Optimal Autoscaling in the IaaS Cloud

Hamoun Ghanbari
Dept. of Computer Science
York University
Toronto, Canada
hamoun@cse.yorku.ca

Bradley Simmons
Dept. of Computer Science
York University
Toronto, Canada
bsimmons@yorku.ca

Marin Litoiu
Dept. of Computer Science
York University
Toronto, Canada
mlitoiu@yorku.ca

Cornel Barna
Dept. of Computer Science
York University
Toronto, Canada
cornel@cse.yorku.ca

Gabriel Iszlai
Centre for Advanced Studies
IBM Toronto Lab
Toronto, Canada
giszlai@ca.ibm.com

ABSTRACT

An application provider leases resources (i.e., virtual machine instances) of variable configurations from a IaaS provider over some lease duration (typically one hour). The application provider (i.e., consumer) would like to minimize their cost while meeting all service level obligations (SLOs). The mechanism of adding and removing resources at runtime is referred to as autoscaling. The process of autoscaling is automated through the use of a management component referred to as an autoscaler. This paper introduces a novel autoscaling approach in which both cloud and application dynamics are modelled in the context of a stochastic, model predictive control problem. The approach exploits tradeoff between satisfying performance related objectives for the consumer's application while minimizing their cost. Simulation results are presented demonstrating the efficacy of this new approach.

1. INTRODUCTION

A *cloud* represents an ultra large-scale system [26] focused on the delivery of computational power (specifically storage, network, and computation, and high-level services on top of these), in an *on-demand* fashion, to a consumer community. At the infrastructure as a service (IaaS) layer of the layered-cloud architecture [24], the cloud provider partitions physical resources into various configurations of virtualized memory, CPU and disk, a virtual machine instance (VMI), and offers them to consumers, at variable prices for a lease period (typically one hour) [1].

An *elasticity policy*[9] governs how and when resources are added to and/or removed from a cloud application. An *autoscaler* is a component of a management framework that is responsible for implementing an application's elasticity policy. In terms of autoscaling an application on the cloud, the current *state-of-the-art* involves specifying rules to implement the elasticity policy for an application server tier. These rules may or may not implicitly minimize the application provider's cost. Further, a single VMI configuration is typically considered (per tier).

[1]Each configuration is associated with its own specification and cost (e.g., m1.small on Amazon Elastic Compute Cloud).

This paper introduces a new approach to autoscaling that utilizes a stochastic model predictive control technique to facilitate effective resource acquisition and releases meeting the service level objectives of the application provider while minimizing their cost. This technique accounts for the delay in resource provisioning times (due to the stochastic nature of the IaaS provider's infrastructure), the stochastic nature of workloads and does not waste instances (i.e., retains instances for the full duration of their lease). A set of simulation results are presented demonstrating the efficacy of this approach.

The remainder of the paper is structured as follows. Section2 describes the problem. Section3 maps the problem to an optimal control one. In Section 4 we provide a solution to the formulated control problem using stochastic model predictive control (MPC). Section 5 presents a case study of a simulated cloud consumer where the technique is applied. In Section 6 we discuss the work and in Section 7 we provide our conclusion on the topic.

2. PROBLEM DEFINITION

In general, the task of the autoscaler is to implement the elasticity policy of the application provider. The elasticity policy is guided by the values of a set of monitored and/or computed metrics. Let y_k denote a column vector of these metrics at discrete time instant k:

$$y_k = \begin{bmatrix} y_k^1 \\ \vdots \\ y_k^j \end{bmatrix}$$

The approach to autoscaling being introduced in this paper focuses on two goals: satisfying the application provider's objectives[2] (as defined in terms of this set of metrics) and

[2]These objectives might take different forms, such as (i) maximization (ii) minimization and (iii) regulation of certain

optimizing for the cost incurred by resource usage[3].

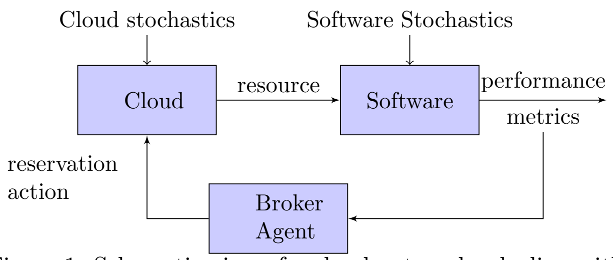

Figure 1: Schematic view of a cloud autoscaler dealing with dynamics of the cloud and application.

A IaaS provider offers several VMI configurations for lease. A configuration represents a specific allocation of CPU, memory, network bandwidth and storage. VMIs are purchased on behalf of the application provider by the autoscaler (i.e. composed of possibly several IaaS providers) in various configurations and over some lease duration (typically one hour). The leasing model can be generally divided into four categories:

1. **Immediate reservation**: where resources are processed right away or rejected.
2. **In-advance reservation**: where resources must be available at specified time. Often an up-front fixed price charge is required to initiate a reservation and a discounted rate is charged for the instances throughout the duration of the reservation.
3. **Best effort reservation**: where request are queued and serviced accordingly.
4. **Auction based reservation**: where customers bid for some number of a particular configuration and as soon as dynamically adjusted resource price lowers the bid amount the resources are allocated.

In all cases the IaaS provider offers a set of possible VMI configurations each associated with a set of `time interval` and `price` tuples. Launching a VMI for this configuration is billed based on the reservation mechanism at the specified price rate. Note that, in general, the outcome of a reservation action in the cloud is not deterministic, and it is not the case that a reservation action always succeeds. For example: immediate and in-advance reservations fail, prices in auction-based reservations are nondeterministic, and in best effort reservations resource may not be provisioned in a timely manner.

To formalize the notion of a reservation action we do the following. Assume that κ_k is the reservation action at time k, and the amount of resource to be used from the cloud is denoted by ϑ_k. The cost to the application provider for using resources depends on the reservation actions performed during application lifetime $(\kappa_0, ..., \kappa_N)$, where application lifetime is denoted by N. An autoscaler seeks to choose a sequence of optimal reservation actions $(\kappa_0^*, ..., \kappa_N^*)$ that minimizes the long term resource cost [4], $\sum_{k=1}^{N} \lambda(\kappa_k)$, where $\lambda(\kappa_k)$ denotes the cost of reservations made at time k.

performance metrics. For example, it is usually desirable to minimize response time of web applications, regulate hardware utilization around certain value, and maximize systems throughput.

[3]The cost incurred for an amount of computation, storage, or network usage in cloud, is not only a function of resources used but also the strategy used to reserve those resources.

3. OPTIMAL CONTROL FORMULATION

In this work, the objectives of the autoscaler are: guiding performance metrics to a desired region and the minimization of application provider's cost. However, upon closer scrutiny, it can be observed that these two set of objectives conflict.

An optimal control approach searches for a solution that finds a good trade-off between a set of goals. In optimal control, the desired behaviour of a system is represented by a single cost function, J, which captures factors of interest over a long horizon of time. More formally, the cost function J that includes the actual cost of resources and the (virtual or actual) cost of deviation from desired performance objectives over the whole period that the application provider leases VMIs from the cloud, can be specified as follows:

$$J((\kappa_0, ..., \kappa_{N-1})) = E\left[\sum_{k=0}^{N-1} \left(\Phi(y_k) + \gamma\lambda(\kappa_k)\right)\right] \quad (1)$$

where N is the lifetime of cloud usage for a specific application provider (i.e., customer), $E[...]$ denotes the expected value, $\Phi(y_k)$ is the virtual (or actual) cost associated with deviation from desired objectives defined in terms of performance metrics, and γ is a coefficient that adjusts the trade-off between performance objectives and cost [5].

Notice that we took the expected value over the cost function due to stochastic effects of software and the cloud. In other words, although the expression representing the stage cost[6] does not seem to have a random component, the underlying system that drives y_k is stochastic.

The idea of optimal control is that by minimizing this cost function over a long horizon, we penalize the *greedy* and *reactive* reservation actions. A greedy reservation action only minimizes the current stage cost, ignoring the effect of current resource reservations κ_k on future performance except for y_{k+1}. This results in a lower cost for the present stage, due to under-reservation; however, this misjudgment drives the system to a sub-optimal future state that is difficult to recover from, incurring very high long-term cost. A reactive reservation action tries to minimize long-term penalty and cost, but it tries to do so only through reservation actions in current or relatively near future states. That is, since it is not able to trade-off the penalty of subsequent steps with the resource cost of current step it will incur sub-optimal resource costs.

In a feedback based scheme, Fig.1, at any given time the controller (i.e., the autoscaler) makes its decisions (i.e., to add/remove resources) based on the information available from the system up to that time and the information deduced and maintained by the controller itself. More precisely, at each time step k this information can be summarized in a collection of variables which we call the *state vector* denoted by x_k[7]. We will discuss the choice of variables to be

[4]while trying to satisfy performance metrics over time (as discussed earlier).

[5]In case Φ represents an actual cost or penalty comparable to resource cost γ is considered 1.

[6]Stage cost refers to the cost for each time increment, k.

[7]$y_k \subseteq x_k$.

included in this state vector later; however, the assumption is that it contains enough information so that the controller can make a decision based on it alone.

Our autoscaler attempts to find a proper reservation action κ_k at any given time k based on the current system state x_k. A stationary reservation *policy* denoted by μ suggests such a reservation action at any given time, k, based on system state as follows:

$$\kappa_k = \mu(x_k)$$

Example In [10] we investigated the efficiency of a autoscaler offered by RightScale[1]. This autoscaler takes the following variables as state:

- ϱ^{ij}: denotes the value of each metric i on each obtained resource $j \in \Lambda$.

- $t'^{ij\mathtt{a}}$: denotes the duration for which a given metric has been breaching a specified threshold (i.e., time spent in the violation zone). It is maintained using the following:

$$t'^{ij\mathtt{a}}_{k+1} = \begin{cases} t'^{ij\mathtt{a}}_k + 1 & \text{if } \varrho^{ij}_k > \mathtt{a} \\ 0 & \text{if } \varrho^{ij}_k < \mathtt{a} \end{cases}$$

- $\psi^{ij\mathtt{ae}}_k$: a vote from the set $\{0, 1, -1\}$ used to indicate interest in acquiring and/or releasing a resource and determined as follows:

$$\psi^{ij\mathtt{ae}}_k = \begin{cases} (-1|1) & \text{if } t'^{ij\mathtt{a}}_k (> | <) \mathtt{e} \\ 0 & \text{otherwise} \end{cases}$$

- ξ^j_k: a resource aggregate vote for each resource based on the aggregation of individual threshold votes:

$$\xi^j_k = \sum_{i\mathtt{ae}} \psi^{ij\mathtt{ae}}_k$$

Resulting in the following state vector (i.e., x_k):

$$x_k = \begin{bmatrix} t'^{ij\mathtt{a}}_k & \forall i, j, \mathtt{a} \\ \psi^{ij\mathtt{ae}}_k & \forall i, j, \mathtt{a}, \mathtt{e} \\ \xi^j_k & \forall j \end{bmatrix}$$

In this case the policy $\mu(x_k)$ defined to derive resource reservation for next step from constructed state was based on the value of the resource votes as follows:

$$\kappa_{k+1} = \begin{cases} \text{acquire} & \text{if } \left(\sum_{j=1..N} \xi^j_k\right)/N > \mathtt{d} \\ \text{release} & \text{if } \left(\sum_{j=1..N} \xi^j_k\right)/N < \mathtt{d} \\ \text{none} & \text{otherwise} \end{cases}$$

Although in [10] we showed how such controller could be used to constrain a certain metric in an operating region, there were definite problems with the approach: (i) the proper number of alerts, and the associated values for $\mathtt{a}$,$\mathtt{e}$,$\mathtt{d}$ could not be deduced automatically, in a way that satisfies performance metrics requirements. (ii) It is not clear how to manage the resource cost by tuning the policy or state definitions. The next section introduces our new approach to autoscaling that harnesses model-predictive control techniques and thereby avoids these negative issues.

4. PROPOSED SOLUTION

In optimal control techniques there exist numerous ways to obtain the optimal policy μ^* such that the cost function is minimized:

$$\mu(x)^* = \text{argmin}_\mu J(\mu(x)) \quad (2)$$

These techniques require that the following are satisfied: (i) the relation between controllable inputs (here κ_k or ϑ_k) and terms of the cost function J (here y_k) should be formulated in certain formats such as linear state-space and (ii) the controller cost function (here including resource cost $\lambda(\kappa_k)$ and performance violation cost $\Phi(y_k)$) should be expressed in certain formats such as quadratic or convex to be able to find a computationally tractable solution.

Model. Regarding item (i), the relation between performance metrics y_k and the used resource ϑ_k over time has been thoroughly investigated in the queuing theory and performance modeling literature [36, 25, 32]. There are several ways to describe the transient effect of resources on well-known performance metrics. For example, it has been shown that transient response time of a simple c-server cluster handling transactional workload where service demands do not depend on the queue lengths, can be modeled as a simple linear difference equation:

$$\begin{aligned} \mu_k &= \sum_{i=1}^{c} \mu^i_k \\ q_{k+1} &= q_k + (w_k - \mu_k).T \\ r_{k+1} &= (1 + q_k)/\mu_k \end{aligned} \quad (3)$$

where μ_k is the aggregate cluster service rate, w_k is arrival rate of the workload, q_k is the number requests in queues of servers, and r_k is the response time at time k.

Since the service rate of each instance depends on the hardware specification of the instance, one cn expect the aggregate service rate to be described in terms of the instance quantity vector ϑ_k.

$$\begin{aligned} \mu_k &= \sum_{i=1}^{c} \mu^i_k \\ \mu_k &= \rho^T \vartheta_k \end{aligned} \quad (4)$$

where ϑ_k represents a vector of the currently obtained quantity of each resource type from the set of the cloud provider's available resource types G: $[\vartheta^i_k | i \in G]$[8] and ρ is the vector of service rates [9] for different types of resource. This assumption will be assessed in the case study.

Item (i) also implies that one has to model the dynamics of the reservation rules as explained in subsection 2. In this paper we targeted modelling the delay in delivery of running instances[10], and the finite property of lease durations. For example, lets assume that an instance has an associated delay of D minutes and a single lease duration of T minutes. We then model the lifecycle of resource delivery using

[8] G can be understood to represent the set of possible VMI configurations offered by a IaaS provider.

[9] This can be derived from specifications such as virtual compute units specified by the cloud provider.

[10] Here, delay refers to a sum of delays specifically: the delay involved in resource delivery, the delay associated with boot-up, the delay associated with running the initial installation scripts, and the delay associated with the initial warm-up.

a graph of nodes, each node representing the number of each type of resource in each minute of *delivery* and *usage* [11]. As time passes, the resources pass through graph nodes, and while they are in usage nodes they affect the performance. A flow model of this form can be represented as an equation of the form:

$$\begin{aligned} X_{k+1} &= AX_k + B\kappa_k \\ \vartheta_{k+1} &= CX_k \end{aligned} \tag{5}$$

where X is $m \times n$ matrix to represent the nodes ($m = T + D$ and $n = |G|$). In this paper our flow model was a simple chain of nodes; thus $A_{m,m}$ has the form:

$$A_{m,m} = \begin{pmatrix} 0 & \cdots & 0 \\ 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$$

$B_{m,n}$ is:

$$B_{m,n} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

and $C_{1,n}$ is:

$$C_{1,n} = \begin{pmatrix} \mathbf{0}_{1,l} & \mathbf{1}_{1,p} \end{pmatrix}$$

In the rest of this paper, we use a combined form of equations 3 and 5 in a state-space format as a model:

$$\begin{aligned} x_{k+1} &= f(x_k, \kappa_k, w_k) \\ y_k &= g(x_k) \end{aligned} \tag{6}$$

where workload w_k is a random term, κ_k is control variable, and x_k includes μ_k, q_k, ϑ_k, and X_k:

$$x_k = \begin{bmatrix} \mu_k \\ q_k \\ \vartheta_k \\ X_k \end{bmatrix}$$

Also our experiments only focus on response time, so we take the vector of metrics of interest as:

$$y_k = [r_k]$$

Cost function. Regarding item (ii), it is usually possible to find a convex stage penalty function $\Phi(y_k)$ for a performance violation, based on the type of the performance objective (i.e. minimization, maximization, or regulation). For example for minimization one can use quadratic form $\Phi(y_k) = y_k^T Q y_k$, while using $\Phi(y_k) = (y_k - \bar{y}_k)^T Q (y_k - \bar{y}_k)$ for regulation.

Control Algorithm. The approach used to minimize the cost function has to (i) take into account the disturbances w_k by considering that an outcome of an action is non-deterministic and computing the expected value (i.e., $E[...]$) over a long horizon of time and (ii) be tractable in terms of computational complexity.

In the stochastic model predictive approach, both of these items are dealt with. One converts the stochastic planning problem into several step-by-step optimizations, and each optimization takes place on arrival at each new state by taking the current observed state as an initial point of planning. Further, in each optimization, the planning is only done for a limited number of steps ahead which is referred to as the *lookahead horizon* and denoted by N'; thus reducing the number of steps involved in optimization from $N - k$ to N' (i.e., this assumes that $N' << N - k$). It is also assumed that the remaining cost from N' to N can be estimated using a simple cost-to-go function $\hat{V}(x_{k+N'})$ which approximates the accumulated cost from the edge of the lookahead horizon (i.e. $k + N'$) to the end of problem N (here application lifetime). This prevents the plan from ending up in an unrecoverable state at the edge of the lookahead horizon. In general, planning over a lookahead horizon inhibits the planner from making reactive or greedy decisions, and that is the essence of predictive control.

The stochastic property of the system is also dealt with during the planning phase. We use the *certainly equivalent control* (CEC) approach to reduce the impact of disturbances. In CEC, at each time step k, process disturbances $\{w_j\}_{j \geq k}$ are fixed at a deterministic value (e.g., their conditional mean) $\overline{w}_j(x_j, u_j) = E[w_k | x_j, u_j]$ which in the simplest case can be considered to be zero. Then, at each step, a perfect information, deterministic optimal control problem with limited horizon is solved once, and the control value to be applied is derived [12]; everything is repeated once a new observation is available. Our adapted version of CEC is presented in algorithm 1.

Algorithm 1: CEC implementation of MPC.

```
input  : system model $f$, violation penalty function $\Phi$,
         resource cost function $\lambda$, trade-off coefficient $\gamma$,
         current system state $x_0$ (i.e. estimated queue
         lengths, already reserved resources) , process
         disturbance mean $\overline{w}_j$, approximate cost-to-go
         function $\hat{V}$
output : optimal reservation action sequence $\{\kappa_k^*\}$
1 initialize: $x_k = x_0$
2 while application uses the cloud do
3     take $k$ as current time
4     given $x_k$, at each time $k$ solve the (planning) problem
          minimize     $\sum_{j=k}^{k+N'} (\Phi(y_j) + \gamma\lambda(\kappa_j)) + \hat{V}(x_{k+N'})$
          subject to   $x_{j+1} = f(x_j, \kappa_j, \overline{w}_j(x_j, \vartheta_j))$
                       $y_j = g(x_j)$
                       $y_j \in \mathcal{Y}$,
                       for $j = k, ..., k + N' - 1$
5     take solution $\tilde{\kappa}_j, ..., \tilde{\kappa}_{j+N'}$ as plan of action for next $N'$
      steps
6     take the first reservation action in plan of action (i.e. $\tilde{\kappa_j}$)
      as $\kappa_k$ and apply it
7     if $\kappa_k$ was successful update $x_{k+1}$ (i.e. $\vartheta_{k+1}$,...) with the
      obtained resources
8     estimate unobserved portion of $x_k$ (queue lengths) and
      update $x_{k+1}$
```

[11] A similar approach is used in inventory control and supply chain management.

[12] In original MPC this value is the first value of the planning sequence.

5. CASE STUDY

To demonstrate the applicability of our approach, we simulated an application provider (i.e., a cloud consumer) that uses purchased VMIs to build a web server cluster to handle transactional workloads. There were two goals: minimization of average response time (i.e., an objective on a metric) and minimization of resource cost.

We simulate, the computing cluster and application users as a network of queues. Previous experience [10] has demonstrated that if the simulator is configured properly its behaviour will adequately match a web server farm hosted on Amazon EC2. The input to the simulator is a set of reserved VMIs of different configurations, the number of users using the system, and the service time per request. The output of the simulator is the average response times and throughput for users, and utilization of each server. Different VMI configurations are simulated based on their specified *virtual processing units*. Presently, Amazon's EC2 offers eleven alternative configurations for lease. Prices were taken according to the hourly rates advertised for on-demand instances. The workload used was a 21 hour excerpt of the FIFA'98 workload [5], day 42 (see 2).

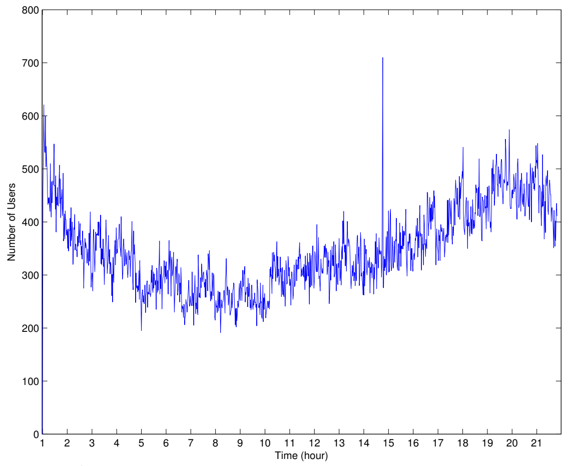

Figure 2: A 21 hours sample workload represented as number of users over time.

Application model. We used equation 3 as the model of the environment (which maps resources to metrics).

Here, ϑ_k denotes a vector of quantities of each VMI configuration class and has 11 elements since we are considering the 11 VMI configuration of Amazon EC2. The service rate μ was assumed to be dependent solely on the VMI configuration (i.e. independent of the workload) and was discovered by performing a least-squares method on data obtained from offline simulations.

Estimation. The queue length is estimated at simulation time on-the-fly using a Kalman estimator[13][14]. The estimation is iterative; the estimate is updated once a new response time measurement is available from the simulator. To demonstrate the performance of the estimation, a sample in which the estimator was applied to track the unknown state variable, queue length, is presented in Figure 3. In this figure, the response time curve generated by acquiring an arbitrary number of `small` instances over a two-hour simulation is presented (red line). For the first hour, response time y_k^1 was observed and the queue length was estimated using a Kalman filter. For the second hour, the tracked state was used as an initial state to simulate the model with the workload w_k and presented as a green line. The difference between the model and simulator's output is due to the simplicity of the model as compared to that of the simulator; however, it will be shown that the model is adequate for effective control.

[13]Kalman estimator is an optimal state estimator for linear models.

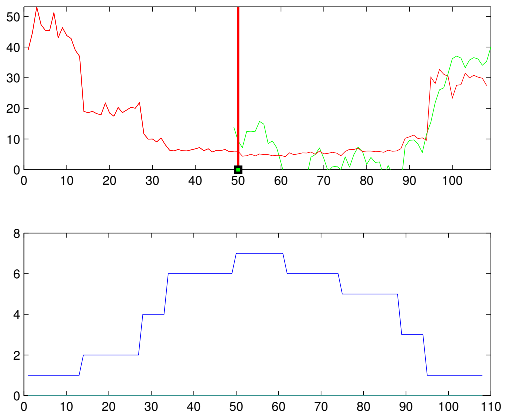

Figure 3: Response time curve simulated using a simulator compared with modeled.

Cost function. Here, we assumed that the performance penalty function has a quadratic form for minimizing response time. The stage resource cost function λ was built based on the hourly lease rates of EC2 instances multiplied by the reservation vector κ_k. The lower and upper limit on the number of purchased instances specified by EC2, was imposed as an input constraint and the fact that queue length cannot be negative was imposed as a constraint on state. Trade-off coefficient γ was lowered in favour of a response time guarantee.

The heuristic. The tool we use for solving the optimization problem was cvx [12] which is a convex optimization solver. In order to use cvx one has to formulate the problem into convex format. During formulation of the problem, we encountered two issues.

The first issue was the fact that number of purchased instances are integer in nature rather than real and cannot be asserted in convex form. Usually such problems are formulated in *integer programming* format and solved in worst case with computational complexity as high as that of exhaustive search. Since, we intended to use cvx we used a heuristic to get around the problem. We initially solved the optimization for virtual compute units purchased (as opposed to VMI configurations), and then use a mapping function $\mathbb{R} \rightarrow \mathbb{N}$ to choose the best choice of VMI configuration for the obtained optimal compute unit to purchase. As it turns out the choice of mapping function also depends on the second issue we encountered.

The second issue was that, unlike what we expected from equation 4, response time in the simulated heterogeneous server cluster was highly affected by the server that had the minimum virtual processing unit. By existence of this small server, all the larger servers act as small ones. This effect is the result of a round robin load distribution between instances. The small server becomes the bottleneck, preventing the rest of cluster from being utilized. Modelling this effect similar to equation 4, while preserving linear affine form is impossible. Thus in our mapping function we considered the current formation of the cluster. The mapping function penalizes choosing a relatively small instance in a cluster of large instances. It basically pushes the system towards using homogeneous instance types, with minimum price per compute unit.

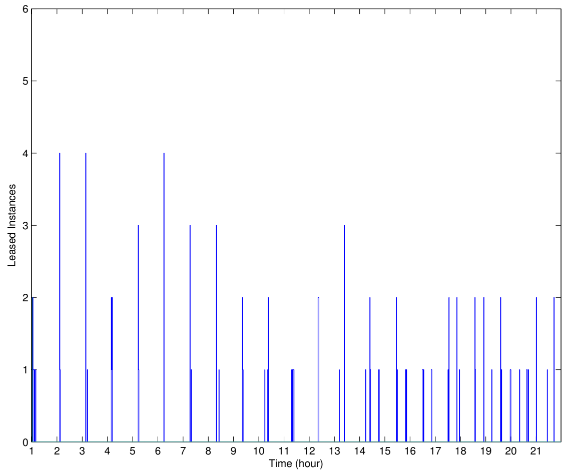

Figure 4: Instances purchased for a 21 hours sample workload of Figure 2.

Figure 4 represents instances purchased for the sample workload (see 2). Initially `m1.small` instances are chosen. All instances purchased after time $k = 10$ (i.e. 10 minutes into the experiment) are high CPU `c1.medium` with five compute unit and $0.17\$/h$ price. The mapping function has selected this VMI configuration because of the low price per compute unit ratio and the divisibility of the optimal compute unit from the controller to the five compute units of the instance. Figure 5 presents the observed response time obtained from the controlled simulator. Note that in this experiment we used $\gamma = 1$ and $N' = 60$.

Figure 6 presents the effect of lookahead horizon (N') on the control cost J for different Cloud dynamics. As the horizon expands from zero to seven, the MPC cost is hugely reduced (order of 10^4). This is not observable in the figure since graph axis is constrained for clarity. For these experiments

the optimal cost, rather than being converged to near infinity, is achieved at a finite prediction horizon of roughly seven minutes. This might be due to the fact that around this optimal horizon, the cost-to-go function $\hat{V}$ gives a better approximation of the future costs than the actual calculated stage costs for subsequent steps. Note that the the random term (i.e. workload) used in calculating subsequent MPC stage costs is an estimate. After the optimal point, the cost increases with an increase in the prediction horizon until it stabilizes around some value. We repeated the experiments for different Cloud reservation dynamics. `Cloud1` (blue dotted line in the figure) has resource delivery delay and lease interval of five minutes ($D = 5$, $T = 5$), `cloud2` and `cloud3` offer 10 and 15 minute leases with the same five minute delay. In general the cost increases as the lease durations

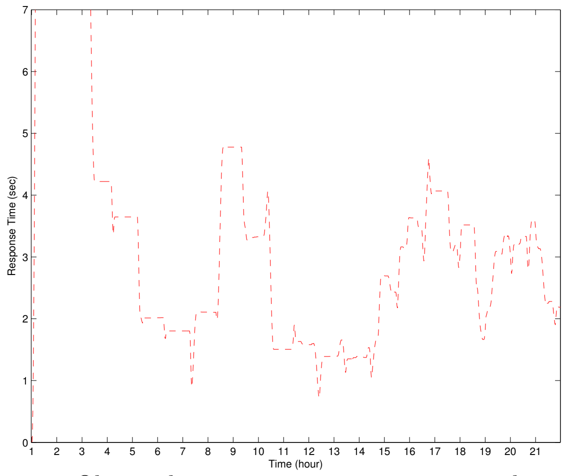

Figure 5: Observed response time in response to the given workload and number of purchased instances presented in Figure 2.

increase.

We also performed an experiment with a hypothetical cloud where resources are delivered and ready to use immediately. In this case increasing the prediction horizon actually had a negative effect on the cost. In such simplified cloud, greedy controller will suffice. However, we suspect if the software model had more complexity, using MPC would be again more efficient.

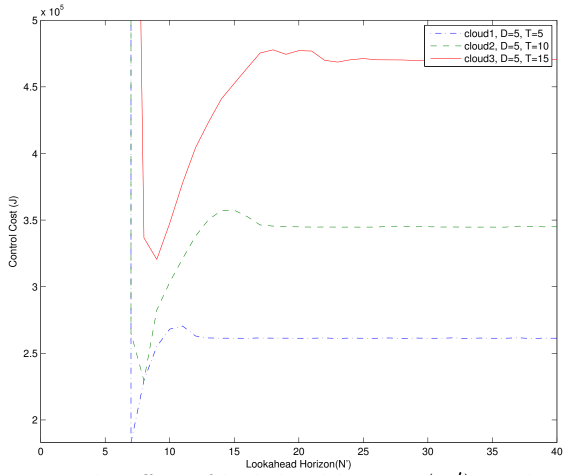

Figure 6: The effect of lookahead horizon (N') on the control cost J for different Cloud reservation dynamics. `Cloud1` (blue dotted line) has resource delivery delay and lease interval of five minutes ($D = 5$, $T = 5$), `cloud2` and `cloud3` offer 10 and 15 minute leases with the same five minute delay.

Figure 7 presents a trade-off curve between resource cost and QoS violation penalty. J_1 represents the resource cost

$$J_1((\kappa_0, ..., \kappa_{N-1})) = \sum_{k=0}^{N-1} \lambda(\kappa_k)$$

and J_2 represents QoS violation penalty based on observed

y_k:

$$J_1((\kappa_0, ..., \kappa_{N-1})) = \sum_{k=0}^{N-1} \Phi(y_k)$$

Each point in on the curve is obtained by performing optimization for a different value of γ.

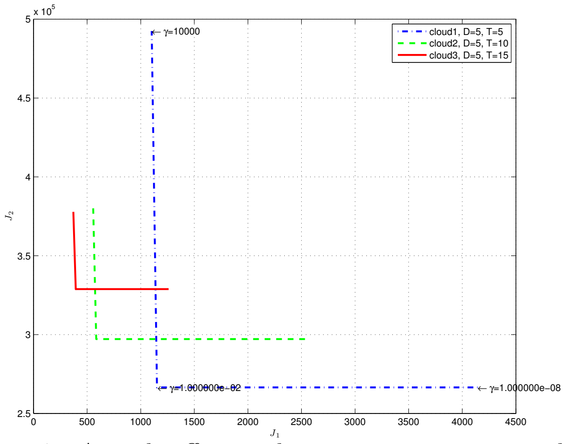

Figure 7: A trade-off curve between resource cost and violation penalty achieved by different values of γ for a specific workload and for different Cloud reservation dynamics. `Cloud1` (blue dotted line) has resource delivery delay and lease interval of five minutes ($D = 5$, $T = 5$), `cloud2` and `cloud3` offer 10 and 15 minute leases with the same five minute delay.

Figure 8 presents the time complexity of a MPC step based on prediction horizon. Each candle represents 10 sample runs, where the bar is 95% confidence interval and the line represent minimum and maximum of samples. This figure approximately matched what we expected; that the time complexity has some polynomial form with respect to the prediction horizon. Notice that the number of variables in convex optimization grows linearly with prediction horizon.

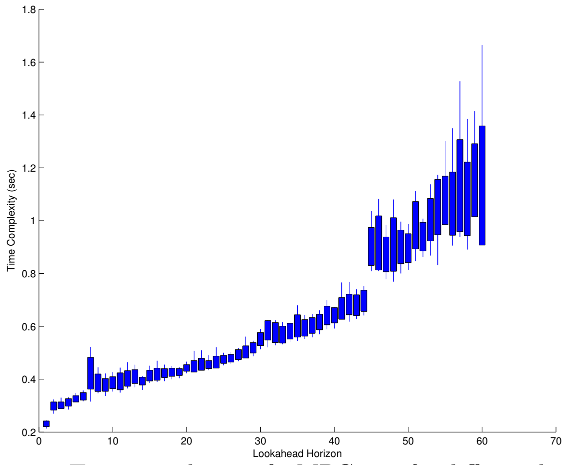

Figure 8: Time complexity of a MPC step for different lookahead horizons (N').

6. RELATED WORK

There is a substantial amount of work on applying optimal control to management of software systems and the associated hardware infrastructure. The most common use case involves keeping servers at high utilizations while avoiding overload. This technique has also been employed in [23, 34] to control processor power management, in [3, 4] to control QoS adaptation in web servers, and in [13, 22, 28] to control load balancing. In virtualized environments the relative share of CPU assigned through a virtual machine manager's (VMM) scheduler has also been used as a control parameter [15]. These controllers can also incorporate *power consumption* metrics (e.g. in [17]) to force hardware to operate in a more energy efficient way (e.g. through changing processor frequency).

The approaches used by these various proposals are referred to as classical linear proportional-integral-derivative (PID) or Linear-Quadratic-Gaussian (LQG) control. In fact, LQG uses an analytical solution of an optimal control problem [14] (like the one we targeted in equation 2) with the limitations that: (i) the objective function has to be quadratic, (ii) expressions describing the system state-space should be in a linear form, (iii) the stochastic term w_k is assumed to be Gaussian, (iv) constraints on control input or system state are ignored and (v) the inputs and state are assumed to belong to $\mathbb{R}^n$ (i.e. as opposed to discrete finite sets). In our case these limitations make application of these technique quite difficult. This experience is not uncommon to alternative fields of engineering [30] as well.

At present, the only examples of using MPC that we are aware of are in infrastructure self-management [6, 2, 16, 7, 29]; however, there they refer to MPC as limited lookahead control (LLC). In [6, 2, 16] MPC is used for managing web server power consumption by changing the frequency at which the CPU is operating. Since, the number of choices of frequencies is quite small (i.e. less than 10 alternatives), changes in frequency are trivial, and the feedback does not have a lot of delay. They were able to use a very small lookahead horizon and a simple model, and calculate the actual expected value of cost over noise (w_j) distribution without getting involved in a state explosion. Unfortunately, this was not possible in our case since in the cloud we have many more choices (i.e., in a single experiment we had 20 instances of 11 VMI configurations) and the effects of those choices is both delayed and long lasting.

In [7] MPC is used to decide about the amount of data each node of a cluster should send through a set of data streams, and the amount to cache to disk. The proposed solution maximizes throughput up to the point that it predicts that the network is being congested. In this paper, the choice of amount of data was continuous and the cost function was quadratic, which greatly helps with solving the MPC problem without any tricks or heuristics.

Paper [29] targets QoS management; but, it focuses on allocating a limited number of sessions to web requests which is quiet different than our problem.

Another set of proposals consider only the steady-state response of the system, thus removing the time aspect from optimization. For example, [20, 19, 11] attempt to minimize cost in virtualized environments subject to performance (response time or throughput) constraints or objectives. Modeling steady-state response simplifies the problem, but it is

[14]This solution called LQ optimal gain is obtained by directly solving the associated Riccatti equation

a well-known fact that the response of a dynamic system to such control converges very slowly. Thus, if forces in the environment have a higher frequency (i.e. change faster than the control takes effect), the controller will be unable to manage. In general, such controllers cannot perform well on short time scales.

Finally, [18] and [35] are example of papers considering multi-objective optimization (MOO) and Pareto-optimal trade-off curves between conflicting goals regarding performance and cost.

7. CONCLUSION AND FUTURE WORK

This paper has presented a novel autoscaling approach that exploits trade-off between satisfying performance related objectives for a consumer's application while minimizing their cost. Autoscaling was formulated as a stochastic model predictive control problem, in which both cloud and application dynamics were modelled. Cost functions were formulated based on objectives of the application provider. The associated problem was solved using the convex optimization solver cvx. This technique accounts for the delay in resource provisioning times (due to the stochastic nature of the IaaS provider's infrastructure), the stochastic nature of workloads and does not waste instances (i.e., retains instances for the full duration of their lease). A set of simulation results are presented demonstrating the efficacy of this approach.

Future work will proceed along several paths. One interesting direction would involve extending the reservation model to include auction-based reservations. The controller in this case would attempt to determine the proper bidding price for resources that is both cheap and reachable by the cloud in the expected time. Another direction would involve utilizing more sophisticated software models. While the model used in this work was very simple it is possible to use far more complicated models that are still in a format suited for estimation and optimization. For example layered and tiered performance models that combine software metrics with hardware have been used in similar context [33, 31, 37, 8, 21]. Moreover, studying the impact of the model on the efficiency of the controller would be an interesting topic. We noticed that the efficiency of the approach presented here is highly dependent on the accuracy of the model. For more complex deployments, such as tiered, distributed and ones with caching, finding a perfect model represents an extremely difficult task. In this situation finding an approach that is resilient to model inaccuracies would be highly useful. A final interesting direction would be to expand the context of control from the case of managing one application to the case of managing a set of applications with different resource constraints and performance objectives. This scenario is similar to a layered implementation of PaaS, where applications are deployed on a set of virtual machines acquired by PaaS provider from IaaS. To our knowledge currently layered implementations of PaaS exist (e.g. [27]) but we have not seen an example that incorporate autoscaling over IaaS.

8. REFERENCES

[1] Rightscale autoscaling using voting tags. `http://support.rightscale.com/12-Guides/Lifecycle_Management/03_-_Understanding_Key_Concepts/RightScale_Alert_System/Alerts_based_on_Voting_Tags/Understanding_the_Voting_Process` [online January 2011].

[2] S. Abdelwahed, N. Kandasamy, and S. Neema. A control-based framework for self-managing distributed computing systems. In *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pages 3–7. ACM, 2004.

[3] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, pages 80–96, 2002.

[4] T. F. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23(3):74–90, June 2003.

[5] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. *Network, IEEE*, 14(3):30 –37, may. 2000.

[6] J. Bai and S. Abdelwahed. Efficient algorithms for performance management of computing systems. 2009.

[7] V. Bhat, M. Parashar, et al. Enabling self-managing applications using model-based online control strategies. In *2006 IEEE International Conference on Autonomic Computing*, pages 15–24. IEEE, 2006.

[8] H. Ghanbari, M. Litoiu, M. Woodside, T. Zheng, J. Wong, and G. Iszlai. Tuning tracking of performance model parameters using dynamic job classes. In *Proceedings of the second ACM/SPEC International Conference on Performance Engineering (ICPE 2011), to appear*. ACM, 2011.

[9] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai. Exploring alternative approaches to implement an elasticity policy. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 716 –723, 2011.

[10] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai. Exploring alternative approaches to implement an elasticity policy. In *Proceedings of the 4th IEEE International Conference on Cloud Computing*, Washington DC, USA, 2011. IEEE.

[11] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai. Feedback-based optimization of a private cloud. *Future Generation Computer Systems*, In Press, Corrected Proof, 2011.

[12] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 1.21. `http://cvxr.com`, March 2012.

[13] J. Hellerstein and I. ebrary. *Feedback control of computing systems*. Wiley Online Library, 2004.

[14] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35,45, 1960.

[15] E. Kalyvianaki, T. Charalambous, and S. Hand. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th international conference on Autonomic computing (ICAC '09)*, pages 117–126, Barcelona, Spain, June 2009. ACM.

[16] N. Kandasamy, S. Abdelwahed, and J. Hayes. Self-optimization in computer systems via on-line control: application to power management. In

Autonomic Computing, 2004. Proceedings. International Conference on, pages 54–61. IEEE.

[17] N. Kandasamy, S. Abdelwahed, and J. P. Hayes. Self-optimization in computer systems via online control: Application to power management. In *First International Conference on Autonomic Computing (ICAC'04)*, pages 54–61, New York, New York, May 2007. IEEE Computer Society.

[18] H. Li, G. Casale, and T. Ellahi. SLA-driven planning and optimization of enterprise applications. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 117–128. ACM, 2010.

[19] J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai. Performance model driven QoS guarantees and optimization in clouds. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 15–22. IEEE Computer Society, 2009.

[20] J. Z. Li, J. Chinneck, M. Woodside, and M. Litoiu. Fast scalable optimization to configure service systems having cost and quality of service constraints. In *Proceedings of the 6th international conference on Autonomic computing*, pages 159–168. ACM, 2009.

[21] T. K. Liu, S. Kumaran, and Z. Luo. Layered queuing models for enterprise javabean applications. In *Proc. 5th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 4–7.

[22] C. Lu, G. Alvarez, and J. Wilkes. Aqueduct: online data migration with performance guarantees. In *Proceedings of the Conference on File and Storage Technologies*, pages 219–230. USENIX Association, 2002.

[23] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 156–163. ACM, 2002.

[24] P. Mell and T. Grance. The NIST definition of cloud computing (draft). NIST special publication 800-146, 2011.

[25] J. E. Neilson, C. M. Woodside, D. C. Petriu, and S. Majumdar. Software bottlenecking in client-server systems and rendezvousnetworks. *IEEE Transactions on Software Engineering*, 21(9):776–782, 1995.

[26] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and et al. Ultra-large-scale systems - the software challenge of the future. *Technical report Software Engineering Institute Carnegie Mellon University ISBN*, 0, 2006.

[27] N. Pang, C. Soman, and R. Wolski. AppScale design and implementation.

[28] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Systems*, 23(1):127–141, 2002.

[29] T. Patikirikorala, L. Wang, A. Colman, and J. Han. Hammerstein–wiener nonlinear model based predictive control for relative qos performance and resource management of software systems. *Control Engineering Practice*, 2011.

[30] S. Qin and T. Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.

[31] S. Ramesh and H. G. Perros. A multi-layer client-server queueing network model with synchronous and asynchronous messages. In *Proceedings of the 1st international workshop on Software and performance*, pages 107–119. ACM New York, NY, USA, 1998.

[32] J. Rolia and V. Vetland. Correlating resource demand information with ARM data for application services. In *Proceedings of the 1st international workshop on Software and performance*, pages 219–230. ACM New York, NY, USA, 1998.

[33] J. A. Rolia and K. C. Sevcik. The method of layers. *IEEE Transactions on Software Engineering*, 21(8):689–700, 1995.

[34] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware QoS management in Web servers. In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pages 63–72. IEEE.

[35] A. Soror, U. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. *ACM TRANSACTIONS ON DATABASE SYSTEMS*, 35(1):Article 7, 2010.

[36] C. M. Woodside, E. Neron, E. D. S. Ho, and B. Mondoux. An" active server" model for the performance of parallel programs written using rendezvous. *Journal of Systems and Software*, 6(1):125–132, 1986.

[37] J. Xu, A. Oufimtsev, M. Woodside, and L. Murphy. Performance modeling and prediction of enterprise JavaBeans with layered queuing network templates. *ACM SIGSOFT Software Engineering Notes*, 31(2), 2006.