Reflective Control for an Elastic Cloud Application: An Automated Experiment Workbench

Azbayar Demberel, Jeff Chase, and Shivnath Babu
Duke University

Abstract

This paper addresses "reflective" control for applications that use server resources from a shared cloud infrastructure opportunistically. In this approach, an external reflective controller launches application functions based on knowledge of what resources are available from the cloud, their cost, and their value to the application through time. As a driving example, we consider reflective control for an important use of elastic computing: a virtual workbench for digital experiments, focusing on automated benchmarking. We report progress on a Workbench Automation/Intelligence Framework (Waif), and show how it can adapt to available cloud resources by planning and launching experiments in parallel.

Waif is part of the ongoing Automat project – an open testbed for programmable hosting centers, built on the ORCA resource leasing platform. We designed a prototype Waif, directed at constructing server performance models by mapping server behavior within a multi-dimensional parameter space. The planner estimates the value and cost of candidate experiments based on the results of completed experiments. In this setting, we show the potential of reflective control to accelerate progress toward a benchmarking objective in a way that balances speed, accuracy, and cost.

1 Introduction

Elastic cloud computing infrastructure is a powerful platform for elastic applications. Applications are "elastic" when their resource needs vary widely through time. Several groups have studied feedback-controlled dynamic resource provisioning/configuration for elastic applications [8, 10, 11, 14, 15, 17]. For example, clouds can offer surge protection for network services to cope with flash crowds, e.g., by modulating the amount of virtual resources acquired from a cloud service to meet a performance target as application load changes.

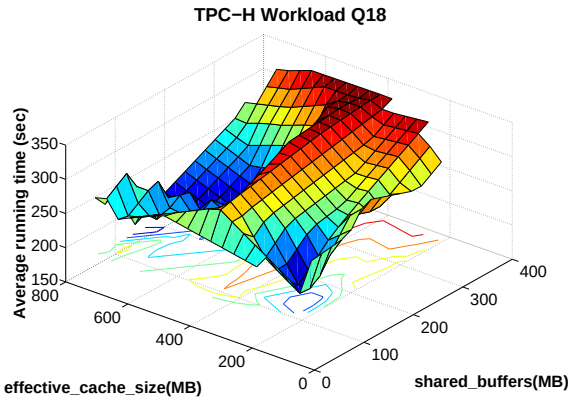

Figure 1: Response surface map (RSM) generated from TPC-H experiments. RSMs are important for cloud automation as they expose interactions of system parameters (effective cache size vs. shared buffer size) and "sweet spots" of system configurations.

In most cases, dynamic provisioning for elastic applications affects the application's performance, but it does not change what the application does. This paper considers another important class of elastic applications with different control needs: *reflective* applications that can change the work that they do based on the resources that are available to them. These applications are well-matched to the capabilities of elastic cloud infrastructure services because they can adapt to use surplus resources opportunistically. They require a new type of reflective control that is more deeply integrated with the application, and has the ability to balance cost/benefit tradeoffs in meeting application goals. For example, we are particularly interested in cloud applications that can plan their resource usage under variable pricing, e.g., congestion pricing for server resources or energy.

In this paper, we address the issues of controlling reflective elastic applications, with a specific fo-

cus on automation of *digital experiments*: experiments that can be launched programmatically from a reflective controller. We discuss a Workbench Automation/Intelligence Framework (Waif) that provides reflective control to digital experiments and enables the cloud to serve as a "virtual workbench" to host experiments of multiple users as well as multiple parallel experiments of a single user. Waif exploits virtual workbench to harness resources and launch experiments automatically. Additionally, it provides plug-in modules for intelligent control that can plan experiments, monitor their progress, and adjust to resource changes and new results, online. Waif takes a step toward allowing experimenters to "close the loop" and automate gathering of experimental data for well-defined objectives.

Waif is part of the ongoing Automat project [18] – an open testbed for programmable autonomic hosting services based on the ORCA resource leasing platform [9, 7, 10]. We designed a prototype Waif directed at automating comprehensive server benchmarking in a multi-dimensional parameter space. The system obtains virtual infrastructure to plan and conduct a schedule of experiments for a sampling of points in the parameter space in order to map some performance metric across the space. Figure 1 depicts an example of the resulting performance model as a response surface map (RSM). While we use automated response surface mapping as an example of a reflective elastic application, the response models are also useful for automating cloud performance management [6, 8, 10, 11, 14, 15, 17]. Waif extends the ideas of our earlier work in workbench automation [13] to a virtual cloud setting, and exposes new tradeoffs associated with reflection and elasticity of the cloud.

2 Overview

The domain of our focus is digital experiments in a virtual cloud workbench. A "digital" experiment is a run of a system, model, data query, or simulation with some set of parameters in order to discover a new result that adds to knowledge of some behavior. One property of digital experiments is that they can be launched automatically whenever suitable resources are available. When resources have a price, it is also desired to launch experiments when the price is right, i.e., the result is worth more than the cost to run the experiment. The choice of experiments to run may depend on the resources available to run them.

Automation of virtual labs is already attracting commercial interest as a driving application of virtual cloud computing [2, 3, 4]. The idea is also applicable to network testbed initiatives such as NSF's GENI [1]. In effect, GENI broadens the virtual cloud idea to a diverse set of virtualizable, programmable substrate components

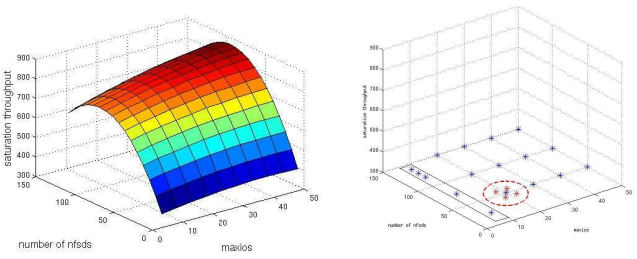

Figure 2: Response surface map (left) generated from Fstress experiments and potential sampling points (right). RSM displays interactions of number of nfsd threads and the maxio configuration parameters. Each point on the surface represents the saturation throughput obtained by running tests for various workload levels for the given configuration.

that includes other networked elements in addition to servers. Each GENI experiment runs within a "slice" of the testbed cloud acquired on demand and configured to order with virtual servers and other elements needed to conduct the experiment.

Our approach involves a controller that launches experiments automatically on resources obtained from the cloud. Since it requests and configures the cloud resources bound to each of its slices, and launches the experiments that run as "guests" within the slice, we call it a *slice controller* or *guest controller*. This controller is not part of the cloud service itself: it runs externally on behalf of the user or experimenter and uses the interfaces exported by the cloud platform to its customers. It uses cloud APIs to determine what resources are available, and matches them against a list of candidate experiments. The controller for response surface mapping plans experiments in part by estimating the value (utility) of each candidate experiment at a particular time based on what it knows about the results of completed experiments. This estimate of the value of work is an essential ingredient for reflective control.

Consider automated sampling of a parameter space to map a response surface. For example, commercial database systems are shipped with hundreds of configuration parameters. As good configuration can enhance performance by orders of magnitude, administrators are motivated to experiment with a wide range of configuration settings (Figure 1) to optimize performance [16]. Similarly, server benchmarking requires repeated experiments for various combinations of resource $\vec{R}$, configuration $\vec{C}$, and workload $\vec{W}$ parameters to expose impacts of system design on performance (Figure 2). The parameter space is too large to explore exhaustively, but guided sampling can yield accurate models: if the planner does its job well, then more resources or more time yield more

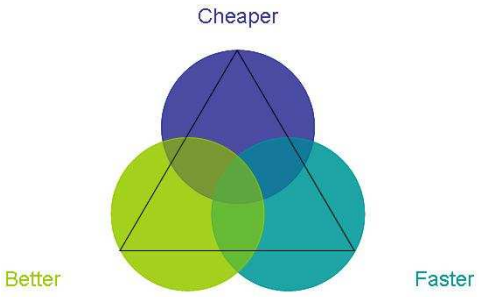

Figure 3: Better? Cheaper? Faster? A reflective controller can adjust the behavior of an elastic cloud application to balance tradeoffs of quality, cost, and time. For example, an experiment planner can determine when it is worthwhile to use cloud resources to run speculative experiments in parallel, based on the expected yield from those experiments and the cost of the resources.

accurate models. Since experiments at different points in the parameter space run independently, the planner may run them in parallel when sufficient resources are available.

Planning parallel experiments on an elastic cloud, however, exposes new tradeoffs of speed vs. cost. The planner refines its value estimates for candidate experiments continuously as earlier experiments complete and their results become available. With more information the planner can make more efficient choices about the next experiments to run. For example, Figure 2 illustrates a response surface and some sampling points to map this surface. We can see that any point in the circle provides enough information about the neighboring area. Sampling too many points from the circle for the sake of parallelism would be wasteful given that other higher-utility experiments could be run instead. In contrast, sampling along the nfsds axis (points in the rectangle) provides useful information about the general shape of the surface. The planner may map the response surface faster with higher levels of parallelism, but the tradeoff of parallelism is that the planner has less information when it selects new experiments; thus it may make choices that yield a less efficient experiment schedule with a higher overall cost. Our research explores techniques to navigate such tradeoffs (Figure 3).

3 Reflective Control in Waif

The Waif controller is designed as a set of interacting modules that instantiate new slices and configure and launch experiments to run within those slices. It has plug-in policy modules to estimate experiment utilities, plan experiments, and evaluate progress toward a high-

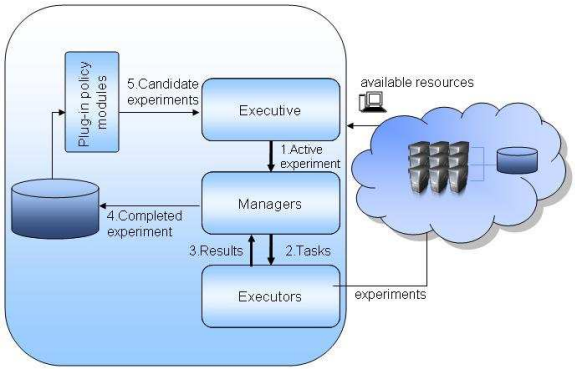

Figure 4: Waif controller structure

level objective. We implemented policy modules geared to RSM benchmarking for servers [13].

The Waif slice controller is designed hierarchically so that decoupled control elements can be stacked and combined. There are three main elements in the controller: control executive, control managers, and executors (Figure 4). The control executive evaluates candidate experiments and matches them to resources available from the cloud. It builds the experiment schedule to balance tradeoffs in accuracy, cost, and time, using the planner module to select candidate experiments based on estimates of cost and benefits.

Once selected, each experiment is assigned to a control manager to launch and monitor its execution. Digital experiments often consist of multiple tasks. For example, in the server RSM benchmarking example, each experiment to map a point on the response surface requires multiple tests for different workload levels to determine the saturation throughput at the given configuration [13]. The control manager devises tasks to complete the experiment and executors run the tasks and process the results.

As each experiment completes, the controller invokes a validator module to evaluate progress toward a high-level objective. For the RSM example, we implemented a hybrid of mean squared error and cross-validation techniques to test convergence, i.e., whether the planner has sufficient results to build an accurate RSM model.

The reflective Waif planner adjusts plans dynamically according to resource availability and completed results. The current planner implements an “Active-sampling” approach [12]. The utility generator algorithm (Figure 5) generates a list of top-k candidates that promise to add the most new information to the results already known from completed experiments. To do this it trains multiple regression models from different subsets of the completed results, and selects new experiments for which the model predictions disagree (high variance). It estimates utility of the candidates from their variance, then uses a simple greedy heuristic to match candidates to suitable

Algorithm *Utility generator*
Input: (i)*smpls*: Samples from completed experiments (ii) *candts:* list of candidate experiments, (iii) *M*: model for learning the system
BEGIN
1. Partition the *smpls* into τ random subsets $S_1 ... S_\tau$
2. For each subset S_i learn model M_i
3. For each candidate experiment $e \in$ *candts*
4. For each model M_i predict response value $p(e)_i$
5. Compute variance ***var(e)*** among $p(e)_i$:

$$var(e) = \frac{\sum_{i=1}^{\tau} (p(e)_i - \bar{p(e)})^2}{\tau - 1}$$

6. Sort *candts* according to decreasing variance
END

Figure 5: Utility-generator algorithm in the current prototype. A recent paper [5] offers an alternative using Gaussian Processes.

Algorithm *Experiment Planner*
BEGIN
1. Collect results from an initial set of *I* experiments.
2. Repeat steps 3 to 7 when new results arrive and if the model has not converged
3. Use *Utility-generator* algorithm to generate a sorted list *l* of candidate experiments, where *l*=top(*candts*)
4. For each experiment *e* $\in$ *l*
5. Get status of the knapsack for each machine in the workbench
6. If *e*'s resource requirement can be satisfied by the workbench
7. Assign *e* to the workbench
END

Figure 6: Greedy experiment planner algorithm

resources as they become available (Figure 6).

4 Results

We implemented Waif for running file server benchmarking experiments with Fstress – an NFS workload generator with many parameters. As an example, we mapped the saturation throughput as a function of CPU and memory allocated to the server and NFS thread number and maxiosize configuration parameters. For each candidate experiment, i.e. each parameter configuration, the controller obtains a slice of the cloud, consisting of NFS server, workload generator clients, and experiment director machines, and runs Fstress request loads against the server configuration. Figure 7 illustrates an RSM generated from the learned model using a parallelism level of four. We can see that because the surface is convex, it required only 11 experiments to map the surface with reasonable accuracy.

To compare the tradeoffs in planning, we also ran experiments with a wider range of configuration and workload parameters for various levels of parallelism. For efficiency, we generated a full-factorial experiment result (i.e. discretized grid of experiments for all possible parameter combinations) in advance and during each selected experiment we obtained the results from the stored table, rather than repeating the experiment. Figure 8 compares the impact of parallelism on modeling performance. The figure illustrates that although higher parallelism requires more resources, it achieves model convergence faster. Figure 9 displays the number of experiments required to map the response surface. It shows that higher parallelism required more

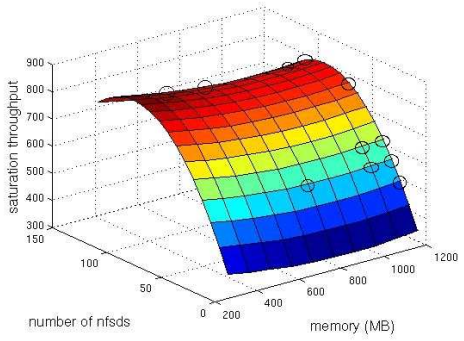

Figure 7: Surface map of NFS file server with 30 percent CPU cap and maxio limit of 20, which was generated from the learned model. The circles illustrate the points where samples were collected.

experiments to achieve the same level of accuracy, i.e. it illustrates the tradeoff of parallelism (speed) and cost. The following table summarizes the results:

Parallelism	Duration	Experiments	Cost
1	71.6 hr	180	71.6 m·hr
5	15.5 hr	200	77.5 m·hr
20	4.8 hr	240	96 m·hr
50	2.4 hr	265	120 m·hr

Our initial results illustrate the tradeoffs in accuracy, cost, and time, and suggest that there is a rich policy space to explore. For example we can see that parallelism of five provides 4.6 times better performance, in terms of time, for a machine·hr cost increase of only six. Additionally, in our experiments we considered a constant cost for the resources. In the future, we are partic-

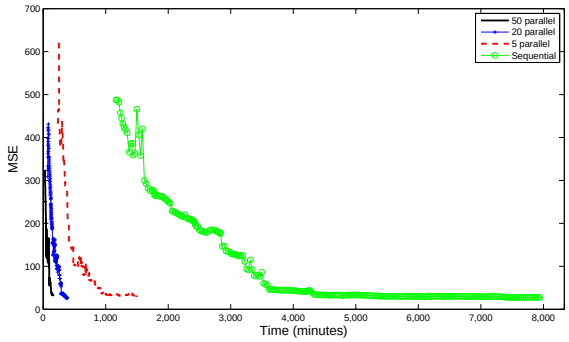

Figure 8: Results of digital experiments for various levels of parallelism. With higher parallelism, the model converges faster.

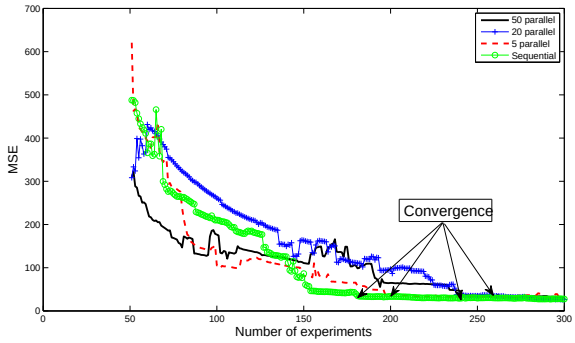

Figure 9: Cost of parallelism. Higher parallelism requires more experiments to achieve the same level of accuracy.

ularly interested in cloud applications that can plan their resource usage under variable pricing, e.g., congestion pricing for server resources or energy.

5 Conclusion

A digital experiment workbench is a compelling application of generalized cloud computing, and a context for exploring reflective elastic control. We illustrate Workbench Automation/Intelligence Framework for intelligent, reflective control of automated digital experiments on a cloud. Our initial results demonstrate the tradeoffs in accuracy, cost, and time, and show that reflective applications provide a rich policy space to explore in the future.

Acknowledgment. This research is supported by the National Science Foundation through CNS-0509408 and CNS-0720829, and by IBM.

References

[1] `http://www.geni.net`.

[2] `http://www.skytap.com`.

[3] `http://www.vmlogix.com/Products/VMLogix-LabManager`.

[4] `http://www.vmware.com/products/labmanager`.

[5] S. Babu, N. Borisov, S. Duan, H. Herodotou, and V. Thummala. Automated Experiment-Driven Management of (Database) Systems. In *Proceedings of 12th Workshop on Hot Topics in Operating Systems*, May 2009.

[6] P. Bodik, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson. Automatic Exploration of Datacenter performance Regimes. In *First Workshop on Automated Control for Datacenters and Clouds*, June 2009.

[7] J. Chase, L. Grit, D. Irwin, V. Marupadi, P. Shivam, and A. Yumerefendi. Beyond Virtual Data Centers: Toward an Open Resource Control Architecture. In *Selected Papers from the International Conference on the Virtual Computing Initiative (ACM Digital Library)*, May 2007.

[8] A. Ganapathi, H. Kuno, U. Dayal, J. Wiener, A. Gox, M. Jordan, and D. Patterson. Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. In *Proceedings of 25th International Conference on Data Engineering*, March 2009.

[9] D. Irwin, J. S. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum. Sharing Networked Resources with Brokered Leases. In *Proceedings of the USENIX Technical Conference*, June 2006.

[10] H. Lim, S. Babu, J. Chase, and S. Parekh. Automated Control in Cloud Computing: Challenges and Opportunities. In *First Workshop on Automated Control for Datacenters and Clouds*, June 2009.

[11] P. Padala, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive Control of Virtualized Resources in Utility Computing Environments. In *Proceedings of 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, March 2007.

[12] P. Shivam. *Proactive Experiment-Driven Learning for System Management*. PhD thesis, Duke University Department of Computer Science, December 2007.

[13] P. Shivam, V. Marupadi, J. Chase, T. Subramaniam, and S. Babu. Cutting Corners: Workbench Automation for Server Benchmarking. In *Proceedings of the USENIX Technical Conference*, June 2008.

[14] A. Soror, U. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic Virtual Machine Configuration for Database Workloads. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, June 2008.

[15] G. Soundararajan, D. Lupei, S. Ghanbari, A. Popescu, J. Chen, and C. Amza. Dynamic Resource Allocation for Database Servers Running on Virtual Storage. In *Proceedings of 6th USENIX Conference on File and Storage Technologies*, February 2009.

[16] R. Thonagi, V. Thummala, and S. Babu. Finding good configurations in high-dimensional spaces: Doing more with less. In *Proceedings of 16th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, September 2008.

[17] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic Provisioning of Multi-tier Applications. In *Proceedings of 2nd IEEE International Conference on Autonomic Computing*, June 2005.

[18] A. Yumerefendi, P. Shivam, D. Irwin, P. Gunda, L. Grit, A. Demberel, J. Chase, and S. Babu. Towards an Autonomic Computing Testbed. In *Workshop on Hot Topics in Autonomic Computing (HotAC)*, June 2007.