Towards a Cloud Optimization Architecture Using Strategy-Trees

Bradley Simmons and Marin Litoiu
York University, Canada
{bsimmons || mlitoiu}@yorku.ca

Dan Ionescu
University of Ottawa, Canada
ionescu@site.uottawa.ca

Gabriel Iszlai
IBM, Toronto Lab, Canada
giszlai@ca.ibm.com

***Abstract*—Cloud computing represents an emerging approach to on-demand computing. This paper proposes an extension to a three-layered cloud computing architecture through the use of strategy-trees. Managers at each layer of the cloud architecture, representing a provider's perspective, will utilize strategy-trees to implement feedback loops to achieve sets of objectives over defined horizons of time.**

I. Introduction

Cloud computing [1–4] represents an emerging approach to *on-demand* computing provided over the Internet that has come into vogue due to both advances in virtualization [5] techniques and the construction of numerous large commodity data centers across the globe [2]. The emergence of the *cloud* offers an opportunity for economies of scale previously unavailable to small and medium size enterprises. Specifically, an enterprise may now purchase resources as needed (scale-up/scale-down) while not procuring or maintaining these resources themselves. This also offers benefits to larger enterprises as well who may either harness the cloud and/or present an outward facing cloud offering to others.

Presently, there are three types of cloud offerings available in the marketplace: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). IaaS offers virtualized resources such as virtual machines (VM) and storage space offered as different types of VM per unit time and storage space per dollar. Amazon's Elastic Compute Cloud (EC2) [6] and Elastic Block Storage (EBS) [7] are representative of this type of service. EC2 is also representative of PaaS in which various containers are deployed withing a VM (i.e., HTTP, application and database servers). Google App Engine [8] can also be considered a PaaS offering. An example of a SaaS offering would be Salesforce.com [9].

The intent to define an architecture based on the three layers of the cloud's current runtime (IaaS, PaaS and SaaS) resulted in the architecture introduced in [10]. Decomposing the cloud into these three layers also has the added benefit of decoupling layer-specific problems from one another. For example, where a IaaS provider may want to maximize the number of VM instances (VMI)s (in order to increase revenue) the PaaS provider may want to minimize the number of VMIs purchased (to minimize its costs) [10]. In alternative cloud architectures [3, 4] this becomes an extremely complicated problem; however, in this fully decoupled, three-layered architecture providers at each layer attempt to best achieve their own objectives over defined time horizons. Specifically, this architecture facilitates optimization according to business objectives [11] at each layer.

While on one hand the perception of an infinite set of resources to draw from is novel, the companies that will succeed in the long term will have strategies for best utilizing their (scarce) resources. This is true of providers at each layer of the cloud. Multiple strategies may exist for achieving a set of objectives (i.e., directive). The strategy-tree [12–14] is an abstraction that encapsulates and relates multiple strategies for achieving a set of objectives over a defined horizon of time.

This paper proposes the use of strategy-trees in managers at each layer of the cloud architecture. Strategy-trees have interesting properties with regards to temporal decomposition of runtime management. They allow for assumptions about performance to be methodically tested and utilized in decision making over multiple granularities of time. Strategy-trees are a good candidate for facilitating feedback loops for providers at each layer of this architecture.

The remainder of this paper is structured as follows. Section II will introduce the business driven cloud optimization architecture. Section III will detail the proposed extensions to this architecture most importantly the addition of strategy-trees to the managers at each architectural layer. Section IV will present a case study of an IaaS provider and explore aspects of the design of its strategy-tree. Section V will offer some closing thoughts.

II. Architecture

The architecture, Figure 1, consists of three layers that mirror the runtime environment (IaaS, PaaS and SaaS) of the contemporary cloud. At each layer of the architecture there are optimization loops to consider [10]. Specifically, each provider (IaaS, PaaS and SaaS) and/or client will have a set of objectives to achieve over various horizons of time. Further, there will typically be a set of constraints impacting the achievement of these objectives.

For example, a SaaS provider must meet its Service Level Agreement (SLA) [15] obligations to clients while minimizing the cost of its PaaS commitments. These obligations will typically involve QoS issues; however, they may also involve other types of issues (i.e., data residency) as well. Specifically, certain laws require that all data about citizens residing in a

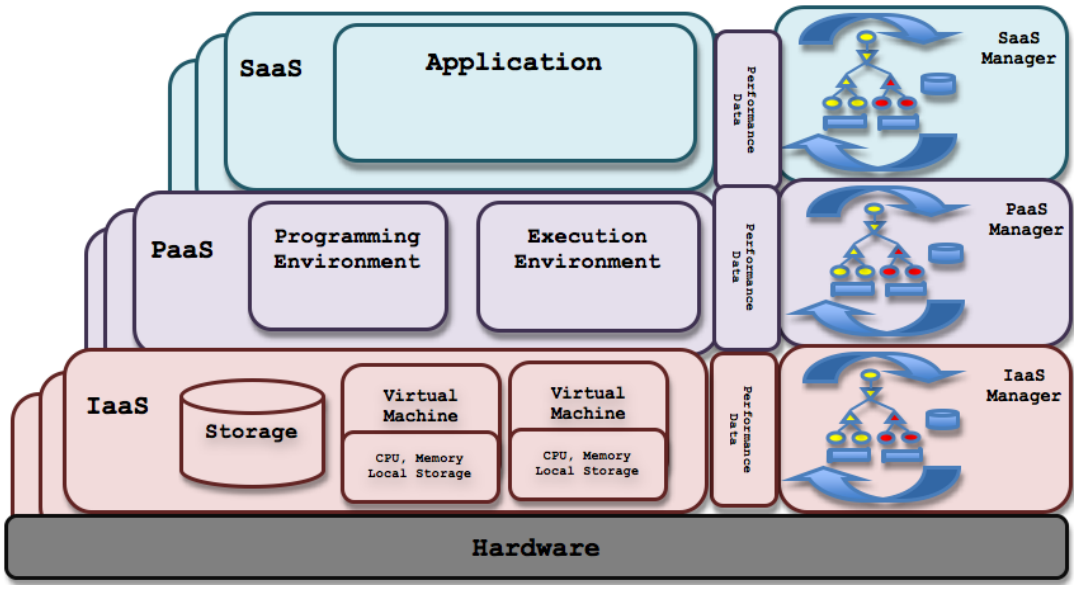

Fig. 1: Three-layered Cloud Architecture. Managers are defined that utilize strategy-trees to achieve a provider's objectives.

particular country must not be moved abroad subject to certain guarantees. This necessitates that all VMIs and storage remain within a specified geographic locale. So, a SaaS provider purchasing PaaS must be guaranteed that the IaaS layer is instantiating VMIs and storage accordingly.

Similarly, a PaaS provider must meet its SLA obligations to its clients while minimizing the cost of its IaaS commitments. Consider the situation in which VMIs can be purchased more cheaply from provider A than from provider B. It makes sense for the PaaS provider to utilize these VMIs; however, this must be permissible in accordance with the SLAs it holds with its SaaS clients. Another consideration may be degrees of redundancy. If a SaaS application is meant to provide specific types of redundancy (i.e., failover, etc.) it may make more sense for the PaaS provider to ensure that various containers are not hosted on one machine or one rack (or in one geographical location for that matter) by the IaaS provider. Again this must be conveyed through contractual specification in the SLA.

Finally, the IaaS provider must meet its SLA obligations to its clients while best utilizing its hardware infrastructure. While on the surface it may seem obvious that this is a packing problem (i.e., pack the maximum number of instances onto fewest hardware resources) in fact, it is more complicated. Specifically, there are constraints with regards to VMI placement to consider related to energy costs (i.e., jurisdictional price differences and aspects of power management) further, performance issues related to heavy loading of VMIs onto individual hardware entities must be considered as well. Constraints may also exist relating to issues of contractual obligations to PaaS providers who need to ensure computation within specific locales or provide guaranteed redundancy. One other issue that IaaS providers must consider is the impact of tightly packing hardware nodes such that performance bottlenecks materialize. For example, work done [16] demonstrates that on EC2, background load impacts I/O performance.

A layer-specific manager (there may be multiple managers which behave as a single manager logically) representing the provider's interests is defined at each layer of the architecture. The manager encapsulates an optimization-based control of its particular domain. It utilizes sensors and actuators to monitor and affect control. A model captures the dependencies between actuators and objectives. There are multiple ways to express these dependencies through formal linear, non-linear, queuing models or as policy [17, 18] sets. A strategy may represent one of many paths from sensors to actuators. Each actuator is associated with various models.

Key to the optimization over time is feedback. A monitoring infrastructure is required for each layer of the cloud. This system must be scalable, and lightweight allowing for the collection of metrics about performance which can be used by the various stakeholders to perform optimizations at each layer. Possible types of monitoring infrastructures exist ranging from centralized [19] approaches to completely distributed approaches [20].

Control is exerted through actuators. At the IaaS layer a provider has control over things like VMI and storage allocation and configuration. At the PaaS layer a provider has control of things like how many and what type of VMI it has ownership of, what containers and how many containers it instantiates within a VMI and the configurations of the containers it instantiates. At the SaaS layer a provider has control over things like how many container instances are running and configuration parameters for the software that is run within the container.

III. Feedback Loops with Strategy-Trees

The architecture presented in this section represents an evolution of the architecture presented in [10]. The evolution

is concerned with adding strategic management capabilities to providers at each layer of the architecture to better achieve their business objectives over various time horizons.

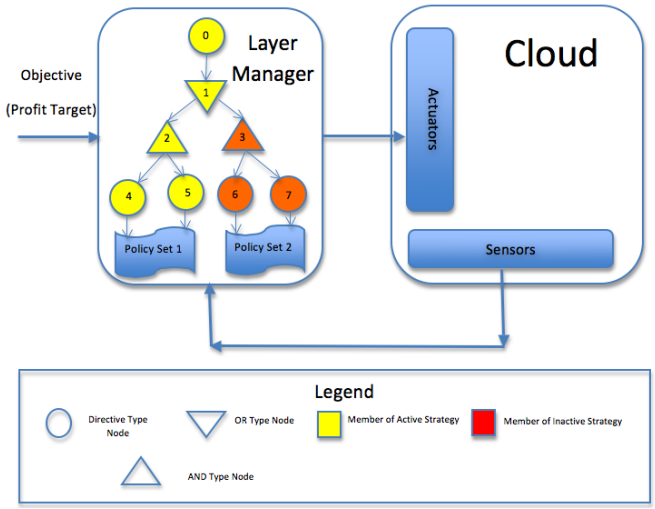

Fig. 2: Strategy-trees as autonomic managers for cloud layers.

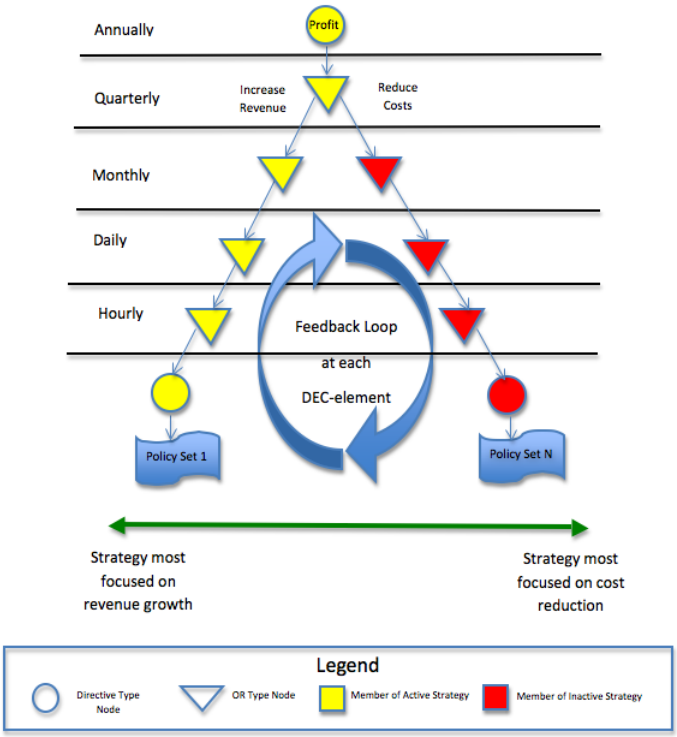

Fig. 3: Conceptual example of strategy-tree design. Notice that any specific implementation may or may not use the selected temporal granularities (although these are sensible choices in the context of a cloud provider). Also, notice that a feedback loop is operational at every DEC-element (i.e., OR Type Node) in the tree.

At each layer of the architecture there is a feedback loop, Figure 2. Specifically, it is a *model adaptive feedback loop* in which the system model, optimization law and parameters change over time [10]. The feedback loop is attempting to achieve a high-level goal. However, a goal may have multiple decompositions resulting in different policy sets [21]. The selection of a single pathway through the goal graph encapsulates assumptions and biases that must be considered [14]. It has been demonstrated [13] that there is not an optimal policy set to handle all situations.

A. *Strategy-Trees*

The strategy-tree was introduced to address these issues. Specifically, a strategy-tree is an abstraction that encapsulates and relates multiple strategies for achieving a set of objectives over a defined horizon of time. A strategy-tree has been added to managers at each layer of the architecture to facilitate dynamic alternation among possible strategies for achieving the provider's objectives over time. Strategy-trees will be used to facilitate feedback loops at each layer of the cloud architecture.

A strategy-tree is composed of three types of nodes: Directives, AND and OR. The Directive type represents the super-type for the other node types and encapsulates a set of objectives to achieve over some horizon of time (epoch). Further, every node in a strategy-tree has a *quantum attribute value* which defines its temporal evaluation schedule (i.e., equivalent to the duration of the node's epoch) in terms of the *management time unit* (MTU). The MTU represents the shortest interval on which monitoring is performed. Leaf nodes of a strategy-tree are bound to policy sets. A policy can be understood to represent "...any type of formal behavioural guide" that is input to the system [22]. A policy set represents a *strategy* for achieving a set of objectives.

For each node of a strategy-tree the administrator defines a SAT-element (i.e., satisfaction element) which encapsulates the evaluation of a set of objectives to be achieved. These SAT-elements are evaluated once per epoch (based on the quantum attribute value of the node). A second type of element, a DEC-element (i.e., decision making element), is defined for OR type nodes in the tree. Like SAT-elements, DEC-elements are evaluated according to the node's quantum attribute value. The decision making defined within these DEC-elements analyzes performance, effectiveness of the current approach and makes a choice about whether to persist in using the current strategy or to switch to an alternative.

A strategy-tree provides an infrastructure by which run-time evaluation of the *active* strategy may be made at multiple temporal granularities. Further, due to its temporal partitioning it allows for different objectives to be evaluated on various time scales to better achieve the root directive.

Performance models are used to capture dependencies within a system. Under some conditions models can fail. Further, models may have different degrees of efficacy under varying conditions. Models must also be configured. Different configurations may lead to quite different degrees of effectiveness at runtime. Strategy-trees provide a framework through which alternating among multiple models and/or configurations of models can be performed as needed and as specified

explicitly in the DEC-elements. An aspect of these DEC-elements (and SAT-elements) is that they may be changed at runtime and thereby allow dynamic evolution of a particular strategy-tree. Evolution may be made in response to poor perceived performance of the strategy-tree as a whole based on external analyses or on insights gained via observations over time. As the cloud represents an incredibly dynamic and always running environment this ability is potentially quite useful.

Service providers have both global and local optimization loops to consider. From the provider's point of view asynchronous actions (e.g., satisfying a request for the addition and/or removal of resources, or recovering from a hardware failure (i.e., single box, rack, disk, ...) must be handled as they occur and so are more representative of local optimization loops. The effectiveness of responding to the asynchronous situations (i.e., local optimizations) can be utilized in decision making at the global scale in terms of incorporating monitored information about the resolutions of these situations into SAT- and/or DEC-element evaluations.

IV. Case Study

Consider the simplified case of a provider operating a single data center and offering IaaS to a set of clients in our three-layered cloud architecture. In order for a strategy-tree to be utilized the middleware that controls the IaaS services must be dynamically configurable (preferably through policy based management (PBM)). Specifically, a strategy is equivalent to a deployed policy set. Changes in policy set represent change in active strategy. The general process of designing a strategy-tree is presented in Figure 3. The following sections consider the specific case of designing a strategy-tree for the IaaS provider in this example.

A. Define the Root Directive

A directive encapsulates a set of objectives to be achieved over some time interval. Strategy-trees attempt to coordinate among multiple strategies over various time intervals to achieve a root directive. In order to construct a strategy-tree for this particular IaaS provider a root directive must be defined over some horizon of time. Since most businesses operate on a schedule of annual reports supplemented by quarterly reports it seems like a reasonable horizon of time to consider is one year. For this example, the set of objectives to achieve will be:

- *Increase revenue by 5% per annum*
- *Reduce cost by 2% per annum*

B. Policy Sets

Aspects of IaaS middleware that we feel might be dynamically configurable include (but are not limited to) the following:

- Queuing system used to accept requests from clients (e.g., FIFO or some form of prioritized queuing, admission control)
- Placement algorithm used for assigning VMIs to the hardware infrastructure (e.g., random placement, least loaded rack/machine, placement in relation to other VMIs purchased by a single client)
- Defragmentation of infrastructure (i.e., how often are VMIs replaced in an organized fashion onto the physical hardware and how often is this performed). This may also involve migrations (i.e., what triggers a migration).
- Power management (i.e., what machines, racks are on/off)
- Resource allocation in periods of plenty (i.e., are extra resource given to clients while less than some percentage of physical infrastructure is in use)
- Resource allocation in periods of contention (i.e., are certain configurations treated as more important)
- Threshold setting (i.e., acceptable load on a single physical machine, limits on inter-instance bandwidth, ...)
- Whether hints are pushed to the client with regards to running out of resources (i.e., suggesting purchasing more VMIs, or releasing VMIs) and to what degree
- Recovery algorithms for handling various forms of failures (e.g., machine failures, rack failures, does a priority get imposed for more important clients, etc.)

C. Economic Model Underpinning Strategies

Consider the simplified case of a provider operating a single data center and offering IaaS to a set of clients (i.e., PaaS provider in our three-layered cloud architecture). Changes in policy set represent a change in the *active* strategy. An overview is provided in Figure 3 of how a strategy-tree might be designed for the general case (i.e., any service provider). Notice in the figure that there are potentially numerous strategies (we only show two).

For this example, the provider only offers three VMI configurations: C_0, C_1 and C_2. These classes are ordered $C_0 >> C_1 >> C_2$ in terms of revenue generation to the provider. Clients may purchase as many of a single type of VMI per request. Leases are automatically renewed hourly. The client must explicitly terminate instances. An SLA exists between the provider and each client which encapsulates guarantees about the VMI being provided and its expected performance. Further, the SLA contains clauses which define penalties should performance be degraded or non-available [1].

The hourly revenue stream may be computed as follows:

$$r = \sum_{s \in SLA} (w_0 * c_0 + w_1 * c_1 + w_2 * c_2) \quad (1)$$

Where the coefficients w_0, w_1, w_2 denote the weight coefficients on the numbers of VMIs of a particular class. Specifically, c_i denotes the number of VMIs belonging to class C_i. The hourly total cost may be computed as follows:

$$t = m + v \quad (2)$$

[1]This is an extremely simplified example in order to introduce concepts about strategy-tree design. In a more realistic example a data center would charge for ingress bandwidth, egress bandwidth, block storage, ...

Where cost t is proportional to the wear on the machines m being used (i.e., fixed-costs, power, wear and tear, ...) and the penalties v charged during the hour for SLA violations. The hourly profit may be computed as follows:

$$p = r - t \tag{3}$$

D. Design of a Strategy-Tree

It has become apparent that a methodical approach to designing strategy-trees would be quite useful. This has led to the the approach presented here which utilizes a systematic breakdown of (i) time and (ii) emphasised objectives as presented in Figure 3.

For example, consider the strategy-tree presented in Figure 3. Traversing down the tree from root to leaf it is partioned temporally (i.e, yearly down to hourly). Similarly, traversing left to right, it is partitioned in terms of which objective is most emphasized [2]. Specifically, the left-most strategy presented in Figure 3, is focused on increasing revenue generation (to the exclusion of any consideration for cost reduction). Similarly, the right-most strategy is myopically focused on cost reduction (to the exclusion of any consideration for revenue generation).

In a strategy which emphasizes revenue generation singularly, the focus may be on servicing clients quickly, focusing more on those clients who are paying larger sums of money (i.e., purchasing VMIs of C_0 or many VMIs of C_2). Perhaps, a strategy like this would involve monthly defragmenting of the data center's hardware resources. However, should this not be effective, this would be realized at the monthly evaluation of strategy effectiveness and perhaps an alternative strategy would be selected by the DEC-element in which a bi-weekly de-fragmentation regiment is followed instead.

With regards to a cost reducing strategy perhaps an energy aware placement algorithm is in use. However, should numerous SLA violations be occurring due to an increase in client VMI requests and this is detected at the hourly interval a switch might be made to an admission control policy and the energy efficient placement algorithm might be disabled for some period of time (e.g., three hours or some period without a violation cost above some threshold being detected).

Once a set of strategies have been defined and rationales for altering among them determined at various temporal granularities, the SAT- and DEC-elements must be generated, edited and deployed. This process requires coding the logic within SAT- and DEC-elements to (i) evaluate the assumptions at various temporal granularities and (ii) decide between various strategies (i.e., based on feedback loops). This and the algorithm for evaluating the strategy-tree is covered fully in [12–14].

V. Discussion and Further Work

A strategy is defined to achieve a set of objectives under a set of assumptions. Multiple strategies may be defined to achieve the same directive. Designing strategies is a non-trivial task. A strategy-tree allows the administrator's assumptions to be tested at different temporal granularities from short intervals at the leaf nodes to longer temporal intervals traversing up the tree. This temporal decomposition is useful in that it allows a more comprehensive understanding of the system to be constructed facilitating more informed decision making at the various DEC-elements (with regards to alternating among strategies).

Determining when to evaluate assumptions, which assumptions to evaluate and when to make decisions are all currently open questions (and likely unique to particular strategies). Deciding how to alternate among the defined strategies is a further complicating aspect. Attempting to gain insights into these aspects of strategy and strategy-tree design is an important area of future research.

Regardless of whether a strategy-tree is being used, strategies must be defined and their effectiveness evaluated in order to improve the performance of large systems in general. The strategy-tree allows for assumptions underpinning these deployed policy sets to be tested at runtime at multiple temporal granularities and to construct a history which can then facilitate more effective alterations among policy sets at runtime (based on observed performance, and future prediction).

A provider has little control over the behaviour and expectations of its clients. Further, the pattern of client requests is not necessarily predictable beyond certain general characterizations. However, it is expected that either through extensive experimentation and/or simulation [23] the provider will be able to classify general types of workloads and likely to detect them algorithmically (to some degree based on pattern detection and trends analysis over time). While in general, the objectives are to increase revenue while limiting costs over the year, the strategy-tree designer must have determined various configurations of the system to best meet these objectives in various contexts.

Logically, metric values are written to a management database (MDB); however, the scale of the cloud may make using a centralized data store difficult (in terms of single point of failure, bottleneck, etc.) so it may be prudent for this to be implemented in a more distributed manner than previously (likely through various aggregation algorithms and gossip protocols). The hierarchical nature of a strategy-tree makes it a good fit for various aggregating approaches for data collection. Specifically, SAT-elements could be defined to evaluate periodically on a all active hypervisors and over time this could be aggregated so that a rack level understanding would be achieved.

Business objectives must be met over various horizons of time. This is a fact of business and not a unique problem of the cloud. Optimizations must be performed on both local and global scales [10]. An example of a local optimization might be adding an application to a the cloud while an example of a global optimization might be some periodically occurring maintenance phase where all applications are redeployed optimally [10]. We feel that optimization will play

[2]This progression across the permutations may omit some possibilities and add others but the idea is to move across the set of objectives in an orderly manner.

an important role in the long-term viability of the cloud.

While the work presented in this paper is at an early stage we are optimistic about its benefits. This paper has provided an overview of a three-layered cloud architecture augmented by the addition of strategy-trees which facilitate the achievement of objectives over defined time horizons for managers representing providers' perspectives at each architectural layer. We have begun experimenting with a cloud simulator [23] and are also in the process of building a private PBM-aware cloud at York University. Re-engineering of the *StrategyTreeEditor* for use on various clouds (i.e., ours, Amazon, Google) is also ongoing.

ACKNOWLEDGMENT

This research was supported by OCE, the Ontario Centres of Excellence, and by the IBM Toronto Centre for Advanced Studies, as part of the program of the Centre for Research in Adaptive Systems (CERAS).

REFERENCES

[1] B. Hayes, "Cloud computing," *Commun. ACM*, vol. 51, no. 7, pp. 9–11, 2008.

[2] 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," Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, February 2009.

[3] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, pp. 4:1 –4:11, july 2009.

[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Comp. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

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

[6] Amazon, "Elastic compute cloud (ec2)," 2010. http://aws.amazon.com/ec2/ [online May 10].

[7] "Amazon elastic block store (ebs)." http://aws.amazon.com/eb2 [online May 2010].

[8] "Google app engine." http://code.google.com/appengine/ [online May 2010].

[9] "Salesforce.com." http://www.salesforce.com/ [online May 2010].

[10] M. Litoiu, M. Woodside, J. Wong, J. Ng, and G. Iszlai, "A business driven cloud optimization architecture," in *25th Symposium on Applied Computing*, ACM, March 2010. 6 pages.

[11] J. Sauve, A. Moura, M. Sampaio, J. Jornada, and E. Radziuk, "An introductory overview and survey of business-driven it management," in *Business-Driven IT Management, 2006. BDIM '06. The First IEEE/IFIP International Workshop on*, pp. 1 – 10, 07-07 2006.

[12] B. Simmons and H. Lutfiyya, "Strategy-trees: A feedback based approach to policy management.," in *MACE* (S. van der Meer, M. Burgess, and S. G. Denazis, eds.), vol. 5276 of *Lecture Notes in Computer Science*, pp. 26–37, Springer, 2008.

[13] B. Simmons and H. Lutfiyya, "Achieving high-level directives using strategy-trees," in *MACE 2009: Modelling Autonomic Communications Environments, Fourth IEEE International Workshop, MACE 2009, Venice, Italy, October 26-27, 2009. Proceedings*, pp. 44–57, 2009.

[14] B. Simmons, *Strategy-Trees: A Novel Approach To Policy-Based Management*. PhD thesis, The University of Western Ontario, 2010.

[15] J. Sauvé, F. Marques, A. Moura, M. C. Samaio, J. Jornada, and E. Radziuk, "Sla design from a business perspective," *DSOM 2005: Proceedings of the 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, pp. 72–83, October 2005.

[16] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *OSDI*, pp. 29–42, 2008.

[17] M. Sloman, "Policy driven management for distributed systems," *Journal of Network and Systems Management*, vol. 2, pp. 333–360, December 1994.

[18] R. Boutaba and I. Aib, "Policy-based management: A historical perspective," *J. Network Syst. Manage.*, vol. 15, no. 4, pp. 447–480, 2007.

[19] "Nagios." http://nagios.org [online May 2010].

[20] R. van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Trans. Comput. Syst.*, vol. 21, no. 2, pp. 164–206, 2003.

[21] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, and G. Pavlou, "A methodological approach toward the refinement problem in policy-based management systems," *IEEE Communications Magazine*, vol. 44, pp. 60–68, October 2006.

[22] J. O. Kephart and W. E. Walsh, "An artificial intelligence perspective on autonomic computing policies," in *POLICY '04: Proceedings of the 5th IEEE International Workshop on Policies for Distributed Systems and Networks*, pp. 3–12, IEEE Computer Society, June 2004.

[23] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, "Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, 2010.