Virtual Machine Placement in Cloud Environments

Wubin Li

李务斌

LICENTIATE THESIS, MAY 2012
DEPARTMENT OF COMPUTING SCIENCE
UMEÅ UNIVERSITY
SWEDEN

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

wubin.li@cs.umu.se

ISBN 978-91-7459-453-9
ISSN 0348-0542
UMINF 12.13

Printed by Print & Media, Umeå University, 2012

Abstract

With the emergence of cloud computing, computing resources (i.e., networks, servers, storage, applications, and services) 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 is one of the most commonly used resource carriers in which business services are encapsulated. Virtual machine placement optimization, i.e., finding optimal placement schemes for virtual machines, and reconfigurations according to the changes of environments, become challenging issues.

The primary contribution of this licentiate thesis is the development and evaluation of our combinatorial optimization approaches to virtual machine placement in cloud environments. We present modeling for dynamic cloud scheduling via migration of virtual machines in multi-cloud environments, and virtual machine placement for predictable and time-constrained peak loads in single-cloud environments. The studied problems are encoded in a mathematical modeling language and solved using a linear programming solver. In addition to scientific publications, this work also contributes in the form of software tools (in EU-funded project OPTIMIS) that demonstrate the feasibility and characteristics of the approaches presented.

Preface

This thesis consists of an introduction to cloud computing, a brief discussion of virtual machine placement 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)*, pages 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. Technical Report UMINF-12.14, May, 2012. Department of Computing Science, Umeå University, 2012.

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.

Acknowledgments

The accomplishment of this licentiate thesis has been one of the most significant academic challenges I've ever had to face. Without the support, patience and guidance of numerous people, it 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 his timely meetings, discussions and emails, for his 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 did on proofreading my papers.

Peter Svärd, for your nice personality which always makes me feel easy and relaxed when working and travelling with you.

Daniel Espling and **Lars Larsson**, for sharing your knowledge and experience, for answering me tons of questions 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.

Other group members (in no particular order), **Ewnetu Bayuh Lakew**, **Lei Xu**, **Mina Sedaghat**, **Ahmed Ali-Eldin**, **Francisco Hernández**, **Peter Gardfjäll**, **Lennart Edblom**, and **Tomas Forsman**, for all their contributions to our collective effort.

All floorball team players, for creating the exciting and passionate moments every Tuesday in IKSU.

Lastly, I would like to thank my family for all their love and encouragement. Thank you.

Umeå, May 2012
Wubin Li

Contents

Introduction

By provisioning of shared resources as a metered on-demand service 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 it is expected that their total IT costs will decrease. From a cloud provider's or an agent's perspective, however, due to the scale of resources to manage, and the dynamic nature of service behaviours (with rapid demands for capacity variations and resource mobility), as well as the heterogeneity of cloud systems, resource allocation and scheduling are becoming challenging issues, e.g., to find optimal placement schemes for resources, and resource reconfigurations in response to the changes of the environment [11].

There is a multitude of parameters and considerations (e.g., performance, cost, locality, reliability and availability, etc.) involved in the decision of where and when to place and reallocate data objects and computation resources in cloud environments. Some of the considerations are consistent with one another while others may be contradicting. This work investigates challenges involved in the problem of resource placement and scheduling in cloud environments, tackles the problem using combinatorial optimization techniques and mathematical modeling. Thesis contributions include scientific publications addressing, e.g., modeling for dynamic cloud scheduling via migration of Virtual Machines (VMs) in multi-cloud environments, as well as to optimal virtual machine placement within datacenters for predicable and time-constrained load peaks. In addition, this work also contributes in the form of software tools (in the EU-funded project Optimis [13]) that demonstrate the feasibility and characteristics of the proposed solutions.

Chapter 1

Cloud Computing

Cloud Computing provides a paradigm shift following the shift from mainframe to client-server architecture in the early 1980s [14] [32] and 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.

1.1 Hardware Virtualization

Virtualization is a technology that separates computing functions and implementations from physical hardware. It is the foundation of cloud computing, since it enables isolations between hardware and software, between users, and between process and resources. These isolation problems are not well solved by traditional operating systems. Hardware virtualization approaches include *Full Virtualization*, *Partial virtualization* and *Paravirtualization* [31]. With virtualization, software capable of execution on the raw hardware can be run in a virtual machine. Cloud systems deployable services can be encapsulated in virtual appliances (VAs) [18], and deployed by instantiating virtual machines with their virtual appliances [17]. This new type of service deployment provides a direct route for traditional on-premises applications to be rapidly redeployed in a Software as a Service (SaaS) mode. By decoupling the hardware and operating system infrastructure provider from the application stack provider, virtual appliances allow economies of scale on the one side to be leveraged by the economy of simplicity on the other.

1.2 The XaaS Service Models

Commonly associated with cloud computing are the following service models:

- Software as a Service (SaaS)

In the SaaS model, software applications are delivered as services that execute on infrastructure managed by the SaaS vendor. Consumers are enabled to access services over various clients such as web browsers and programming interfaces, and are typically charged on a subscription basis [6]. The implementation and the underlying cloud infrastructure where it is hosted is transparent to consumers.

- 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 [5]. 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 maintains the control over the deployed applications and possibly configuration settings for the application-hosting environment [24].

- Infrastructure as a Service (IaaS)
  In 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 cloud infrastructure but have to control its own virtual infrastructure typically constructed by virtual machines hosted by the IaaS vendor. This thesis work mainly focus on this model, although it may be generalized to also apply to the other models.

1.3 Cloud Computing Scenarios

Based on the classification of cloud services into SaaS, PaaS, and IaaS, two main stakeholders in a cloud provisioning scenario can be identified, i.e., the *Infrastructure Provider* (IP) who offers infrastructure resources such as Virtual Machines, networks, storage, etc. which can be used by *Service Providers* (SPs) to deliver end-user services such as SaaS to their consumers, these services potentially being developed using PaaS tools. As identified in [7], four main types of cloud scenarios can be listed as follows.

- Private Cloud

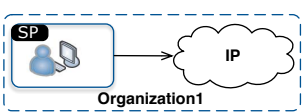

Figure 1: Private cloud scenario.

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

- Cloud Bursting

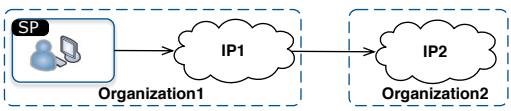

Figure 2: Cloud bursting scenario.

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 2. Typically, less sensitive tasks are executed in the public cloud instead while tasks that requiring higher levels of security are provisioned the private infrastructure.

- Federated Cloud

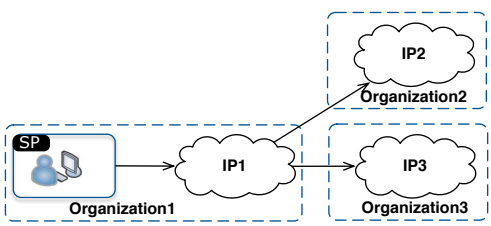

Figure 3: Cloud federation scenario.

Federated clouds are IPs collaborating on a basis of joint load-sharing agreements enabling them to offload capacity to each others [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, an 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 3 illustrates a federation between three IPs.

- Multi-Cloud

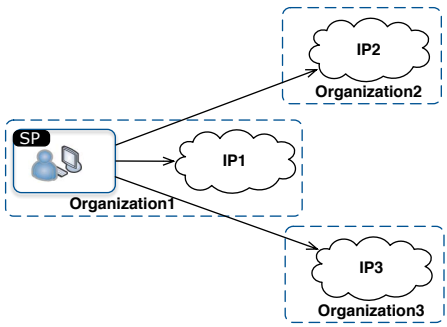

Figure 4: Multi-cloud scenario.

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.

It should be remarked that the multi-cloud and federated cloud scenarios are commonly considered only for the special case where organization 1 does not possess an internal IP, corresponding to removing IP1 from figures 3 and 4.

Chapter 2

Virtual Machine Placement

Given a set of admitted services and the availability of local and possibly remote resources, there are a number of placement problems to be solved to determine where to store data and where to execute VMs. The following sections describe the challenges and state of the art of VM placement and scheduling in cloud environments.

2.1 Parameters and Considerations

There are a multitude of parameters and considerations involved in the decision of where and when to place/reallocate data objects and computations in cloud environments. An automated placement and scheduling mechanism should take into account the considerations and tradeoffs, 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 price or performance given a set of constraints, often including the one of price and performance that is subject to optimization. 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 placement schemes of virtual machines, the performance achieved may differ a lot [29].
- **Cost:** The price model was dominated by fixed prices in the early phase of cloud adoption. However, cloud market trend shows that dynamic pricing schemes utilization is being increased [23]. Investment decreases 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 cost for VM placement, e.g., interference and overhead that one VM causes on other concurrently running VMs on the same physical host, should also be taken in to account.

- **Locality:** In general, for considerations of usability and accessibility, VMs should be located close to users (which could be other services/VMs). However, due to e.g., legal issues and security reasons, locality may become a constraints for optimal placement.

- **Reliability and continuous availability:** Part of the central goals for VM placement is service reliability and availability. To achieve this, VMs may be placed/replicated/migrated across multiple (at least two) geographical zones. During this procedure, factors such as the importance of the data/service encapsulated in VMs, its expected usage frequency, and the reliability of the different data centers, must be taken in to account.

2.2 Challenges

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

- Firstly, there exists no generic model to represent various scenarios of resource scheduling, especially when users' requirements are vague and hard to encode through modeling languages.

- Secondly, 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, in for a multi-cloud scenario that includes n cloud providers and m VMs, $m * n^2$ assignments are needed to express the VM migration overheads ignoring possible changes of VM sizes. Therefore, mechanisms that can help to automatically capture those values are required.

- Thirdly, the VM placement problem is typically formulated as a variant of the class constrained multiple-knapsack problem that is known to be NP hard [9]. Thus, 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., Amazon EC2 [4], the leading cloud provider, has approximately 40,000 servers and schedules 80,000 VMs every day [12].

2.3 State of the art

Virtual machine placement in distributed environments has been extensively tudied in the context of cloud computing. Such approaches address distinct problems, such as initial placement, consolidation, or tradeoffs between honoring service level agreements and constraining provider operating costs, etc. [25]. Studied scenarios are usually encoded in mathematical models and are finally

solved either by algorithms such as approximation, greedy packing and heuristic method, or by existing programming solvers such as Gurobi [1], CPLEX [2] and GLPK [3]. Those related work can be separated into two sets: (1) VM placement in single-cloud environments and (2) VM placement in multi-cloud environments.

In single-cloud environments, given a set of physical machines and a set of services (encapsulated within VMs) with dynamically changing demands, on-line placement controllers that decide how many instances to run for each service and where to put and execute them, while observing resource constraints, are NP hard problems. Tradeoff between quality of solution and computation cost is a challenge. To address this issue, various approximation approaches are applied, e.g., by Tang et al. [9] propose an algorithm that can produce within 30 seconds high-quality solutions for hard placement problems with thousands of machines and thousands of VMs. 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. [15] 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 [16], 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. [19] study the impact of VM placement policies on cooling and maximum data center temperatures, 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.

For VM placement across multiple cloud providers, information about the number of physical machines, the load of these physical machines, and the state of resource distribution inside the IP side are normally hidden from SP, and hence not parameters that can be used for placement decisions. Only provision-related information such as types of VM instance, price schemes, are exposed to SP. Hence, most works on VM placement across multi-cloud environments are focusing on cost aspects. Chaisiri et al. [8] 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. Bossche et al. [10] examine the workload outsourcing problem in a multi-cloud setting with deadline-constrained, and present cost-optimal optimization to maximize the utilization of the internal data center and to minimize the cost of running the outsourced tasks in the cloud, while fulfilling the applications quality of service constraints. Tordsson et al. [30], propose a cloud brokering mechanisms for optimized placement of VMs to obtain optimal cost-performance tradeoffs across multiple

cloud providers. Similarly, Vozmediano et al. [27] [26] explore the multi-cloud scenario to deploy a computing cluster on top of a multi-cloud infrastructure, for solving 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 3

Summary of Contributions

3.1 Paper I

In Paper I [22], we investigate dynamic cloud scheduling use cases where parameters are continuously changed, and propose 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. 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 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, is also evaluated in scenarios based on commercial cloud providers' offerings.

3.2 Paper II

In Paper II [21], the VM placement problem for load balancing of predictable and time-constrained peak workloads is studied for placement of a set of virtual machines 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. Notably, two VM sets (i.e., VMs provisioned to fulfill services demands) may use the same physical resources if they do not overlap in runtime. We use an approximation based on discrete time slots to generate all possible overlap sets. Finally, 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 reduce the time required to compute a final solution, enabling larger problems to be solved. Evaluations based on synthetic workload traces suggest that our algorithms are feasible, and that these can be combined to achieve desired tradeoffs between quality of solution and execution time.

3.3 Paper III

The cloud computing landscape has 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 for performing the management operations and 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 [20], 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.

Chapter 4

Future Work

Future directions for this work include to model the interconnection requirements that can precisely express the relationships between VMs to be deployed. Another area of future work is approximation algorithms based on problem relaxations and heuristic approaches such as greedy formulation for considerations of tradeoff between quality of solution and execution time. Additionally, for VM placement problems, interference and overhead that one VM causes on other concurrently running VMs on the same physical host should be taken in to account. 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 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. The hard and soft constraints can, e.g., be used to specify collocation or avoidance of co-location of certain VMs. We are also investigating how to apply multi-objective optimization techniques to this scenario.

Bibliography

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

[2] IBM ILOG CPLEX Optimizer, `http://www.ibm.com/software/integration/optimization/cplex-optimizer/`, visited October 2011.

[3] GNU Linear Programming Kit, `http://www.gnu.org/s/glpk/`, visited October 2011.

[4] Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`, visited May, 2012.

[5] Platform as a Service. `http://en.wikipedia.org/wiki/Platform_as_a_service`, visited May, 2012.

[6] Software as a Service (SaaS). Cloud Taxonomy. `http://cloudtaxonomy.opencrowd.com/taxonomy/software-as-a-service/`, visited May, 2012.

[7] M. Ahronovitz et al. Cloud computing use cases white paper, v4.0. www.cloudusecases.org, visited May 2012.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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, New York, NY, USA, 2009. ACM.

[16] 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.

[17] 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.

[18] 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.

[19] 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*, SC '11, pages 22:1–22:12, New York, NY, USA, 2011. ACM.

[20] W. Li, P. Svärd, J. Tordsson, and E. Elmroth. A General Approach to Service Deployment in Cloud Environments. Technical Report UMINF-12.14, May, 2012. Department of Computing Science, Umeå University, 2012.

[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] 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.

[24] P. Mell and T. Grance. The NIST definition of cloud computing. *National Institute of Standards and Technology (NIST)*, 2011.

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

[26] 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.

[27] 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.

[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] 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.

[30] 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.

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

[32] 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.

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.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.

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

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 evalu-

978-0-7695-4622-3/11 $26.00 © 2011 IEEE
DOI 10.1109/CloudCom.2011.31

ations 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 reallocate 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.

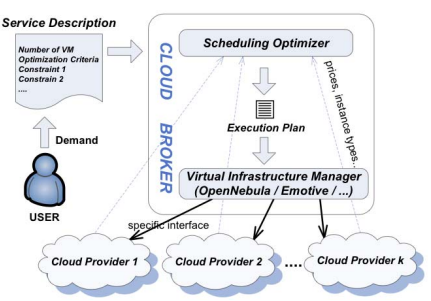

Fig. 1. Architecture overview for cloud brokerage scenario.

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:

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

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

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

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

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

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

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

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:

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

$Subject\ to:$

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

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 β_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].

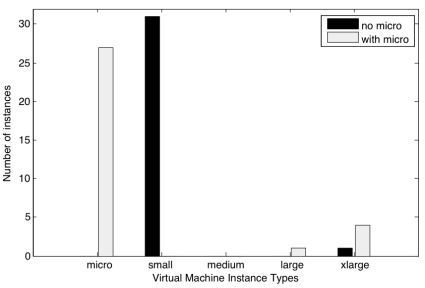

Fig. 2. VM placement with and without the micro instance type.

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

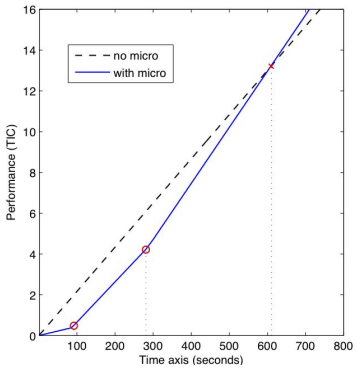

Fig. 3. Performance improvement with and without the micro instance type.

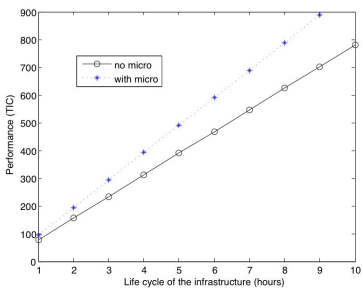

Fig. 4. Performance improvement with and without the micro instance type.

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,

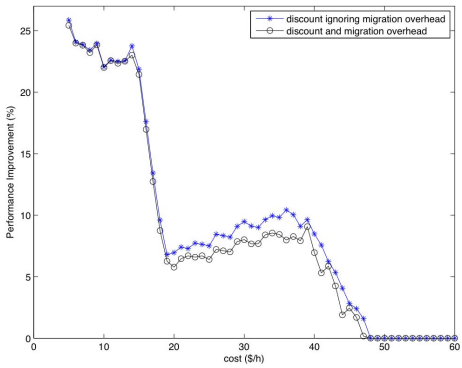

Fig. 5. VM placement with and without price discount.

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.

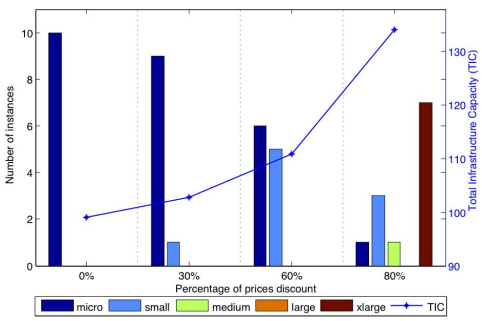

Fig. 6. VM placement with varying prices discount by GOGRID.

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.

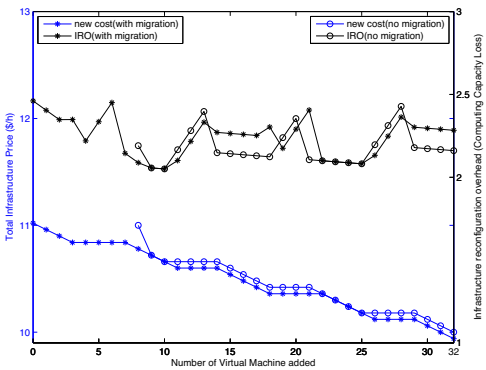

Fig. 7. Illustration of performance scale-up (LB constraint: 0%).

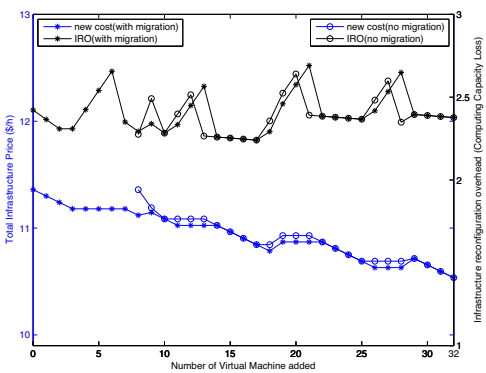

Fig. 8. Illustration of performance scale-up (LB constraint: 10%).

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%)

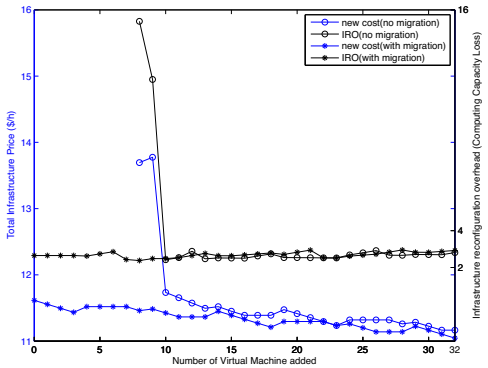

Fig. 9. Illustration of performance scale-up (LB constraint: 20%).

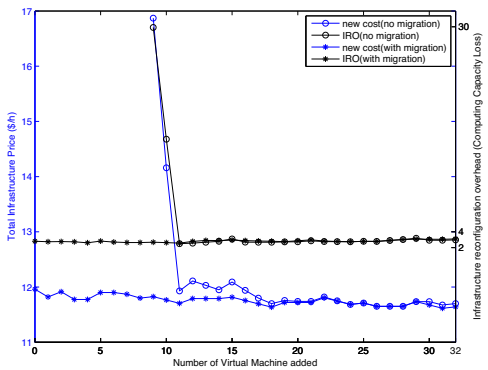

Fig. 10. Illustration of performance scale-up (LB constraint: 30%).

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.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.

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

K. Vanmechelen, J. Altmann, and O.F. Rana (Eds.): GECON 2011, LNCS 7150, pp. 120–134, 2012.

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

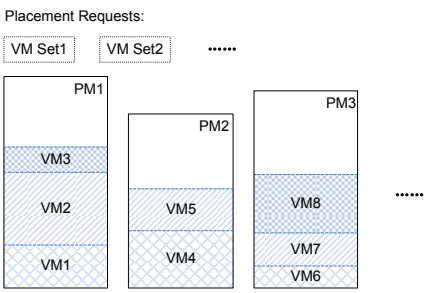

Fig. 1. Studied scenario illustration

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, ..., r_n}.
   Output: Placement Scheme for R.
 1 Sort all the requests by start-time s;
 2 for 1 ≤ i ≤ n do
 3     for 1 ≤ 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 ← 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*).

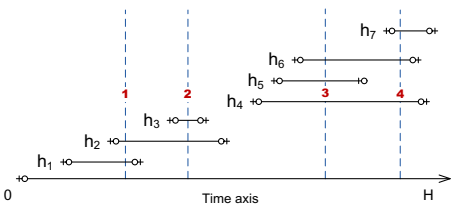

Fig. 2. Illustration for coexistence of placement requests

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 $R = \{r_1, r_2, \ldots, r_n\}$.
Output: Placement Scheme for R.

```
1 Sort all the requests by start-time s;
2 for 1 ≤ i ≤ n do
3     for 1 ≤ 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** $1 \le i \le T$ **do**
5 $\quad ts \leftarrow S + (i-1) * interval$;
6 $\quad te \leftarrow S + i * interval$;
7 $\quad currentSet = \{r \in R \mid r$ starts in $[ts, te]\}$;
8 $\quad should\_add \leftarrow true$;
9 $\quad$ **foreach** P *in* $OverlapSets$ **do**
10 $\quad\quad$ **if** $P \subset currentSet$ **then**
11 $\quad\quad\quad OverlapSets \leftarrow OverlapSets \setminus P$;
12 $\quad\quad$ **end**
13 $\quad\quad$ **if** $currentSet \subseteq P$ **then**
14 $\quad\quad\quad should\_add \leftarrow false$;
15 $\quad\quad$ **end**
16 $\quad$ **end**
17 $\quad$ **if** $should\_add$ **then**
18 $\quad\quad OverlapSets \leftarrow OverlapSets \cup currentSet$;
19 $\quad$ **end**
20 **end**

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

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

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, ..., 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  γ ← 0;
3  for 1 ≤ i ≤ n do
4      for 1 ≤ 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 γ < Load(r_i) then
10         γ ← Load(r_i);
11     end
12 end
13 OverlapSets ← GenerateOverlapSets(R,T);
14 Minimize μ 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, $71.9\% \pm 6.1\%$, whereas the three other algorithms all result in loads above 80%, with Sequential the second best at $80.5\% \pm 7.6\%$. 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

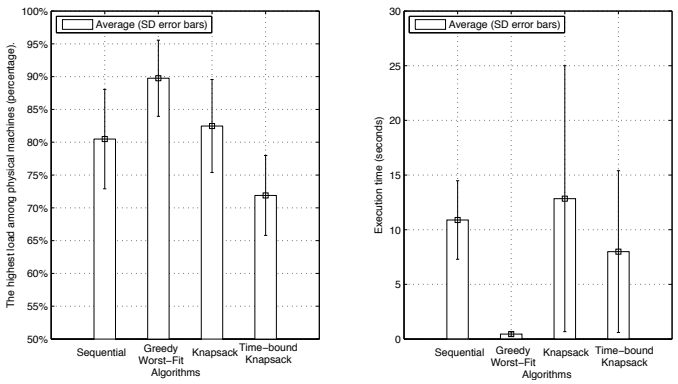

Fig. 3. Performance and execution time comparison for 1000 tests

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

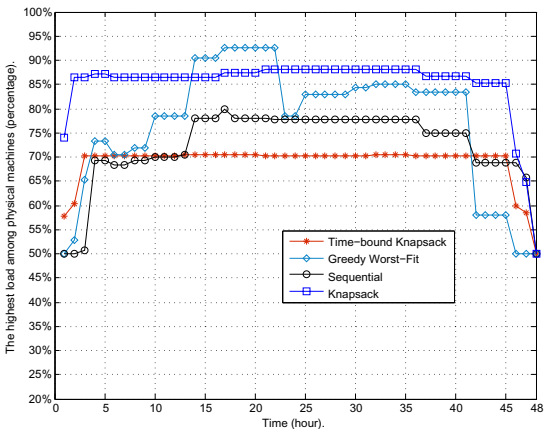

Fig. 4. Illustration of highest loads of the infrastructure evolve with time

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

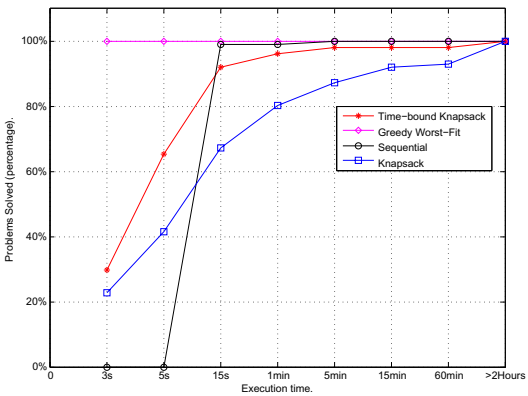

Fig. 5. Execution time vs. percentage of problems solved for 100 tests

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

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

* Technical Report UMINF-12.14, May, 2012. Department of Computing Science, Umeå University.

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 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.**

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

I. Introduction

With the rise of virtualization as a platform for hosted services provision in the context of cloud computing, deployable cloud services are encapsulated in virtual appliances (VAs), and deployed by instantiating Virtual Machines (VMs) with virtual appliances [18]. This new manner of service deployment provides a direct route for traditional on-premises applications to be rapidly redeployed in a Software as a Service (SaaS [9]) mode. By decoupling the hardware and operating system of the infrastructure provider from the application stack provider, virtual appliances allow economies of scale on the one side to be leveraged by the economy of simplicity on the other.

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 brokered clouds. We identify the 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, e.g., data management, service contextualization, service management in its orchestration of the service deployment process. Our proposed approach is validated by implementation and integration in a private cloud, a bursted cloud, and a brokered multi-cloud scenario using resources at Atos (Barcelona), BT (London), Flexiant (Edinburgh), and Umeå University (Umeå), as well as tools from the OPTIMIS Toolkit [16] providing the required complementing functionality.

The remainder of the paper is organized as follows: Section II introduces background cloud services and deployment. Section III describes core requirements for service deployment in cloud environments. Section IV presents the design of our service deployment solution. Section V describes a validation study of our approach in the context of OPTIMIS toolkit. Section VI outlines the related work of service deployment. Finally, our conclusions are presented in Section VII followed by a presentation of future work, acknowledgments, and a list of references.

II. Background

A. Cloud Services

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. 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 an SP or an IP deploys services to an IP providing IaaS.

Deployable Services: Cloud systems offering IaaS are based on virtualization technology which means that a deployable cloud service is in fact a VM or a collection of VMs together with an description of the service, a *service manifest*. The manifest typically consists of sections describing what components the service is composed of along with functional and non-functional requirements for a deployment target. We refer to 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 service manifest can also define elasticity rules for the service, i.e. upper and lower bounds for how many instances of a component that are allowed to run. Commonly, associated with elasticity bounds are elasticity rules for when to scale up or down, which 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, etc.

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 service manifest and these instances can be shutdown or restarted as needed.

B. Deployment scenarios

Cloud environments can be set up differently depending on the types of interaction between the collaborating sites [27], [11]. For example, an SP can set up a cloud infrastructure for its own internal use, commonly referred to as a *private cloud*, which is illustrated in Figure 1. Private clouds can circumvent many of the security and privacy concerns related to hosted sensitive information in public clouds, the latter a case where the SP leases IaaS resources publicly available IPs. Private clouds may also offer stronger guarantees on control and performance as the whole infrastructure can be administered within the same domain.

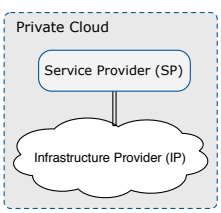

Fig. 1. Private cloud.

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 *cloud bursting* as seen in Figure 2. Typically, less sensitive tasks are executed in the public cloud instead while tasks that requiring higher levels of security are provisioned the private infrastructure.

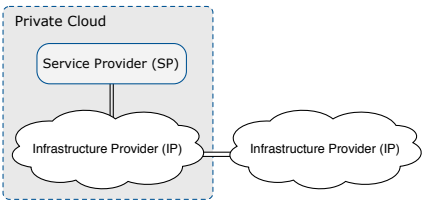

Fig. 2. Bursted (private) cloud.

Federated clouds are IPs collaborating on a basis of joint load-sharing agreements enabling them to offload capacity to each others [27] 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, an 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 3 illustrates a federation between three IPs.

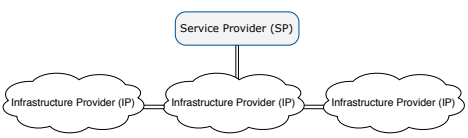

Fig. 3. Cloud federation.

If the IP 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 4, 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.

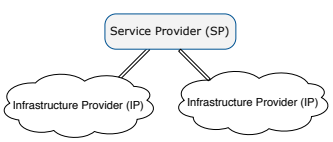

Fig. 4. Multi-cloud scenario.

A related scenario is when a *cloud broker* [32] handles the complexity of prioritization and selection of IPs, and may also offer value-add services to IPs and SP. In this case, the broker

has 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 an SP-style one to IPs, as illustrated in Figure 5.

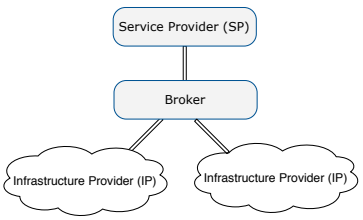

Fig. 5. Brokered scenario.

III. Requirements for Service Deployment

Based on the service lifecycle and the various cloud architectures discussed in Section II, we identify the below list of requirements for service deployment. Notably, the order and exact details of what is performed in each step of the service deployment process may vary with the deployment scenario, but the following tasks are always performed.

- *Discovery of IPs*. This step is about identifying 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 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).
- *Construction of deployment descriptor*. Each service must be defined a service description that specifies the functional and non-functional parameters of the service. A service description 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 description, 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*. It must be possible for an SP to negotiate with available IPs and ask these to provide offers for hosting the service (or parts of it). Based on the results these negotiations and data such as reputation statistics that could be evaluated by a third-party entities, the SP must be able to make a decision about where to deploy the service.
- *Service contextualization*. Before a service can be deployed, some information required to launch the service successfully, which is not known at the moment of VM image generation must be propagated to the IP. A possible mechanism for this *contextualization* process is to embed various scripts in the VM images that upon boot dynamically retrieves information such as network parameters, security credentials, etc., enabling the VM to self-contextualize.
- *Service data transfer.* It must be possible to transfer the contextualized VM images along with any other data required by the service to the IP. To be able to guarantee properties such as confidentiality and integrity of data during this transfer, a set of security mechanisms are required.
- *SLA creation*. To ensure that the service operates according to the SP's expectations, it must be possible to establish and SLA that governs the relationship between the SP and IP for the provisioning of the service. Penalties may be agreed upon in the case of non-compliance with the SLA. An SLA for service provisioning commonly includes segments that address: a definition of the service, performance measurements, problem management, customer duties, warranties, disaster recovery, as well as conditions for termination of the agreement [1].

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 [29].

IV. SDO architecture

Based on the requirements study in the previous sections, we derive a general sequence for service deployment, containing the tasks to be performed. In this process, illustrated by the sequence diagram in Figure 6, the complexity of each task vary with the deployment scenario (private cloud, federation, bursting, etc.).

Notably, Step 1, service construction, is identifical no matter the scenario, an SP constructs (implements, packages, etc.) a service the same way no matter how it will be deployed. Step 2, identification and filtering of suitable IPs, is trivial for SPs in the private cloud case, as there is a single, well-known IP available. This step is also relatively easy in cloud brokering scenarios, but more difficult in federation and multi-cloud cases.

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 6). For the scenarios where the SP interacts with a single provider (the private cloud IP or the broker), these tasks are simplified. Conversely, for federation, bursting, 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, etc., to approaches based on auctions. An in-depth analysis of negotiation protocols is beyond the scope of this paper. Further details on this topic are given, e.g., by Sarangan et al. [28] and Jennings et al. [17]. 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 [14], [32], but tend to scale very poorly with the number of IPs. Others recommend heuristic solutions [23] that trade optimality for faster decision-making.

Once the most suitable provider (or potentially, set of providers in the multi-cloud case) 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 may be more complicated if split deployment is performed for multi-clouds, as an external rendezvous mechanism typically is required in order to initialize cross-provider networking for the VMs of the service.

After VM images are properly configured, these are uploaded to the target provider(s) as illustrated in Step 6 of Figure 6. As VM images typically are very large files, significant performance gains can be achieved by proper tuning of network parameters. In private clouds where a network file system may connect IP and SP, image transfer is much less of an issue. Alternatively, if some public 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, the SP confirms the offered negotiated in Step 3 and an SLA is created between the SP and the IP for the provisioning of the, as illustrated in Step 7. Once again, this step is more complex for multi-clouds, where the SP need to aggregate multiple SLAs from different IPs.

Finally, Step 8 in Figure 6 illustrates that once the service is deployed, the SP stores some state information about it, to enable subsequent service monitoring, management, and undeployment.

To fulfill the requirements of service deployment and perform the steps in Figure 6, we propose a Service Deployment Optimization (SDO) architecture. The purpose of the SDO tool 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 in IP(s) according to the deployment plan. In order to separate the placement optimization from the deployment coordination functionality, the SDO is split into two components, the *Service Deployer* (SD), and the *Deployment Optimizer* (DO),

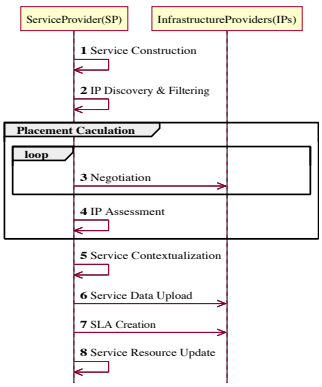

Fig. 6. Sequence Diagram.

both illustrated in Figure 7. 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 6.

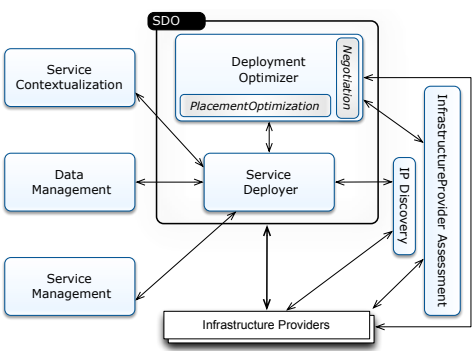

Fig. 7. Overview of SDO architecture.

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 SD is designed 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 6). 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 6 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 inputs from the SD include a service manifest, the optimization objective, and available IP info, etc. Based on those parameters, 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 6. Optimization techniques such as combinatorial optimization, problem relaxations and heuristic approaches such as greedy formulation can be applied in this component.

V. 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 [16], 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 [31]. 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 aminoacid 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 [13] algorithm on them (component GD). Additionally, there is one component for coordination (component GP). Each component can be encapsulated in a VM sized approx 9.8 GB. To avoid repetitive data transmission, only one VM image is transferred from SP to IP if there are multiple components deployed to the IP, while in this case multiple VM instances are to be started using the same image with different contextualized 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

In the integration with OPTIMIS, the SDO interacts with the following external components altogether providing the functionality of the external components illustrated in Figure 7.

- *IP Discovery:* IP information is registered in a simple online 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*: This 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 and a list of software dependencies [12].
- *Data Manager*: A Front-end, Hadoop-based [7] Data Management service enriched with RESTful APIs in front of Hadoop and a series of components that aim to extend Hadoop's functionality beyond its well known back-end, heavy data processing scope [21].
- *SLA Management*: A service and client based on WS-Agreement protocol [10] for negotiating and creating Service Level Agreements between IP and SP [22].
- *Infrastructure Provider Assessment*: In OPTIMIS, deployment decision is based on four key factors - trust, risk, eco-efficiency and cost (TREC). TREC parameters are used by DO to perform IP assessments.

B. Scenarios Descriptions and Statistics

- Private Cloud

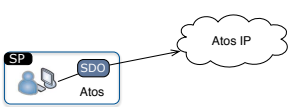

Fig. 8. Private scenario.

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 an SP. The other one is located in the BT cloud, which plays the role of a Cloud broker. The SP submits the

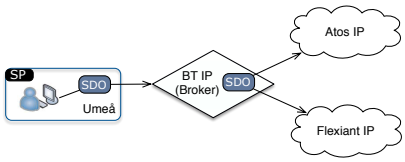

Fig. 9. Cloud brokerage scenario.

gene detection service deployment request to the Umeå cloud. Instead of deploying the service by itself, the 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

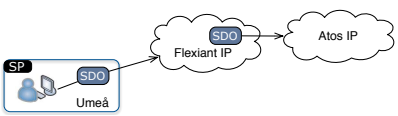

Fig. 10. Cloud bursting scenario.

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 this table, we conclude that the major part of the time 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 placement scheme. For the private cloud scenario, all components are deployed to Atos. Only one VM image needs to be transferred internally. For Multi-Cloud scenario, two VM images are transferred, one from BT to Atos (for components GA, GB, and GP), the other one from BT to Flexiant (for components GC and GD).

TABLE II
ILLUSTRATIVE PERFORMANCE RESULTS FOR THE DEMONSTRATED DEPLOYMENT SCENARIOS (SECONDS)

Deployment Phases	Private	Multi-Cloud	Cloud Bursting
IP Discovery	0	2	1
Placement Calculation	2	108	13
VM Contextualization	11	19	15
Data Upload	598	1546	701
Service Resource Creation	4	4	2
Agreement Creation	12	23	17

Another observation is that placement calculation becomes more complex in the multi-cloud case, where the number of potential service configurations is much larger than for the private and bursting cases. During the brokering case, multiple negotiations are performed between BT cloud and Atos cloud, and Flexiant cloud for cost inquiry. In addition, IP assessment is also based on IP evaluations in terms of trust, risk-level, and eco-efficiency which are independently verified by querying a trusted database containing historical information pertaining to these factors .

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 cloud brokerage and federation 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 [20], eco-efficiency and cost.

VI. Related Work

Talwar et al. [30] review approaches for service deployment before the emergence of Cloud Computing. They compare and evaluate four types of service-deployment approaches, i.e., manual, script-, language-, and model-based solutions, in terms of scale, complexity, expressiveness, and barriers for first usage. They also conclude that service deployment technologies based on scripts and configuration files have limitation to express dependencies and verify configurations, and usually result in erroneous system configurations, while language- and model-based approaches address these challenges with comparatively higher barriers for first usage.

With the emergence of Cloud Computing, services are provisioned using virtual machines. Service deployment can be done by initializing virtual machines with their virtual appliances. Cloud users are enabled to deploy applications without confronting the usual obstacles of maintaining hardware and system configurations. Lots of work have been done in the context of this new service-deployment technology. Most of these are focusing on approaches to optimization, e.g., Kecskemeti et al. [19] who propose an automated virtual appliance creation service that aids the service developers to create efficiently deployable virtual appliances. They reduce deployment time of the service by rebuilding the virtual appliance of the service on the deployment target site. For optimal

virtual machine placement across multiple cloud providers, Chaisiri et al. [14] propose an stochastic integer programming (SIP) based algorithm that can minimize the cost spent in each placement plan for hosting virtual machines in a multiple cloud provider environment under future demand and price uncertainty. Similarly, Vozmediano et al. [26] [25] explore the multi-cloud scenario to deploy a computing cluster on top of a multi-cloud infrastructure, for solving 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.

Our previous contributions in this field include a cloud brokering mechanisms [32] for optimized placement of VMs to obtain optimal cost-performance tradeoffs across multiple cloud providers in static scenarios, and 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 [24], as well as an approach to optimal virtual machine placement within datacenters for predicable and time-constrained load peaks [23].

However, although algorithms for optimizing service deployment is a very active area of research, and a lot of interest is given to the various deployment architectures in general, we have not been able to identify any results on the topics of this contribution, namely architectures and tools general enough to support all current deployment scenarios.

VII. Conclusion 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 [15]. 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 [24]. 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 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. Financial support has in part been provided by the European Community's Seventh Framework Programme ([FP7/2010-2013]) under grant agreements no. 257115 (OPTIMIS [16]), 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] Software as a Service (SaaS). Cloud Taxonomy. http://cloudtaxonomy.opencrowd.com/taxonomy/software-as-a-service/, visited May, 2012.
[10] Web Services Agreement Specification (WS-Agreement). http://www.ogf.org/documents/GFD.107.pdf, visited May, 2012.
[11] M. Ahronovitz et al. Cloud computing use cases white paper, v4.0. www.cloudusecases.org, visited May 2012.
[12] 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.
[13] E. Birney, M. Clamp, and R. Durbin. GeneWise and Genomewise. *Genome Research*, 14(5):988–995, May 2004.
[14] 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.
[15] 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.
[16] 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.
[17] 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.
[18] 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.
[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[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] 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.

[28] 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.

[29] J. Schopf. Ten actions when grid scheduling. In *Grid resource management*, pages 15–24. Kluwer Academic Publishers, 2003.

[30] 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.

[31] 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.

[32] 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.