Automatic Control Realistic Through Statistical Machine Learning

Ujjal Aloke Sarkar, Bhupender Kumar Singh

Ph.D Research Scholar, Dept. of Computer Science and Engg. , Dean JPIET, Meerut

Shri Venkateshwara University , Gajraula (UP)

Abstract : The horizontally extensible Internet service on a commercial computer cluster appears to be very suitable for automatic control. There is a target output (service-level agreement), an observation output (actual delay), and a gain controller (adjustment the number of servers). However, there are few data centers that are actually automated in this way in practice, due in part to well-founded skepticism about whether the simple models often used in the research literature can capture complex real-life workload/performance relationships and keep up with changing conditions that might invalidate the models. We agree these shortcomings can be solved by introducing modeling, control, and analysis techniques from statistics and machine learning. In particular, we applied a rich statistical model of application performance, a simulation-based approach to finding the optimal control strategy, and a change point to find abrupt changes in performance. Preliminary results of running a Web 2.0 benchmarking application driven by actual workload tracking in the Amazon EC2 cloud show that our approach can effectively control the number of servers, even in the face of performance anomalies is showing.

Keyword: Web 2.0 benchmarking, Amazon EC2 cloud, statistical machine learning (SML)

1 Introduction

Most Internet applications have strict performance requirements, such as the 95th Percentile service level agreement (SLAs) in response time. The application is designed to be as extensible as possible to meet these requirements. This means you can add more applications, even in the face of greater demand. However, these additional resources can be costly, especially given the growing popularity of public computing such as Amazon's EC2, applications are incented to minimize resource usage because they incur cost for their incremental resource usage. A natural way to minimize used the while meeting performance requirements is to automatically allocate resources based on current needs. However, despite increasing research on automatic control of Internet applications [2, 11, 6, 5, 4, 10], application operators are still skeptical of these methods, and configuration is usually done manually.

In this article, we argue that the data center operators suspicion is based on two key limitations of previous automated configuration attempts. First, the commonly used performance models, such as linear and simple queueing models, are so impractical that they cannot control complex Internet applications without sacrificing SLA's. Second, previous automatic control attempts cannot demonstrate the robustness of the application and its environment over time, such as usage patterns changes, hardware failures, application changes, and sharing resources with other applications in the cloud environment.

We claim that both issues can be solved of modeling, control and analysis techniques based on statistical machine learning (SML).We propose a control framework consisting of three elements. First, based on our framework, there is a wealth of statistical performance models that predict system performance for future configurations and workloads. Second, to find control strategies that minimize resource using while maintaining performance, use a control strategy simulator that simulates performance models to add different strategies for adding and removing resources. Finally, to achieve robustness, we used model management techniques in the SML literature, such as online training and change detection, to adjust the model to changes in application performance. The important thing is that control and model management are model-agnostic: we use common statistics programs that allow us to "plug and play" a different variety of performance models.

Our new contributions are, first, to demonstrate the power of applying established SML techniques for rich models to this problem space, and second, to show how to use these techniques to augment the conventional closed-loop control framework, the actual high level of automatic control more robust and practical to use in real, highly variable, rapidly-changing Internet applications.

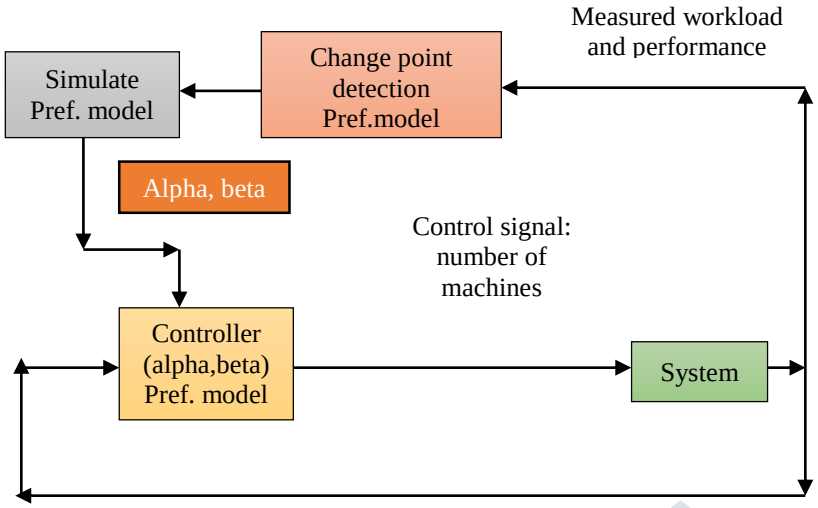

Figure 1: Architectural diagram of the control framework.

2. Modeling and Model Management

The controller needs to respond to several types of changes in the expected performance of data center applications. "Expected" changes, such as diurnal or seasonal workload patterns and their impact on performance, can be accurately measured and captured in historical models. However, there is no way to plan for unplanned continuous spikes ("Slashdot effects"), and even "planning" events such as bug fixes may have unpredictable effects on workload performance relationships, or a combination of workloads. (e.g., new features). In addition, the controller needs to handle performance anomalies and resource bottlenecks.

We propose a resource controller (see Figure 1) that use an accurate performance model of the application to dynamically adjust the resource allocation in response to changes in the workload. We find the optimal control strategy and adapts the performance model to change-point detection technology. The control loop is executed every 20 seconds as follows.

Step 1. First, predict the next 5 minutes of work using a simple linear regression over the last 15 minutes. (More sophisticated historical workload models can be easily incorporated here.)

Step 2. Next, the predicted workload is then used as input to a performance model that estimates the number of servers needed to process the forecasting workload (section 2.1). However, many complex factors can affect application performance, such as workload combination, database size, and application code changes. Instead of capturing all this in a single model, we detect that the current performance model no longer accurately simulates actual performance, and estimate the new model from the production data collected from the survey of the configuration space. We explain this model management process is described in section 2.2.

Step 3. Add or remove servers based on performance model recommendations of the performance model, which we call $_s$ targets. To prevent wild oscillations in the controller, we use hysteresis with gains α and β. More formally, maintain s, this is a continuous version of the server that tracks the $_s$ target by goals

$$s_{new} \leftarrow s_{old} + \begin{cases} \alpha(s_{target} - s_{old}) if s_{target > s_{old}} \\ \beta(s_{target} - s_{old}) \; otherwise \end{cases} \quad (1)$$

Where α and β are hysteresis parameters used to determine the rate at which the controller add to removes servers. Section 2.3 describes how to use the simulator to find optimal values for α and β.

The proposed change-points and simulation methods are model-agnostic, suitable for most existing statistical performance model choices, and can be any model that predicts the mean and variance of performance. This makes the proposed framework very flexible; advances in statistical machine learning can be applied directly to modeling, control or model management without affecting the other components.

2.1 Statistical Performance Models

The performance model estimates the fraction of requests slower than the SLA threshold, given input of the form {workload, # servers}. Each point in the training data represents the amount of work observed at 20 second intervals, the number of servers, and system performance. Use a smooth spline based performance model [3]. This is an established technique for non-linear regression without having to specify the shape of the curve in advance. Use this approach to estimate a curve that maps the number of workloads and servers directly to represent performance (see, for example, Figure 3). In addition to forecasting average performance, forecasting variance is equally important as it represents forecasting of "typical" performance spikes. After fitting the average performance, we estimate the variance by calculating the square of the difference between the predicted average performance of each training point and the model, resulting in a training measurement of the variance. Finally, fit a non-linear regression model (especially LOESS regression [3]) and map the mean performance to the variance. In this method, you can use to important capture fact that high workloads not only increase average performance, but also increase a higher variance.

2.2 Detecting Changes in Application Performance

Our performance model should be discarded or modified when it no longer accurately captures the relationship among

workload, number of servers and performance. In fact, this relationship can change due to software upgrades, temporary hardware failures, or other changes in the environment. Note that this is different from just detecting changes in performance alone: 10% increase in workload they can reduce performance, decrease, but we do not want to flag it as a change point.

The accuracy of a model is usually estimated from residuals, the difference between the measured performance of the application and the predictions of the model. At steady state, the residual must follow a static distribution, so an increase in the mean or the variance in the distribution indicates that the model is no longer accurate and should be updated. The online change point detection technique [1]uses statistical hypothesis testing to compare residual distributions in two consecutive time intervals. If the differences between distributions are statistically significant, start training a new model. The magnitude of the change affects the detection time; sudden changes should be detected within a few minutes, and it may take several days to detect slow and gradual changes.

Because we are deploying models online from production data rather than small test deployments, there is a strong demand for an approach that will quickly collect the training data needed for new models. To address this limitation, use aggressive search strategies until a performance model is established. In explore mode, the controller is very cautious about the number of machines needed to process the current workload, starting with a large number of machines to ensure good application performance, and then to the current workload level Remove the machine slowly to find the required minimum value. As the accuracy of the performance model improves, the controller switches from exploration to optimal control.

2.3 Control Policy Simulator

The individual accurate performance models do not guarantee the good performance of the control strategy in a production environment, as the various parameters in the control loop (such as hysteresis gains α and β) have a major impact on control. Although this is a standard problem of control theory called gain scheduling, it is difficult to find the optimal value of parameters in a complex control domain because the operation delay and the time scale and cost function used by the controller are different. To solve this problem, we use Pegasus [8], a strategy search algorithm that uses simulation to compare different control strategies. Use the Performance Model to estimate the alpha and beta parameters of the various values using the workload estimation performance of the application and return the amount spent on the machine and the number of SLA violations. By using a local search heuristics such as hill-climbing, you can find optimal values for alpha α and β to minimize the total cost of running the application.

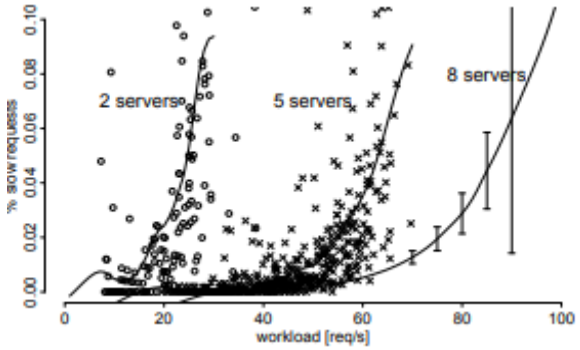

Figure 3: The Smoothing spline performance model estimated from observations; circles and x are observations from 2 and 5 servers, respectively. Each curve represents the mean performance estimated by the model. The error bars in the eight-server line represent our estimate of the standard deviation of performance.

3 Preliminary Results

In this section shows the initial results and introduces an automatic resource allocation method. First, as described in Section 2.1, we first present a resource management experiments using a smooth spline-based performance model. Next, we show that the simulator is effective in selecting the optimal value of the β hysteresis parameter. Finally, we use variable point detection techniques to identify performance changes during performance anomalies.

All experiments used the Cloudstone Web 2.0 Benchmark [9], written in Ruby on Rails and deployed on Amazon EC2. Cloudstone includes a workload generator called Faban [7]. This is used to replay the 3 days of actual workload data obtained from Ebates.com. The tracking playback time has been reduced to 12 hours to improve efficiency.

3.1 Automatic Resource Allocation

First, we show that the SML performance model can be used for automatic resource allocation. Model the business cost of violating a single SLA as a $ 10 penalty per 10 minutes with a 95th percentile delay of> 1 second. We used our offline benchmark data to train our initial performance model, which was used to derive the relationship between workload and optimal number of servers. The controller tries to minimize the total cost of combining hardware costs and penalties for SLA violations. Set $\alpha = 0.9$ to respond quickly respond to workloads, but $\beta = 0.01$ is very conservative when removing machines. Figure 2 shows that with these choices, the execution is successful, there are no SLA violations, and there are few controller operations to change the number of servers. However, as shown in Figure 4, the controller is very sensitive to the values of α and β, so next we will explain how to use the simulator to automatically find the best values for these parameters.

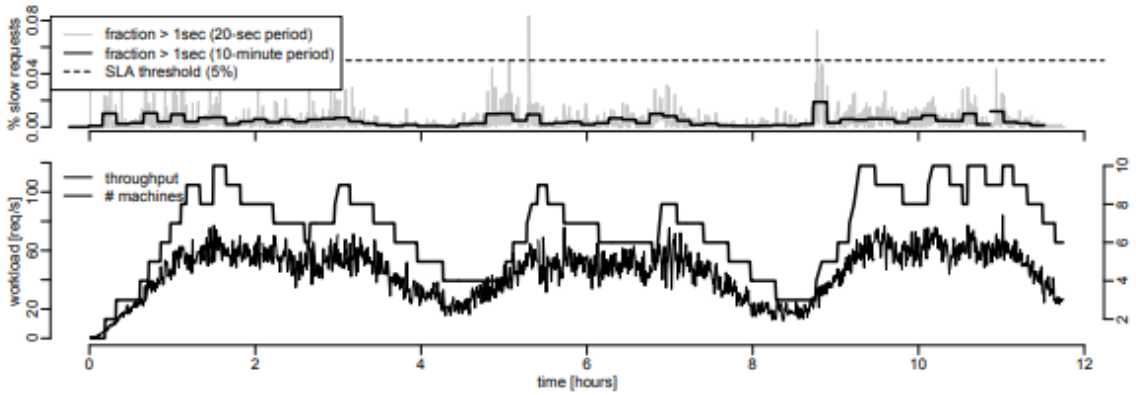

Figure 2: Results of a 12 hours workload reproduced in a 12-hour experiment The following diagram shows the number of servers (thickness, step size curve) requested by the workload and the controller. The figure above shows that the required ratio is slower than 1 second between a 20 second interval (thin line, gray line) and during the ten-minute SLA evaluation interval (thickness, step curve). The SLA does not violate because the percentage of slow requests during the ten-minute interval is always less than 5% the SLA was not violated. During the whole experiment 0.52% of the request were slower than one second. The controller performs 55 actions.

3.2 Control Policy Simulator

In this section describes how to use the Control Strategy Simulator (Section 2.3) to find the optimal value for controller strategy parameters. In this experiment, using a simulator, the optimum value of β, which is the hysteresis gain when the machine was removed, was determined while maintaining α = 0.9. For each value of β, 10 simulations were performed to calculate the average total cost of the control strategy (dashed line in Figure 4). If β = 0.01, the minimum average total cost is $ 78.05. To validate this result, we measured the actual performance of the application on EC2 using the same workload and the same beta value (solid line in Figure 4). The results are almost completely consistent, confirming that the simulator finds the optimal value for β.

3.3 Detecting Changes in Performance

In this section shows you how to use change point detection for performance anomalies observed when running Cloudstone on Amazon EC2. Workload and control parameters are the same as described in Figure 2, but performance anomalies of up to 3 hours were observed, during which the percentage of slow requests increased significantly (the time in the upper graph in Figure 5) The result of the change point test for each t is the p value, the lower the p value, the higher the probability that the average of the normalized performance signal will change significantly. The lower graph in Figure 5 shows the p-values calculated on a logarithmic scale, but the degradation corresponds to the beginning and end of the performance anomaly. Furthermore, the p value remains the same except for performance anomalies the p-values. This result show prime the p-value could indeed a use of practice to direct model management.

4 Related Work

Most of the work in dynamic configuration of Web applications used analytical performance models such as queuing models and did not consider adapting to environmental changes. In contrast, our statistical models have numerical features that allow more natural use of statistical methods to optimize control parameters and model management.

Muse [2] adds, removes, closes, or reassigns servers to maximize energy efficiency using control policies, but limits SLA on the quality of service for each application in a co-managed service plan Receive Muse assumes that each application already has a utility function that represents the monetary value of additional resources, including the monetary value of improved performance. In our work, we learn the performance impact of additional resources.

In [6], the author uses a simple queuing model (single M / G1 / 1 / processor sharing queue) and adaptive admission control for layer 3 Web applications using a proportional integral (PI) controller I designed a strategy. However, one queue cannot simulate the effects of a bottleneck. For example, if additional application servers no longer help, you can statistical model can incorporate naturally model.

[11] Use a more complex system analysis performance models (G / G / 1 queue networks) for resource allocation. [5] provides a virtual machine integrated controller based on a simple performance model and advanced control. This is similar to the simulator. [10] Apply reinforcement learning to train resource allocation controller traces from other controllers and improve their performance. The system learns the direct relationship between observation and action, but since it explicitly models the performance of the application, this approach is more modular and interpretable, simulating virtual future workloads.

[4] Shows an example of using change detection in a thread pool controller design to accommodate concurrency levels and changes in workload. In our control strategy, change point detection is used to indicate when the performance model needs to be modified.

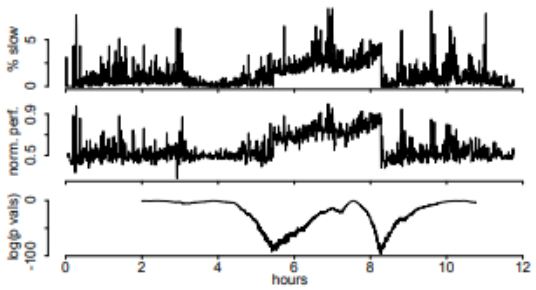

Figure 5: Top graph: Performance of the application during the 12 hour experiment with performance anomalies within 6 to 8 hours. Center map: Performance observed using model predictions. Bottom: p-values of hypothesis by the test report.

5. Conclusion

We have demonstrated that the shortcomings of using closed-loop control to automate data centers can be solved by replacing simple modeling and model management methods with more sophisticated methods that are derived from statistical machine learning. An important goal of our framework and methodology is to enable SML to move further into this area. We encourage the increasing interaction between control theory, machine learning, and systematic research groups.

References

[1] M. Basseville and I. V. Nikiforov. Detectiong of Abrupt Changes. Prentice Hall, 1993.

[2] J. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In Symposium on Operating Systems Principles (SOSP), 2001.

[3] T. Hastie, R. Tibshirani, and J. H. Friedman. The Elements of Statistical Learning. Springer, August 2001.

[4] J. L. Hellerstein, V. Morrison, and E. Eilebrecht. Optimizing concurrency levels in the .net threadpool: A case study of controller design and implementation. In Feedback Control Implementation and Design in Computing Systems and Networks, 2008.

[5] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. In ICAC '08: Proceedings of the 2008 International Conference on Autonomic Computing, pages 3–12, Washington, DC, USA, 2008. IEEE Computer Society.

[6] X. Liu, J. Heo, L. Sha, and X. Zhu. Adaptive control of multi-tiered web applications using queueing predictor. Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP, pages 106– 114, April 2006.

[7] S. Microsystems. Next generation benchmark development/runtime infrastructure. http://faban.sunsource.net/, 2008.

[8] A. Y. Ng and M. I. Jordan. Pegasus: A policy search method for large mdps and pomdps. In UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, pages 406–415, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[9] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, S. Patil, A. Fox, and D. Patterson. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0, 2008.

[10] G. Tesauro, N. Jong, R. Das, and M. Bennani. A hybrid reinforcement learning aproach to autonomic resource allocation. In International Conference on Autonomic Computing (ICAC), 2006.

[11] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic provisioning of multi-tier internet applications. In ICAC, 2005.