Towards Model-adaptive Control in Online Service Environments

Dara Kusic, Student Member, IEEE, Nagarajan Kandasamy, Member, IEEE, and Sherif Abdelwahed, Senior Member, IEEE, and Guofei Jiang, Member, IEEE

Abstract—There is a growing interest in implementing online control frameworks that manage distributed computing systems for power and performance objectives. While such frameworks continuously manage the system to optimize resource allocation and respond to dynamic environment input, they often rely upon static models of application behavior that do not adapt to slow behavior changes that occur during normal operation. By introducing adaptive models that dynamically adjust to the changing performance profile of an application, a robust controller can maintain performance objectives through normal changes that can occur in production and those introduced by software errors. In this paper, we characterize the effects of events that change an application’s performance profile over time. Such studies motivate the need for model-adaptive control to maintain system power and performance objectives over time under dynamic operating conditions.

Key words: Adaptive model learning, dynamic control, robust control, resource provisioning

I. INTRODUCTION

Hosting centers often share hardware and software resources among multiple online applications such as retail commerce and online gaming. Virtualization provides a promising approach maximizing the utility of such resources by consolidating multiple online services onto fewer computing resources within the hosting center. This developing technology allows a single server to be shared among multiple performance-isolated platforms called virtual machines (VMs), where each virtual machine can, in turn, host multiple enterprise applications. Virtualization also enables on-demand or utility computing—a just-in-time resource provisioning model in which computing resources such as CPU, memory, and disk space are made available to applications only as needed and not allocated statically based on the peak workload demand [1]. By dynamically provisioning virtual machines, consolidating the workload, and turning servers on and off as needed, hosting center operators can maintain the desired performance objectives while achieving higher server utilization and energy efficiency. These dynamic resource provisioning strategies complement the more traditional off-line capacity planning process [2].

A growing number of automated control strategies have been proposed that use virtualization [3]–[5] to dynamically assign resources on-demand, allocating only those resources needed to support a service at a given time. To do so, these strategies rely on models of application behavior to estimate an amount of resources to sustain an acceptable performance level. For model-based control strategies to be of practical value, however, they must adapt to dynamic changes in the application environment that can slowly degrade performance and compromise model accuracy over time. Some common examples of such changes include small memory leaks in the software that accumulate over time to consume memory resources, and growing database file sizes that increase application response times for dynamic content retrieval and transaction commitments.

Behavioral changes that continue to accrue will compromise the system’s ability to meet quality-of-service (QoS) goals and may eventually render application components inoperable. By introducing model-adaptive control into the system management, however, data center operators can prolong acceptable QoS performance while responding to the root causes of performance degradation.

In this paper, we discuss some of the common causes of application performance degradation to motivate the need for adaptive-model control. Experimental results show that an application’s performance profile can change most rapidly on the order of a few minutes or more slowly over the course of a day, dependent on the application and the magnitude of the environmental changes. A model-adaptive approach to online control will continuously monitor the actual system state against an estimated system state, and use this data to re-train the application model as necessary. A recurrent neural network is one example of a modeling approach that utilizes real-time data to re-train the model as updated information becomes available.

The paper is organized as follows. Section II discusses related work to compel the need for adaptive-model control. Section III describes the application environment and testing setup for our experiments. Section III presents results showing the effects of some common causes of performance degradation, and Section V concludes the paper.

II. RELATED WORK

Recent research on resource provisioning in virtualized computing environments in [6]–[11] offers a variety of solutions to manage such systems for power and performance objectives. Such strategies, however, often rely upon static models of application behavior to estimate the amount of computing resources (CPU, memory, I/O) that are required to service a given workload for a targeted power or performance goal. In this paper, we motivate the need to construct adaptive
models to incorporate into existing resource provisioning strategies for robustness to changing application profiles.

The authors of [4] propose an online method to select a VM configuration that is triggered by threshold-based events such as CPU utilization to revise the placement of VMs. Similarly, in [12], the authors propose an optimization scheme to allocate CPU shares to VMs processing for enterprise applications using fuzzy-logic models to estimate CPU requirements for the current workload intensity. Monitoring the CPU utilization for processes that are CPU-intensive works well under normal operating conditions, but events such a memory leaks can cause performance to degrade even as CPU utilization remains constant. Resource provisioning techniques for batch processing in a virtualized environment, such as in [5], that rely upon estimates of job completion time may similarly suffer from the effects of growing database file size or memory leaks.

III. THE TEST ENVIRONMENT

Recently published work by the authors [3] has shown the effectiveness of an online control framework to manage a small virtualized hosting environment for power and performance objectives. The resource-provisioning problem is posed as one of sequential optimization under uncertainty and solved using limited lookahead control (LLC), a predictive control approach previously introduced in [13]. This approach allows for multi-objective optimization under explicit constraints and is applicable to computing systems with non-linear dynamics where control inputs must be chosen from a finite set.

The testbed hosts multiple web-based services, comprising front-end application servers and back-end database servers. The applications perform dynamic content retrieval (customer browsing) as well as transaction commitments (customer purchases) requiring database reads and writes, respectively. The hosting hardware consists of a cluster of heterogenous Dell PowerEdge servers running VMware ESX Server 3.5 virtualization software to logically partition the computing resources (CPU, memory, disk, etc.). Incoming requests from workloads such as those shown in Fig. 2 for each service are dispatched to a dedicated cluster of virtual machines (VMs) distributed upon the host machines.

The virtual clusters generate revenue as per the non-linear pricing scheme or service-level agreement (SLA) shown in Fig. 1 that relates the average response time achieved per transaction to a dollar value that clients are willing to pay. Response times below a threshold value result in a reward paid to the service provider, while response times violating the SLA result in the provider paying a penalty to the client.

The control objective is to maximize the profit generated by this system by minimizing both the power consumption and SLA violations. To achieve this objective, the online controller decides the number of physical and virtual machines to allocate to each service where the VMs and their hosts are turned on or off according to workload demand, and the CPU share and fraction of incoming workload to allocate to each VM.

Experimental results show that the hosting environment, when managed using the implemented LLC approach saves, on average, 26% more power over a twenty-four hour period when compared to a system operating without dynamic control. An uncontrolled system is defined as one in which all available host machines remain powered on. Power consumption during run-time was estimated from models of the measured power consumption of the host machines in various operating states (e.g. "idle", "in boot", "n virtual machines serving workloads"). These power savings are achieved with very few SLA violations, 1.6% of the total number of service requests. An uncontrolled system incurs some SLA violations due to normal variability in application performance—about less than 1% to 5% of the total requests made to the system. Results of the energy savings and SLA violations of two applications for five different workloads similar to those shown in Fig. 2 are summarized in Table I.

A key aspect of implementing the LLC approach is to formulate a model that will estimate the future state of the system in order to evaluate control decisions. Fig. 3 illustrates the basic LLC concept where the environment input $\lambda$ is estimated over a prediction horizon $h$ and used by the system model to forecast future system states $\hat{x}$. At each time step $k$, the controller finds a feasible sequence $\{u^{\ast}(l) | l \in [k + 1, k + h]\}$ of control actions within the prediction horizon that maximize the profit generated by the cluster. Then, only the first control action in the chosen sequence, $u(k + 1)$, is applied to the system and the rest are discarded. The entire process is repeated at time $k + 1$ when the controller can adjust the trajectory, given new state information and an updated workload forecast.

In order for the LLC approach or any model-based control strategy to be of practical value over time, it must adapt to
changes in the operating conditions that can compromise the accuracy of predictions about application behavior. Without robustness to application dynamics that increase resource utilization and response times, the number of SLA violations shown in Table I will increase even as workload intensity remains constant. The performance effects of some common conditions that effect change in an application’s performance profile are presented in the next section.

RESULTS

We use three multi-tier, transaction-based applications for testing in our virtualized hosting environment. The first, IBM’s Trade6 benchmark—a multi-threaded stock-trading application that allows users to browse, buy, and sell stocks—is used in the work presented in [3]. As shown in Fig. 4, Trade6 is a transaction-based application integrated within the IBM WebSphere Application Server V6.0 and uses DB2 Enterprise Server Edition V9.2 as the database component. This execution environment is then distributed across multiple servers comprising the application and database tiers.

The second application hosted within our virtualized cluster is Dell’s DVD Store [14], an open source simulation of an online e-commerce site that we host on an Apache Tomcat 6.0.14 application server and have adapted the database component to IBM’s DB2 Enterprise Server Edition V9.2. The experimental results for all three applications reflect application workloads requiring a 50/50 mix of database reads and writes.

We first sought to characterize the effects of database file size growth on an application performance. The size of an application’s database may increase due to user activity over the lifetime of the online service as new transactions accumulate. The database size affects the retrieval time for dynamic content, as well as the time to commit new transactions. Fig. 5 clearly shows the effects of database size on the average response time per request for the Trade6 application. The default database for the Trade6 application consists of 500 users and 1,000 stocks and is scaled upward from that size to exceed the actual number (about 6,000) of combined stock ticker symbols in the NASDAQ and NYSE combined. As the size of the database grows, the maximum workload arrival rate that can be tolerated within the bounds of the SLA shown in Fig. 1 decreases. An adaptive online controller would scale back the workload directed to each node to accommodate the increased response time per transaction.

While growing database sizes can be considered an effect of normal operation, memory leaks caused by software bugs are a common source of errant operation in an application, particularly when using the C/C++ programming languages [16]. A memory leak is defined as memory that has been...
dynamically allocated by an application, such as by using the `malloc()` operation in the C language, but that is never released back to the operating system. Memory leaks that accumulate will eventually allocate the resource to capacity and the operating system will reserve the swap space (virtual memory). It is typically at this point when the application has reserved the system memory to capacity that performance significantly degrades.

Fig. 6 and Fig. 7 show the response times measured per-transaction for the Trade6 and RUBBoS applications, respectively. The solid black lines through Fig. 6 and Fig. 7 indicate the response time averaged over the last 20 time samples. In both applications, the memory leak was effected at the application server and database tiers and the workload arrival rate was held constant. The point at which application performance degrades significantly is commensurate with the rate of the memory leak, occurring at about $t = 150$, or two and a half hours into a 5 MB per minute memory leak, and at half that time, occurring at about $t = 70$, or an hour and fifteen minutes into a 10 MB per minute memory leak. The 1 MB per minute memory leak shown in Fig. 7(a) exhibits a more gradual increase before causing the application server to crash, at about $t = 1100$.

In both applications, the number of per-transaction SLA violations, those requests that exceed the acceptable response time thresholds in Fig. 1, jumps from less than one hundredth of a percent to $2 - 5\%$ of the total requests over a given time period. Because SLA goals are set somewhat conservatively, the percentage of per-transaction SLA violations may not be large. However, the increases in average response times are a more dramatic measure of performance degradation. The Trade6 application exhibits somewhat modest increases in average response times (about 60% to 75% above normal) as the memory is allocated to capacity, however, the RUBBoS application exhibits more severe increases in average response times (about 150% to 500% above normal) under the same conditions.

IV. DEVELOPING A MODEL-ADAPTIVE CONTROL FRAMEWORK

To make good provisioning decisions, an online controller must maintain an accurate model of application behavior under various workload intensities when hosted by a VM given a particular CPU and memory allocation. In previous work, we have captured this behavior using supervised learning, i.e., by extensively simulating the VM (and the underlying application) in offline fashion and using the observations to train an approximation structure such as a neural network for use at run time [17]. However, it is generally not possible, via offline simulations, to create an exact model of system dynamics using a limited set of training data. Therefore, we propose to integrate online parameter tuning and model-learning techniques within the LLC control framework to: (1) improve the accuracy of partially specified system models, and (2) maintain the correctness of the model against slow behavioral changes to system components over time.

We can use supervised online learning to construct (approximate) dynamical models of the application components under control. These models capture the complex and nonlinear relationship between the response time achieved by a VM, and the CPU and memory share provided to it, under dynamically changing operating conditions. We propose to incorporate
the online control framework with an online model-learning component as shown in Fig. 8 aimed at developing autonomic systems that anticipate changes in operating conditions and react to gradual behavior changes in the system components.

Fig. 8 shows an online learning structure adjoined to the LLC framework that inputs the actual state of the system, \( x \) and compares it against the estimated system state as output by the system model, denoted as \( \hat{x} \). The difference between the actual and estimated state, \( \delta(x) \), is then applied to a model updating component. The process by which the model is updated will be dependent upon the modeling approach chosen. For example, for a simple lookup table, the adaptation process is a matter of updating the table entries. For more abstract models, such as a recurrent neural network, the adaptation process involves re-training the neural network as new data becomes available. The cost in terms of the time to perform the adaptation will again depend upon the modeling approach chosen.

The key challenge is to learn the new system behavior under dynamic conditions and update the model used by the controller in real time. An accurate system model helps to avoid the excessive switching a controller will introduce into the system to correct for misguided control actions, and to reduce the number of SLA violations. The key challenge will be for the model adaptation unit to learn an accurate new model of the system in a timely manner. As shown in Fig. 7, changes in the application behavior can occur suddenly. To tackle this problem, it is useful to study several modeling techniques (non-linear regression, auto-regressive moving averages, etc.) for their estimation accuracy and convergence time.

V. Conclusion

This results presented in this paper support the position that to obtain good control performance, the dynamical models of application components should be refined to match the time-varying behavior of the real system. We propose the development of an online learning structure that continuously refines system or component models using feedback from the system. It observes the current system state and compares it to the estimated system state to update the underlying approximation structure (e.g., lookup tables, neural networks). We also propose an analysis of how the convergence speed of a given learning structure impacts control performance, since, while the dynamical model converges to the behavior of the physical system, the controller will operate with inaccurate
models. The potential compensatory behavior of the controller can also be considered.

REFERENCES


