Towards an Autonomic Computing Testbed

Aydan Yumerefendi, Piyush Shivam, David Irwin, Pradeep Gunda
Laura Grit, Azbayar Demberel, Jeff Chase, and Shivnath Babu
Duke University
{aydan,shivam,irwin,pradeep,grit,asic,chase,shivnath}@cs.duke.edu

Abstract

This paper introduces Automat, a testbed architecture and prototype for research in autonomic services and hosting centers. Automat is an interactive web-based laboratory in which users allocate resources from an on-demand server cluster to experiment with controller policies for sense-and-respond monitoring and adaptation by hosted services and, crucially, by the hosting center itself. Users may explore the interactions of components selected from a menu of applications, workloads, faultloads, and controllers, or install their own components. Components may include view extensions as plugins to a Web portal interface, enabling users to monitor and interact with controllers during an experiment.

1 Introduction

Attendees at the 2006 ICAC conference agreed that development of common testbeds and test scenarios is a key step to accelerating progress in autonomic computing research. Although no testbed can serve the needs of all researchers, there is an opportunity for new software to address many common needs and to facilitate development and evaluation of new services and self-management algorithms.

This paper introduces Automat[1], a community testbed architecture targeted for research into mechanisms and policies for *autonomic hosting centers* that configure, monitor, optimize, diagnose, and repair themselves under closed-loop control. The Automat project seeks to enable researchers to define services and control policies at various levels of a computing service utility. Testbed users may install and launch their own application services or environments (*guests*) and specify *guest controller* modules that drive provisioning, adaptation, and repair for guests under their control. Users exploring self-management at the hosting center level can instantiate *virtual data centers* comprising subsets of the center's hardware, and define controllers that govern resource arbitration, resource selection, and dynamic migration within their VDCs.

[1]An Automat is an automated vending facility introduced in the 1930s, in which clients select from menus of precooked items and compose them to form a meal. In this case, the "meal" is a reproducible experiment drawn from a menu of packaged test scenarios and dynamic controllers instantiated on demand in a virtual data center.

Automat strives to improve researcher productivity by enabling researchers to focus on the novel aspects of their work, rather than on the infrastructure needed to realize their objectives. Our premise is that a standard "harness" to package and share test applications, workloads, faultloads, and system prototypes—and deploy them at the push of a button—will catalyze the growth of a repository of shared test cases and approaches within the community. A successful testbed can enable repeatable experiments, provide a common means to measure progress toward shared goals, facilitate sharing of research outputs, and establish paths for technology transfer.

2 Overview

Advances in virtualization of hosts and other data center components (network elements and storage) continue to extend the mechanisms for controlled resource management and dynamic adaptation. For example, Automat leverages virtual machine (VM) technology, which offers powerful mechanisms to manage shared clusters. The leading VM systems support live migration, checkpoint/restart, and fine-grained allocation of server resources as a measured and metered quantity (e.g., Xen [2, 5], VMware [10]).

These capabilities create a rich policy space for adaptive hosting centers and guest applications. Automat is an open framework to facilitate experimental exploration of this space, and other aspects of dynamic "orchestration" for autonomic services and hosting centers. More generally, it defines essential elements of an architecture for programmable hosting centers.

Figure 1 depicts the components of an Automat testbed on a hosting center comprising physical resources. A Web portal enables users to install software components, launch and manage experiments, and

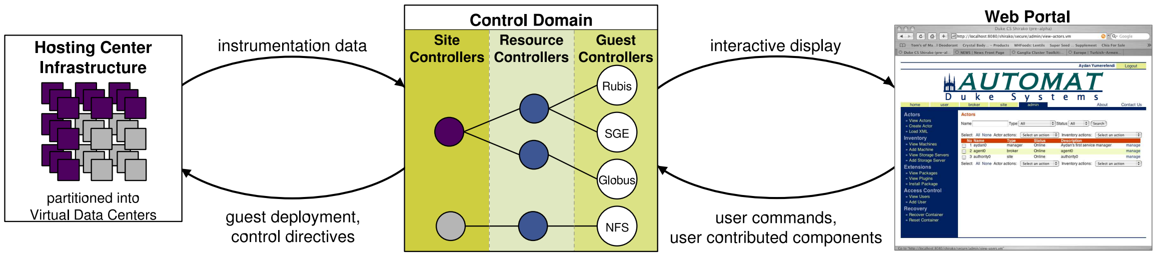


Figure 1: Automat is an open testbed architecture for adaptive services and autonomic hosting centers. Users access the system through a Web portal to launch experiments on a shared cluster. Users can install and instantiate guest applications or adaptive middleware services, workloads, and *controller* modules that govern various sense-and-respond policies for self-management and self-repair, including guest adaptation and resource arbitration. A common instrumentation plane supports publish/subscribe of metric streams to drive controller policies. The infrastructure may be partitioned into multiple *virtual data centers* (VDC) managed by *site controllers* for isolated experiments that require direct control over hardware.

maintain and share libraries of test scenarios and components. This basic structure is similar to other testbeds for systems research, e.g., PlanetLab [3] and Emulab [11]. The novel feature of Automat is foundational support for pluggable user-supplied *controllers* that manage dynamic adaptation, dynamic resource management, and/or diagnosis and repair. Automat exposes and virtualizes the underlying control points to serve these needs, while enabling users of the testbed to control key aspects of policy through common event interfaces and APIs. Table 1 summarizes the user-supplied or replaceable components in Automat.

Controllers incorporate policies and configuration actions to meet desired objectives under changing conditions. Controllers are specified as Java classes that use various support interfaces in the testbed. Users instantiate and link controllers through the Web portal. Each controller instance is driven by a clocking signal and other event notifications, and is endowed with specific powers for some period of time, e.g., access to control points for some hosted guest, power to control allocation within some share of testbed resources, or direct control over a partition of the hosting infrastructure. Controllers may subscribe to instrumentation streams published by the infrastructure elements or the guest applications and operating systems. These data streams act as a feedback signal to drive the control policies.

Some of the possible areas of autonomic computing research enabled by Automat are listed below.

Provisioning and adaptation in self-optimizing systems: Automat provides a platform for experiments with automated resource allocation to meet higher-level objectives (e.g., QoS levels) under dynamic workloads (e.g., [1, 8]). The general approach follows the *MAPE* loop: collect data through *monitoring*, build models of system behavior by *analyzing* this data, use the learned models to *plan* how desired objectives can be met, and *execute* the actions chosen.

Understanding complex systems: Automat provides context to explore the stability and robustness of network services under automated control, complex interactions and emergent behavior of dynamic control policies acting on behalf of various stakeholders in the system, and analysis of instrumentation data to infer system models. Examples include the use of statistical learning to identify metrics correlated with constraint violations and failures (e.g., [6]), and techniques for answering high-level queries automatically (e.g., [9]).

Self-configuring and self-healing systems: Automat supports research into automated fault attribution, which is a key step toward self-repair (e.g., [4]). The testbed can also be used to explore how system components can self-configure and integrate themselves automatically into a running system (e.g., [12]), which is integral to Automat's design.

Placement, migration, and data center management: User-supplied site controllers enable experimentation with policies to assign workload to specific physical hosts, e.g., for energy management or hotspot reduction. In principle, site controllers could also introduce fault injection for low-level resources.

3 Automat Components

This section describes the main elements of Automat's architecture: exposing infrastructure and guest control points, providing the means to define and install controllers that can access exposed actuators, and offering a common collection of tools for testing and monitoring the behavior of controllers.

Automat extends a resource control substrate called Shirako developed in our earlier research [7]. The Shirako core is a Java toolkit for building *resource leasing* services. The prototype includes plugin modules to lease computing resources from shared clusters, and to configure and instantiate OS images and services on physical servers and Xen virtual machines.

A lease is a contract granting the holder rights to ex-

Component	*Description*
Guest	Packaged application software suitable for hosting on the testbed. *Example*: a network application service, workload generator, or faultload.
Guest controller	Drives resource provisioning, configuration, monitoring, fault attribution, and/or self-repair for some guest instance under its control. *Example*: feedback-driven control to provision a Web service to meet service targets under changing workload. A separate guest controller may drive a workload or faultload on a test service.
Resource controller	Arbitrates resource requests from multiple guest controllers contending for a pool of typed abstract resources. *Example:* allocate resources to hosted applications to optimize a global objective such as overall utility.
Site controller	Exercises direct control over physical resources in a *virtual data center* (VDC), delegating arbitration powers to one or more resource controllers. *Example*: assign best-effort virtual machines to specific physical servers, and migrate them reactively to eliminate hotspots.
View	A Web portal plugin that defines a graphical Web interface associated with a controller module, including real-time display of metric streams. Views may provide controls to interact with the controller, observe its status, or change its behavior.

Table 1: Summary of user-supplied or replaceable components. Automat facilitates experiments with control policies at all levels of the hosting center and applications, and their interactions.

clusive control over some quantity of a resource over a specific period of time. Leases are dynamic and renewable by mutual consent between the resource provider and guest. The leasing abstraction applies to any set of computing resources that is "virtualized" in the sense that it is partitionable as a measured quantity.

Each guest runs within a virtual execution context on leased resources. The resources assigned to each guest are determined by the interaction of the user-selected controller modules, which incorporate policies for guest adaptation, resource arbitration, and resource configuration. Table 1 outlines the types of controller modules in the Automat architecture.

The Automat prototype defines a common framework for developing, installing, upgrading, and composing controllers written in Java. Controllers are driven by a clocking signal and may use Automat-provided classes for calendar scheduling, resource allocation, processing of instrumentation streams, and common heuristics for optimization and placement. In addition, each controller registers to receive upcalls as new resources join or leave its domain, and to notify it of changes in resource status.

3.1 Controlling the Infrastructure

The infrastructure exports control points to create and destroy virtual machines (VMs), assign network addresses and physical resource shares to VMs, migrate VMs, and control the functioning of the underlying physical resources, e.g., to image, boot, configure, and power-cycle physical servers, and other elements such as storage and network elements, if applicable.

Site controllers have direct access to the control points for some set of physical elements assigned to Virtual Data Centers (VDC) from the underlying hosting center. To enable more flexible user control over resource management policy, they export limited power over resource provisioning and resource arbitration to resource controllers. Each resource controller is delegated control over a resource inventory and draws on this inventory to satisfy requests from guests and guest controllers. Multiple resource controllers may be active on behalf of different users or groups, concurrently controlling different shares of the resources in the testbed. In this way, control over resources diffuses dynamically and hierarchically through the network of active controllers.

The resource controllers exercise control over provisioning and migration for abstract resources, independent of how they are materialized within each site (VDC). Site controllers decide how to map requests provisioned by resource controllers to the underlying physical resources. Automat researchers may experiment with site controller policies to determine virtual machine placement, such as a virtual machine migration policy to determine where and when to move virtual machines on physical resources [5]. Sites may use a migration policy for node maintenance and failures, and thermal and energy management.

3.2 Controlling Guests

Adaptive application environments and workload generators—guests—must also expose control points to user-supplied controllers. Automat defines a standard means for guests to expose their own control points to guest controllers that manage their configuration, adaptation, and repair. An important goal is to enable separate development of the guest controllers, and even of generic controllers that manage whole classes of guests. Thus Automat also defines a common set of instrumentation and control interfaces for guest components.

Drivers. In addition to the resource leasing APIs, Automat provides basic support for dynamically installable and upgradeable *guest drivers* to enable dynamic control of guests. A guest driver defines a set of control actions

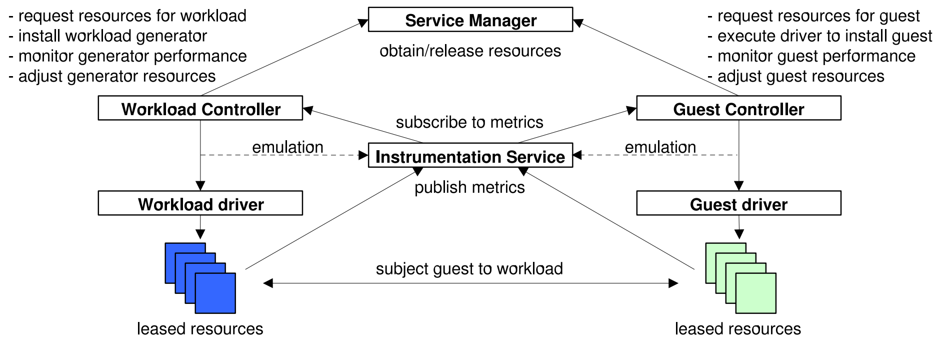

- request resources for workload
- install workload generator
- monitor generator performance
- adjust generator resources
Service Manager
obtain/release resources
- request resources for guest
- execute driver to install guest
- monitor guest performance
- adjust guest resources
Workload Controller
subscribe to metrics
Guest Controller
emulation
Instrumentation Service
emulation
publish metrics
Workload driver
Guest driver
subject guest to workload
leased resources
leased resources

Figure 2: Each guest controller obtains resources for a guest service, invokes the guest driver to install the service, and monitors the service status and performance. A controller can subscribe to instrumentation data and use instrumentation metrics to adjust the resources allocated to the guest. Multiple controllers may share a common naming scope. Controllers in the same scope may interact: for example, a workload controller can subject a guest service to some workload and publish streams of performance metrics that guide another controller to adapt the service.

on a guest instance (e.g., a virtual machine running the guest software). A guest driver exports standard actions to instantiate and configure the guest, probe its status, and also to attach and detach managed resources (servers, storage) to and from the service, and to notify it of VM resizing or migration. The testbed binds controllers to their guest instances securely, so that only a properly authorized guest controller can issue control operations on a guest. Figure 2 depicts guest controllers and drivers for an application service and an associated workload generator.

Workloads and faultloads. Automat simplifies packaging and deployment of workloads and faultloads. The system treats these classes of applications as guest services: each defines a driver and a controller. Users instantiate workloads and faultloads using the web portal. Performance and availability metrics computed by a generator can be exported to customized views accessible through the portal. Workload generators may export control interfaces to modulate the load signal via their guest drivers. Similarly, faultloads can offer the means to inject faults in specific components of an application.

Emulation. Automat provides a means to run controllers in emulation mode on an emulated data center. Guests active in an emulation must provide `null` drivers and a module to generate synthetic instrumentation streams for the controllers to subscribe to and observe, based on expected behavior under the configured workload and the resource set assigned to it by the controller. These modules "short-circuit" the gathering of instrumentation data from a running guest (Figure 2). All other components run unchanged.

3.3 Web Portal

Automat's extensible web portal allows users to interact with the system. Using the web portal a user can install guest packages (hosted applications, workloads, and faultloads) and their drivers and controllers. To launch a guest, a user instantiates a controller for the guest, links it to an active resource controller, and activates it.

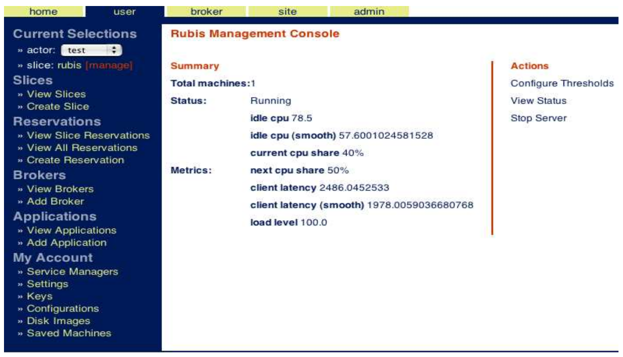
Users can also request powers to define the resource management functions for portions of the hosting center for periods of time. If granted, these functions are embodied in user-selected resource controllers and VDCs with user-selected site controllers.

An Automat center operator also uses the Web portal to monitor the testbed and its users, manage the default site controller and resource controller, and move hardware components in and out of service.

Automat portal users can extend the user interface to interact with their guests and controllers. Each user-supplied controller defines an optional *view* as a Web portal plugin (Figure 3). View plug-ins may include server-side templates or applets. For example, the guest controller plug-ins can implement a graphical interface to set attributes or parameters for the guest, e.g., to modulate a workload generator or switch among alternative provisioning policies.

4 Example

As an example, we briefly outline a Rubis experiment on our initial Automat prototype. Rubis is a popular Web application that implements the core functionality of an auction website. We built a a guest controller for Rubis that leases virtual machines from a VDC, instantiates the Rubis service, subscribes to the service's performance metrics, and uses feedback control to adapt the server resources bound to the service under varying load. The workload is driven by another guest controller that provisions, launches, and controls a workload generator for Rubis. These controllers interact to sequence the experiment.



```
Rubis Management Console

Summary
Total machines:1
Status:      Running
             idle cpu 78.5
             idle cpu (smooth) 57.6001024581528
             current cpu share 40%
Metrics:     next cpu share 50%
             client latency 2486.0452533
             client latency (smooth) 1978.0059036680768
             load level 100.0
```

Figure 3: Rubis guest controller view.

To deploy Rubis on Automat, we created two packages—one for the Rubis service and one for the workload generator—and deployed them as guests on the Automat prototype. The Rubis guest package contains: (1) the Rubis archive, a JBoss application server, and MySQL; (2) scripts and configuration files to install and manage the service; (3) a guest driver for the Rubis service which implements control actions (*e.g.*, install, start, stop) associated with code which typically invoke scripts bundled into the service package. The workload generator is packaged similarly.

We deploy the guest controllers for Rubis and its workload in the same scope so that the workload guest controller can obtain the IP addresses of virtual server running the Rubis web tier. The guest controller deploys the service, then controls the CPU allocation of the application server dynamically based on moving averages of request latency published by the workload generator through a Ganglia instrumentation layer. Figure 3 shows a screenshot of the view for this controller, displaying the status of the server allocated to this guest, metrics subscribed to by the controller, and control actions that can be taken on the guest. Figure 4 shows the resulting client latency with and without the controller.

5 Summary

Automat is an architecture for a common testbed for self-managing hosting centers and autonomic services. Automat users can install dynamic controllers governing various management functions, drawing on sensors and actuators exposed by the testbed. It is a first step toward an exportable community-wide software tool to lower the barriers to entry for research and education in autonomic computing. Automat is a work in progress: we have constructed a proof-of-concept prototype as a controller toolkit and Web portal using the Shirako resource control plane, with core extensions for Virtual Data Centers.

Acknowledgment. This research is supported by the National Science Foundation through ANI-0330658 and CNS-0509408, and by IBM, HP Labs, and Network Appliance. Laura Grit is a National Physical Science Consortium Fellow.

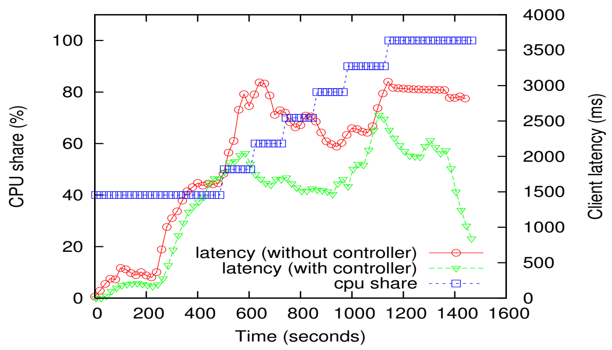


Figure 4: Rubis client latency with and without controller adaptation. For the same workload, a simple controller manages to reduce client latency by changing the amount of CPU cycles bound to the Rubis virtual machine. The controller performs its functions by subscribing to an instrumentation stream and interacting with the control plane.

References

[1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach. *IEEE TPDS*, 13(1), January 2002.

[2] P. Barham, B. Dragovic, K. Faser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *SOSP*, October 2003.

[3] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating System Support for Planetary-Scale Network Services. In *NSDI*, March 2004.

[4] G. Candea, E. Kiciman, S. Kawamoto, and A. Fox. Autonomous Recovery in Componentized Internet Applications. *Cluster Computing*, 9(1), February 2006.

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *NSDI*, May 2005.

[6] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. Chase. Correlating Instrumentation to System States: A Building Block for Automated Diagnosis and Control. In *OSDI*, December 2004.

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

[8] P. Pradala, 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 *EuroSys*, March 2007.

[9] E. Thereska, M. Abd-El-Malek, J. J. Wylie, D. Narayanan, and G. R. Ganger. Informed Data Distribution Selection in a Self-Predicting Storage System. In *ICAC*, June 2006.

[10] C. A. Waldspurger. Memory Resource Management in VMware ESX Server. In *OSDI*, December 2002.

[11] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *OSDI*, December 2002.

[12] J. Wildstrom, P. Stone, E. Witchel, R. J. Mooney, and M. Dahlin. Towards Self-Configuring Hardware for Distributed Computer Systems. In *ICAC*, June 2005.