On the Use of Fuzzy Modeling in Virtualized Data Center Management

Jing Xu*, Ming Zhao*, José Fortes*, Robert Carpenter†, Mazin Yousif†
*University of Florida, {jxu, ming, fortes}@acis.ufl.edu
†Intel Corporation, {robert.e.carpenter, mazin.s.yousif}@intel.com

Abstract

One of the most important goals of data-center management is to reduce cost through efficient use of resources. Virtualization techniques provide the opportunity of carving individual physical servers into multiple virtual containers that can be run and managed separately. A key challenge that comes with virtualization is the simultaneous on-demand provisioning of shared resources to virtual containers and the management of their capacities to meet service quality targets at the least cost. This paper proposes a two-level resource management system with local controllers at the virtual-container level and a global controller at the resource-pool level. Autonomic resource allocation is realized through the interaction of the local and global controllers. A novelty of the controller designs is their use of fuzzy logic to efficiently and robustly deal with the complexity of the virtualized data center and the uncertainties of the dynamically changing workloads. Experimental results obtained through a prototype implementation demonstrate that, for the scenarios under consideration, the proposed resource management system can significantly reduce resource consumption while still achieving application performance targets.

1. Introduction

The need to manage multiple applications in shared data centers creates the challenge (and also the opportunity) of on-demand resource provisioning and allocation in response to dynamically changing workloads. It is often desirable for application providers to be able to lease data-center resources under a "pay-as-you-go" model, and for the data-center providers to be able to multiplex shared resources in a way that guarantees the expected performance of applications. To realize this, the data center must provide flexible and manageable execution environments that are specialized for each application without compromising its ability to share resources among applications and delivering to them the necessary performance, security and isolation.

Virtualization is key to this vision, by allowing physical servers to be carved into multiple virtual resource containers, and enabling a virtualized data center where applications are hosted and managed in their dedicated virtualized containers. In particular, virtual machines (e.g. [15][7][2]), which provide strong isolation, security and customizability, can be dynamically created to serve as virtual containers; while the management of these containers, e.g. lifecycle management and resource allocation, can be conducted through the interface provided by the virtualization platform.

Applications served by a data center are usually business-critical applications with Quality of Service (QoS) requirements, e.g. e-commerce services. Such applications have time-varying resource demands with typically high peak-to-mean ratios, leading to low resource utilization if over-provisioning is used to meet peak demands. The resource allocation needs to not only guarantee that a virtual container always has enough resources to meet its application's performance goals, but also prevent over-provisioning in order to reduce cost and allow the concurrent hosting of many applications. Static allocation approaches that consider a fixed set of applications and resources cannot be used because of changing workload mixes, and solutions that only consider runtime behavior individual applications fail to capture the competition for shared resources by virtualized containers.

This paper presents a two-level autonomic resource management system that enables automatic and adaptive resource provisioning in accordance with Service Level Agreements (SLA) specifying dynamic tradeoffs of service quality and cost. Resource control functions are integrated in a data center at two different levels of abstraction: virtual containers and resource pools. A local controller, created per virtual container, is responsible for determining the resources needed by its application and making resource requests accordingly. By doing so, the local controller minimizes leasing costs by avoiding over-provisioning

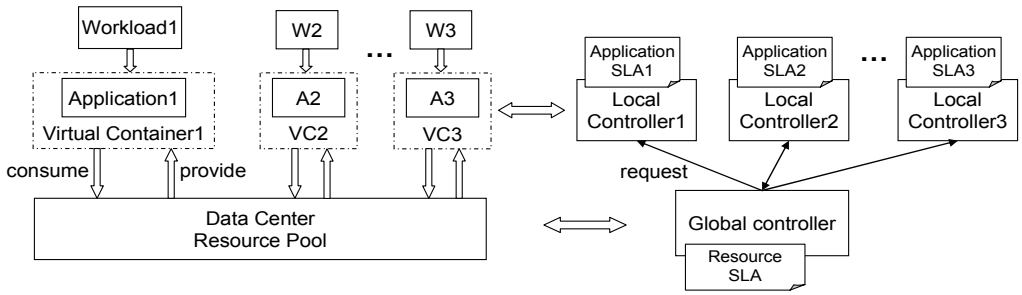

Figure 1. Data center with virtual resource containers to host applications, and a two-level controller architecture to allocate physical resources to containers.

for the application running on the container. A global controller manages the virtual containers hosted on the same physical resources. It responds to the local controllers' requests and allocates the shared resources to them in a way that maximizes the total profit (by leasing resources to a large number of containers).

The key to cost-effective resource allocation is the ability to efficiently find the minimum amount of resources that an application needs to meet the desired QoS. The paper presents a fuzzy-logic-based control system that applies fuzzy modeling to characterize the relationship between application workload and resource demand. A prototype of the proposed two-level resource management system has been deployed on a virtualized data center testbed. Typical e-business applications with synthetic workloads and real-world traces were used to evaluate the fuzzy modeling in the local controller and the resource allocation in the global controller. The results show that the proposed approach can effectively allocate resources to virtual containers under dynamically changing workloads, and significantly reduce resource cost while still achieving desired application performance.

The rest of this paper is organized as follows. Section 2 provides an overview of the two-level resource management system. Section 3 and Section 4 describe in detail the designs of the local controller and the global controller. Section 5 presents an evaluation of the prototype. Section 6 examines related work and Section 7 concludes the paper。

2. Two-level Autonomic Resource Control

A data center, illustrated in Figure 1, serves a number of applications. Each delivers a distinct service to its customers using (virtual) resources provided by its dedicated container, which is a virtual machine to host the application. The data center's resource pool allocates the physical resources to its virtual containers based on their applications' resource needs.

To achieve performance isolation and guarantee an *Application SLA* (between an application provider and the client) independently of the load on other containers, a local resource controller is employed in each virtual container to estimate the resources needed by the application's workload and to make resource requests accordingly. The global controller makes resource allocation decisions among competing requests, trying to avoid violations of *Resource SLAs* (between application providers and data center). The underlying assumption is that if the global controller does not allocate enough physical resources requested by the local controller resulting in application's SLA violation, the data center provider will be penalized instead of the application provider.

This two-level resource control system is preferred over the more obvious centralized approach in which all the control functions are implemented at one centralized location. Since local containers are independent of each other, heterogeneous local controllers' implementations are possible. All of the internal complexities of control functions in virtual containers are compressed by local controllers into straightforward resource requests, which specify the amount of resources needed. This approach makes it easy to add, change or remove virtual containers and their local controllers without affecting the global controller.

The system handles two different types of optimizations independently. The local controller tries to minimize the resource consumed by the virtual container to reduce the resource cost while still satisfying SLAs of its clients. The global controller seeks to maximize its own profit, which is the revenue received from allocating its resources among virtual containers minus the cost of penalties incurred from resource SLA violations. The following sections explain our approach to the design of the local and global controllers.

3. Local Resource Controller

Interaction between the local and global controllers enables a virtual container to augment its resources in

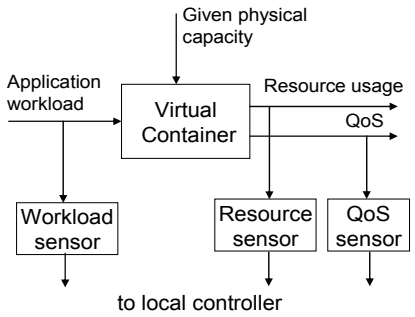

Figure 2. Inputs and outputs of a virtual container.

response to increased workload, and to reduce its resources when no longer needed. The main task of the local controller is to optimize the set of resources needed by an application running in the container. Our approach to the design of such a controller is based on fuzzy modeling, as discussed next.

3.1. Background

Fuzzy logic [19] is a tool to deal with uncertain, imprecise, or qualitative decision-making problems. Unlike in Boolean logic, where an element x either belongs or does not belong to a set A, the membership of x in A has a degree value in a continuous interval between 0 and 1. Fuzzy sets are defined by membership functions that map set elements into the interval [0 1].

One of the most important applications of fuzzy logic is the design of fuzzy rule-based systems. These systems use "IF-THEN" rules (fuzzy rules) whose antecedents and consequents use fuzzy-logic statements to represent the knowledge or control strategies of the system. A fuzzy model is a qualitative model constructed from a set of fuzzy rules to describe the relationship between system input and output [14].

The process of applying fuzzy rules on the system is called inference mechanism. Because fuzzy rules describe the relationship between system variables in fuzzy values, two functions are necessary for translating between numeric values and fuzzy values. The process of translating input values into one or more fuzzy sets is called fuzzification. Defuzzification is the inverse transformation which derives a single numeric value that best represents the inferred fuzzy values of the output variables.

3.2. Modeling based on Fuzzy Logic

To determine the resource needs of an application hosted in a virtual container, the local controller needs to learn the behavior of the virtual container under dynamically changing workloads. Figure 2 shows the abstracted inputs and outputs of a virtual container that hosts a running application. The virtual container receives the application workload from its clients, and utilizes the physical resources provided by data center resource pool to process the workload. The achieved QoS of the application depends on the amount of allocated resources and the workload. The goal is to find the relationship between workload and resource usage for acceptable QoS.

The proposed approach uses fuzzy logic to model the behavior of a virtual container from its input-output data without requiring *a-priori* knowledge about the system, which is a so-called "black box" approach. It automatically learns the relationship between workload and the corresponding resource demand needed to achieve desired QoS. Figure 3 (in Page 4) illustrates the key functions used to construct and apply fuzzy models in the local controller. The data monitored by the virtual container sensors are first processed by the filtering and clustering functions. Then the produced data clusters are used by the modeling function to create IF-THEN rules which are stored in a rule base. The cluster centers and ranges are used to determine the fuzzy model's parameters and are also stored in a database. Finally, the fuzzy inference functions take this learned model to determine the resource needs from the currently monitored workload. The rest of this section explains these functions in detail.

3.3. Online Monitoring and Data Filtering

Monitoring sensors periodically measure the workload (W), the application performance (P), and the resource usage (U) of a virtual container. For a typical data center application, its workload can usually be described by the rate and mixture of the requests. For instance, a Web server's workload can be characterized by the HTTP request rate as well as the ratio between the requests to static and dynamic Web content. The metrics for performance measurement are often directly taken from the SLA, e.g. the throughput (number of completed transactions per second) and/or average service response time.

The metrics for resource utilization are associated with the different types of consumed physical resources, including CPU percentage, memory size, disk storage, disk I/O rate and network bandwidth. However, an application's virtual resource usage (the values collected inside of the virtual container) does not necessarily represent its physical resource consumption. For example, an application's network I/O consumes not only the physical network bandwidth, but also the physical CPU cycles. In the proposed approach, an application's resource usage is obtained by directly monitoring the physical resource

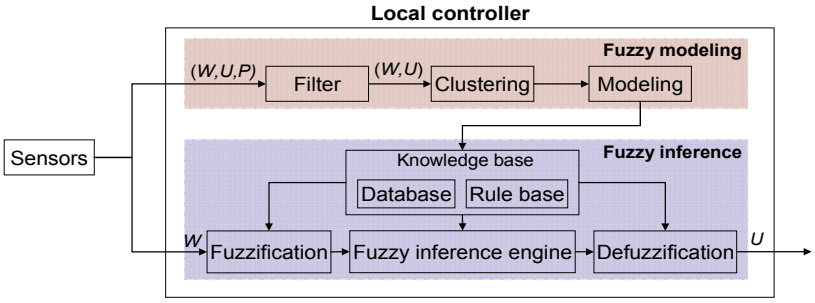

Figure 3. Fuzzy modeling and inference functions in a local resource controller.

consumption of its virtual container. This is sensible because in the envisioned data center a virtual container is dedicated to an application's service.

A sequence of data points (*W*, *U*, *P*) are produced by the sensors at constant time intervals (20 seconds in our experiments), and then filtered to generate a subset of points to be used to learn the fuzzy model that captures the relationship between workload and resource demand under a certain SLA. The data points are kept or filtered out depending on whether the measured performance satisfies the SLA or not, respectively. A data point's *P* is satisfactory, only if the resource capacity allocated to the virtual container at that point is sufficient for the corresponding workload for the given SLA. In this case, the monitored resource utilization represents the actual resource needs, and thus the data (*W*, *U*) can be used for model learning. On the contrary, an SLA violation indicates that the allocated resources are not enough to achieve the SLA target. In this case, the resource consumption is capped by the allocated capacity, so the monitored values are less than the desired resources and cannot be used in fuzzy modeling.

3.4. Data Clustering and Fuzzy Modeling

After the collection of a certain amount of paired data (*W*, *U*) are collected, the mapping between workload and resource demand can be learned by first clustering the data points and then building the fuzzy model based on the data clusters. The purpose of clustering is to produce a concise representation of the system's behavior. Several classic clustering algorithms can be used, e.g. hierarchical and k-means clustering. In the proposed local controller design, subtractive clustering [5] is chosen for its speed and robustness.

This clustering method assumes that each data point is a potential cluster center and chooses the data center based on the density of surrounding data points. The algorithm selects the data point with the highest potential to be the first cluster center and then removes all data points in the vicinity of the first cluster center in order to determine the next data cluster and its center location. This process continues until all of the data is within *radius* of a cluster center. The variable *radius* represents a cluster center's range of influence in each of the data dimensions, assuming the data falls within a unit hyperbox. Small *radius* values generally result in finding a large number of small clusters. This value is set to 0.5 in the local controller's implementation.

Since each cluster exemplifies a characteristic of system behavior it can be used as the basis of a fuzzy IF-THEN rule that describes system behavior. If *n* data clusters are formed, *n* rules can be produced in which the *i*th rule is expressed as:

Rule *i*: IF input data *w* is close to cluster *i*, THEN output data *u* is close to cluster *i*.

Each cluster specifies a fuzzy set with its membership functions determined by cluster center and range. Using the Gaussian membership function,

$$\mu_i(x) = e^{-\frac{(w-c_i)^2}{2\sigma_i^2}},$$

the center of the membership function c_i for fuzzy set *i* equals the center of cluster *i* and the weight of membership function σ_i equals the *radius* of cluster *i*.

The model described by the above fuzzy rule is called zero-order Sugeno-type fuzzy model [14]. The modeling accuracy can be improved significantly by using the first-order Sugeno model, in which the output of each rule is a linear function of the input variables. The rules are rewritten as follows,

Rule *i*: IF input data *w* is close to cluster *i*, THEN output data $u = aw + b$.

The parameters *a* and *b* in the linear equations are estimated by the least-squares method. This model is implemented in the local controller's fuzzy modeling function.

3.5. Fuzzy Inference

Once the fuzzy model relating workload to resource demand is learned from the selected data relating workload to resource usage, it can be used in a rule-

based fuzzy inference module which, given the application's workload (W), produces the estimated resource demand (U) for the application's container.

The fuzzy controller consists of four basic functions (Figure 3). The knowledge base includes a database which contains the membership functions of the fuzzy sets and a rule base where the fuzzy rules are specified. In the fuzzification function, the input (W) collected from the sensor is mapped to fuzzy values using the membership functions. A decision-making unit, called the fuzzy inference engine, infers from a fuzzy inputs to resulting fuzzy outputs according to the rules stored in the knowledge base. The defuzzification function aggregates the outputs and converts them to a crisp output. The final output (U) of the system is the weighted average of all rule outputs with the weight of ith rule being the membership value of the input in cluster i.

In summary, using fuzzy modeling and fuzzy inference shown in Figure 3, the local controller estimates the resource needs for the current workload measured by the sensor, and sends requests to the global controller to either ask for more resources if the current allocation is not sufficient to satisfy the SLA or to withdraw resources when no longer needed.

3.6. Adaptive Modeling

The discussion so far has only considered offline model learning from collected data. As the workload or system condition changes, the model describing the system's behavior needs to capture the changes accordingly. The adaptive modeling is employed by the local controller in which the model is repeatedly updated based on online monitored information.

The clustering function takes new data into consideration as soon as they arrive (after the filtering) and keeps updating, so that up-to-date clusters are always provided for the modeling. Whenever the data clusters are updated, the fuzzy model' parameters are changed accordingly in the database. If a new cluster is added, a corresponding rule is then added into the rule base; and similarly, if a cluster no longer exists, the rule associated with it is removed from the rule base.

In the case when the allocated resources are insufficient for the workload, the monitored data becomes disqualified and is filtered out because of the performance degradation. The shortage of qualified data will hurt the model's learning speed and quality. To avoid this situation, whenever the filter function detects that the percentage of qualified data is less than 50%, the controller asks for an additional predefined percentage (10% is used in our prototype) of current resource allocation from the global controller to improve the application's performance back to the desired level.

4. Global Resource Controller

The global controller receives requests for physical resources from local controllers and allocates the resources among them. It seeks to make allocations that maximize its own profit, which is the revenue received from allocating its resources minus the penalties due to resource SLA violations.

Suppose that K virtual containers are concurrently active in the data center. Let d_k denote the resource demand from container k, and c_k be the amount of resources granted to it. The global controller receives revenue of r_k for every allocated resource unit. But if the global controller cannot satisfy the requested d_k, it also pays a penalty of p_k per unit of unmet resource demand, according to the resource SLA. The profit that can be made by the global controller's allocation is,

$$profit(c_1, c_{2,\ldots}, c_k) = \sum_{k=1}^{K} [r_k c_k - p_k (d_k - c_k)],$$

$$s.t.\ 0 \le c_k \le d_k,\ \ C = \sum_{k=1}^{K} c_k \le I$$

where C is the total amount of resources allocated to the virtual containers, and I is total available resource capacity in the data center.

The profit equation can be rewritten as follows,

$$profit(c_1, c_{2,\ldots}, c_k) = \sum_{k=1}^{K} (r_k + p_k) c_k - \sum_{k=1}^{K} p_k d_k$$

Consider $r_k + p_k$ as the profit rate for the virtual container k and assume that the global controller can allocate any resource fraction to the virtual containers, a greedy algorithm that allocates resources in the order of decreasing profit rates is an optimal allocation (this is similar to the case of a fractional knapsack problem [10]).

5. Experimental Evaluation

This section summarizes the experimental evaluation of the suitability of the proposed two-level control system for dynamic resource allocation in a data center environment with time-varying workloads. Section 5.2 discusses experiments that evaluate the ability of the local controller to track the resource needs of changing workloads. Section 5.3 considers the maximal profit approach of Section 4 when the global controller must share limited resources among several containers.

5.1. Setup

5.1.1. Data Center Environment

The testbed consists of a pool of servers that provide virtual containers for applications, and several workload-generating clients. VMware ESX Server 3.0.1 is installed in each server node which has dual hyperthreaded Intel Xeon 3.2GHz CPUs and 4GB memory. Virtual machines are created on the servers and used as virtual containers to host applications. The clients are created on VMware-Server-1.0.0-based virtual machines, hosted on another cluster of dual-2.4GHz hyperthreaded Intel Xeon nodes. Workloads are generated by the clients and issued to the applications across a Gigabit Ethernet network.

5.1.2. Application and Workload

The Java Pet Store [24] is chosen to represent a typical e-business application, which implements an online store that allows users to browse and make orders, and managers to manage orders, suppliers and inventory. It is a reference application that has been developed on various Java EE platforms. The version of Java Pet Store used in the experiments is 1.3.1 and built with J2EE 1.3.1.

Synthetic HTTP workloads that mimic the key aspects of real-world workloads are created with various client sessions generated by *httperf* [11]. Each individual session consists of a sequence of requests generated by repeating and mixing the following customer actions: go to the storefront, sign in, browse products, add some products to shopping cart, and checkout. Two key parameters can be adjusted to vary a session's workload on the application: the user think time (the time between two consecutive requests) can be changed to generate different request rates; the ratio of dynamic requests (e.g., sign in, check out and search product) versus static requests (e.g., browse static Web pages and view images) can be varied to impact a session's workload characteristics. A *Perl* program is developed to create different workloads and drive *httperf* to issue the requests.

Traces collected from '98 World Cup sites are also used in the experiments to represent real-world workloads. The logs provided by an Internet repository [23] consist of about 1.3 million requests made to the '98 World Cup Web site between April 30, 1998 and July 26, 1998. A real-time log replayer is used to generate workloads according to the trace [22].

5.1.3. Controller Prototype

The virtual containers are monitored and controlled through the Web-service-based management interface provided by VMware ESX Server. It allows the allocation of a server's available physical resources among its hosted virtual machines (e.g. setting the minimum, maximum and proportional resource shares of a virtual machine), and also the real-time monitoring of a virtual machine's resource utilization.

The proposed two-level controllers are implemented in Java, running along with the virtual containers. Every virtual machine has a local controller to manage the virtual container it provides, and every ESX server node has a global controller to manage the shared resources for the virtual containers hosted on it. The sensors, also developed in Java, monitor the workload (request rate and mixture), the application throughput (reply rate), and the resource consumption (CPU usage). The monitoring period is set to 20 seconds. Because the concerned workloads are mostly CPU intensive, the experiments focus on the utilization and allocation of CPU resources.

5.2. Experiment on Local Controller

5.2.1. Workload with Static Web Requests

In the first experiment, the workload generator issues a new session to the Pet Store every 10 seconds, up to a total of 15 sessions. The sessions only issue static Web page requests with a user think-time between 0.1 to 1 second, and each session lasts around 1 minute. After a group of 15 sessions are completed, another group is generated similarly but with a decreasing average think-time (and hence an increasing request rate). This setup emulates the burstyness of workloads in real world. The entire experiment lasts for 4000 seconds.

The workloads used in this experiment consist of only static Web requests. In this case, the CPU usage is highly correlated with request rate, which is used as the only parameter to characterize the workload. The first 50 pairs of data points (the request rate and CPU usage) collected from the sensors are used to initialize the learning of the fuzzy model. Afterwards, the model is continuously updated every 200 seconds (in which 10 new data points become available from the sensors). Figure 4 illustrates the model learned at the beginning and the end of the experiment, which shows an approximate linear relationship between the request rate and CPU usage [1] in the range of 0 to 100 requests/second.

The local controller continuously estimates the CPU demand based on the current workload and the latest learned fuzzy model, and dynamically adjusts the CPU requests to the global controller. Because the available

[1] In VMware ESX Server, the amount of CPU allocation and usage can be expressed in CPU frequencies (Hz).

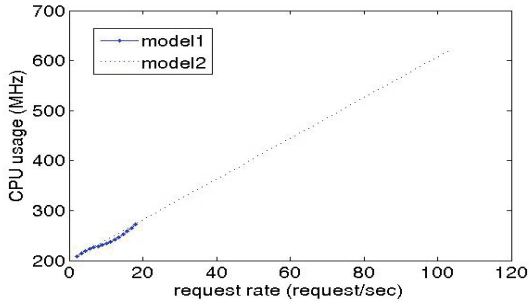

Figure 4. Fuzzy model learned from the workload with static Web requests at the beginning (model1) and the end (model2) of the experiment.

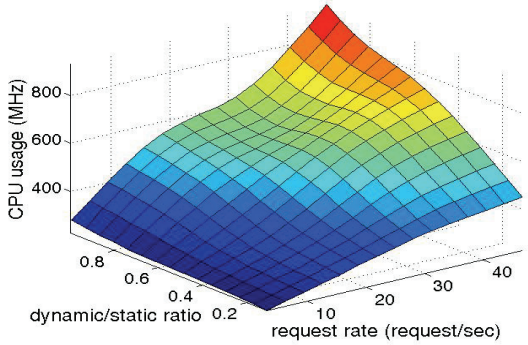

Figure 6. The surface of the fuzzy model learned from the workload with dynamic Web requests.

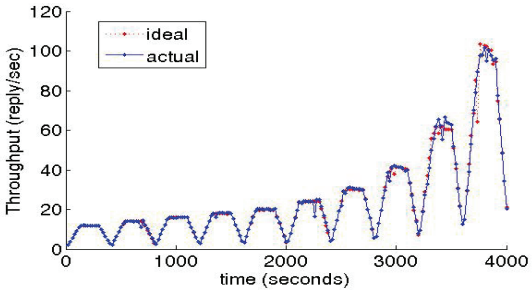

Figure 5. Comparison of the throughput achieved by using local controller and the ideal throughput for the workload with static Web requests.

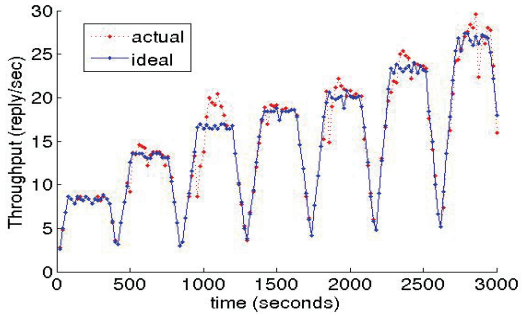

Figure 7. Comparison of the throughput achieved by using local controller and the ideal throughput for the workload with dynamic Web requests.

resources are sufficient in this experiment, the global controller always allocates to the virtual container the exact amount of CPU requested by the local controller. To prove the accuracy of the fuzzy modeling, the same experiment is repeated on the virtual container which is statically allocated with a large amount of CPU (3.2GHz) in order to obtain the ideal throughput for the same workload.

The throughputs from these two runs are compared in Figure 5, indicating that the actual throughput obtained by using the local controller is almost identical to the ideal throughput. Compared to the static allocation of CPU with the peak value (overprovision based on the highest load), the dynamic approach saves about 55% of CPU cycles otherwise needed for this experiment. It proves that the online fuzzy modeling can accurately learn the relationship between the workload and resource demand, and effectively guide the resource allocation for the virtual container.

5.2.2. Workload with Dynamic Web Requests

In the second experiment, the workloads are generated similarly to the previous one, except that dynamic Web requests are also considered. Every group of sessions differs not only in the request rate but also the proportion of dynamic requests in the workload: the ratio of dynamic to static requests grows from 0 to 1 across the groups. Servicing dynamic Web content requires processing by the application server and database, and thus typically consumes more resources than servicing static Web content. If the fuzzy modeling still uses only request rate as the input, the resulting model cannot effectively represent the actual relationship between workload and resource demand. The experiment results (observed but not shown here) also confirm that the throughput achieved by using such a model is much worse than the ideal throughput for the workload.

In contrast, using both the request rate and dynamic/static request ratio to characterize the workload, a 3D fuzzy model can be constructed to describe the relationship between workload and resource demand more accurately. Figure 6 shows the surface of the model learned at the end of the experiment. One of the advantages of fuzzy modeling demonstrated by the above experiments is that fuzzy models can effectively learn simple as well as non-linear and complex relationships between inputs and outputs.

Figure 7 compares the application's throughput to the ideal throughput obtainable for the workload. The graph shows that the throughput achieved is again very close to its ideal level (the difference is under 6%). It is

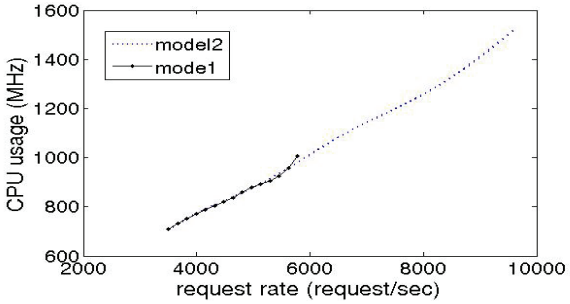

Figure 8. Fuzzy model learned from the trace-based workload at the beginning (model1) and the end (model2) of the experiment.

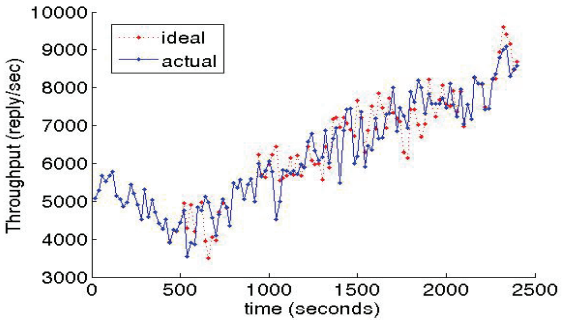

Figure 9. Comparison of the throughput achieved by using local controller and the ideal throughput for the trace-based workload.

also noticeable that when the workload is high the difference becomes relatively larger. This is because of the delay between the change of workload and resource allocation, which is largely due to the granularity of the online monitoring and control. When the workload is heavy, this delay causes the application's throughput to fluctuate a little around the ideal one. However, the overall error is still very low. About 33% of CPU cycles are saved by this dynamic allocation, compared to a fixed allocation where overprovision is based on the highest load.

5.2.3. Trace-Driven Workload

In the third experiment, the '98 World Cup Web site trace collected on May 31 from 5am to 5pm (local time in Paris) is used to generate workload, and it is played at 12X speedup to enhance its intensity. All the documents requested by the trace are created by the log replayer tool based on the sizes recorded in the trace. Because only static Web pages are requested in the trace replaying, the workload is characterized by the request rate.

During the experiment, the first 30 measurements of workload and CPU consumption are used to initialize the fuzzy model. After that, the model is updated every 200 seconds. Figure 8 illustrates the model learned at the beginning and the end of the experiment. Figure 9

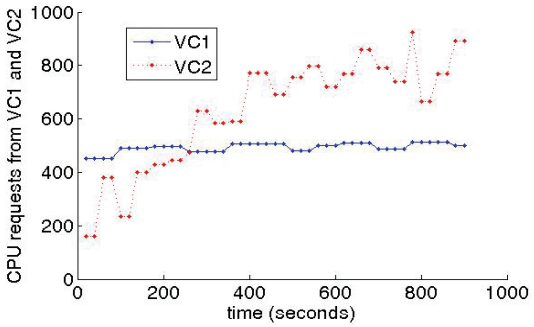

Figure 10. CPU requests from two virtual containers (VC1, VC2) that share limited resources.

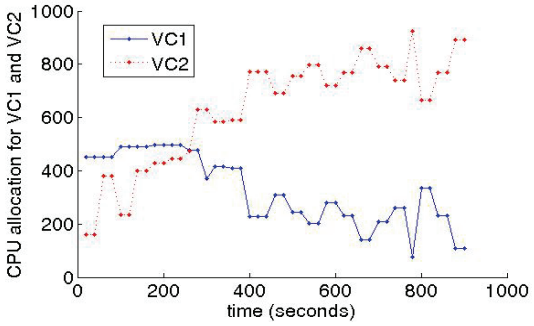

Figure 11. CPU allocation that favors VC2.

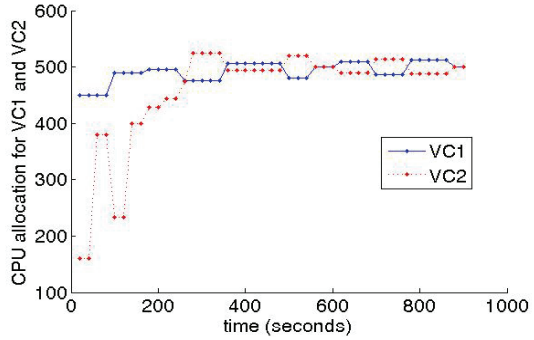

Figure 12. CPU allocation that favors VC1.

shows that the application's throughput achieved by using the local controller is close to the ideal throughput obtainable for the workload (the difference is within 5%), indicating the effectiveness of the fuzzy approach under real-world workloads. The dynamic allocation uses less than 75% of the CPU cycles used by a static approach that allocates maximum CPU fraction based on the highest workload.

5.3. Experiments on Global Controller

The last set of experiments investigates the global controller's allocation of limited resources among multiple virtual containers. Two virtual containers (VC1, VC2) running on the same server node compete for the available CPU cycles (1GHz). VC1 serves a fixed workload, which has a constant request rate of 30

requests/sec; while VC2 receives an increasing workload with a request rate rising from 10 up to 60 requests/sec. The workloads used in this experiment only consider static Web requests.

Both local controllers of these two containers employ the fuzzy modeling approach to dynamically estimate their CPU demands for the workloads, and the amounts of resources requested during the experiment are plotted in Figure 10. The local controller of VC1 requests around 500MHz of CPU throughout the entire experiment; while VC2 increases its CPU request from about 200MHz to more than 800MHz as its workload grows.

When the CPU needed by VC2 goes beyond 500MHz, the global controller responds to the resource shortage by reducing the allocation to VC1 or VC2. The allocation policy of the global controller is to maximize its profit by employing the greedy algorithm discussed in Section 4. Two simple scenarios are considered in the experiments. In the first case, the profit rate of VC2 is higher than VC1; therefore, the global controller decides to satisfy the resource requests from VC2 by reducing the allocation for VC1 whenever a CPU shortage happens. Figure 11 shows the actual CPU allocations for the two containers throughout the experiment. The second case considers the opposite situation where VC1 has a higher profit rate and thus is favored in the resource allocation. In this case, VC2 suffers from the resource shortage when the global controller cannot allocate enough resources for both containers (Figure 12).

6. Related Work

To the best of our knowledge there is no prior work using a fuzzy modeling approach to data center resource management. The following briefly summarizes other work with some common elements with this paper's approach.

Rule-based systems: This approach uses a set of event-condition-action rules (defined by system experts) that are triggered when some precondition is satisfied (e.g., when some metrics exceed a predefined threshold). For example, the HP-UX Workload Manager [21] allows the relative CPU utilization of a resource partition to be controlled within a user-specified range, and the approach of Rolia et al. [12] observes resource utilization (consumption) by an application workload and uses some "fixed" threshold to decide whether current allocation is sufficient or not for the workload. With the growing complexity of systems, even experts are finding it difficult to define thresholds and corrective actions for all possible system states.

Control theory: Approaches based on control theory have been applied to resource management to achieve performance guarantees. Most of the work assumes a linear relationship between the QoS parameters and the control parameters, and involves a training phase with a given workload to perform system identification. Typically, control parameters must be specified or configured offline and on a per-workload basis. Abdelzaher et al. [1] investigated this approach for QoS adaptation in Web servers. In [17][20], a nonlinear relation between response time and CPU allocation to a Web server is studied, and a bimodal model is used to switch between underload and overload operating regions. To deal with time-varying workloads, more recent work applies adaptive control theory, in which models are automatically adapted to changes using online system identification.

Model-based: Previous research efforts [4][8][13][18][16] have been trying to model computer systems from different perspectives. Bennani et al. [3] predicts the response time and throughput for both online and batch workloads using multiclass open queueing networks. Liu et al. [9] uses AR models to map CPU entitlement percentage to the mean response time with a fixed workload. Chandra et al. [4] models the resource using a time-domain queueing model which relates the resource requirements to its workload. Some of these approaches make simplifying assumptions such as using a single queue to model the whole system, which may fail to capture complexities of the relationship between application workload and resource usage. Some models are validated only using simulations.

Fuzzy control: Diao et al. [6] proposed a profit-oriented feedback control system for maximizing SLA profits in Web server systems. The control system applies fuzzy control to automate the admission control decisions in a way that balances the loss of revenue due to rejected work against the penalties incurred if admitted work has excessive response time.

The proposed resource management system differs from the prior work in the following aspects:

- The resource control functions are separated between resource provider and application provider, which makes the design of data center resource management more flexible and robust. Each local controller tries to maximize its profits by requesting adequate resources for satisfying application SLAs as well as reducing unnecessary resource cost. The global controller takes into account the tradeoff between revenue obtained from satisfied resource requests and cost from violations of resource SLAs.

- Fuzzy modeling provides a generic approach to representing the relationship between system variables. It can be easily applied to any type of applications hosted in virtual containers. This approach makes no underlying assumption of the workload characteristics, and can learn any type of relationship very fast. Especially, the fuzzy system is well suited for modeling the nonlinear system with dynamically changing operating conditions.
- The resource management process is automatically done without any human intervention. The fuzzy modeling relates the application resource requirements to their dynamically changing workload characteristics using online monitored data. The learned model is updated continuously as new data arrives, which enables the model to capture transient or unexpected workload changes.

7. Conclusions

This paper presents a flexible two-level resource management system that is able to provide high quality of service with much lower resource allocation costs than worst-case provisioning. To accurately estimate resource demands, fuzzy models are used by the controllers to learn the relationship between the workload and resource needs of virtual containers and to guide resource allocation based on online measurements. Our approach, in conjunction with virtualization techniques, can provide application isolation and performance guarantees in the presence of changing workloads by dynamically allocating resources at fine time granularity, which results in high utilization and low cost as well. The proposed resource management system is implemented on a virtualized data center testbed and evaluated using applications that are representative of e-business and Web-content delivery scenarios. Both synthetic and real-world Web workloads are used to evaluate the effectiveness of the approach. The experimental results show that the system can significantly reduce resource cost while still guaranteeing application QoS in various scenarios.

8. Acknowledgements

This work was supported in part by an Intel grant (IT R Council), National Science Foundation grant CNS-0540304, the BellSouth Foundation and equipment awards from DURIP and IBM. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, BellSouth Foundation, Intel, IBM, or VMware.

9. References

[1] T. Abdelzaher et al., "Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach", IEEE Trans. on Parallel and Distributed Systems, 2002.

[2] P. Barham et al., "Xen and the Art of Virtualization", Proc. Symp. on Operating Systems Principles, 2003.

[3] M. N. Bennani, D. A. Menascé, "Resource Allocation for Autonomic Data Centers using Analytic Performance Models", ICAC, 2005.

[4] A. Chandra, W. Gong, P. Shenoy, "Dynamic Resource Allocation for Shared Data Centers Using Online Measurements", IWQoS, June 2003.

[5] S. Chiu, "Fuzzy Model Identification Based on Cluster Estimation", J. of Intell. & Fuzzy Systems, Vol. 2, 1994.

[6] Y. Diao, J. L. Hellerstein, and S. Parekh, "Using Fuzzy Control To Maximize Profits In Service Level Management", IBM Syst. J., Vol. 41, No. 3, 2002.

[7] J. Dike, "A User-mode Port of the Linux Kernel", 4th Linux Showcase and Conference, USENIX 2000.

[8] R. Doyle et al., "Model-Based Resource Provisioning in a Web Service Utility", USENIX Symp. on Internet Technologies and Systems, Mar 2003.

[9] X. Liu et al., "Adaptive Entitlement Control Of Resource Containers On Shared Servers", IFIP/IEEE Intl. Symp. on Integrated Network Management, 2005.

[10] S. Martello, P. Toth, "Knapsack Problems: Algorithms and Computer Implementations", John Wiley & Sons.

[11] D. Mosberger, T. Jin, "httperf: A Tool for Measuring Web Server Performance", First Workshop on Internet Server Performance, 1998.

[12] J. Rolia et al., "Configuring Workload Manager Control Parameters for Resource Pools", 10th IEEE/IFIP Network Operations and Management Symposium, 2006.

[13] L. Sha, et al., "Queueing Model Based Network Server Performance Control", Real-Time Systems Symp., 2002.

[14] M. Sugeno, T. Yasukawa, "A Fuzzy-Logic-Based Approach to Qualitative Modeling", IEEE Transactions on Fuzzy Systems, Vol. 1, Issue. 1, 1993.

[15] J. Sugerman et al., "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor", Proc. USENIX Annual Technical Conf., 2001.

[16] B. Urgaonkar et al., "An Analytical Model for Multi-tier Internet Services and its Applications", ACM SIGMETRICS, Jun 2005.

[17] Z. Wang et al., "Utilization and SLO-Based Control for Dynamic Sizing Of Resource Partitions", DSOM 2005.

[18] W. Xu et al., "Predictive Control for Dynamic Resource Allocation in Enterprise Data Centers", NOMS, 2006.

[19] L. A. Zadeh, "Fuzzy Sets", Info. & Ctl., Vol. 8, 1965.

[20] X. Zhu, Z. Wang, S. Singhal, "Utility-Driven Workload Management using Nested Control Design", American Control Conference, June 2006.

[21] HP-UX Workload Manager, http://docs.hp.com/en/5990-8153/ch05s12.html

[22] http://www.cs.virginia.edu/~rz5b/software/software.htm

[23] http://ita.ee.lbl.gov/html/contrib/WorldCup.html

[24] https://blueprints.dev.java.net/petstore/