

JustRunIt: Experiment-Based Management with Xen

Wei Zheng[§] Ricardo Bianchini[§] Yoshio Turner[‡] Jose Renato Santos[‡] G. Janakiraman[†]
[§]Rutgers University [‡]Hewlett Packard Laboratories [†]Skytap

Introduction

Managing data centers is a challenging and often manual endeavor. In particular, many management tasks involve selecting a proper resource allocation or system configuration from many possible alternatives. Evaluating each alternative often requires understanding its performance, availability, and energy consumption implications.

This paper presents *JustRunIt*, an infrastructure for experiment-based management of data centers. JustRunIt uses Xen virtual machine (VM) cloning and workload duplication to run experiments on a few machines in a large data center to predict the impact of configuration changes. We argue that this approach for systems management is often cheaper and more accurate than previous approaches that rely on analytical modeling. The full paper will present case studies using JustRunIt for two common tasks: server consolidation/expansion and evaluating hardware upgrades. The evaluation shows that JustRunIt can produce results realistically and transparently, and nicely complements automated management systems.

Architecture Overview

Our target environment is virtualized data centers that host multiple independent Internet services. Each service comprises multiple tiers: for example, a Web tier, an application tier, and a database tier. All services are hosted in VMs for performance and fault isolation, easy migration, and resource management flexibility.

A *management entity* (i.e., an automated management system or the system administrator) can use JustRunIt to experimentally determine appropriate management actions. To do so, the management entity specifies to JustRunIt the resource types and the range of resource allocations to experiment with, and how long experiments should be run. JustRunIt then creates a sandboxed environment in which experiments will be run on a small number of machines without affecting the on-line system. To run an experiment, JustRunIt clones a small subset of the on-line VMs (e.g., one VM per tier of the service) and migrates them to the sandbox. JustRunIt controls the resources allocated to the VMs in the sandbox and offers the same workload to them that is offered to similar VMs on-line. As an example, Figure 1 depicts a virtualized data center in which one VM instance from each tier of a service is being cloned into the sandbox for testing.

Figure 2 shows an overview of the system infrastructure of JustRunIt. There are four components: experimenter, driver, interpolator, and checker. The *experimenter* implements the VM cloning and workload duplication mechanism to run experiments. Each experiment tests a possible configuration

change to a cloned software server under the current workload. A configuration change may be a different resource allocation (e.g., a larger share of the CPU) or a different hardware setting (e.g., a machine with a higher CPU frequency). The results of each experiment are reported as the server throughput, response time, and energy consumption observed under the tested configuration.

The experiment *driver* chooses which experiments to run to efficiently explore the configuration parameter space. The driver tries to minimize the number of experiments needed while ensuring that the experiments complete within a user-specified time bound. Together, the driver and experimenter produce an experimental results matrix. The coordinates of the matrix are the configuration parameter values for each type of resource, and the values recorded at each point are the performance and energy metrics observed for the corresponding resource assignments.

Blank matrix entries are filled in by the *interpolator*, based on linear interpolation from the experimental results. The filled matrix is provided to the management entity for use in deciding resource allocations for the on-line system.

If the management entity uses any of the interpolated performance or energy values, the *checker* invokes the experimenter to run experiments to validate those values. If it turns out that the difference between the experimental results and the interpolated results exceeds a user-specified threshold value, then the checker notifies the management entity.

VM Cloning and Workload Duplication

Cloning is accomplished by minimally extending Xen’s live migration technology mechanism. In live migration, after state transfer is complete execution transfers to the new VM, and the original VM is destroyed. Our cloning mechanism changes live migration to resume execution on both the new VM and the original VM. Since cloning is transparent to the VM, the clone VM inherits the same network identity (e.g., IP/MAC addresses) as the on-line VM. To avoid address conflicts, cloning sets up network address translation to transparently give the clone VM a unique identity exposed to the network and conceal the clone VM’s internal addresses. The disk storage used by the clone VMs must also be replicated. During the short pause of the on-line system VM at the end of state transfer, cloning creates a copy-on-write snapshot of the block storage volumes used by the on-line VM, and assigns them to the clone VM. We implemented this using the Linux LVM snapshot capability and by exporting volumes to VMs over the network using iSCSI or ATA Over Ethernet. All of this is transparent to the on-line VM that is being cloned.

For testing, the experimenter replays workload to the VMs

in the sandbox using proxies. Two low-overhead proxies, called in-proxy and out-proxy, are inserted into communication paths in the on-line system to replicate traffic to the sandbox. The proxies are application protocol-aware. The in-proxy mimics the behavior of previous tiers before the sandbox, and the out-proxy mimics the behavior of following tiers. The operation of this mechanism will be presented in the full paper, including explanations of the control of message timing and techniques for tolerating commonly encountered non-determinism in the application layer. Our evaluation will also show that the proxies impose little interference on the performance of the on-line system.

Evaluation

We have implemented simple automated management systems for two common tasks in virtualized hosting centers: server consolidation/expansion (i.e., partitioning resources across the services to use as few active servers as possible) and evaluation of hardware upgrades. These tasks are currently performed by most administrators in a manual, labor-intensive, and ad-hoc manner. Due to lack of space, we present here only an overview of the server consolidation case.

The goal in this case is to consolidate the hosted services onto the smallest possible set of nodes, while satisfying all service level agreements (SLAs). To achieve this goal, the system constantly monitors the average response time of each service, comparing this average to the corresponding SLA. Because workload conditions change over time, the resources assigned to a service may become insufficient and the service may start violating its SLA. Whenever such a violation occurs, our system initiates experiments with JustRunIt to determine what is the minimum allocation of resources that would be required for the service's SLA to be satisfied again.

To demonstrate the behavior of our manager, we show the results of experiments with four instances of a 3 tier auction service running on virtual machines on 9 physical servers. Figure 3 shows the response time of each service during our experiment; each point represents the average response time during the corresponding minute. We initially offered 1000 requests/second to each service. This offered load results in an average response time hovering around 40 ms. Two minutes after the start of the experiment, we increase the load offered to service 0 to 1500 requests/second. This caused its response time to increase beyond a desired limit of 50ms during the third minute of the experiment. At that point, the manager started JustRunIt experiments to determine the CPU allocation that would be required for the service's application servers (the second tier is the bottleneck tier) to bring response time back below 50 ms under the new offered load. The set of JustRunIt experiments lasted 3 minutes, allowing CPU allocations of 60%, 80%, and 100% of a core to be tested. Values for 70% and 90% shares were interpolated based on the experimental results. Based on the response-time results of the experiments, the manager determined that the application server VMs of the offending service should be given 60% of a core. Because of the extra CPU allocation requirements, the man-

ager decided that the system should be expanded to include an additional physical machine that was powered off. To populate this machine, the manager migrated 2 virtual machines to it. Besides the 3 minutes spent with experiments, VM cloning and VM migration took about 1 minute altogether. As a result, the manager was able to complete the resource reallocation 7 minutes into the experiment. The experiment ended with all services satisfying their SLAs.

Related Work

Current management systems rely on analytical modeling, feedback control, and/or machine learning to at least partially automate certain management tasks. Modeling has high complexity and accuracy limitations, and feedback control is not applicable to many types of tasks. Although machine learning is useful for certain management tasks, such as fault diagnosis, machine learning can only learn about system scenarios and configurations that have been seen in the past and about which enough data has been collected. Nevertheless, machine learning can be used to improve the interpolation done by JustRunIt, given enough data to derive accurate models.

Previous work [1] considered validating operator actions in an Internet service by using request duplication to a sandboxed extension of the service. Tan *et al.* [2] considered a similar infrastructure for verifying file servers. Instead of operator-action validation in a single, non-virtualized Internet service, our goal is to experimentally evaluate the effect of different resource allocations, parameter settings, and other potential system changes (such as hardware upgrades) in virtualized data centers. Thus, JustRunIt is more broadly applicable than previous works. As a result, our infrastructure is quite different than previous systems. Most significantly, JustRunIt can explore a large number of scenarios that differ from the on-line system, while extrapolating results from the experiments that are actually run, and verifying its extrapolations if necessary.

Conclusions

JustRunIt enables an automated management system or the system administrator to answer "what-if" questions experimentally during management tasks to select the best course of action. The current version of JustRunIt can be applied to many management tasks, including resource management, hardware upgrades, and software upgrades.

We plan to extend JustRunIt to allow cross-tier communication between sandboxed servers. We also plan to create infrastructure to allow experimentation with request mixes other than those observed on-line. The idea here is to collect a trace of the on-line workload offered to one server of each tier, as well as the state of these servers. Later, JustRunIt could install the states and replay the trace to the sandboxed servers. During replay, the request mix could be changed by eliminating or replicating some of the traced sessions. Finally, we plan to build an in-proxy for a database server, starting with code from the C-JBDC middleware.

References

- [1] Kiran Nagaraja, Fábio Oliveira, Ricardo Bianchini, Richard P. Martin, and Thu D. Nguyen. Understanding and Dealing with Operator Mistakes in Internet Services. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation*, December 2004.
- [2] Y.-L. Tan, T. Wong, J. D. Strunk, and G. R. Ganger. Comparison-based File Server Verification. In *Proceedings of the USENIX Annual Technical Conference*, June 2005.

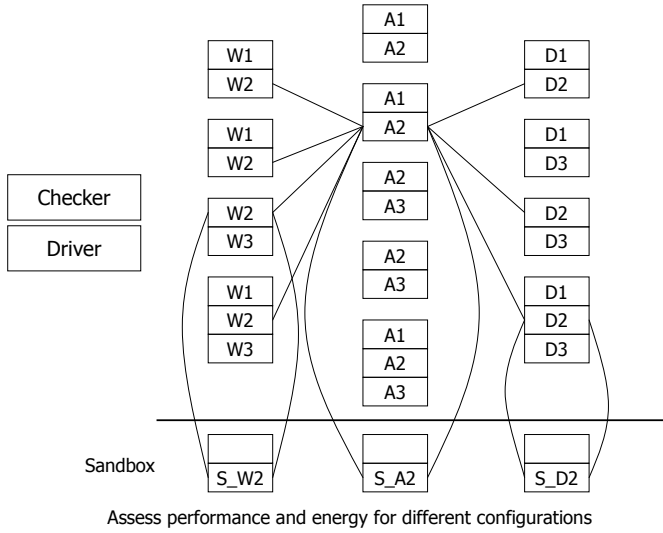


Figure 1: Virtualized data center and JustRunIt sandbox. Each box represents a VM, whereas each group of boxes represents a PM. “W2”, “A2”, and “D2” mean Web, application, and database server of service 2, respectively. “S_A2” means sandboxed application server of service 2.

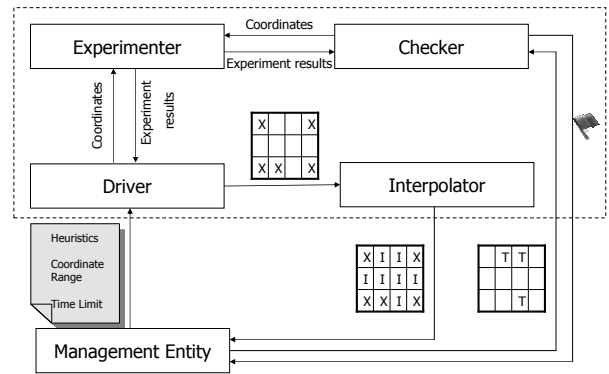


Figure 2: Overview of JustRunIt. “X” represents a result obtained through experimentation, whereas “I” represents an interpolated result. “T” represents an interpolated result that has been used by the management entity.

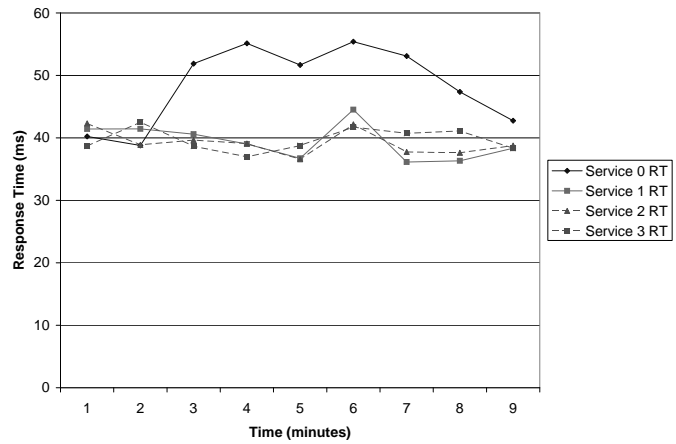


Figure 3: Server expansion using JustRunIt.