. All components (GA, GB, GC, GD, and GP) are deployed to the Atos IP.

- Cloud brokerage (multi-cloud):
In the cloud brokerage scenario, two SDO instances are running: one in the Umeå cloud, which plays the role of the SP. The other one is located in the BT cloud, which plays the role of a cloud broker. The SP submits the gene detection service deployment request to the Umeå cloud. Instead of deploying the service by itself, the

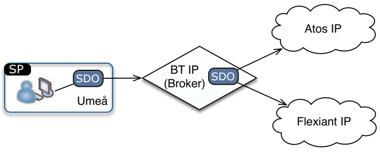

Fig. 10. Cloud brokerage scenario.

SDO in the Umeå cloud calls the SDO on the broker to complete the deployment. There are three IPs registered in the IP registry which can be queried by the SDO in the BT cloud. After the by Deployment Optimizer's calculations (including IP assessment, negotiation, and placement optimization) two IPs are selected to host the service. Specifically, two components (GC and GD) are to be deployed to Flexiant cloud, the other three (GA, GB, and GP) are to be deployed to the Atos cloud. For the purpose of this demonstration, VM images are stored on the broker in advance.

- Cloud bursting:

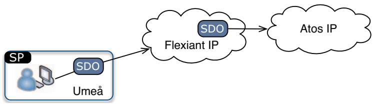

Fig. 11. Cloud bursting scenario.

In the cloud bursting scenario, the service is already deployed in the Flexiant cloud. To fulfill a demand for increasing service capacity from the SP, the Flexiant cloud needs to launch two more instances respectively for two of the five components (i.e., GC and GD) in the service. For financial reasons, the Flexiant cloud decides to outsource this demand to a more cheaper cloud provider, i.e., Atos cloud, while maintaining its SLA-agreement with SP.

C. Experimental Results

In order to assess the performance of the SDO and the complexity of the service deployment process as such, we measure the duration of the main steps of deployment for each studied cloud architecture. Table II presents statistics of time consumed in each phase of service deployment for each scenario.

From our experiments, we conclude that the major part of the time (around 90-95% depending on the scenario) is used to transfer VM images from the SP to the IP. Notable, the differences in image transfer time among the scenarios are due to the complexity of the deployment solution. For the private cloud scenario, all components are deployed to Atos and only a single VM image thus needs to be transferred, over an internal network. In contrast, for the multi-cloud scenario, two VM images are transferred, one from BT to Atos (for components

TABLE II
ILLUSTRATIVE PERFORMANCE RESULTS FOR THE DEMONSTRATED DEPLOYMENT SCENARIOS. ALL TIMES ARE IN SECONDS.

Deployment Phase	Private	Multi-cloud	Cloud bursting
IP discovery	0	2	1
Deployment optimization	2	108	13
VM contextualization	11	19	15
Data upload	598	1546	701
Agreement creation	12	23	17
Update service resource	4	4	2

GA, GB, and GP), the other one from BT to Flexiant (for components GC and GD), both transfers taking place over Internet.

Another observation is that placement optimization becomes more complex in the multi-cloud case. Due to the possibility of split deployment, the number of potential service configurations is much larger in a multi-cloud scenario than for the private and bursting cases. In the brokering case, multiple negotiations are performed between the broker deployed at BT and the Atos and Flexiant IPs. The actual assessment of the IPs' suitability for provisioning the service includes complex statistical modeling techniques [18], [27] to assess the trust, risk, eco-efficiency and cost factors for each potential placement. These models all have in common that they in the assessments make extensive use of a database with historical information about past IP behavior. Notably, the details of the optimization techniques used in this section are outside the scope of this paper, however, a detailed elaboration can be found in our contribution [23].

In summary, the private cloud scenario demonstrates how the SDO can be used to complete a service deployment in general. The cloud brokerage scenario demonstrates brokerage across multiple cloud providers. The cloud bursting scenario shows how organizations can utilize the SDO to scale out their infrastructure, using resources from third-party providers based upon a range of factors such as trust, risk assessment, eco-efficiency and cost [18], [27].

VIII. RELATED WORK

Talwar et al. [32] review approaches for service deployment before the emergence of Cloud Computing. They compare and evaluate four types of service-deployment approaches: manual, script-, language-, and model-based solutions, in terms of scale, complexity, expressiveness, and barriers for first time usage. They also conclude that service deployment technologies based on scripts and configuration files have limitations in expressing dependencies and verifying configurations, and usually result in erroneous system configurations. Conversely, language- and model-based approaches handle these challenges better, but at the expense of comparatively higher barriers for first usage.

With the emergence of cloud computing, services are provisioned using VMs. Service deployment can be done by initializing VMs with their virtual appliances. Cloud service developers are thus enabled to deploy applications without confronting the usual obstacles of maintaining hardware and system configurations. Much work have been done in the context of this new service-deployment technology. Most focus has been on deployment optimization approaches. For example, Kecskemeti et al. [17] propose an automated virtual appliance creation service that aids the service developers to efficiently create deployable virtual appliances. They reduce the deployment time of the service by rebuilding the virtual appliance of the service on the deployment target site. For optimal VM placement across multiple cloud providers, Chaisiri et al. [13] propose an stochastic integer programming (SIP) based algorithm aimed at minimizing the cost for a placement plan for hosting VMs in a multiple cloud provider environment under future demand and price uncertainty. Similarly, Vozmediano et al. [26] [25] explore the use case of deploying a compute cluster on top of a multi-cloud infrastructure, for use by loosely-coupled Many-Task Computing (MTC) applications. They conclude that cluster nodes can be provisioned with resources from different clouds to improve the cost-effectiveness of the deployment, or to implement high-availability strategies.

Our previous contributions in this field include cloud brokering mechanisms [34] for cost- and performance-optimal placement of VMs across multiple cloud providers in static scenarios, and extensions to this with a linear programming model to dynamically reschedule VMs (including modeling of VM migration overhead) upon changed conditions such as price changes, service demand variation, etc. in dynamic cloud scheduling scenarios [22]. In addition, we have proposed an approach to optimal VM placement within data centers for predictable and time-constrained load peaks [21].

Although algorithms for optimizing service deployment is a very active area of research, with lot of interest is given to the various deployment architectures in general, the topic of this contribution, namely architectures and tools general enough to support multiple cloud deployment scenarios, has received far less attention to date.

IX. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a general approach to automatic service deployment in cloud environments, based on our study of cloud architectures and deployment scenarios and the core requirements for service deployment derived from these. A validation study performed in the context of the OPTIMIS Toolkit verifies the feasibility of a general service deployment component that can be reused across multiple cloud architectures. Our validation study also gives some indications about the performance aspects of cloud service deployment, identifying transfer of VM images as the most time-consuming task.

Future directions for this work includes in-depth studies of algorithms for optimized selection of deployment targets. Another topic of future research is the incorporation of *re-deployment*, i.e., migration of the full service, or some of its components, to other IP(s) during operation [14]. Reasons for re-deployment include improved performance, and improved cost-efficiency. In such scenarios, a careful tradeoff between re-deployment overhead and expected improvement must be considered [22]. Additionally, a model of interconnection requirements that can precisely express the relationships between components within a service to be deployed can be another promising direction to investigate. Such a model can help SDO optimizing the service deployment with e.g., less communication cost between service components. In addition, we are working on a specific scenario where cloud users can specify hard constraints and soft constraints when demanding resource provisions. A hard constraint is a condition that has to be satisfied when deploying services, i.e., it is mandatory. In contrast, a soft constraint (also called a preference) is optional. An optimal placement solution with soft constraints satisfied is preferable over other solutions. We are also investigating how to apply multi-objective optimization [24] techniques to this scenario.

ACKNOWLEDGMENTS

We acknowledge the members of the OPTIMIS team for their valuable help in integration and testing for the validation scenario. We also thank Atos, BT, and Flexiant for providing servers for the distributed testbed, as well as Daniel Espling for helping with figures 1-6. Financial support has in part been provided by the European Community's Seventh Framework Programme ([FP7/2010-2013]) under grant agreements no. 257115 (OPTIMIS), and the Swedish Government's strategic effort eSSENCE.

REFERENCES

[1] An outline of the core elements of an SLA, http://www.sla-zone.co.uk/, visited May 2012.
[2] Atos, 2012, http://atos.net, visited May 2012.
[3] BT Group, 2012, http://www.bt.com/, visited May 2012.
[4] Flexiant Cloud, 2012, http://www.flexiant.com/, visited May 2012.
[5] Xen, 2012, http://www.xen.org/, visited May 2012.
[6] Nagios, 2012, http://www.nagios.org/, visited May 2012.
[7] Apache Hadoop Project. http://hadoop.apache.org/, visited May, 2012.
[8] Oracle WebLogic Server. http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html, visited May, 2012.
[9] Web Services Agreement Specification (WS-Agreement). http://www.ogf.org/documents/GFD.107.pdf, visited May, 2012.
[10] M. Ahronovitz et al. Cloud computing use cases white paper, v4.0. www.cloudusecases.org, visited May 2012.
[11] D. Armstrong, K. Djemame, S. Nair, J. Tordsson, and W. Ziegler. Towards a Contextualization Solution for Cloud Platform Services. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, CloudCom'11, pages 328–331, Washington, DC, USA, 2011. IEEE Computer Society.
[12] E. Birney, M. Clamp, and R. Durbin. GeneWise and Genomewise. *Genome Research*, 14(5):988–995, May 2004.
[13] S. Chaisiri, B.-S. Lee, and D. Niyato. Optimal virtual machine placement across multiple cloudproviders. In *Proceedings of the 4th IEEE Asia-Pacific Services Computing Conference*, pages 103–110.
[14] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI '05*, pages 273–286. ACM, May 2005.
[15] 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*, 28(1):66–77, 2012.
[16] N. Jennings, P. Faratin, A. Lomuscio, S. Parsons, M. Wooldridge, and C. Sierra. Automated negotiation: prospects, methods and challenges. *Group Decision and Negotiation*, 10(2):199–215, 2001.
[17] G. Kecskemeti, G. Terstyanszky, P. Kacsuk, and Z. Neméth. An Approach for Virtual Appliance Distribution for Service Deployment. *Future Gener. Comput. Syst.*, 27(3):280–289, March 2011.
[18] M. Kiran, M. Jiang, D. J. Armstrong, and K. Djemame. Towards a Service Lifecycle Based Methodology for Risk Assessment in Cloud Computing. In *Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, DASC'11, pages 449–456, Washington, DC, USA, 2011. IEEE Computer Society.
[19] G. Kousiouris, G. Vafiadis, and T. Varvarigou. A Front-end, Hadoop-based Data Management Service for Efficient Federated Clouds. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, CloudCom'11, pages 511–516, Washington, DC, USA, 2011. IEEE Computer Society.
[20] A. Lawrence, K. Djemame, O. Wäldrich, W. Ziegler, and C. Zsigri. Using Service Level Agreements for Optimising Cloud Infrastructure Services. In *Proceedings of the 2010 international conference ServiceWave*, ServiceWave'10, pages 38–49, Berlin, Heidelberg. Springer-Verlag.
[21] W. Li, J. Tordsson, and E. Elmroth. Virtual Machine Placement for Predictable and Time-Constrained Peak Loads. In *Proceedings of the 8th international conference on Economics of grids, clouds, systems, and services (GECON'11).* Lecture Notes in Computer Science, Vol. 7150, Springer-Verlag, pp. 120-134, 2011.
[22] 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)*, pages 163–171, 2011.
[23] W. Li, J. Tordsson, and E. Elmroth. Permutation-pack based Optimization for Service Deployment in Multi-Cloud Environments, 2012. to be submitted.
[24] R. Marler and J. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26:369–395, 2004.
[25] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Elastic management of web server clusters on distributed virtual infrastructures. *Concurrency and Computation: Practice and Experience*, 23(13):1474–1490, 2011.
[26] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Multicloud deployment of computing clusters for loosely coupled mtc applications. *IEEE Transactions on Parallel and Distributed Systems*, 22:924–930, 2011.
[27] P. Pawar, M. Rajarajan, S. Nair, and A. Zisman. Trust model for optimized cloud services. In *IFIP Advances in Information and Communication Technology, Volume 374, Trust Management VI*, pages 97–112. Springer, 2012.
[28] 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. Galan. The RESERVOIR model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):1–11, 2009.
[29] V. Sarangan and J. Chen. Comparative study of protocols for dynamic service negotiation in the next-generation internet. *Communications Magazine, IEEE*, 44(3):151–156, 2006.
[30] J. Schopf. Ten actions when grid scheduling. In *Grid resource management*, pages 15–24. Kluwer Academic Publishers, 2003.
[31] S. Strauch, O. Kopp, F. Leymann, and T. Unger. A taxonomy for cloud data hosting solutions. In *Proceedings of the IEEE International Conference on Cloud and Green Computing, CGC 2011, 12-14 December 2011, Sydney, Australia*, pages 577–584. IEEE Computer Society, 2011.
[32] V. Talwar, D. Milojicic, Q. Wu, C. Pu, W. Yan, and G. Jung. Approaches for Service Deployment. *IEEE Internet Computing*, 9(2):70–80, Mar. 2005.
[33] E. Tejedor, J. Ejarque, F. Lordan, R. Rafanell, J. Alvarez, D. Lezzi, R. Sirvent, and R. Badia. A cloud-unaware programming model for easy development of composite services. In *2011 Third IEEE International Conference on Coud Computing Technology and Science*, pages 375–382. IEEE, 2011.
[34] J. Tordsson, R. Montero, R. Moreno-Vozmediano, and I. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems*, 28(2):358–367, 2012.

IV

Paper IV

Cost-Optimal Cloud Service Placement under Dynamic Pricing Schemes[*]

Wubin Li, Petter Svärd, Johan Tordsson, and Erik Elmroth

Department of Computing Science, Umeå University
SE-901 87 Umeå, Sweden
{*wubin.li, petters, tordsson, elmroth*} *@cs.umu.se*
http://www.cloudresearch.org

Abstract: Until now, most research on cloud service placement has focused on static pricing scenarios, where cloud providers offer fixed prices for their resources. However, with the recent trend of dynamic pricing of cloud resources, where the price of a compute resource can vary depending on the free capacity and load of the provider, new placement algorithms are needed. In this paper, we investigate service placement in dynamic pricing scenarios by evaluating a set of placement algorithms, tuned for dynamic pricing. The algorithms range from simple heuristics to combinatorial optimization solutions. The studied algorithms are evaluated by deploying a set of services across multiple providers. Finally, we analyse the strengths and weaknesses of the algorithms considered. The evaluation suggests that exhaustive search based approach is good at finding optimal solutions for service placement under dynamic pricing schemes, but the execution times are usually long. In contrast, greedy approaches perform surprisingly well with fast execution times and acceptable solutions, and thus can be a suitable compromise considering the tradeoffs between quality of solution and execution time.

Key words: Cloud Computing; Dynamic Pricing; Service Placement; Deployment Optimization

[*] By permission of IEEE Computer Society Press.

Cost-Optimal Cloud Service Placement under Dynamic Pricing Schemes

Wubin Li, Petter Svärd, Johan Tordsson, and Erik Elmroth
Department of Computing Science, Umeå University
SE-901 87 Umeå, Sweden
Email: {wubin.li, petters, tordsson, elmroth}@cs.umu.se

***Abstract*—Until now, most research on cloud service placement has focused on static pricing scenarios, where cloud providers offer fixed prices for their resources. However, with the recent trend of dynamic pricing of cloud resources, where the price of a compute resource can vary depending on the free capacity and load of the provider, new placement algorithms are needed. In this paper, we investigate service placement in dynamic pricing scenarios by evaluating a set of placement algorithms, tuned for dynamic pricing. The algorithms range from simple heuristics to combinatorial optimization solutions. The studied algorithms are evaluated by deploying a set of services across multiple providers. Finally, we analyse the strengths and weaknesses of the algorithms considered. The evaluation suggests that exhaustive search based approach is good at finding optimal solutions for service placement under dynamic pricing schemes, but the execution times are usually long. In contrast, greedy approaches perform surprisingly well with fast execution times and acceptable solutions, and thus can be a suitable compromise considering the tradeoffs between quality of solution and execution time.**

***Keywords*-Cloud Computing; Dynamic Pricing; Service Placement; Deployment Optimization**

I. Introduction

Cloud services are typically encapsulated in virtual machines (*VMs*), and are deployed by instantiating VMs in a virtualized infrastructure. When deploying such services, it is desirable to find an optimal placement, that is an optimal choice of cloud provider(s), considering for example Service Level Agreement (*SLA*) terms, power consumption and performance. By deploying cloud services across several cloud providers instead of using just one, users can gain benefits like cost reduction, load balancing and better fault tolerance, and also avoid vendor lock-in.

In the expanding cloud computing market, there are many cloud providers with comparable offers, for example GoGrid [1] and Amazon [2] which offer capacity on a hourly, monthly, semi-annual, and annual base. Historically, most providers have used fixed pricing schemes, i.e., the price of a compute unit is constant regardless of the available capacity at the provider. Recently, the concept of dynamic resource pricing is becoming increasingly popular. Such schemes enable cloud providers to attract more customers by offering a lower price if they have excess capacity. Amazon for example has introduced spot instances which enable users to bid for unused Amazon EC2 capacity. Instances are charged the Spot Price, which is set by Amazon and fluctuates periodically depending on the supply of and demand for the spot instance capacity [3]. Such types of dynamic pricing schemes provide cloud customers with the flexibility of ad-hoc provisioning while receiving significant price savings.

As a consequence of the static pricing schemes used by commercial providers, most cloud service placement research has focused on static pricing scenarios where cloud providers offer fixed pricing schemes for their resources. In this paper however, we propose methods and algorithms to find cost-optimal deployment of services across multiple cloud providers in dynamic pricing scenarios. We study a number of algorithms for placement optimization and evaluate them by running deployments on a cloud platform using our general approach to service deployment, the Service Deployment Optimizer (SDO), presented in a previous contribution [14].

The remainder of the paper is organized as follows. Section II gives a short background on cloud services and deployment. Related work is discussed in Section III. Section IV briefly defines the studied problem and outlines our optimization algorithms. Section V presents the experimental evaluation in an environment with three clouds. Our conclusions are given in Section VI followed by acknowledgments, and a list of references.

II. Background

A. Cloud services

In virtualized cloud environments, a cloud service is provisioned as a VM or a collection of VMs. A VM of a certain type is known as a component and a service can consist of multiple components. For example, a typical three-tier web application has a presentation layer component, a business layer component and a database component. Note that there can be several instances of each component. Information about the service composition in terms of components, functional and non-functional requirements and elasticity bounds may be described in a document, the service manifest. The elasticity bounds are upper and lower limits for how many instances of a component that are allowed to be provisioned at a given time and are commonly associated with elasticity rules for when to scale up or down the number of instances of a component. The service manifest may also contain any constraints on the service, e.g., geographical location or requirements for data protection. An extensive implementation of this kind of a service manifest can be found in [19].

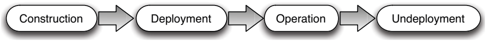

Fig. 1. The lifecycle of a cloud service.

Figure 1 illustrates the different phases in the lifecycle of a cloud service. The packaging the service into components and building the service manifest is known as the construction phase of the service lifecycle. In order to get the service running, it needs to be deployed. During the deployment phase, a suitable cloud provider, or a set of providers, is identified. The service components are then contextualized and transferred to the selected provider where they are installed. Once the VMs have been booted and are accessible to outside peers, the deployment phase is complete. The service lifecycle then moves to the operation phase in which the service is managed by the cloud provider to ensure efficient and robust service delivery. Notably, VM recontextualization may be needed in the operation phase to enable adaptation of VM behavior in response to internal changes in the service to which the VM belongs or to external changes affecting the execution environment of the VM [6]. When the service is no longer needed, it can be undeployed. During the undeploy phase, the cloud provider shuts down the running VMs and removes the service assets such as disk images. This paper focuses on the service deployment phase which is discussed in more detail in the upcoming sections.

B. Service Deployment

Providers offering services to customers are known as Service Providers (SPs). Since most SPs often do not control enough hardware resources, they deploy their services to Cloud Providers. This deployment process can be complex. In a previous contribution [14], where we design and implement a general approach to service deployment, the Service Deployment Optimizer, we divide the deployment process into six stages. These stages are cloud provider discovery and filtering, service manifest construction, negotiation and deployment optimization, service contextualization, service data transfer and SLA creation. We then identify requirements for all of these stages as well as for the deployment process as a a whole.

Of interest to this contribution is the requirement for the *negotiation and deployment optimization* stage which states that the deploying party must be able to negotiate with available providers for service hosting offers. The requirement also states that it is not always desirable to deploy the whole service to the same provider but for reasons such as security, performance, and fault tolerance it can be preferable to split the service between several providers. This means that a negotiation can be performed for part of a service, not necessarily the whole manifest. Based on the results of these negotiations and data such as reputation statistics, which could be gathered and evaluated by third-party entities, the deploying party must then decide where to deploy the service. In this contribution, we refer to this process as *Service Placement*.

C. Service Placement

During the deployment of a service, a decision is taken on which provider, or combination of providers, is to be used to host the service. If several providers are used, the service manifest is split to create a number of sub-services which are then independently mapped to providers. Note that constraints such as affinity and anti-affinity can limit how the service can be decomposed [9].

Taking into account these requirements, the placement algorithm usually strives to optimize a given objective, for example to minimize the cost or the risk by splitting the service and finding the optimal combination of providers. The selection process is commonly performed as a negotiation process, where the SP asks the providers for offers on hosting a service or parts of a service. Based on the results of these negotiations and other possibly available information such as previous experience with the providers (reputation assessments), the SP decides on where to deploy the service.

III. Related Work

Over the last years, there has been a significant research effort in optimizing allocation of VMs in clouds, commonly with cost and performance as optimization objectives [8], [15], [17], [22], [24]. This research field, commonly referred to as cloud placement, scheduling, and/or brokering, started out focusing on static environments. Recently, the field has been extended to also include dynamic scenarios, including changes in cloud provider prices. Examples of the latter include work by Andrzejak et al., who propose a probabilistic model to optimize cost and performance under dynamic pricing schemes. They use an SLA model with tasks bound by deadline and budget, where varying numbers of VMs can be allocated to optimize these goals [5]. An evaluation based on historical spot instance prices from Amazon EC2 combined with publicly available grid workloads demonstrates how users can achieve large cost savings by bidding for high-CPU instances, and also achieve a balance between cost and service level (job deadline).

Similarly but from a cloud provider's perspective, to optimize the revenue and energy cost while satisfying the demands of customers, Zhang et al. presents a MPC (Model Predictive Control) based resource management mechanism to dynamically adjust the capacity allocated to each VM type [26]. Experimental evaluations show that, compared with static allocation strategies, the proposed approach combining market economy and optimal control theory is very promising.

Service placement in multi-cloud scenarios has also been studied extensively in the past. Our previous contribution on this topic includes a novel cloud brokering approach that optimizes placement of virtual infrastructures across multiple clouds [23], which compared to single cloud deployment improves performance, lower costs, or provide a combination thereof. For scenarios where parameters such as pricing schemes and VM types are continuously changed, we propose a linear integer programming model for dynamic cloud scheduling via migration of virtual machines [16]. The proposed model can be applied in various scenarios through selections of corresponding objectives and constraints, and offers the flexibility to express different levels of migration overhead when restructuring an existing virtual infrastructure where services are being hosted. Lucas-Simarro et al. go a step further to implement a scheduler capable of taking autonomous placement decisions based on different pricing schemes. In case of dynamic pricing scenarios, the scheduler decisions are based on a prediction model that estimates the price of the VMs in the next period [18].

A field closely related to the optimization of VM placement is study of the actual cloud provider pricing mechanisms. Wee studies the development of spot instance prices on Amazon EC2 [25], the most well-known cloud system with real-time pricing. While Wee observes that spot instances are around 50% cheaper than reserved instances, the observed deviations in spot instance prices over a year are very small, with only a few percent reduction in the cheapest prices. Ben-Yehuda et al. analyze the historical prices of EC2 spot instances and reverse engineer the pricing scheme. They conclude that prices are not market-driven but rather randomly generated within a tight interval [7]. Analogously, a statistical model of spot instance prices in public cloud environments is presented by Bahman et al. in [12], which fits Amazon's spot instances prices well with a good degree of accuracy. To capture the realistic value of the cloud compute commodities, Bhanu et al. employ financial option theory and treat the cloud resources as real assets. The cloud resources are then priced by solving the finance model [21].

IV. Algorithms

In this paper, we study the cost-optimization problem from the perspective of a service provider, which can be simply formulated as follows: Given n cloud providers provisioning resources with dynamic pricing schemes, our goal is to find the placement solution that minimizes the cost of deploying a service with q components across those cloud providers. Notably, unlike the related works mentioned in the previous section, we assume dynamic pricing of provisioning requests, and thus, we assume that the price of hosting a service or a part of a service is not known to the service provider prior to negotiation with the target provider.

To investigate the effects of our dynamic pricing strategy, a number of placement algorithms for cost-minimization are evaluated in this contribution. We define an optimal algorithm (Permutation) that through exhaustive search finds the best solution. We also define a greedy heuristic and another approximation (First-fit) to the optimal algorithm. For the sake of comparison in the later evaluation, we also introduce two naive algorithms: Round-robin and Random. For the sake of clarity, we omit requirements on affinity or anti-affinity between deployed instances but we remark that such requirements can be included as additional constraints, as presented in [9].

A. Random

The Random algorithm outlined in Algorithm 1 partitions the service into components and deploys each component to a random cloud provider (see lines 3 and 4 in Algorithm 1). After that, negotiations with cloud providers are performed (see line 9) accordingly. Note that in the presented algorithms, ***negotiate***(X, I) is an atomic operation that represents a bargaining action with a service request X against cloud provider I. This action returns the cost of hosting X in cloud provider I. If cloud provider I does not accept the request X (e.g., due to insufficient capacity), the cost is denoted as $+\infty$, indicating that hosting X in I is infeasible (see lines 10 and 11 in Algorithm 1).

Algorithm 1: Random($Components, Clouds$)

```
   Input: $Components = \{C_0, C_2, \ldots, C_{q-1}\}$,
          $Clouds = \{I_0, I_2, \ldots, I_{n-1}\}$
   /* Randomly map service components to
      clouds.                                 */
1  $mapping \leftarrow \emptyset$;
2  for $C \in Components$ do
3      $I_p \leftarrow$ Randomly select a cloud $I_p \in Clouds$;
4      $mapping[C] \leftarrow I_p$;
5  end
6  for $I \in Clouds$ do
7      $X \leftarrow \{C \in Components \mid mapping[C] = I\}$;
8      if $X \neq \emptyset$ then
9          $cost \leftarrow$ negotiate$(X, I)$;
10         if $cost = +\infty$ then
11             return N/A;
12         end
13     end
14 end
15 return $mapping$;
```

B. Round-robin

The Round-robin algorithm maps the service components to cloud providers in a circular fashion. While this algorithm does not in anyway strive to find an optimal solution, it is simple and fast. The algorithm is outlined in Algorithm 2. For a component C in the service, if there is no available cloud provider that can host it within n rounds of negotiation, the algorithm fails (see lines 18 and 19).

C. Greedy

The Greedy algorithm strives to find the best match for each component, without considering how this affects the other components. For many problem classes, this type of algorithm tends to give good results while being simple and easy to implement. The Greedy algorithm used in this evaluation is outlined in Algorithm 3. For each component C in the service, the lowest cost provider is selected to host it (see lines $6-13$).

Algorithm 2: Round-robin($Components, Clouds$)

```
Input: $Components = \{C_0, C_2, \ldots, C_{q-1}\}$,
       $Clouds = \{I_0, I_2, \ldots, I_{n-1}\}$
/* Map service components to clouds
   using a Round-robin strategy.          */
$mapping \leftarrow \emptyset$;
$r \leftarrow 0$;
for $C \in Components$ do
    $round \leftarrow 0$;
    while $round < n$ do
        $p \leftarrow r\%n$;
        $X \leftarrow \{c \in Components \mid mapping[c] = I_p\}$;
        $X \leftarrow X \cup \{C\}$;
        $cost \leftarrow$ negotiate$(X, I_p)$;
        if $cost = +\infty$ then
            $r$++;
            $round$++;
            continue;
        end
        $mapping[C_i] \leftarrow I_p$;
        break;
    end
    if $round \geq n$ then
        return N/A;
    end
end
return $mapping$;
```

Algorithm 3: GreedyAlg($Components, Clouds$)

```
Input: $Components = \{C_0, C_2, \ldots, C_{q-1}\}$,
       $Clouds = \{I_0, I_2, \ldots, I_{n-1}\}$
/* Map service components to clouds
   using a greedy strategy.               */
$mapping \leftarrow \emptyset$;
$costMap \leftarrow \emptyset$;
for $C \in Components$ do
    $cost \leftarrow +\infty$;
    $destination \leftarrow$ N/A;
    for $I \in Clouds$ do
        $currentCost \leftarrow costMap[I]$
        $X \leftarrow \{c \in Components \mid mapping[c] = I\}$;
        $X \leftarrow X \cup \{C_i\}$;
        $newCost \leftarrow$ negotiate$(X, I)$;
        if $newCost - currentCost < cost$ then
            $cost \leftarrow newCost - currentCost$;
            $destination \leftarrow I$;
        end
    end
    if $cost = +\infty$ then
        return N/A;
    end
    $costMap[destination]$ += $cost$;
    $mapping[C] \leftarrow destination$;
end
return $mapping$;
```

D. Permutation

The optimal Permutation algorithm, outlined in Algorithm 5 evaluates all possible permutations of components on all providers to find the global maximum. It first generates all possible partitions of a set of components. In combinatorics, the number of possible partitions of a set of size n is referred to as the *Bell number*, and denoted by B_n. This can be calculated as $B_n = \sum_{i=0}^{n} \left\{ {n \atop i} \right\}$, where $\left\{ {n \atop i} \right\}$ is the *Stirling number of the second kind* which is the number of ways to partition a set of n objects into i non-empty subsets [11]. For optimization purpose, given the number of cloud providers k, we only need to generate $B'_{n,k} = \sum_{i=0}^{k} \left\{ {n \atop i} \right\}$ partitions for a service with n components using Algorithm 5.

All generated partitions are then evaluated via a recursive exhaustive search algorithm presented in Algorithm 4. To narrow the search space as much as possible, a branch-and-cut strategy is adopted. For each component set Ω in a partition, two different branches are created to find a better solution. Each non-occupied cloud provider I is evaluated (see lines $11 - 26$). The first branch just skips placing Ω in I, and continues with the next non-occupied cloud provider, keeping the cumulative cost unchanged. The other branch evaluates whether the sum of the cumulative cost and the cost of placing Ω in I is higher than the optimum already obtained. If so, the current branch is stopped; otherwise, the algorithm continues with mapping Ω to cloud I and adding the corresponding cost to the cumulative cost.

E. First-fit

The First-fit algorithm is a simplification of the Permutation algorithm. As soon as a feasible solution is found, the algorithm exits. This means that there is no guarantee a global optimum is found. This algorithm is outlined in Algorithm 5 (same as the optimal Permutation algorithm) but uses a parametric setting $firstfit = true$ to halt upon finding the first feasible solution.

V. Evaluation

Multiple external factors affect the results of the service deployment algorithms, including provider pricing schemes and workloads, SLA-tiered pricing, and specifications of the service(s) to deploy. We try to make reasonable assumptions about these factors in our evaluation and to avoid bias, the results are interpreted at a higher level, focusing on the overall trends rather than on exact numbers. The purpose of the evaluation is to highlight the conceptual differences between the proposed algorithms in as realistic environments as possible. The evaluation setup is discussed in detail below.

A. Evaluation setup

1) Testbed configuration: The tests are run on 3.30 GHz Intel Core i5-2500 machines with 8 GB of RAM and Gigabit Ethernet. The operating system is Linux stable 3.6. We host

Algorithm 4: TraversePartition($tmapping$, $partition$, $Clouds$, $indicator$, $cost$, $firstfit$)

```
   /* $partition = \{\Omega_0, \Omega_1, \ldots, \Omega_{m-1}\}$ where
      $Components = \bigcup_i \Omega_i$ and
      $\forall i, j \in [0, m-1], i \neq j : \Omega_i \cap \Omega_j = \emptyset$        */
 1 if $firstfit = true$ && $optimum \neq +\infty$ then
 2     return;
 3 end
 4 if $indicator \geq m$ then
 5     if $tmapping.size() = m$ then
 6         $optimum \leftarrow cost$;
 7         $mapping \leftarrow tmapping$;
 8     end
 9 end
10 else
11     for $I \in Clouds$ do
12         $r \leftarrow cost$;
           /* check if $I$ is not occupied.   */
13         if $\forall\Omega \in partition$ && $tmapping[\Omega] \neq I$ then
14             $c \leftarrow$ negotiate$(\Omega_{indicator}, I)$;
15             if $cost + c < optimum$ then
16                 $tmapping[\Omega_{indicator}] \leftarrow I$;
17                 $cost \leftarrow cost + c$;
18                 TraversePartition($tmapping$, $partition$,
                   $Clouds$, $indicator + 1$, $cost$);
19             end
20         end
21         else
22             continue;
23         end
24         $cost \leftarrow r$;
25         $tmapping[\Omega_{indicator}] \leftarrow \emptyset$;
26     end
27 end
```

Algorithm 5: PFFAlg($Partitions, Clouds, firstfit$)

```
   Input: $Partitions = \{G_0, G_1, \ldots, G_{p-1}\}$,
          $Clouds = \{I_0, I_1, \ldots, I_{n-1}\}$
   /* Traverse each patition $G \in Partitions$,
      and store the optimal mappings
      between $Partitions$ and $Clouds$          */
 1 $mapping \leftarrow \emptyset$;
 2 $optimum \leftarrow +\infty$ ;
 3 for $G \in Partitions$ do
       /* $tm$ is a temporary mapping.      */
 4     $tm \leftarrow \emptyset$;
 5     TraversePartition( $tm$, $G$, $Clouds$, 0, 0, $firstfit$ );
 6 end
 7 if $optimum \neq +\infty$ then
 8     return $mapping$;
 9 end
10 return N/A;
```

TABLE I
HARDWARE METRICS FOR INSTANCE TYPES.

Instance Type	small	medium	large	xlarge
CPU(#cores)	1	2	4	8

TABLE II
CONFIGURATIONS.

Tiers	no. of instances	instance types	no. of configurations
FE	$1 \sim 4$	S, M	14
LO	$1 \sim 4$	M, L	14
DB	$1 \sim 2$	L, XL	5

three cloud providers on this testbed. These providers are configured with the Optimis cloud toolkit [10], which is a set of independent components. The Optimis toolkit can be adopted, either in full or in part, by cloud providers that provide infrastructure resources and by service providers that use these capacities to deliver services. In the evaluation, we use three components from the Optimis toolkit. The first is *SLA Management*, a service and client based on the WS-Agreement protocol [4] which is used for negotiating and creating SLAs between cloud providers and service providers [13]. The second is *Admission Control*, which is responsible for accepting or rejecting services for deployment in a provider. For clarity, the Admission Control algorithm used is a simple threshold-based function that accepts service requests if there is enough capacity. The third is the (previously discussed) SDO, which implements the service deployment and placement processes. For the purpose of this evaluation, the SDO is modified by implementing the placement algorithms discussed in Section IV.

2) Services: We use a service consisting of a typical three-tier Web application comprised of a front-end (FE), a logic (LO) and a database (DB) tier in the evaluation. To evaluate the algorithms against different services with diverse configurations, we vary the number of instances and the instance types for each tier, as shown in Table II. In addition, we use four different sizes of VMs, defined in terms of number of CPU cores used, as presented in Table I. We also use three different service availability SLAs, namely Bronze, Silver and Gold, which are summarized in Table III.

By varying these parameters, we get a considerable amount of different services. For example, the number of instances in the FE tier ranges from 1 to 4, and for each instance, its type is defined to be *small* or *medium*. Thus, there are 14 ($\sum_{i=1}^{4} C_{i+1}^{1}$) different configurations for the FE tier. Finally, incorporating the SLA types listed in Table III, we end up with 2940 ($14 \times 14 \times 5 \times 3$) different services to evaluate our placement algorithms with.

3) Cloud providers: Three cloud providers with different average background load and free capacity are used in our experiments. In this contribution, we assume that the total capacity of each provider is uniform, i.e., 96 unit-capacity VMs. To be able to define a dynamic provider pricing scheme based on real-world conditions, the background load of the cloud providers must be realistic. To model the background loads for these cloud providers, we use the Google cluster data trace [20], which consists of 30 days of usage data for a $12k$-machine cell in May 2011.

The workload consist of traces for jobs, tasks and machines and contains over 60 GB of data. To get the total CPU load of the cluster as a relative value, we aggregate the CPU usage of all running tasks and divide this absolute value with the total capacity of the cluster. We use data from a 24 hour period and as seen in Figure 2. The load, represented by the blue line, varies between $15\% - 60\%$ during this interval. In order to highlight the impact of the dynamic provider pricing schemes, we scale the CPU load by a factor of 1.6 so that the load instead varies between $25\% - 96\%$. This scaled load is represented by the black line in Figure 2.

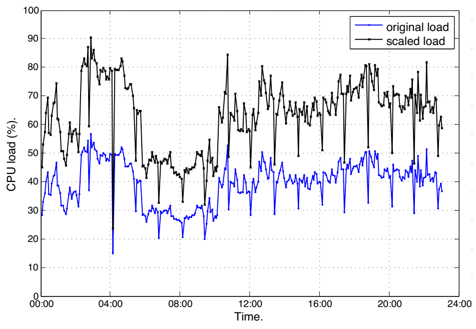

Fig. 2. CPU loads of the cluster.

In each of the 2940 tests, one point in Figure 2 is randomly chosen as the background loads for each of the three cloud providers. These background loads are then combined with the service size and required SLA level by the provider when pricing a service request, as discussed below.

4) Dynamic pricing strategy: In order to evaluate the five placement algorithms, the providers must support dynamic pricing. Since little work has been done in researching how such schemes should work asides from modeling how prices are set for Amazon spot instances through reverse engineering [7], we therefore chose to implement a simplistic dynamic pricing scheme for the purpose of the evaluation.

Our dynamic pricing scheme is a straightforward function where the price set for a service offer is defined as a function of three factors: the background workload of the cloud provider, the size of the service request, and the required SLA level (availability). A step-function is defined to incorporate the required SLA level by multiplying the unit price depending on the currently available capacity. This means that the price for deploying a service with a high availability requirement is inversely proportional to the free capacity of the provider, and that highly loaded cloud providers charge more for high availability. The pricing function is illustrated in Table III.

TABLE III
SLA PRICING SCHEME: FACTOR g VS. BACKGROUND LOADS + AVAILABILITY.

Background \ Availability	Bronze (90%)	Silver (95%)	Gold (98%)
$0\% \sim 50\%$	1	1	1
$50\% \sim 70\%$	1	2	4
$70\% \sim 85\%$	1	4	8
$85\% \sim 100\%$	1	8	16

The pricing scheme for the cloud providers are given by:

$$f(X, I) = g(X, I) \times \frac{X}{C(I)},$$

where X denotes the size of the service request, $C(I)$ represents the available capacity of cloud provider I, and g is the step-function defined in Table III.

With these configurations, we evaluate the behavior of the studied algorithms in terms of execution time, rounds of negotiations, and quality of solutions.

B. Evaluation results

Figure 3 and Figure 4 present the execution time (in seconds) for each algorithm with respect to the number of components in a service request. The value is given both as the average value and the standard deviation. As illustrated, the execution time for the Permutation algorithm increases dramatically as the size of the service becomes larger. This is due to the fact that Permutation algorithm is in essence a brute-force approach, which iterates through the search space that contains all possible partitions of components set. The number of partitions $B'_{n,k}$ rises rapidly as the number components n increases, e.g., when $n = 10$, and the number of cloud providers $k = 3$, the number of components are $B'_{10,3} = 9842$.

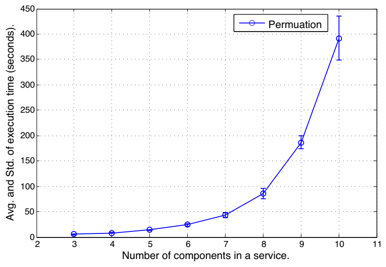

Fig. 3. Execution time for algorithms.

In contrast, the Random, Greedy, Round-robin, and First-fit algorithms require much shorter execution time. The execution time of Greedy and Round-robin increases linearly with different slopes for increasing number of components. Considering the fact that the execution time for an algorithm is proportional to the number of negotiations, we do not present the numbers of negotiations here. In our evaluation, a round of negotiation lasted for around 0.6 seconds.

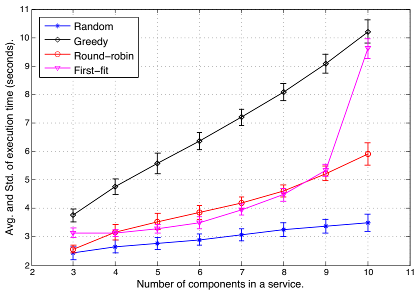

Fig. 4. Execution time for algorithms.

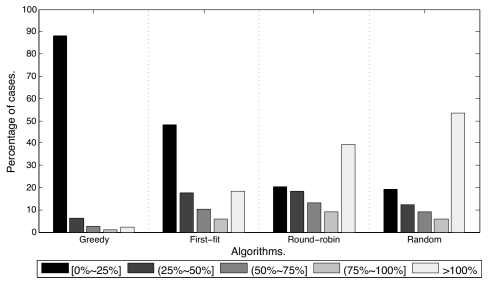

Fig. 5. Distribution of additional costs for deployment for non-optimal algorithms compared with permutation one.

To evaluate the ability of algorithms to find optimal solutions, we reduce the total capacity of the providers by half, i.e., 48 single core VMs, and repeat the same experiments. As presented in Table IV, some of the algorithms now fail to find an optimal solution due to this capacity constraint of the providers.

TABLE IV
PERFORMANCE RESULTS FOR THE DEMONSTRATED ALGORITHMS.

Algorithms	Optimal solution	Suboptimal solution	No solution
Permutation	2927	0	13
First-fit	23	2904	13
Round-robin	2	2866	72
Greedy	1630	1259	51
Random	6	2125	809

After investigating the ability of the algorithms to find a solution at all, we next study the quality of the found solutions. We quantify the distance between suboptimal solutions and optimal solutions by defining the cost overhead, α, as

$$\alpha = \frac{cost - optimum}{optimum}, \tag{1}$$

where $cost$ is the cost of deploying the service request gained by the algorithm, and $optimum$ is the cost of the optimal solution for the service. The cost for cases where no solution is found is denoted by $+\infty$. A lower α indicates a better solution, as it is closer to the optimal solution.

Notably, the Permutation algorithm always finds the optimum solution if one exists. Therefore, for the Permutation algorithm, $\alpha = 0$ is always true. For the other algorithms, since $cost \geq optimum$, we have $\alpha \geq 0$.

We divide α values into 5 intervals, i.e., $[0\% - 25\%]$, $(25\% - 50\%]$, $(50\% - 75\%]$, $(75\% - 100\%]$, and $(100\% - +\infty)$. We then calculate the percentage of solutions whose α-values fall into each interval for each algorithm and present the results in Figure 5. The Permutation algorithm is not included since its solutions always have an α-value of 0.

Finally, to compare the performance of the algorithms with each other, we employ a similar metric as Equation (1):

$$\beta = \frac{cost_1 - cost_2}{cost_2},$$

where $cost_1$ an $cost_2$ are two solutions gained by two compared algorithms respectively.

TABLE V
PERFORMANCE COMPARISON.

Algorithms	Permutation	First-fit	Round-robin	Greedy	Random
Permutation	0	76.6%	173.6%	7.62%	163.2%
First-fit	–	0	52.9%	−22.6%	47.0%
Round-robin	–	–	0	−40.1%	−0.86%
Greedy	–	–	–	0	155.2%
Random	–	–	–	–	0

We aggregate the results and present the average of β values in Table V. A value in the table represents an average β, i.e., the comparison result for the algorithms in the corresponding column and row, respectively.

From Figure 5 and Table V, we conclude that the Greedy algorithm performs very well. For 90% of all 2940 services, the deployment cost using Greedy is within 25% of the optimal cost. The corresponding number for first-fit is around 50%. Conversely, Round-robin and random perform much worse, with deployment costs twice that of the optimal solution in almost half of the cases.

To summarize the evaluation, Permutation is the best algorithm for finding optimal solutions. As it evaluates the entire search space, it either finds optimal solutions, or confirms that no solution is available. The downside of the Permutation algorithm is that the number of negotiation rounds grows rapidly and thus, execution time quickly grows infeasible. The First-fit algorithm on the other hand terminates when the first feasible solution is found, if one exists. This means that the quality of the solution always is the same as the first solution obtained by the Permutation algorithm. Consequently, the results shown in Table IV demonstrates that most of the solutions generated by the First-fit algorithm are suboptimal. Similar results are observed on Round-robin and Random and we also remark that while the Random algorithm by chance might generate the optimal solution, it also has the largest percentage of no solution found cases. Interestingly, the very fast Greedy algorithm finds optimal solutions in more than half of all cases, and for 90% of the rest of cases, the quality of solution is within 25% from optimal. Greedy thus seems to be a very good trade-off between the quality of the solution and execution time.

As discussed previously, the exact numbers in the evaluation depend on the provider pricing schemes, background workload, size and composition of services, etc. However, we observe that the Greedy algorithm seems to perform very well.

VI. Concluding remarks

In this contribution, we study a series of algorithms for cost-optimal cloud service deployment under dynamic pricing schemes. We perform an experimental evaluation using simulated deployments on cloud providers with dynamic pricing schemes. We then compare the algorithms with respect to execution time, ratio of successfully solved deployment cases, and the quality of the solution. Our experiments suggest that the greedy algorithm is a promising approach as it is very fast and also finds good solutions in most cases.

We believe that results of this research could be helpful in the design of scheduling algorithms and mechanisms in cloud environments with dynamic pricing schemes.

VII. Acknowledgments

We are grateful to the anonymous reviewers for their constructive and valuable feedback, improving the quality of this work. We also thank Ahmed Ali-Eldin for sharing his knowledge and materials on workload analysis. Financial support has in part been provided by the European Community's Seventh Framework Programme ([FP7/2010-2013]) under grant agreements no. 257115 (OPTIMIS), the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control, and the Swedish Government's strategic effort eSSENCE.

References

[1] GoGrid, http://www.gogrid.com, visited July 2013.
[2] Amazon, http://aws.amazon.com, visited July 2013.
[3] Amazon EC2 Spot Instance, http://aws.amazon.com/ec2/spot-instances/, visited July 2013.
[4] Web Services Agreement Specification (WS-Agreement). http://www.ogf.org/documents/GFD.107.pdf, visited July, 2013.
[5] A. Andrzejak, D. Kondo, and S. Yi. Decision model for cloud computing under sla constraints. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 257–266, 2010.
[6] D. Armstrong, D. Espling, J. Tordsson, K. Djemame, and E. Elmroth. Runtime Virtual Machine Recontextualization for Clouds. In *Euro-Par Workshops*, volume 7640 of *Lecture Notes in Computer Science*, pages 567–576. Springer, 2012.
[7] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing amazon ec2 spot instance pricing. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom 2011)*, pages 304–311, 2011.
[8] D. Breitgand and A. Epstein. SLA-aware Placement of Multi-Virtual Machine Elastic Services in Compute Clouds. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 161–168, May 2011.
[9] D. Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth. Modeling and Placement of Structured Cloud Services. Submitted to IEEE Transactions on Network and Service Management.
[10] 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*, 28(1):66–77, 2012.
[11] R. Jain and N. S. Chaudhari. A New Bit Wise Technique for 3-Partitioning Algorithm. *IJCA Special Issue on Optimization and On-chip Communication*, OOC(1):1–5, February 2012. Foundation of Computer Science, New York, USA.
[12] B. Javadi, R. K. Thulasiramy, and R. Buyya. Statistical Modeling of Spot Instance Prices in Public Cloud Environments. In *Proceedings of the Fourth IEEE International Conference on Utility and Cloud Computing (UCC 2011)*, pages 219–228, 2011.
[13] A. Lawrence, K. Djemame, O. Wäldrich, W. Ziegler, and C. Zsigri. Using Service Level Agreements for Optimising Cloud Infrastructure Services. In *Proceedings of the 2010 International Conference ServiceWave*, ServiceWave'10, pages 38–49. Springer-Verlag.
[14] W. Li, P. Svärd, J. Tordsson, and E. Elmroth. A General Approach to Service Deployment in Cloud Environments. In *Proceedings of the 2nd International Conference on Cloud and Green Computing (CGC 2012)*, pages 17–24, Washington, DC, USA, 2012. IEEE Computer Society.
[15] W. Li, J. Tordsson, and E. Elmroth. Virtual Machine Placement for Predictable and Time-Constrained Peak Loads. In *Proceedings of the 8th International Conference on Economics of grids, clouds, systems, and services (GECON'11)*. Lecture Notes in Computer Science, Vol. 7150, Springer-Verlag, pp. 120-134, 2011.
[16] 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)*, pages 163–171, 2011.
[17] J. Lucas Simarro, R. Moreno-Vozmediano, R. Montero, and I. Llorente. Dynamic placement of virtual machines for cost optimization in multi-cloud environments. In *Proceedings of the International Conference on High Performance Computing and Simulation (HPCS)*, pages 1–7, 2011.
[18] J. L. Lucas-Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Cost Optimization of Virtual Infrastructures in Dynamic Multi-cloud Scenarios. *Concurrency and Computation: Practice and Experience*, 2012.
[19] H. Rasheed, A. Rumpl, O. Wäldrich, and W. Ziegler. OPTIMIS Service Manifest Scientific Report. Scientific Report, Fraunhofer-Institute for Algorithms and Scientific Computing (SCAI), 2013. http://www.optimis-project.eu/sites/default/files/content-files/document/optimis-public-deliverable-service-manifest-scientific-report.pdf, visited July 2013.
[20] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google cluster-usage traces. http://code.google.com/p/googleclusterdata/, visited July 2013.
[21] B. Sharma, R. K. Thulasiram, P. Thulasiraman, S. K. Garg, and R. Buyya. Pricing Cloud Compute Commodities: A Novel Financial Economic Model. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012)*, pages 451–457, 2012.
[22] B. Sotomayor, R. S. Montero, I. Llorente, and I. Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. *Internet Computing, IEEE*, 13(5):14–22, 2009.
[23] J. Tordsson, R. Montero, R. Moreno-Vozmediano, and I. Llorente. Cloud Brokering Mechanisms for Optimized Placement of Virtual Machines across Multiple Providers. *Future Generation Computer Systems*, 28(2):358–367, 2012.
[24] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove. Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads. In *Proceedings of the IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, pages 228–235, 2010.
[25] S. Wee. Debunking Real-time Pricing in Cloud Computing. In *Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011)*, pages 585–590, 2011.
[26] Q. Zhang, Q. Zhu, and R. Boutaba. Dynamic Resource Allocation for Spot Markets in Cloud Computing Environments. In *Proceedings of the Fourth IEEE International Conference on Utility and Cloud Computing (UCC 2011)*, pages 178–185, 2011.

V

Paper V

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
{*espling, larsson, wubin.li, 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 collocation 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 conclusion 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: cloud computing; service management; service structure; placement; affinity; collocation

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 collocation 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 conclusion 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. A *type* in this paper 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.

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

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. 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 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 [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.

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 the scalability issues, 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 as an example of 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 closed 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 affect 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.

1) 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 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. in [19], and Koch et. al. in [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 [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 in this manner. 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 re-placement. 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 [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 presented in [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.

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

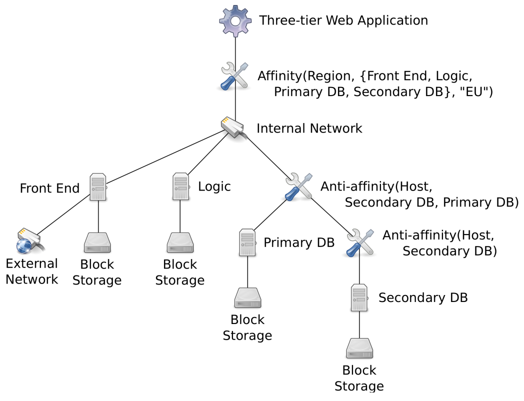
Three-tier Web Application
Affinity(Region, {Front End, Logic, Primary DB, Secondary DB}, "EU")
Internal Network
Front End
Logic
Anti-affinity(Host, Secondary DB, Primary DB)
Primary DB
Anti-affinity(Host, Secondary DB)
External Network
Block Storage
Block Storage
Block Storage
Secondary DB
Block Storage

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 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., 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) AntiAffinity(Region, A, Sweden)
2) Affinity(Site, A, B)
3) Affinity(Site, B)
4) AntiAffinity(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").
- 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.

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

C. Constraint Model

Placement constraints extracted from the service structure can be enforced by a placement engine with ability to handle various constraints, e.g. [27], [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.

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 has 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;

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 $h \in H$
$\text{cap}_h =$ capacity of host $h \in H$
$\text{req}_v =$ 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

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

Subject to:

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

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

$$\begin{aligned} &\forall v, w \in V, \forall t, u \in T, \forall h \in H : \\ &\quad \text{vcons}_{t,u} = -1 \land v_t = t \land w_t = u \implies \\ &\quad m_{v,h} + m_{w,h} \leq 1 \end{aligned} \tag{10}$$

$$\begin{aligned} &\forall v \in V, \forall t, u \in T, t \neq u : \\ &\quad \text{vcons}_{t,u} = 1 \land v_t = t \implies \\ &\quad \exists w \in V, w_t = u, w \neq v : \\ &\qquad \forall h \in H : m_{v,h} = m_{w,h} \end{aligned} \tag{11}$$

$$\begin{aligned} &\forall v, w \in V, \forall t \in T, \forall h \in H : \\ &\quad \text{vcons}_{t,t} = 1 \land v_t = t \land w_t = t \implies \\ &\quad m_{v,h} = m_{w,h} \end{aligned} \tag{12}$$

$$\begin{aligned} &\forall v \in V, \forall t \in T, \forall h \in H : \\ &\quad \text{hcons}_{t,h} = -1 \land v_t = t \implies m_{v,h} = 0 \end{aligned} \tag{13}$$

$$\begin{aligned} &\forall v \in V, \forall t \in T, \forall h \in H : \\ &\quad \text{hcons}_{t,h} = 1 \land v_t = t \implies m_{v,h} = 1 \end{aligned} \tag{14}$$

(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, 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], [28] and heuristics are used to mitigate the effects of scale on the solvability and solution time. In this work we employ ILP techniques which provides the optimal solution without heuristics, and so scalability becomes a major concern. It has been previously shown that ILP based scheduling approaches can achieve very good results for scheduling, especially for medium-sized problems [29].

Improving the scalability of ILP solutions and solvers is outside scope of this work, and instead we mitigate the problem of scalability by performing scheduling on a per-service level. For each scheduling iteration, the background load on hosts is comprised of the capacity requirements of other services that have been deployed previously. Hence, a truly optimal solution that maximizes resource usage needs to consider all components of all services. This is, however, not feasible to do in reasonable time, since it is merely the original NP-hard scheduling problem in a slightly different form. Furthermore, re-scheduling already deployed VMs can cause service interruptions, and the resulting downtime may lead to SLA violations. The relaxation of considering each service in isolation, and only regarding the capacity consumption of all others as background load, makes the input to the scheduler sufficiently small to be manageable.

As previously outlined, our ILP model contains a set of VM types, a set of VM instances, and a set of hosts. The number of VM types in a service rarely exceeds a handful, and does therefore not have a major impact on scalability. The number of VM instances may be in the order of houndreds, but as they are spawned from the same (small set of) VM types, the number of permutations needed to be explored is very much reduced. The critical factor is therefore the number of available hosts, which in a cloud environment may be very large. This results in a trade-off between scalability and optimality of the solution. Subdividing the servers of the data center by, e.g., cluster by racks or by applying a heuristic gives scalability, but not a globally optimal solution.

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 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 VMs instances of each type and their capacity requirements are shown in Table IIIa. As shown in Table IV, 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 IIIb. 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.

The evaluation is carried out by generating a large set of cases with varying amounts of *AA*-constraints and background load

Table III
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 IV
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**

Table V
OVERALL TENDENCY AS IMPACT FACTORS INCREASE.

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

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 [30] modeling language and solved with the Gurobi [31] 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 then 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.

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 V, 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 2 and 4 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 2 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.

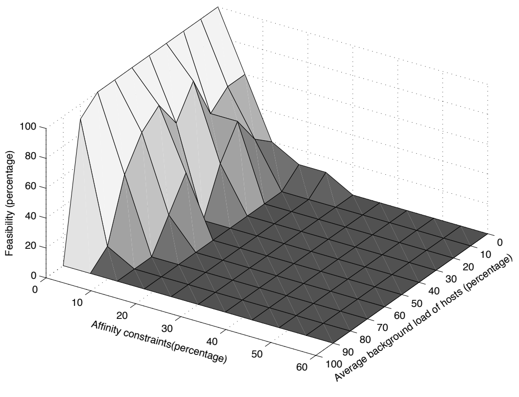

Figure 2. Feasibility depending on affinity constraints and background load.

Figure 3 shows affinity when combined with anti-affinity (at a constant background load of zero). As is evident when comparing Figure 2 and 3, 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.

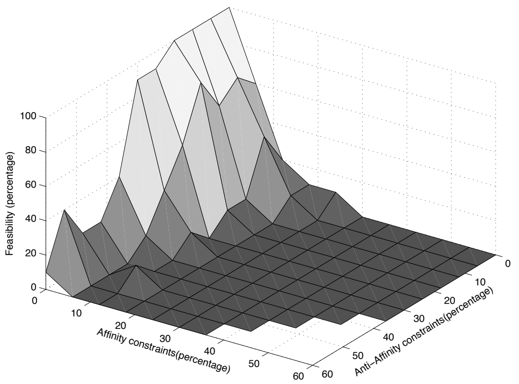

Figure 3. Feasibility depending on affinity- and anti-affinity constraints.

3) Impact of Anti-Affinity Constraints: Another analysis was performed to compare the impact between anti-affinity and background load (illustrated in Figure 4). 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.

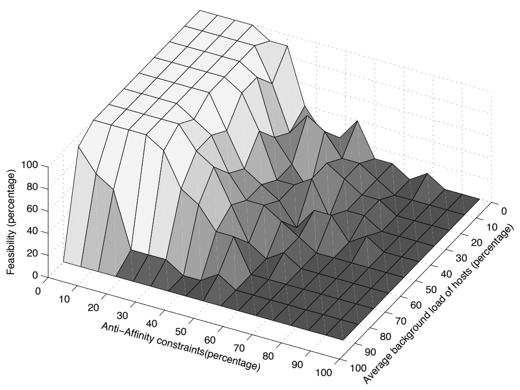

Figure 4. Feasibility depending on anti-affinity constraints and average background load.

4) Timeouts and Execution Time: Figure 5 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:

- 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 2) 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 V, the feasibility of placing a service decreases as the background load and number of *AA*-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

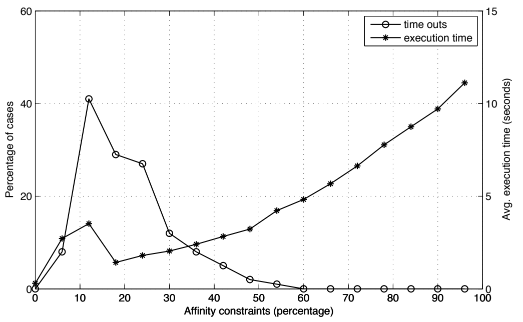

Figure 5. Timeouts and execution time vs. affinity constraints

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.

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 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.

[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] 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.

[28] 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.

[29] 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.

[30] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, Nov. 2002.

[31] 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 is a Ph.D. candidate at the Department of Computing Science, Umeå University, Sweden, and received a Ph.L. (licentiate) degree in 2012. Before that, he received his master’s degree in Computer Science 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 is currently working in 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.

VI

Paper VI

Continuous Datacenter Consolidation[*]

Petter Svärd, Wubin Li, Eddie Wadbro,
Johan Tordsson, and Erik Elmroth

Department of Computing Science, Umeå University
SE-901 87 Umeå, Sweden
{*petters, wubin.li, eddiew, tordsson, elmroth*}*@cs.umu.se*
http://www.cloudresearch.org

Abstract: Efficient mapping of Virtual Machines (VMs) onto physical servers is a key problem for cloud infrastructure providers as hardware utilization directly impacts revenue. Today, this mapping is commonly only performed when new VMs are created, but as VM workloads fluctuate and server availability varies, any initial mapping is bound to become suboptimal over time. We introduce a set of heuristic methods for continuous optimization of the VM-to-server mapping based on combinations of fundamental management actions, namely suspending and resuming physical machines, migrating VMs, and suspending and resuming VMs. Using these methods cloud infrastructure providers can continuously optimize their server resources regardless of the predictability of the workload. To verify that our approach is applicable in real-world scenarios, we build a proof-of-concept datacenter management system that implements the proposed algorithms. The feasibility of our approach is evaluated through a combination of simulations and real experiments where our system provisions a workload of benchmark applications. Our results indicate that the proposed algorithms are feasible, that the combined management approach achieves the best results, and that the VM suspend and resume mechanism has the largest impact.

Key words: Cloud Computing; Scheduling; Heuristic Methods; Consolidation; VM Migration; Power Management;

[*] Technical Report UMINF-14.08, March, 2014. Department of Computing Science, Umeå University.

Continuous Datacenter Consolidation

Petter Svärd, Wubin Li, Eddie Wadbro, Johan Tordsson, and Erik Elmroth

Department of Computing Science, Umeå University
SE-901 87 Umeå, Sweden
Email: {petters, wubin.li, eddiew, tordsson, elmroth}@cs.umu.se

***Abstract*—Efficient mapping of Virtual Machines (VMs) onto physical servers is a key problem for cloud infrastructure providers as hardware utilization directly impacts revenue. Today, this mapping is commonly only performed when new VMs are created, but as VM workloads fluctuate and server availability varies, any initial mapping is bound to become suboptimal over time. We introduce a set of heuristic methods for continuous optimization of the VM-to-server mapping based on combinations of fundamental management actions, namely suspending and resuming physical machines, migrating VMs, and suspending and resuming VMs. Using these methods cloud infrastructure providers can continuously optimize their server resources regardless of the predictability of the workload. To verify that our approach is applicable in real-world scenarios, we build a proof-of-concept datacenter management system that implements the proposed algorithms. The feasibility of our approach is evaluated through a combination of simulations and real experiments where our system provisions a workload of benchmark applications. Our results indicate that the proposed algorithms are feasible, that the combined management approach achieves the best results, and that the VM suspend and resume mechanism has the largest impact.**

***Keywords*-Cloud Computing; Scheduling; Heuristic Methods; Consolidation; VM Migration; Power Management.**

I. Introduction

To date, most research on Virtual Machine (VM) provisioning for cloud datacenters has focused on deploy time scheduling, typically formulated as assignment problems where VMs are mapped to Physical Machines (PMs). Common objectives for these formulations are to optimize criteria such as Service Level Agreements (SLAs) [5], provider revenue [9], performance [23], utilization [37] etc. or a combination thereof. Notably, there are several factors that complicates this problem. First of all, VM scheduling is an online problem as both the arrival rate of new VM requests and the completion time for provisioned VMs is unknown. In addition, resource usage of individual VMs also varies over time. Changes to the server pool, due to failures or energy-management actions such as power-off and frequency-scaling, can also impact the performance of deployed VMs.

These factors imply that any scheduling solution may become suboptimal over time. To address this, we propose a continuous VM remapping approach to optimize VM provisioning as a complement to VM scheduling. Our approach consists of a set of algorithms that enable cloud infrastructure providers to automatically reconfigure the mapping of VMs to PMs and adapt to the changes in workloads and the physical environment. These algorithms are based on a combination of management actions, i.e., suspend and resume of PMs, live VM migration, and suspend and resume of VMs. This *continuous datacenter consolidation* approach aims to maximize cloud provider revenue over time by minimizing power consumption, maximizing PM utilization, and prioritizing important VM requests.

The remainder of the paper is organized as follows. Section II describes our system model and the assumptions made. Section III briefly elaborates on the problem and outlines our optimization algorithms. Section IV describes the architecture and implementation of our continuous datacenter consolidation engine. Section V presents the experimental evaluation on synthetic workload. Section VI surveys the related work on datacenter management, autonomic computing, and VM scheduling. Our conclusions are given in Section VII followed by a presentation of future work, acknowledgments, and a list of references.

II. System Model Overview

This work is based on a model where a datacenter consists of a set of PMs that are used to provision VMs on behalf of users. For each time unit the datacenter provisions a VM for a user, a profit r is generated, whereas a penalty f is imposed to the provider if the VM is not running for a time unit. In addition to any such SLAs, penalties, cloud providers pay electricity costs for running the PMs. To model the power consumption P of a physical server, as a function of the CPU utilization u, we adopt the following model presented by Blackburn [24]:

$$P(u) = P_{idle} + (P_{max} - P_{idle})u, \qquad (1)$$

where P_{max} and P_{idle} are the power consumption for a server at full load and at idle respectively. In this model, server power consumption scales linearly with CPU utilization. So for example, a server with $P_{idle} = 200$ W, $P_{max} = 300$ W, and 10% CPU utilization requires $P(10\%) = 200\text{ W} + (300\text{ W} - 200\text{ W}) \times 10\% = 210$ W. Through empirical measurement of various servers, this approximation has proven by Baroso et al. to be accurate to within $\pm 5\%$ across all CPU utilization rates [4]. They also argue that although the power consumption of the CPU only accounts for roughly 40% of the power usage of a server, it can be used to model the total usage [4].

Finally, the power consumption of the whole data center can be expressed as

$$P_{tot} = \sum_{i|\text{physical machine } pm_i \text{ is active}} P(u_i),$$

where u_i denotes the CPU utilization of physical machine pm_i. The monetary cost of running the data center during a time interval $(0, t)$ is therefore given by

$$C = \frac{m}{1000} \int_0^t P_{tot},$$

where m represents the unit cost per kWh.

When mapping VMs to PMs it is possible to provision more virtual resources than what is physically available in the PM, a concept known as overbooking [33]. However, some infrastructure providers e.g., Amazon EC2, provide a one-to-one mapping between virtual and physical CPU and memory resources [10], which means that PMs are not being overbooked. Within the context of this work, we assume that no overbooking is taking place.

Research on proactive optimization based on the historical data and workload prediction has been performed extensively, by e.g. Ali-Eldin et al. [2]. In our model, we assume that the infrastructure provider has no knowledge of the future workload and no prediction is taking place. However, we believe that it is possible to enhance our approach with such prediction techniques.

A core technology to optimize the mapping of VMs to PMs during operation of the datacenter is live VMs migration [6]. Live VM migration allows a running VM to be moved from one PM to another without shutting it down. Live migration thus reduces SLA penalties as the VM is accessible by users during migration, but at the cost of migration taking considerable time to complete and consuming much hardware resources. To reduce both migration time and resource usage, we use the KVM XBZRLE delta compression migration algorithm [32] in our work as delta compression can significantly reduce the migration downtime, migration time and the amount of transmitted data during live migration for memory-intensive workloads [32], [36]. The total migration time of a VM is given by

$$dt = t_i + t_c + t_s + t_r,$$

where t_i, t_c, t_s, and t_r denote the time for iteratively transferring memory pages, the time for suspending the VM at source, time for CPU/BIOS transfer and the time for resuming the VM at destination, and pulling respectively. The three latter operations are usually fast and vary little between VMs, that is, t_c, t_s, and t_r are typically small. The iterative transfer time is harder to predict, but is usually a function of the active memory usage and the memory size of the VM and the network bandwidth b.

III. The proposed approach and heuristic methods

At any point in time, multiple events can take place. We define three events and prioritize them in descending order as *PM crash, VM exit*, and *VM arrival*. Our proposed approach dynamically handle these events according to their priorities, and adapts the cloud infrastructure to the environmental changes in a reactive manner. Simple management actions are used for optimization of the datacenter, i.e., (i) suspend/resume VMs, (ii) VM migration, and (iii) suspend/resume PMs. For an event of PM crash, a crashed PM not only affects all VMs hosted, but it must also be excluded as a potential destination for VMs. Upon an event of PM crash, we simply suspend all VMs hosted on that PM. A VM exit event has a higher priority than a VM arrival event, as capacity released by a terminated VM can be used to accept more VMs into the datacenter. When a VM terminates, the occupied resources are released, increasing the residual capacity of a PM. All VMs arriving are added to a list, namely, $candidateList$, which also may include VMs suspended in the past. VMs in $candidateList$ can be prospectively executed depending on the decision by the optimization process. A summary of the actions taken on the occurrence of events is shown in Algorithm 1.

Algorithm 1: handleEvents($events$)

```
1 for $e \in events$ do
2     if PM $pm$ crash then
3         Exclude $pm$ and all VMs hosted;
4     else if VM $vm$ quits then
5         Release capacity occupied by $vm$;
6     else if VM $vm$ arrives then
7         Add $vm$ to $candidateList$;
```

Once the event handling procedure is completed, a consolidation action presented in Algorithm 2 is triggered to optimize the profit gained by VM provision. In particular, if an infrastructure provider has too limited capacity, the profit can be maximized by selecting which VMs to run. In order to make an optimal selection, the following two questions need to be answered.

1. **Which VM should be placed first?**
Intuitively, given a set of VMs, the ones that are most profitable should be placed with higher priorities. However, in our model, we also need to consider the penalty of suspending a VM. In this contribution, we prioritize a VM using the sum of its associate profit and penalty. For example, given two VMs, vm_1 with profit r_1 and penalty f_1, and vm_2 with profit r_2 and penalty f_2, our algorithms place vm_1 prior to vm_2 if $r_1 - f_2 > r_2 - f_1$ (even when $r_2 > r_1$).

TABLE I
Example of VM prioritization.

Option	Profit	Penalty	Gain
Run vm_1, and suspend vm_2	r_1	f_2	$r_1 - f_2$
Run vm_2, and suspend vm_1	r_2	f_1	$r_2 - f_1$

Table I compares two different options with respect to potential gains from the perspective of the infrastructure provider. In this case, the first option is preferable if $r_1 - f_2 > r_2 - f_1$, or equivalently $r_1 + f_1 > r_2 + f_2$.

2. **Which VM should be selected to be replaced?**
Given a VM vm (with profit r and penalty f) that can not be hosted by any PM, it is possible to suspend another already running VM vm' (with profit r' and penalty f') in PM pm and instead run vm if (i) pm is capable of running vm (after the suspension of vm') and (ii) it is more profitable to run vm than vm'. Following the strategy aforementioned, the VM with minimum $(r'+f')$ is selected as the victim VM, i.e., the VM to be suspended.

Algorithm 2: consolidation()

```
   /* consolidation                                  */
1  if suspend/resume VM is allowed then
2      Add all suspended VMs to $candidateList$;
3  Sort VMs in $candidateList$ by $(price + penalty)$ in
   descending order;
4  for $vm \in candidateList$ do
5      handleVM($vm$);
6  if suspend/resume PM is allowed then
7      if VM migration is allowed then
           // Release PMs via VM migration.
8          releasePMsbyMigration();
9      else
           // Suspend PMs without VM running.
10         suspendIdlePMs();
```

In order to optimize the datacenter operation, our approach is to generate a list of *consolidation* actions according to Algorithm 2. If the used algorithm allows for suspend/resume of VMs, the first step is to recycle all the currently suspended VMs and enable them to be possibly resumed by adding them to $candidateList$ (see Line 2). All VMs (also referred to as *object* VMs) in $candidateList$ are to be handled sequentially, in a descending order that they are ranked by $(price + penalty)$. There are two possible outcomes of the action **handleVM**, i.e., either suspend the current VM, or place and start VM in some physical server. The final step in a round of optimization is to suspend idle PMs if the feature is enabled (see Line 6–10). More PMs may be released and then suspended, depending on whether VM migration is allowed.

As depicted in Line 2 in Algorithm 3, we use *best-fit* as the baseline strategy to find an active PM for a VM. The motivation for this is to load each PM as much as possible, maximizing the utilization of the PMs and thus minimizing the residual capacity of the whole infrastructure. If this is not feasible (i.e., no PM can host the VM), a simple solution is to try starting a suspended (or a new) PM (see Line 6) and place the VM there. However, in order to decrease the total number of active PMs, prior to starting a new PM, the proposed algorithm strives to readjust the placement of VMs, to see if there exists a PM that can host the VM after migrating some VMs to other PMs (see Line 4). The details of this can be found in Algorithm 4).

Algorithm 3: handleVM(vm)

```
   /* This function is for handling newly
      arrival VMs and VMs that were
      suspended in the previous period.  */
1  $pms \leftarrow$ active PMs;
   // Find a PM for $vm$ using the best-fit
      strategy.
2  $pm \leftarrow$ best-fit($vm, pms$);
3  if $pm$ not found and VM migration is allowed then
4      $pm \leftarrow$ findPMbyMigration($vm$);
5  if $pm$ not found then
6      $pm \leftarrow$ attempt to start a new PM;
7  if $pm$ not found and suspend/resume VM is allowed then
       // Find a victim VM and replace it
          with $vm$.
8      $pm \leftarrow$ findPMwithVictimVM($vm$);
9  if $pm$ found then
10     placeVM($vm, pm$);
11 else
12     suspendVM($vm$);
```

Finally, if no suitable PM is found after trying all of the above approaches, an aggressive approach is applied to pick one of the running VMs as the victim, suspend it, and replace it with the object VM (see Line 8, as described in Algorithm 5). This step is conducted only if replacing the victim with the object VM is feasible and more profitable. Note that our algorithm currently only selects one VM as victim, it is however possible to extend this to enable selection of multiple VMs as victims instead.

To find if re-arranging the mapping of VMs by live migration them can make room for vm on some PM, all active PMs are sorted by residual capacity in descending order in Algorithm 4. The PMs are then evaluated by looking at the feasibility of migrating a set of VMs to other PMs. When evaluating a PM, we only consider migrating VMs that are smaller than vm (see Line 6), as testing a VM larger than vm is meaningless (namely, if a PM can be found for this case, just place vm there without adjusting any VM placement). Also, note that as a machine can be represented by multiple dimensions (CPU, memory, storage, etc), the *size* function in Algorithm 4 can have different definitions depending on the application scenarios. In this work, it is defined in terms of CPU cores while other dimensions (memory, storage, etc.) are used as constraints when evaluating the feasibility of placement on PMs. The algorithm also strives to minimize the number of migrated VMs, as migration takes time and consumes resources. In addition, to further reduce the number of VMs migrated, all potential VMs are sorted by size in descending order (see Line 7). VMs to be migrated are added to a *plan* by function **addToMigratitonPlan** (see Line 13). The evaluation procedure stops when the first suitable PM is found, and the migration plan is executed by **commitMigratitonPlan** (see Line 20). If a PM is not suitable, the migration

Algorithm 4: findPMbyMigration(vm)

```
/* Find a PM that can host vm after
   migrating some VMs to other PMs.  */
1  $pms \leftarrow$ all active PMs;
2  Sort $pms$ by residual capacity in descending order;
3  for $p \in pms$ do
4      $feasible \leftarrow$ FALSE;
5      $vmSet \leftarrow$ VMs hosted in $p$;
6      $vms \leftarrow \{v \in vmSet \mid size(v) < size(vm)\}$;
7      Sort $vms$ by capacity in descending order;
8      $pmset \leftarrow pms \setminus \{p\}$;
9      for $v \in vms$ do
           // Find a PM (not p) to host v
              using the best-fit strategy.
10         $pm \leftarrow$ best-fit($v, pmset$);
11         if $pm$ not found then
12             break;
13         addToMigratitonPlan($v, pm$);
14         if $p$ can host $vm$ then
15             $feasible \leftarrow$ TRUE;
16             break;
17     if not $feasible$ then
18         cancelMigratitonPlan();
19         continue;
20     commitMigratitonPlan();
21     return $p$;
```

Algorithm 5: findPMwithVictimVM(vm)

```
/* Find a PM that can host vm after
   suspending a vm hosted.  */
1  $destination \leftarrow$ null;
2  $minRF \leftarrow price(vm) + penalty(vm)$;
3  $pms \leftarrow$ all active PMs;
4  for $p \in pms$ do
5      $vmSet \leftarrow$ VMs hosted in $p$;
6      $vms \leftarrow \{v \in vmSet \mid price(v) + penalty(v) < minRF\}$;
7      Sort $vms$ by $(price + penalty)$ in ascending order;
8      for $v \in vms$ do
9          if $p$ can host $vm$ after suspending $v$ then
10             $victim \leftarrow v$;
11             $destination \leftarrow p$;
12             $minRF \leftarrow price(v) + penalty(v)$;
13             break;
14 if destination is not null then
15     suspendVM($victim$);
16 return destination;
```

plan is canceled by **cancelMigratitonPlan** (see Line 18).

Algorithm 5 introduces the strategy of finding a victim VM to be replaced by a VM in $candidateList$. The basic idea is to select the VM with the minimum value of $(price + penalty)$ among all selectable VMs (see Line 9-13). Once a VM is selected as a victim VM, it is suspended and moved to a waiting list and may potentially be resumed depending on the future optimization decision.

The final action in each consolidation process is to try to reduce the power consumption of the infrastructure by suspending all idle PMs, or by releasing more PMs by VM migration. To empty an active PM, all hosted VMs need to be migrated to other PM(s). Intuitively, PMs with higher residual capacity are more likely able to be emptied, and thus PMs are evaluated in the order of residual capacity (see Line 2 in Algorithm 6). Once again, we use a *best-fit* strategy whenever finding a new location for a VM (see Line 8).

Finally, by enabling or disabling the three management actions identified, we end up with 8 algorithms to evaluate in Section V, as listed in Table II.

Notably, scheduling algorithms based on bin-packing [7] or knapsack approaches are exponential in complexity, limiting their applicability for large-scale problems. As our algorithms are all with polynomial complexity this means that they can be used for larger problem sizes than what is studied in our simulations.

Algorithm 6: releasePMsbyMigration()

```
/* Release PMs through VM migration  */
1  $pms \leftarrow$ all active PMs;
2  Sort $pms$ by residual capacity in descending order;
3  for $p \in pms$ do
4      $feasible \leftarrow$ TRUE;
5      $vms \leftarrow$ VMs hosted in $p$;
6      $pmset \leftarrow pms \setminus \{p\}$;
7      for $vm \in vms$ do
           // Find a PM (not p) to host vm
              using the best-fit strategy.
8          $pm \leftarrow$ best-fit($vm, pmset$);
9          if $pm$ not found then
10             $feasible \leftarrow$ FALSE;
11             break;
12         addToMigratitonPlan($vm, pm$);
13     if not $feasible$ then
14         cancelMigratitonPlan();
15         continue;
16     commitMigratitonPlan();
       // Suspend PM p when it is idle.
17     suspendPM($p$);
```

TABLE II
ALGORITHMS AND MANAGEMENT DIMENSIONS

algorithm	suspend/resume PMs	suspend/resume VMs	VM migration
baseline			
pmSR	✓		
vmSR		✓	
vmM			✓
vmSRpmSR	✓	✓	
vmMpmSR	✓		✓
vmMvmSR		✓	✓
combined	✓	✓	✓

IV. Architecture and Implementation.

To verify that our algorithms are valid in real-world scenarios, we design and implement a software package, Automatic Continuous Datacenter Consolidation (ACDC), capable of managing PMs and VMs using the algorithms described in Section III. PMs can be suspended and resumed, and VMs can be started, shutdown and migrated. The ACDC is built on the open-source KVM [18] hypervisor in combination with libvirt [22] to provide the virtualization backend.

The architecture consists of three components, as shown in Figure 1. All components are implemented in Java and the complete package runs on one PM, the controller. To perform consolidation actions on the worker PMs, where ACDC is not running, ssh remote invocation by means of shared key authentication is used. This means that remote libvirt virsh [22] commands for suspending and resuming VMs and migration of VMs, as well as scripts to suspend and resume PMs, can be executed by the controller PM.

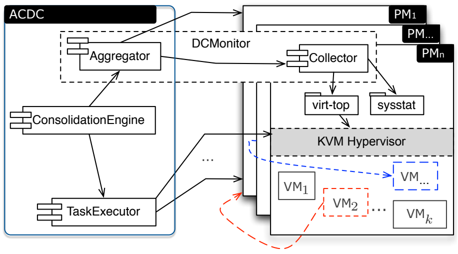

Fig. 1. Architecture overview.

A. DCMonitor.

To continuously collect the state information of the data center, we implement a monitoring system that consists of two subcomponents. A *Collector* installed in each physical machine, and an *Aggregator* is co-located with the ConsolidationEngine. The collector is built on two libraries, virt-top [13] and sysstat [1]. virt-top is a top[1]-like utility for showing stats of virtualized domains. It is employed to collect the cpu, memory and IO usage info of virtual machines. Note that virt-top only collects the total memory allocated to the guest, not the memory being used. The sysstat utilities are a collection of performance monitoring tools for Linux hosts. They are used to gather the resource usage info of physical hosts. Using the statistics collected by the Collector in each physical host, the Aggregator constructs a global view of the state of the data center.

[1]*top* is a task manager program inherent in many Unix-like operating systems.

B. ConsolidationEngine.

The ConsolidationEngine analyzes the data from the DCMonitor and decides on what actions to take in order to optimize the operation of the datacenter, according to the selected algorithm. All algorithms in Table II are evaluated through simulation. In addition to this, the combined algorithm and the baseline algorithm are also used to verify the simulation results for small-scale real experiments. The output from the ConsolidationEngine is an Execution Plan which is an ordered list of optimization actions.

C. TaskExecutor.

The TaskExecutor is designed to perform the actions produced by the ConsolidationEngine. It is running as a daemon, waiting to accept Execution Plans from the ConsolidationEngine. Once an Execution Plan arrives, it is processed and the tasks are performed in order. The TaskExecutor controls the underlying virtualized infrastructure by using libvirt virsh commands.

V. Evaluation

We compare the eight algorithms defined in Table II by comparing the impact of each management action in isolation and combination on provider profit, PM utilization, number of running and suspended VMs. This is done both through simulations and real experiments. Note that, in order to increase the readability of the figures, data values are aggregated for each hour, unless otherwise specified.

A. Overall experiment setup.

We model the datacenter as a set of PMs where each server has 32 cores and 56G of memory. A limitation factor ($\chi = 0.9$) is set to restrict the maximum number of cores loaded for each PM, i.e., for each PM, 28 cores are allowed to be occupied by VMs. This is reasonable as some resources are needed by the hypervisor, for example to emulate the underlying hardware environment and to migrate VMs, and by the host operating system. The server power consumption is 100 W when idle and 560 W when fully utilized, which is consistent with power usage for the HP ProLiant DL165G7 servers used in the real tests. The price of electricity is set to be \$0.07 per kW/h.

We consider 7 different VM instance types similar to offerings by Amazon EC2. Their hardware characteristics are illustrated in Figure 2, and their hourly prices as well as the associated SLA penalties for downtime are listed in Table III. VM arrival is modeled using one Poisson process for each instance type, seven processes in all. A parameter associated with these processes is λ, which indicates the average arrival interval. Note that the time unit used in simulation is hours, while minutes are used in real test.

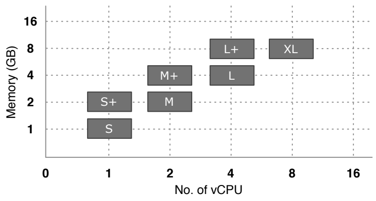

Fig. 2. VM instance types.

TABLE III
SETTING OF PRICES AND PENALTIES FOR VM INSTANCES.

VM types	S	S+	M	M+	L	L+	XL
Price (\$/h)	0.04	0.06	0.09	0.14	0.20	0.31	0.46
Penalty (\$/h)	0.40	0.60	0.90	1.40	2.00	3.10	4.60

B. Simulations.

For the simulations, we use 48 PMs, which is a common size for a rack in a datacenter [4]. We study three different scenarios with $\lambda = 0.4$ (high load scenario), $\lambda = 0.5$ (medium load scenario), and $\lambda = 0.6$ (low load scenario), respectively. Each new VM instance is assigned a lifetime ranging from 1 hour to 60 hours, uniformly distributed. By aggregating the capacity over all cores for each instance type (22 cores), we arrive at an average capacity requirement of $\frac{22}{\lambda} \times \frac{1+60}{2} = 1342$ cores per hour for the medium load scenario. These parameters are selected to make the average workload demand $(1342/(48 * 32) = 87.4\%)$ close to the selected infrastructure limitation factor $(\chi = 0.9)$. All VMs arrive during the first 252 hours, and then terminate within 60 hours after its arrival.

To compare the performances of algorithms on increasing the profit for the infrastructure provider, we run 10 tests for each value of λ. Note that the workload in each test case is the same for all eight algorithms. However, the workloads for any two tests are different, although they have the same VM arrival interval parameter.

1) Simulation Results: In the following sections we use one of the 10 medium load tests $(\lambda = 0.5)$ as an example to investigate the behavior of the algorithms in detail. We study the profit (p') for each of the seven other algorithms compared with the profit (p) achieved by the baseline algorithm using a metric defined by $\alpha = (p' - p)/p$. The average α values of the 10 medium load tests are presented in Figure 3. Some interesting findings can be observed in Figure 3, including (i) that algorithms with suspend/resume of VMs enabled show more significant improvements with average α values higher than 20%, (ii) enabling suspend/resume PMs increase profit only slightly, by less than 5%, and (iii) using live VM migration can improve the profit by more than 10%.

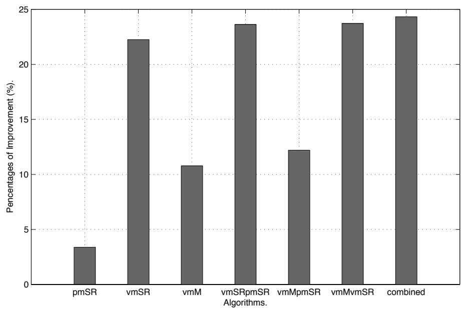

Fig. 3. Average profit improvement (all other algorithms vs. baseline).

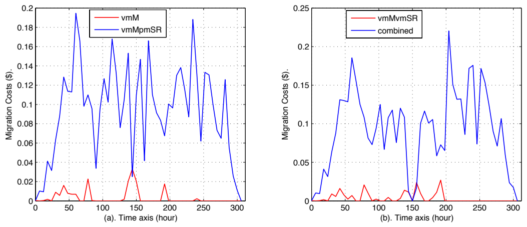

Fig. 7. VM migration cost: with/without suspend/resume PMs.

We next study the impact of the suspend/resume PM management action in more detail. Figure 4 shows that the average PM CPU usage with suspend/resume of PMs enabled is higher than without. In particular after 252 hours, these differences become larger, as there are no arriving VM requests resulting in a larger number of PMs that can be suspended. This is evident when VM migration is enabled (see subgraphs (c) and (d) in Figure 4, and subgraphs (c) and (d) in Figure 5). Enabling suspend/resume of PMs is also beneficial for the power consumption of the whole infrastructure. Similar to the CPU usage and number of active PMs, the differences are larger when the PMs are under low load compared with when the load is high, as illustrated in Figure 6. Also as expected, the difference for PM expenses is even larger when VM migration is enabled (see subgraphs (c) and (d) Figure 6);

Figure 7 demonstrates the collected average migration cost over time for algorithms with VM migration enabled. In order to improve the readability the migration costs are aggregated every 2 hours. Even so, we see that the migration costs are very low due to the short migration times. For example, it only takes 8 seconds to migrate an XL instance, resulting in a monetary cost of \$0.0102. From Figure 7, we also observe that, for algorithms with suspend/resume PMs, VM migration costs are much higher than others, as Algorithm 6 migrates all VMs before suspending an active PM. Looking at subgraphs (c) and (d) in Figure 5, it is observed that benefiting from VM migration, more PMs can be suspended compared with other algorithms without VM migration.

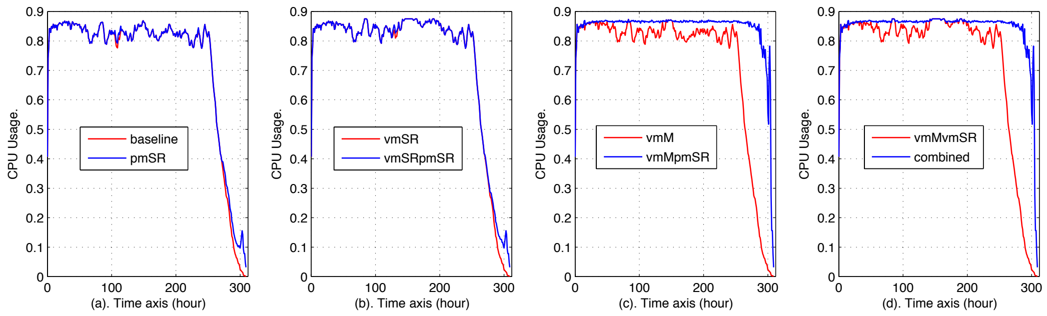

Fig. 4. CPU usage: with/without suspend/resume PMs.

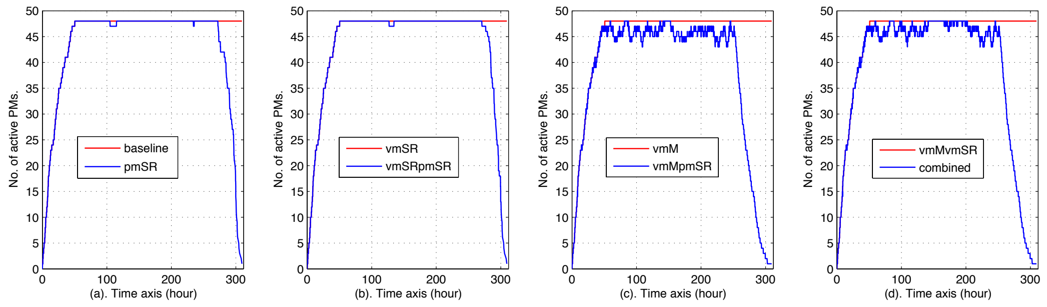

Fig. 5. No. of active PMs: with/without suspend/resume PMs.

Regarding the impact of VM migration, Figure 8 illustrates, for each policy, how CPU usage changes when migration is used. The impact is largest for the policies that allow suspend/resume of VMs (vmM)pmSR, and for the combined policy (as compared to vmSRpmSR). Looking at subgraph (c), we can see that there is almost no difference between vmSR and vmMvmSR. This is because in our current settings, suspend/resume VM is much more profit-efficient than VM migration, and it is dominating. This is also consistent with the data plot in Figure 3, where the differences among vmSR, vmSRpmSR, vmMvmSR, and combined are very small.

Turning to the impact of suspending and resuming VMs, Figure 9 plots the number of VMs suspended over time, as well as the their total number of cores. The subgraph (a) shows that, in general, the algorithms without the feature of resuming VMs (e.g., pmSR, and vmMpmSR) suspend fewer VMs than others. In particular, looking at the combined algorithm, we can see that it suspends the largest number of VMs most of the time (see the curve in bold). However, the total number of cores that belongs to suspended VMs is usually fewer than other algorithms (see subgraph (b) in Figure 9).

This observation indicates that VMs suspended by algorithms with this feature enabled are comparably smaller instances. This is consistent with our algorithm design, as when the capacity of the infrastructure is tight, Algorithm 5 selects VMs with smaller values of $(price + penalty)$ to suspend in order to release more capacity and run VMs with higher $(price + penalty)$ values. According to the settings in Figure 2 and Table III, smaller instances are using smaller $(price + penalty)$ values (but this is not a necessity). This is also consistent with the penalty plots in Figure 10.

Finally, we study the impact of the data center workload by varying the datacenter workload parameter (λ). Table IV presents aggregated results of 10 runs each with parameters $\lambda = 0.4$ (high load), $\lambda = 0.5$ (medium load), and $\lambda = 0.6$ (low load), respectively. In the high load scenario, the number of active PMs is high along with CPU usage. As there are quite a few VMs not running, the datacenter pays significant penalties and the profits are negative. Here, we note that baseline, pmSR, vmM, and vmMpmSR all perform very similar, with an average loss around \$50 per hour. In contrast, the algorithms that can start and resume VMs (vmSR, vmMvmSR, vmSRpmSR, and combined) and thus replace low-profit VMs when more important VM workload arrive, all perform much better with average losses around \$11 per hour. In the medium load case, we observe similar resource usage in the high load scenario, but for medium load, there are only a few VMs that are suspended as the average penalties are much lower and the

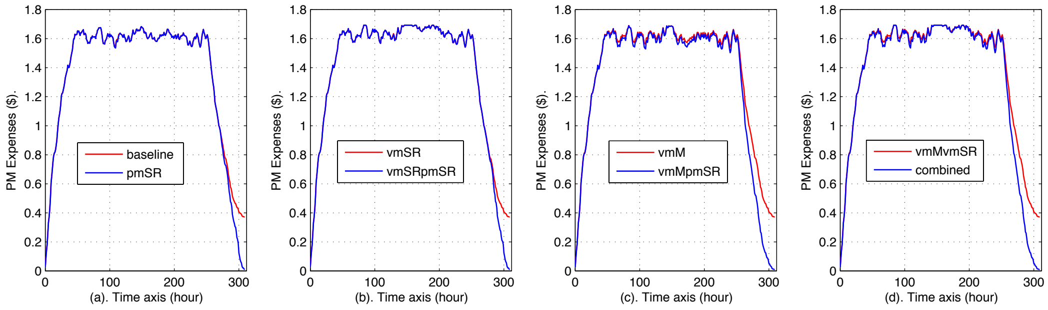

Fig. 6. PM expenses: with/without suspend/resume PMs.

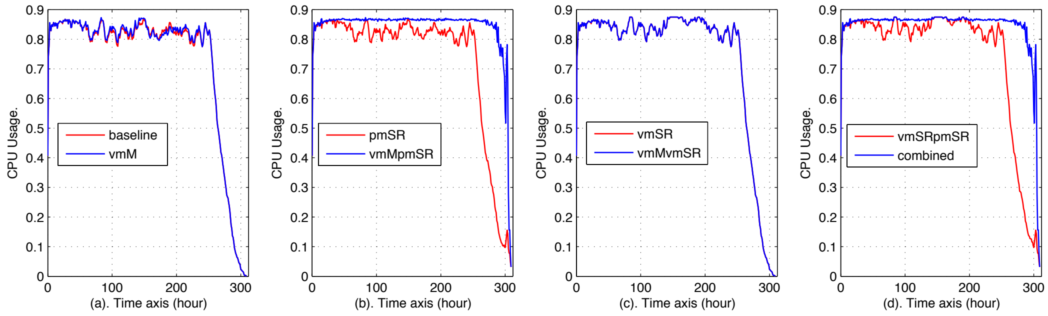

Fig. 8. CPU usage: with/without VM migration.

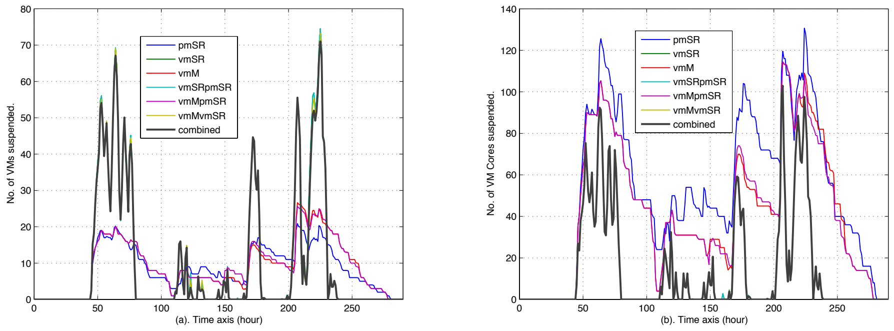

Fig. 9. No. of VMs suspended and No. of VM cores suspended.

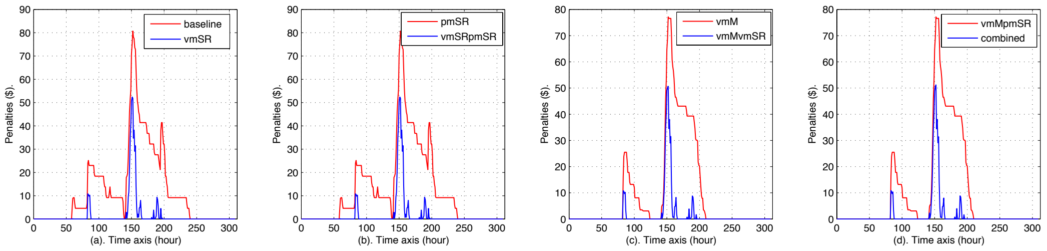

Fig. 10. Penalties with/without suspend/resume VMs.

TABLE IV
AGGREGATED RESULTS FOR DIFFERENT WORKLOADS.

λ	metrics	baseline	pmSR	vmSR	vmM	vmSRpmSR	vmMpmSR	vmMvmSR	combined
0.4	profit ($/h)	-56.15	-56.13	-11.33	-48.60	-11.31	-49.73	-11.40	-11.30
	penalty ($/h)	118.20	118.20	77.42	111.32	77.42	112.39	77.48	77.42
	no. act. PMs	45.7	43.3	45.7	45.7	43.2	39.4	45.7	39.2
	CPU usage (%)	74.20	74.71	76.74	75.07	77.29	84.36	76.75	84.94
0.5	profit ($/h)	48.10	49.72	58.80	53.28	59.46	53.96	59.50	59.80
	penalty ($/h)	12.05	12.05	1.72	9.67	1.72	9.67	1.66	1.61
	no. act. PMs	45.1	43.1	45.0	45.1	42.6	38.8	45.1	38.6
	CPU usage (%)	72.73	73.47	73.85	73.00	74.53	84.24	73.87	85.31
0.6	profit ($/h)	50.66	50.68	50.66	50.66	50.68	50.71	50.66	50.71
	penalty ($/h)	0	0	0	0	0	0	0	0
	no. act PMs	39.7	36.6	39.7	39.7	36.7	31.8	39.7	31.8
	CPU usage (%)	69.88	71.66	69.88	69.89	71.66	83.72	69.89	83.72

datacenter is profitable. Regarding the different algorithms, we note that also here, the ability to suspend and resume VMs is the key, with these four algorithms keeping penalties around \$1.7 per hours and profits around \$59 per hour. Notably, as the PMs are less loaded in this scenario, migration of VMs actually make a difference, with vmM and vmMpmSR having penalties of \$9.7 as compared to \$12 for baseline and pmSR. In the low load scenario, no penalties are paid as there for all algorithms always are enough resources to run all VMs. There are some differences in average number of PMs used and subsequently in CPU usage. The ability to suspend and resume PMs brings the average number of PMs down from 39.7 (algorithms baseline, vmSR, vmM, and vmMvmSR) to 36.7 (pmSR) and 36.7 (vmSRpmSR). Combining this with VM migration to be able to achieve consolidation has even greater impact, with 31.8 PMs used on average for vmMpmSR and the combined algorithm.

C. Real-world Demonstration.

In order to verify the validity of our approach, we perform a real-world test on a small testbed with 5 nodes, using our software described in Section IV. The PMs used are HP ProLiant DL165G7 @ 2.1 GHz with 32 cores and 56 GB of RAM each, connected by a top-of-the-rack Gigabit Ehternet switch. One node is functioning as controller and the other four nodes are worker nodes, hosting VMs. The setup of the testbed in terms of nodes and installed software is shown in Figure 11.

One debatable point is the setting of the peer-to-peer connection bandwidth among PMs, as network congestion issues can reduce the available bandwidth. However, we argue that by tuning the over-subscription factor and constructing the network topology in a proper manner [4] the effect of this problem can be reduced. Also, as VM migrations are carried out in a sequential order using our approach, network congestion potentially introduced by parallel VM migration can be mitigated.

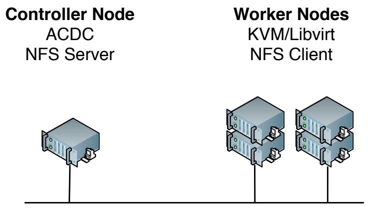

Fig. 11. Setup of the testbed.

With a limitation factor $\chi = 0.9$, the maximum number of cores available for VM provisioning is $4 * 32 * 0.9 = 115$. The test starts with an empty datacenter with new VMs arriving at an average rate of 6 minutes per instance type, namely, following a Poisson process with $\lambda = 0.1$. The VM lifetime is set to $24 - 36$ minutes normally distributed and the number of VMs to be provisioned during 96 minutes is 69. All VMs arrive in the first 60 minutes. For the real-world demonstration we use the same instance types as in the simulation and the number of instances of each type is limited to 10. Using this configuration, the average demand of cores during the test duration of 96 minutes is 105 which is close to the maximum 115.

Each VM is running a synthetic benchmark *bw_mem*, which is a memory write benchmark from the LMBench [25] suite. The bw_mem benchmark allocates twice the specified amount of memory, zeros it, and then copies the first half to the second half. For each of the instance types, the amount of memory allocated to bw_mem and the number of parallel threads used is tuned in order to consume 50% of the VMs memory and 100% of the vCPU.

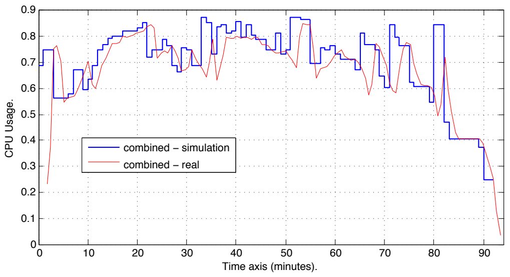

Fig. 12. CPU usage with *combined*: simulation vs. real.

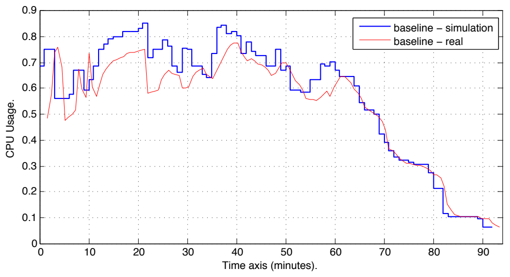

Fig. 13. CPU usage with *baseline*: simulation vs. real.

Figure 12 and Figure 13 present the CPU usage of the whole testbed over time in simulation and real tests. First of all, we remark that the behaviour of our algorithms (combined and baseline) in simulation is consistent with that in real tests. Another observation is that, in most cases the CPU usage in the real-world test is lower than in the simulation. The reason for this is that, in the simulation the CPU usage of a PM is defined by the proportion of cores occupied by VMs and these numbers are constant. However, in the real test, the workload fluctuates a bit, for example it takes up to 60 seconds from when a VM is provisioned until it is consuming the maximum amount of resources, because of the boot-up delay. Due to the same reason, it is also illustrated that there is a lag (around 60 seconds) between the curves representing the results in real tests and that of simulations.

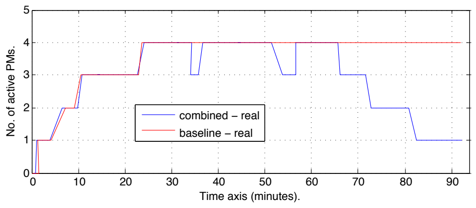

Fig. 14. No. of active PMs (real): combined vs. baseline.

Figure 14 presents the number of active PMs over time during the tests. We can see that the combined algorithm suspends one PM roughly between $35th$ minute and $36th$ minute and one PM roughly between the $54th$ minute and the $57th$ minute, resulting in higher CPU usage of the whole testbed, which is consistent with Figure 12. Additionally, benefiting from VM migration, some PMs can be suspended by the combined algorithm and the CPU usage can stay at a high level even after the $60th$ minute. In contrast, CPU usage achieved by the baseline algorithm keeps dropping as no new VMs arrive after the $60th$ minute and more PMs become idle when VMs terminate.

TABLE V
COMPLEMENTARY RESULTS FOR REAL TESTS: COMBINED VS. BASELINE.

	VMs suspended	Cores suspended	CPU usage (%)
combined	0.001%	<0.0001%	73.58
baseline	1.642%	6.103%	52.96

Table V summarizes some complementary results for the real tests. For the baseline algorithm, an average of 1.642% of VM instances are suspended per minute, while it is only 0.001% for the combined. Looking at the average percentage of VM cores suspended, the difference between the combined and the baseline algorithm is even larger. The reason behind this is that, if it is necessary to suspend VMs, the combined algorithm tends to suspend small instances that with fewer cores. Regarding the average CPU usage during the test, the combined algorithm also outperforms the baseline algorithm.

Finally, we also remark that in the simulations, the total time spent on VM migration was 77 seconds, while the duration in the real-world was 178.2 seconds. This difference is because migration time in the simulations are modeled on post-copy migration which transfers each memory page only once. The real-world tests are run using the standard KVM hypervisor, version 1.5.0, which does not include a post-copy migration algorithm. Because of this, pre-copy migration is used and this type of live migration algorithm uses an iterative transfer where memory pages can be sent multiple times, thus increasing migration time and making it hard to predict the migration time [32].

VI. Related Work

Optimal mapping of admitted VMs to a set of PMs in order to gain maximum profit while complying to all SLAs specified by customers is challenging for cloud providers as it is in general a NP-hard problem [20], [27]. Various algorithms have been proposed to produce near-optimal placement schemes, e.g., by Jing et al. [38], who present an improved genetic algorithm aimed to optimize possibly conflicting objectives, including making efficient usage of multidimensional resources, avoiding hotspots, and reducing power consumption. On the other hand, given the dynamic nature of clouds, with significant changes over time both in workload demands and available resources, the mapping of VMs to PMs need to be revisited regularly. Live migration of running VMs is therefore a necessity. A comprehensive study on principles and performance of live migration mechanisms (including precopy, postcopy and hybrid) is presented in [26], which also discusses how migration downtime can be reduced. Li et al.[21] define a framework for joint optimization of data center deployment, VM assignment, and migration. Based on fluctuations in network performance (latency), they propose a method based on network flow maximization to estimate VM migration cost by amortizing it to the latency of every access. Song et al.[30] define a bin packing approach to allocation and migration of VMs in data centers and similarly, Sato et al.[28] combine bin packing with resource usage prediction to dynamically optimize VM placement. Auto-regressive models are used for predictions and the number of VM migrations is minimized (avoiding ping-pong effects) based on the predictions. Li et al. [19] aim to minimize VM completion time using a knapsack formulation for VM placement. A hybrid on-line off-line scheme is used where VM migrations are combined with the knapsack placement algorithm. A defragmentation approach by Shanmuganathan et al. [29] include two algorithms whereas Avin et al. [3] propose simple destination swap strategies for VMs in order to reduce network traffic.

A large amount of effort has been devoted to server consolidation methods based on workload analysis, aiming at improving efficiency in cloud infrastructures [14], [31], [35]. Additional mechanisms for isolation of resources in hardware [12], [15], or software [34] have been developed to reduce the performance degradation introduced by consolidation of multiple VMs on a same server. Further, Roytman et al. present a polynomial time algorithm to determine the best suited VM combinations to be co-located [27], yielding server energy saving and VM performance preservation. However,these approaches commonly operate in off-line manners which are not able to dynamically and efficiently adapt the cloud to the changes (including workload variations, system failures, etc). They neither take VM pricing schemes and monetary penalty for SLA violation into consideration. Khanna et al. [17] propose a framework to detect application performance deviations and a VM migration mechanism to handle these. They proposed a set of server consolidation heuristics based on VM migration costs and server residual capacity. Xiao et al. [37] introduce some skewness metrics and use these to avoid hot and cold spots in datacenters and migrate VMs around.

In 2001, the Autonomic Computing [11] initiative was initiated by IBM, who also introduced the MAPE-K reference model. Its goal is to build computing systems that can manage themselves given high-level objectives from administrators [16]. In the MAPE-K model, with the support of a Knowledge base, the system Monitors the managed elements, analyzes the data monitored, and finally Plans and Executes suitable actions to ensure the system is in a desired state. Although considerable progress has been achieved in the past few years, the original vision remains unfulfilled as even more complexity is added to the system due to the convergence of new technologies and new applications [8]. The main goal of this work is to maximize the monetary profit of running a datacenter by automatic adaption to both internal and external changes, and thus it is within the scope of autonomic computing. The design and implementation of the proposed system (see Section IV) follows the MAPE-K model.

VII. Conclusions and Future Work

In this paper we present a continuous management approach for cloud infrastructure providers to maximize their PM utilization and increase profits. Based on a set of fundamental management actions, our approach can rearrange the VM to PM mapping during operation to increase the resource utilization of the cloud infrastructure, thereby increase the revenue by prioritizing more profitable workloads and reducing energy consumption. The feasibility and performance of our work, consisting of optimization algorithms and a continuous datacenter consolidation software, is evaluated by simulations and real-world experiments on a testbed. Results indicate that overall utilization of the datacenter is increased using out approach and that power consumption is reduced. The testbed results are consistent with the simulation results, which validates our simulation and indicates that our approach is applicable in real-world scenarios.

We have identified several interesting subjects for the future work, e.g., (i) to investigate the impact of the penalty model on our algorithms, (ii) to extend our work to support other pricing models and to integrate our algorithms in open-source cloud middlewares such as CloudStack and OpenStack, and (iii) to study the feasibility of incorporating our work with auto-scaling and workload prediction techniques, targeting proactive optimization and even better performance.

VIII. Acknowledgments

Financial support has in part been provided by the European Community's Seventh Framework Programme ([FP7/2010-2013]) under grant agreements no. 257115 (OPTIMIS), the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control, and the Swedish Government's strategic effort eSSENCE.

References

[1] Sysstat. http://sebastien.godard.pagesperso-orange.fr, visited March 2014.

[2] A. Ali-Eldin, J. Tordsson, and E. Elmroth. An Adaptive Hybrid Elasticity Controller for Cloud Infrastructures. In *Proceedings of the IEEE Network Operations and Management Symposium (NOMS 2012)*, pages 204–212. IEEE, 2012.

[3] C. Avin, O. Dunay, and S. Schmid. Simple Destination-Swap Strategies for Adaptive Intra-and Inter-Tenant VM Migration. *CoRR*, 2013.

[4] L. A. Barroso, C. Jimmy, and U. Hoelzle. *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2nd edition, 2013.

[5] 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 (M'07)*, pages 119–128. IEEE, 2007.

[6] C. Clark, K. Fraser, S. Hand, J. G. 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 and Implementation*, pages 273–286. ACM, 2005.

[7] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation Algorithms for NP-hard Problems. chapter Approximation Algorithms for Bin Packing: A Survey, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997.

[8] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey. Fulfilling the Vision of Autonomic Computing. *Computer*, 43(1):35–41, 2010.

[9] I. Goiri, J. Guitart, and J. Torres. Characterizing Cloud Federation for Enhancing Providers' Profit. In *Proceedings of the IEEE 3rd International Conference on Cloud Computing (CLOUD'10)*, pages 123–130. IEEE, 2010.

[10] A. Gulati, G. Shanmuganathan, A. Holler, and I. Ahmad. Cloud-scale Resource Management: Challenges and Techniques. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*, pages 3–3. USENIX Association, 2011.

[11] IBM Corp. An Architectural Blueprint for Autonomic Computing. Technical report, USA, Oct 2004. Tech. Rep.

[12] R. Iyer, R. Illikkal, O. Tickoo, L. Zhao, P. Apparao, and D. Newell. VM^3: Measuring, Modeling and Managing VM Shared Resources. *Computer Networks*, 53(17):2873–2887, 2009.

[13] R. Jones. Virt-top. http://people.redhat.com/~rjones/virt-top/, visited March 2014.

[14] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. A Cost-sensitive Adaptation Engine for Server Consolidation of Multitier Applications. In *Middleware 2009*, pages 163–183. Springer, 2009.

[15] F. Kamoun. Virtualizing the Datacenter Without Compromising Server Performance. *Ubiquity*, 2009, Aug 2009.

[16] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.

[17] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application Performance Management in Virtualized Server Environments. In *Proceedings of 10th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 373–381. IEEE, 2006.

[18] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: the Linux Virtual Machine Monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[19] K. Li, H. Zheng, and J. Wu. Migration-based Virtual Machine Placement in Cloud Systems. In *Proceedings of the IEEE 2nd International Conference on Cloud Networking (CloudNet)*, pages 83–90. IEEE, 2013.

[20] W. Li, J. Tordsson, and E. Elmroth. Virtual Machine Placement for Predictable and Time-Constrained Peak Loads. In *Proceedings of the 8th International Conference on Economics of Grids, Clouds, Systems, and Services (GECON'11)*, 2011. Lecture Notes in Computer Science, Vol. 7150, Springer-Verlag, pp. 120-134.

[21] Y. Li, M. Yao, and C. Lin. Joint Study on Optimizations of Data Center Deployment, VM Assignment and Migration. In *Proceedings of the IEEE/ACM 21st International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2013.

[22] Libvirt. The virtualization API. http://libvirt.org, visited March 2014.

[23] J. L. Lucas-Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Scheduling Strategies for Optimal Service Deployment across Multiple Clouds. *Future Generation Computer Systems*, 29(6):1431–1441, 2013.

[24] Mark Blackburn, 1E. Five Ways to Reduce Data Center Server Power Consumption, April 2008. White Paper.

[25] L. McVoy. LMbench - Tools for Performance Analysis. http://www.bitmover.com/lmbench/, visited March 2014.

[26] P. Svärd, J. Tordsson, E. Elmroth, S. Walsh, and B. Hudzia. The Noble Art of Live VM Migration - Principles and Performance of Precopy and Postcopy Migration of Demanding Workloads. 2014. Submitted.

[27] 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*, pages 83–94, Berkeley, CA, 2013. USENIX.

[28] K. Sato, M. Samejima, and N. Komoda. Dynamic Optimization of Virtual Machine Placement by Resource Usage Prediction. In *Proceedings of the 11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 86–91. IEEE, 2013.

[29] G. Shanmuganathan, A. Gulati, and P. Varman. Defragmenting the Cloud using Demand-based Resource Allocation. In *Proceedings of the ACM SIGMETRICS/international Conference on Measurement and Modeling of Computer Systems*, pages 67–80. ACM, 2013.

[30] W. Song, Z. Xiao, Q. Chen, and H. Luo. Adaptive Resource Provisioning for the Cloud using Online Bin Packing. *IEEE Transactions on Computers*, 99(PrePrints), 2013.

[31] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware Consolidation for Cloud Computing. In *Proceedings of the 2008 Conference on Power aware Computing and Systems*, volume 10. USENIX Association, 2008.

[32] 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 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '11)*, pages 111–120. ACM, 2011.

[33] L. Tomás and J. Tordsson. Improving Cloud Infrastructure Utilization Through Overbooking. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, CAC'13, pages 5:1–5:10. ACM, 2013.

[34] A. Verma, P. Ahuja, and A. Neogi. Power-aware Dynamic Placement of HPC Applications. In *Proceedings of the 22nd Annual International Conference on Supercomputing*, pages 175–184. ACM, 2008.

[35] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari. Server Workload Analysis for Power Minimization using Consolidation. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, pages 28–28. USENIX Association, 2009.

[36] 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. In *VEE '11: The 2011 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 121–132. ACM, 2011.

[37] Z. Xiao, W. Song, and Q. Chen. Dynamic Resource Allocation using Virtual Machines for Cloud Computing Environment. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1107–1117, 2013.

[38] J. Xu and J. A. Fortes. Multi-objective Virtual Machine Placement in Virtualized Data Center Environments. In *Proceedings of the 2010 IEEE/ACM International Conference on Green Computing and Communications & 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing*, pages 179–188. IEEE, 2010.