Optimal Autoscaling in a 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 modeled in the context of a stochastic, model predictive control problem. The approach exploits trade-off 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.

Categories and Subject Descriptors

I.6 [**Simulation and Modelling**]: Model Development; C.4 [**Computer Systems Organization**]: Performance of Systems

Keywords

Application, Cloud, Autoscaling, Cost, Performance, Model Predictive Control, Optimization

1. INTRODUCTION

A *cloud* represents an ultra large-scale system 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, 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* [5] 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).

This paper introduces a new approach to autoscaling that utilizes a stochastic model predictive control (MPC) 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).

The remainder of the paper is structured as follows. Section 2 describes the problem. Section 3 maps the problem to an optimal control one. In Section 4 we provide a solution to the formulated control problem using stochastic MPC. Section 5 presents a brief description of case study where the technique is applied for a simulated cloud consumer. In Section 6 we discuss the related 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

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

ICAC'12, September 18–20, 2012, San Jose, California, USA.

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

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: (i) **immediate reservation**: where resources are processed right away or rejected. (ii) **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. (iii) **best effort reservation**: where request are queued and serviced accordingly. (iv) **auction based reservation**: where customers bid for some number of a particular configuration and should the dynamically adjusted resource price be less than the bid price, 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. Note that, in general, the outcome of a reservation action in the cloud is non-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 non-deterministic, 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.

3. OPTIMAL CONTROL FORMULATION

In this work, the objectives of the autoscaler are: guiding performance metrics to a desired region and the minimization of the 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:

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

where N is the lifetime of cloud usage for a specific application provider (i.e., customer), $E[...]$ denotes the expected value (in statistical sense), $\Phi(y_k)$ 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, at any given time instant 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. We will discuss the choice of variables to be 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 ϕ suggests such a reservation action at any given time, k, based

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

[4]This is done 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.

on system state as follows:

$$\kappa_k = \phi(x_k)$$

In [5] we investigated the efficiency of an autoscaler where ϕ was composed of a set of user defined rules, and described the issue of finding the optimal policy and state definitions which was not addressed by the technique. The next section introduces our new approach to autoscaling that harnesses model-predictive control technique to address the optimality issue.

4. PROPOSED SOLUTION

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

$$\phi^*(x) = \text{argmin}_\phi J(\phi(x)) \tag{2}$$

To find a computationally tractable solution, these techniques require that the following conditions are satisfied: (i) the relation between controllable inputs (here κ_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.

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. 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 homogenous c-server cluster[7] handling transactional workload where service demands do not depend on the queue lengths, can be modeled as a simple linear difference equation:

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

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 can expect the aggregate service rate to be described in terms of the instance quantity vector ϑ_k:

$$\mu_k = \sum_{i=1}^{c} \mu_k^i = \rho \vartheta_k^T \tag{4}$$

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 [8] and ρ is the vector of service rates[9] for different types of resource.

[7] c is the number of servers in the cluster.

[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.

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 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 $n = |G|$), and $A_{m,m}$, $B_{m,n}$, and $C_{1,m}$ have the form[12]:

$$A_{m,m} = \begin{pmatrix} 0 & \cdots & 0 \\ 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}, B_{m,n} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, C_{1,m} = \begin{pmatrix} \mathbf{0}_{1,D} & \mathbf{1}_{1,T} \end{pmatrix}$$

and κ_k is a $1 \times n$ vector of reservation amounts of resources. 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, where Q is a user defined matrix of coefficients.

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.

[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.

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

[12] This form represents a simple chain of nodes which was our flow model in this paper.

In the stochastic MPC 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[13]; 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 Φ, resource cost function λ, trade-off coefficient γ, 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

$$\begin{aligned} \text{minimize} \quad & \sum_{j=k}^{k+N'} (\Phi(y_j) + \gamma\lambda(\kappa_j)) + \hat{V}(x_{k+N'}) \\ \text{subject to} \quad & x_{j+1} = f(x_j, \kappa_j, \overline{w}_j(x_j, \vartheta_j)) \\ & y_j = g(x_j)\ ,\ y_j \in \mathcal{Y}, \\ & \text{for } j = k, ..., k+N'-1 \end{aligned}$$

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 κ_k and apply it

7 if κ_k was successful update x_{k+1} (i.e. ϑ_{k+1},...) with the obtained resources

8 estimate unobserved portion of x_k (queue lengths) and update x_{k+1}

5. CASE STUDY

We simulated an application provider (i.e., a cloud consumer) that uses purchased VMIs of multiple configurations to build a web server cluster to handle transactional workloads. Both computing cluster and application users were simulated as network of queues. There were two goals: minimization of average response time (i.e., an objective on a metric) and minimization of resource cost.

The input to the simulator at each step is a set of reservation actions (in terms of VMIs of different configurations), the number of users using the system, and the service time per request. The output of the simulator at each step is the average response times and throughput for users, and utilization of each server. To resemble a real system, the queue length of each server is considered a hidden parameter. Different VMI configurations are simulated based on their specified *virtual processing units*. Presently, Amazon 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 [2], day 42.

Application model. We used equation 3 as the model of the environment. 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 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.[14] The estimation is iterative; the estimate is updated once a new response time measurement is available from the simulator.

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 favor of a response time guarantee.

Challenges. During formulation of the problem, we encountered two issues regarding formulation of the problem into convex format: (i) The first issue was the fact that number of purchased instances is integer in nature rather than real and cannot be asserted in convex form, and (ii) 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. Modelling this effect similar to equation 4, while preserving linear affine form is impossible.

To alleviate the above issues, we first solved the optimization problem in terms of virtual compute units reservation (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 sub-optimal compute unit to purchase. The mapping function considers the current formation of the

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

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

cluster and penalizes choosing a relatively small instance in a cluster of large ones. It basically pushes the system towards using homogeneous instance types, with minimum price per compute unit.

Assessment. We ran several tests to asses the effectiveness of the approach in autoscaling on the simulated cloud. The aspects we tested are as follows: (i) The effect of lookahead horizon (N') on the control cost J for different Cloud dynamics. (ii) The trade-off between resource cost and QoS violation penalty by performing optimization for a different value of γ. (iii) Effectiveness of the approach for different Cloud reservation dynamics (i.e. different resource delivery delay and lease interval). (iv) Time complexity of a MPC step based on prediction horizon. Due to shortage of space, we cannot report the result, however, the result was consistent with MPC literature.

6. RELATED WORK

There is a substantial amount of work on applying optimal control to management of software systems and the associated hardware infrastructure. Classical optimal control has been employed in processor power management [9, 12], QoS adaptation in web servers [1], and load balancing [6, 8, 10]. 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[15] (like the one we targeted in equation 2) but when several limitations are applied to the problem.

MPC has already been used in web server power management (by controlling the CPU frequency) [3, 7], QoS management (using number of sessions as a handle) [11], and network throughput maximization (by controlling the disk cached data) [4]. The main differentiators of the problem we targeted with existing problems are: (i) number of possible actions which in our case was very high (i.e. order of resource type times quantity), (ii) the effect of choices is both delayed and long lasting, thus we were not able to use a small lookahead horizon, (iii) we could not calculate the actual expected value of cost over noise (w_j) distribution because of explosion of states resulting from (i) and (ii), and (iv) we had to use a new technique for modelling lease durations and delivery delay that was not used in the above proposals.

7. CONCLUSION AND FUTURE WORK

This paper has presented a novel autoscaling approach that exploits the trade-off between satisfying performance related objectives for a consumer's application while minimizing their cost. Autoscaling was formulated as a stochastic MPC problem, in which both cloud and application dynamics were modeled. Cost functions were formulated based on objectives of the application. The associated problem was solved using a convex optimization solver. This technique accounts for the delay in resource provisioning and the stochastic nature of workloads.

Future work will proceed along several paths: (i) extending the reservation model to include auction-based reservations, (ii) utilizing more sophisticated software models such as layered and tiered performance models, (iii) studying the impact of the model accuracy on the efficiency of the controller and (iv) expanding 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.

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

8. REFERENCES

[1] 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.

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

[3] J. Bai and S. Abdelwahed. Efficient algorithms for performance management of computing systems. In *Fourth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks FeBID*. IEEE, 2009.

[4] 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.

[5] 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.

[6] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. *Feedback control of computing systems*. Wiley Online Library, 2004.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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*, 20(1):49 – 61, 2012.

[12] 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, 2003.