A Business Driven Cloud Optimization Architecture

Marin Litoiu
York University, Canada
mlitoiu@yorku.ca
Murray Woodside
Carleton University, Canada
Johnny Wong
University of Waterloo, Canada

Joanna Ng, Gabriel Iszlai
IBM Centre for Advanced Studies
IBM Toronto Lab, Canada
{jwng|giszlai}@ca.ibm.com

ABSTRACT.

In this paper, we discuss several facets of optimization in cloud computing, the corresponding challenges and propose an architecture for addressing those challenges. We consider a layered cloud where various cloud layers virtualize parts of the cloud infrastructure. The architecture takes into account different stakeholders in the cloud (infrastructure providers, platform providers, application providers and end users). The architecture supports self-management by automating most of the activities pertaining to optimization: monitoring, analysis and prediction, planning and execution.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures-*domain specific-architectures*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems- *distributed applications*; C.4 [**Computer-Communication Networks**]: Performance of Systems- *modeling techniques, performance attributes*; K.6.3 [**Management of Computing and Information Systems**]: Software Management-*software maintenance*;

General Terms

Management, Measurement, Performance, Optimization

Keywords

Cloud computing, optimization, software as a service, performance engineering, performance modeling

1. INTRODUCTION

Cloud computing is emerging as a new computational model in which software is hosted, run and administered in large web data centers and provided as a service. By using web technologies, both hardware and software are delivered over the Internet seamlessly, similar to running them on a local machine. The long held dream of providing computing as a utility has been made easier by two emerging technologies: *virtualization* and *software as a service*. Virtualization is a process of substitution in which a physical resource is substituted by many logical (virtual) resources. The virtual resources retain the properties of the original resource, but will have less capacity. Software as a service is the delivery of software functionality online, similar to the one installed on a local machine. Depending on the content of the service, a cloud can offer Infrastructure as a Service (raw computing services such as CPU and storage), Platform as a Service (COTS, tools, middleware for developing and deploying applications) and Software as a Service (end user services).

Cloud computing's main motivation is economic. By eliminating the up-front cost and commitment of its users, the cloud allows companies to start small and increase hardware and software resources only when there is an increase in their needs. The pricing models are flexible and predictable to allow end users to plan ahead and know the cost of resource usage well in advance. Depending on the type and level of service, the pricing models include: *per use*, *per subscription* and *per transaction.* At the infrastructure level, the accepted model is pay per use of computing resources (processors, storage, bandwidth). Users can acquire the resources when needed, and release them when no longer used, hence achieving energy and computing resource conservation by letting machines and storage go when they are no longer useful. It is estimated [18] that, by statistically multiplexing the resources in large scale economies, cloud computing uncovers factors of 5 to 7 decrease in cost of electricity, network bandwidth, operations, software, and hardware available at these very large economies.

A cloud environment is by definition a self-serving service, in which the end users develop, deploy and run applications with minimum or even without cloud administrator help. Dynamic and programmatic reservation and release of hardware and software resources, configuration and tuning, uploading and running applications have to be done remotely. Therefore, many of the tasks performed currently by the human administrator have to be implemented by a cloud management infrastructure.

The work that motivated this paper started with the premise that a cloud is driven by business and automation goals. We

SAC'10, March 22-26, 2010, Sierre, Switzerland.

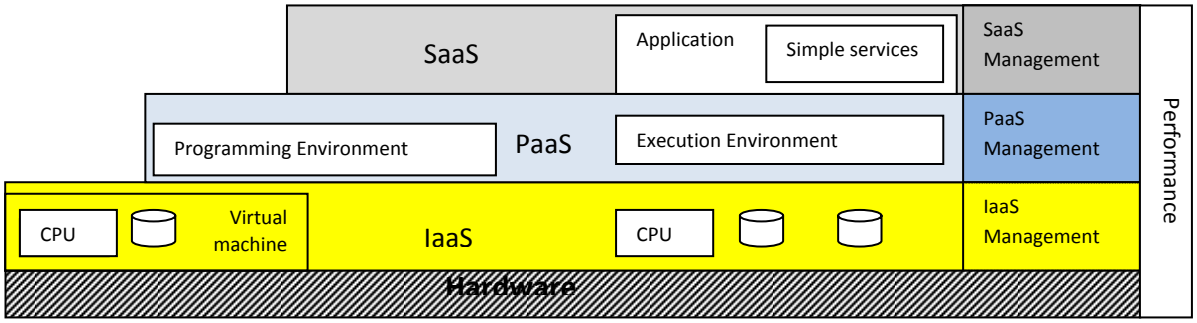

Figure 1. Cloud layers

describe the CERAS cloud[3] architecture for goals optimization. We show how business drivers are expressed as optimization goals and how we can achieve those goals. We consider a layered cloud where various cloud layers virtualize parts of the cloud infrastructure. The architecture takes into account different stakeholders in the cloud (infrastructure providers, platform providers, application providers and end users). It supports self-management by automating most of the activities pertaining to optimization: monitoring, analysis and prediction, planning and execution. The original contributions of the paper are:

-a layered optimization architecture integrated with the layering of the cloud. This layering matches the concerns of the various stakeholders and is well adapted to optimization techniques for complex systems.

-a model adaptive feedback control loop, present at each layer, and the coordination between different feedback loops

-a road map of research issues needed to address optimization in the cloud

-a discussion of practical challenges in implementing the optimization

The paper builds on current trends in cloud computing. Commercialized first by Amazon[1] as Elastic Computing (EC2), Cloud is becoming part of many software companies offerings[19], [20]. Research in cloud computing has recently ramped up, ranging from small scale projects to very large ones[21]. Despite of the spread of cloud computing projects, there is no clear vision of how different layers of the cloud, possibly in different administrative domains, can collaborate to satisfy stakeholders' goals. Li et al. [10][11][12] have shown how to optimize the cost in a private cloud in the presence of physical, logical and licence constraints. Zhang et al.[8] show how to implement a feedback control loop that optimizes the number of physical resources in a private cloud by applying autonomic computing principles [6][7]. This paper looks at public clouds and considers optimization in a layered architecture, where a feedback loop has only a limited view of resources and goals.

The remainder of the paper is organized as follows. Section 2 presents an optimization cloud management architecture, the details of the architecture and the challenges in its implementation. Conclusions are presented in Section 4.

2. ARCHITECTURE

We consider a layered Cloud architecture, with three layers, as shown in Figure 1.

IaaS virtualizes the hardware layer and offers computing services such as storage, CPU and memory. The CPU, memory and local storage are packaged as virtual machines of different sizes, each with a price per hour. Storage is a continuous space with a price per kilobyte [1].

PaaS offers platform services, such as web, application and database servers and a programming model associated with it. Programmers can use this environment to develop, test and deploy applications. PaaS consumes the services of IaaS by requesting Virtual Machines and storage and deploying application containers on the virtual machines. There can be many PaaS deployed on the same IaaS.

SaaS layer consists of simple or composite services (applications) offered to the end users. Those applications are deployed in PaaS containers on topologies specific to each application. There can be many applications sharing the same PaaS. In general, a SaaS user pays a subscription [19].

The management of the services of different levels is implemented by a management pane (to the right of Figure 1). The functions of the management pane are layer specific. In the next sections we details the functions related to optimization.

2.1 Requirements

The cloud optimization infrastructure has to be structured in layers, following the architecture of cloud runtime. There will be therefore 3 optimization layers, corresponding to IaaS, PaaS and SaaS, possibly belonging to different administration domains. It follows that each layer should only use the services of the layer below, in a closed architecture. Reflecting different stakeholders' interests and the level of abstraction, each layer will have different optimization goals and different sensors and actuators. Figure 2 depicts the sensors, goals and actuators in each layer.

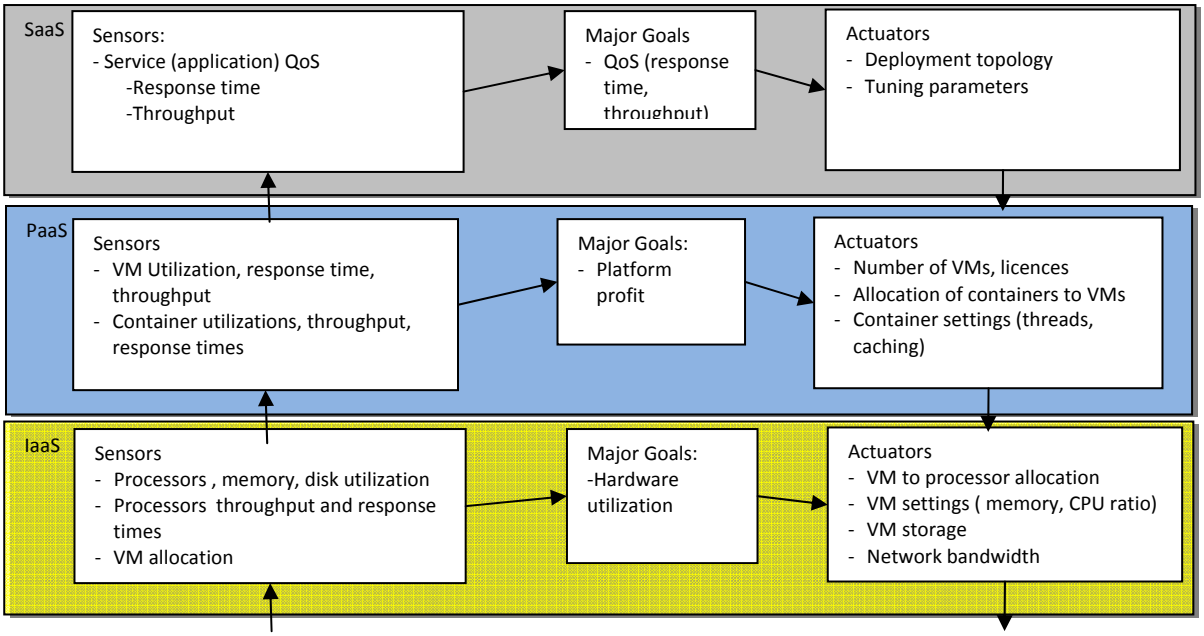

Figure 2 Conceptual optimization in cloud

2.1.1 Goals, Sensors, Actuators

The optimization goals of each layer reflect the layer's owner's economic interests, either to increase profit or maximize end user satisfaction. At the same time, due to abstraction induced by virtualization and by software as a service concepts, the means through which we measure the optimization (sensors) or drive optimization (actuators) are limited in type and numbers.

For the IaaS owner:

Goals. Cost and revenue are functions of number of resources and price per resource. Maximizing the resource multiplexing and therefore increasing the *resource utilization* is an important factor that can contribute to reducing cost and increasing revenue. Reports suggest that only 10% of the IT capacity is currently used, theoretically resource utilization can go up by a factor of 9. In practice, a factor of 5 to 7 is more feasible [18]. Service requests from the layer above are for individual VMs, with performance characteristics: memory, storage, processing capacity. To reach maximum utilization, the management layer has to fit all the requirements into the smallest number of hardware machines.

Constraints. Capacity contracts with PaaS owners, and the available processors, memory, storage, and networks (which may change due to failures or equipment additions).

Sensors. The IaaS layer has access to all hardware counters as well as hypervisor and operating systems counters. The sensors measure the used capacity, the available capacity, the availability and the location of each resource.

Actuators: VM allocation and storage allocation are the main actuators to satisfy the performance requirements and improve the utilization of the layer. Also, activation and deactivation of VMs can be used to increase utilization without affecting the performance requirements.

For the PaaS owner:

Goals. The revenue of this layer comes from the application/services it hosts. The cost is that of the resources (VMs, storage) it consumes from IaaS, the cost of third party licences (Application or Data base servers, for example) and the penalties it has to pay for SLA breaches. Therefore the business goal of this layer is to maximize the profit by maximizing the number of applications it hosts and minimizing the resources it uses, and penalties it pays.

Constraints. SLA contracts with SaaS owners can be treated as hard constraints instead of costs via penalties. There may be provisioning delays in increasing the resources it has, which constrain these resources in the short term.

Sensors. This layer monitors the virtual machines it owns and the containers it deploys on the VMs. It has no access to the hardware resources performance counters or virtualization hypervisors. It is very likely that this layer has a pool of licenses and a pay per use licensing of third parties software and the monitoring of those resources has to be performed as well.

Actuators. This layer achieves its performance goals by acting on several handlers: the number, size and type of VMs, the allocation of containers to the VMs (trying to fit as many in a container, container settings, number of resources in use, etc).

For the SaaS owner:

Goals. Most likely, this layer will charge the end user for subscriptions (revenue proportional to the number of users), or transactions (revenue proportional to the throughput). Its costs include the payment for resources to PaaS which can also be by usage or by subscription. We assume that the application owner will deploy the application together with a set of policies that capture the acquiring of resources from PaaS, the obligation of PaaS to provide those resources, the price per resource type, the penalties for not providing the resources, etc.

Constraints. There may be SLAs with application users that can be treated as hard constraints.
Sensors. Each application monitors its Quality of Service as it tries to optimize its goals (the number of users and the number of transactions). QoS can include response time for each user request and throughput of aggregated functions across multiple requests.
Actuators: To reduce its cost, an application will try to limit the number of resources it requests from PaaS. It should be able to control the number of instances of containers, the deployment of the application on the containers, tune application parameters.

2.1.2 Optimization Scenarios

In a live cloud, the optimization is a continuous activity, being triggered by the deployment of new applications, change in workload conditions, failure of hardware and software components, maintenance activities, etc.

Scenario 1: a burst of load for an SaaS application. The application must identify its need for additional capacity or better load balancing (by optimizing its use of resources in the new situation), the PaaS layer must find an optimal decision to provide it from resources it already possesses (deploying additional containers) or request more from the IaaS layer, which must deploy and allocate additional VMs in an optimal manner.

Scenario 2: a node failure suddenly reduces resources available at all layers. The IaaS layer must recognize the failure and act to replace the lost node by deploying new VMs and possibly copying state data. The PaaS layer then may respond by deploying replica containers in these VMs, and finally SaaS layer will deploy replica software. While this is happening, the PaaS and SaaS layers may have to cover a certain delay by shifting to an optimal temporary degraded mode of operation under the constraint of the missing resources.

Depending on the trigger of the scenario, the optimization can be local or global. Figure 3 shows two optimization modes: (a) *local optimization* in red, triggered by the deployment of a new application in a busy cloud (b) *global optimization* in blue, triggered by periodic maintenance and where all applications are optimally redeployed. The deployment/ optimization module computes the deployment plans and forwards them to a deployment engine which executes them. The optimization decisions are based on the *state* of the cloud which includes information about the applications and resources already allocated and also, in scenario (b), a tracked performance model for each application in the cloud. The state is measured continuously so the optimization service has the latest information when making decisions.

2.2 Adaptive Feedback Optimization and Control

In general, a feedback loop architecture satisfies the requirements of autonomic computing[6][7]. However, since the layers of the cloud have a time variant performance behaviour, the optimization and automation have to be *adaptive*, that is the model of the system, the optimization law and parameters change over time. In this section we introduce a variation of Model Identification Adaptive Control (MIAC) [5] that can be used for optimization and control of each layer in the cloud. In Figure 4, a p*erformance model* reflects the quantitative dependencies in each layer. The model is *identified* on-line by the *state estimator* and *workload classifier* components. Classes of service within a layer are denoted by *c*. A class might denote one application or a statistically similar set of applications or scenarios. *Perturbations* p_c include variable workloads for class *c*; *control* parameters u_c include tuning parameters or hardware resource allocation changes; *state* parameters x_c include per class service times and resource utilization; *output* parameters y_c include response times and class throughputs. The controlled changes u_c are computed by the *Optimization & Control* component, based on the current model of the layer (indicated in the figure by the oblique line) and the goals of the optimization. There are also parameters for the resources, including capacity limits and latencies.

The next sections illustrate the research challenges for each of the components of the adaptive feedback loop and for different layers of the cloud as well as the approaches that mitigate them.

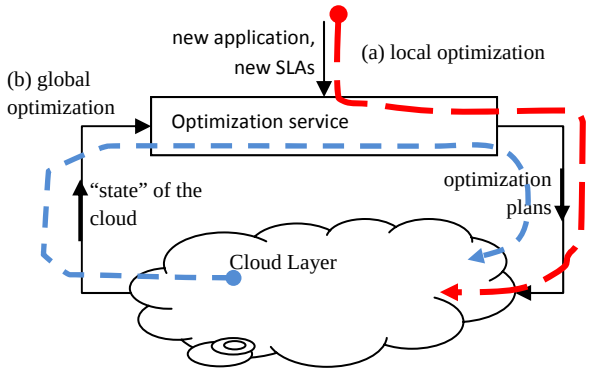

Figure 3. Two optimization scenarios in a cloud

2.3 Research Challenges

To implement a feedback adaptive loop for each layer, research is required in several areas. Research challenges include the realization of the components of Figure 4, their interaction and the achievement of local and global properties of the cloud.

2.3.1 Sensors and Monitoring

In order for each layer to implement its optimization functions, it should have access through an adequate set of sensors to metrics from the layer below. The sensors should be accessible through a standard interface which allows access to the performance counters with different sampling rates and different statistics. In general, monitoring interfaces such as JMX [22] and ARM [2], are good candidates for the PaaS and SaaS while hardware and operating system counters can satisfy the requirements for IaaS. However, the volume of the monitored data, the distribution of the data to a central management node, and the availability of the data for different control and optimization schemas raise serious challenges.

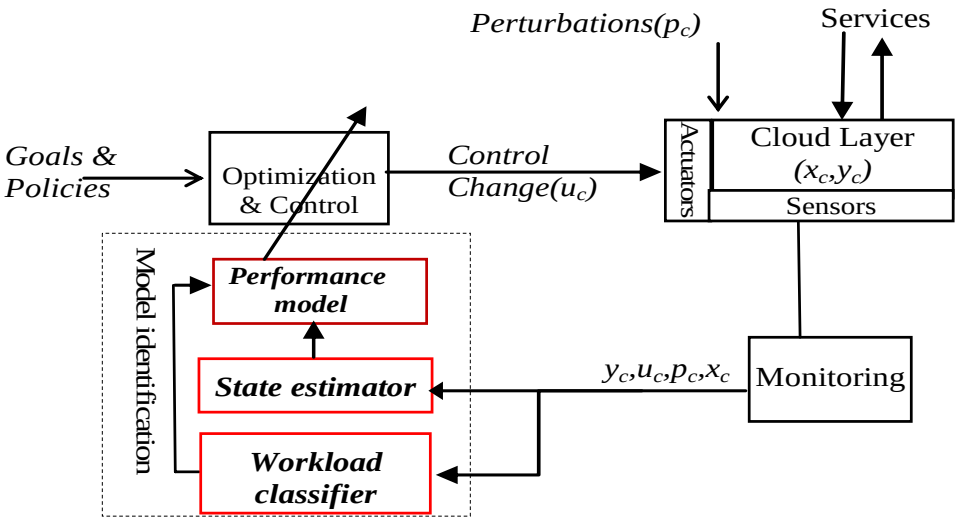

Figure 4. A feedback loop architecture for one cloud layer

2.3.2 Model identification

To implement its role, an adaptive feedback loop needs an explicit model to evaluate the state of the layer, evaluate the trends, and estimate the impact of possible changes. Prediction of the future workloads or breaches of SLA allows the optimization component to take proactive steps to provision more hardware and software resources to avoid loss of revenue or users. We take the approach that models, estimation and prediction techniques are specific to each layer. For example, Queuing Network Models and Layered Queuing Models models are appropriate at PaaS and SaaS layers. However, the assumption is that the performance parameters such as service time are available. In a layered model, that might not be the case, since the need for estimating those parameters from indirect measurements. At the IaaS layer, the number of components, interactions and type of workloads might be too large to model with traditional performance prediction techniques. IaaS should be treated as a large scale system [4] and techniques from this domain should be investigated. For all layers, to reduce the complexity, application with similar statistical behaviour or similar. SLA are grouped in classes of services by workload classifiers.

2.3.3 Optimization

Optimization is the process of monitoring the state of the layer, comparing with the target goals and deciding what has to be changed in the system so the target goals are met. Simple classic controllers such as Proportional Integrative Derivative (PID) might work for simple local loops, but at the layer level, we need more complex optimization techniques. Linear, non-linear and constraint programming augmented with predictive algorithms can be applied to achieve the business goals of each layer. Initial steps towards that have been shown in [12], [10][11][14] where instant optimization is addressed in a private cloud and in [14] where a model predictive control is defined. Main challenges remain in terms of scaling the techniques to ultra large number of components, to express the business goals in formal ways (using goals models), in including a risk model when VM migration is involved.

2.3.4 Coordination of the layer optimizations

While the layers are viewed here as separate control/optimization problems they are connected by contracts between layers, the system propagating the effects of decisions, and potentially by data or hints passed between layers by the optimizers. Explicit coordination can provide a way to address conflicts and partial centralization of decision-making, discussed next.

2.3.5 Conflicting optimization objectives

Assuming different administration domains, the optimization objectives of different layers can easily be in conflict with each other. For example, IaaS would like as many VMs as possible, to increase the revenue while PaaS will want as few as possible to reduce the cost. The layering and the contracts between the layers should make sure the provisioning and unprovisioning of the resources has to be the responsibility of the payer. At the same time the service provider has to honour the request type in a timely manner so the intermediate layer can honour the request of the layer above. Still, an open architecture and an auditing process that build trust across layers need to be in place. A SOA governance model, similar to the one proposed in [14] can partially satisfy the auditing and the trust building effort.

2.3.6 Centralized versus Decentralized Architectures

The architecture in Figure 4 can be realized either centralized or decentralized. At the SaaS layer, one optimization loop per application can be realized using a centralized implementation. There are going to be many independent optimization loops at the SaaS layers, one per application. At the IaaS layer, a decentralized architecture

performs better. The hardware can be partitioned in clusters, eventually distributed geographically, each cluster being managed locally by optimization loops like that in Figure 4. Communication and coordination between neighbouring clusters can be done either through peer to peer protocols or using a federating layer that communicates with PaaS. At the PaaS layer, a combination of centralized and decentralized implementation is feasible, depending of the size of each PaaS hosted in the cloud.

2.3.7 Metamodels in the Cloud

For a successful optimization, each layer has to have a good understanding of the behaviour and performance characteristic of the layer below. Metamodels can help in that regard. A step in that direction is shown in [11] in the case of a private cloud. Figure 5 shows a simplified metamodel of the cloud as an UML class diagram which can be used by the decision maker engine to find relevant information about each layer.

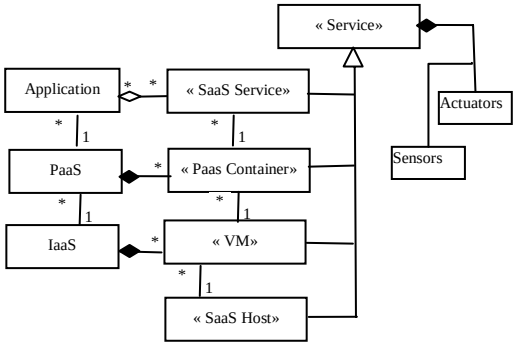

Figure 5. A simplified meta model of cloud

3. CONCLUSIONS

The optimization architecture described above solves several problems in designing cloud management schemes. It solves the problem of scale by partitioning the problem between layers and, at the cloud user layers, between cloud users. It coordinates the separate problems through virtualization of resources in the optimization. It supports approximate optimization using available techniques such as linear programming, to balance complex business factors and come at least close to optimal profits at all layers. It provides self-adaptation through feedback mechanisms at each layer.

4. REFERENCES

[1] Amazon Web Services, http://aws.amazon.com/, Aug 2009.

[2] Application Response Measurement, ARM, http://www.opengroup.org/management/arm/, Aug 2009.

[3] CERAS project, https://www.cs.uwaterloo.ca/twiki/view/CERAS, Aug 2009.

[4] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau, *Ultra-Large-Scale Systems: The Software Challenge of the Future*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006.

[5] Brun Y. et al. "Engineering Self-Adaptive System through Feedback Loops," in *Software Engineering for Self-Adaptive Systems*, Cheng B. et al. (Eds), pp 48-70, Springer Verlag, 2009.

[6] J. Kephart and D. Chess. The vision of autonomic computing. IEEE Computer Journal, 36(1):41–50, January 2003.

[7] S. White et al. An architectural approach to autonomic computing. In Proc. 1st IEEE Intl. Conf. Autonomic Computing, pages 2–9. IEEE Computer Society, 2004.

[8] T. Zheng, M. Woodside, and M. Litoiu. Performance model estimation and tracking using optimal filters. IEEE Transactions on Software Engineering, 34(3):391–406, 2008.

[9] Y. Diao et al., An Adaptive Feedback Controller for SIP Server Memory Overload Protection, Proceedings of the ACM/IEEE International Conference on Autonomic Computing, pages 23-32 Barcelona, June 2009.

[10] J. Li, J. Chineck, M. Woodside, M. Litoiu, Fast Scalable Optimization to Configure Service Systems Having Cost and Quality of Service Constraints, Proceedings of the ACM/IEEE International Conference on Autonomic Computing, Barcelona, June 2009, pages 159-168.

[11] Li J., Chinneck J., Woodside M, Litoiu M., Iszlai G., "Performance Model Driven QoS Guarantees and Optimization in Clouds," *ACM/IEEE ICSE Workshop on Cloud Computing*, Vancouver, May 2009.

[12] Li J., Chinneck J., Woodside M, Litoiu M., "Deployment of Services in a Cloud Subject to Memory and License Constraints," *IEEE International Conference on Cloud Computing*, Bangalore, Sept. 2009.

[13] Ye H., Wong J., Iszlai G., Litoiu M., "Resource Provisioning for Cloud Computing," *Proceedings of CASCON 2009*, November 2009.

[14] Litoiu M., Litoiu M. V., "Cloud Optimization- A Governance View," *Proceedings of ISGIG 2009*, Prague, Sept. 2009.

[15] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer, 37(10):46-54, 2004.

[16] D. Q. Mayne, J. B. Rawlings, C. V. Rao and P. O. Scokaert., "Constrained model predictive control: Stability and optimality", *Automatica*, 36, 2000, pp. 789-814.

[17] D. J. Watts and S. H. Strogatz., *Collective dynamics of 'small-world' networks*, Nature Vol 393 (1998) 440-444.

[18] Armbrust M. et al., *Above the Clouds: A Berkeley View of Cloud Computing*, Technical Report, http://radlab.cs.berkeley.edu/, February 10, 2009.

[19] Salesforce Cloud Computing Platform, http://www.salesforce.com/platform/, Aug, 2009

[20] IBM Cloud Computing, http://www.ibm.com/ibm/cloud/, Aug 2009.

[21] EU's FP7 RESERVOIR project, http://www.reservoir-fp7.eu/, Aug 2009.

[22] Java Management Extensions, JMX, http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/, Aug 2009.