Cluster Comput (2008) 11: 213–227
DOI 10.1007/s10586-008-0060-0

Autonomic resource management in virtualized data centers using fuzzy logic-based approaches

Jing Xu · Ming Zhao · José Fortes · Robert Carpenter · Mazin Yousif

Received: 31 May 2008 / Accepted: 16 June 2008 / Published online: 16 July 2008

Abstract Data centers, as resource providers, are expected to deliver on performance guarantees while optimizing resource utilization to reduce cost. Virtualization techniques provide the opportunity of consolidating multiple separately managed containers of virtual resources on underutilized physical servers. A key challenge that comes with virtualization is the simultaneous on-demand provisioning of shared physical 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 to dynamically allocate resources to individual virtual containers. It uses local controllers at the virtual-container level and a global controller at the resource-pool level. An important advantage of this two-level control architecture is that it allows independent controller designs for separately optimizing the performance of applications and the use of resources. Autonomic resource allocation is realized through the interaction of the local and global controllers. A novelty of the local controller designs is their use of fuzzy logic-based approaches to efficiently and robustly deal with the complexity and uncertainties of dynamically changing workloads and resource usage. The global controller determines the resource allocation based on a proposed profit model, with the goal of maximizing the total profit of the data center. 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.

Keywords Autonomic computing · Fuzzy modeling · Resource management · Virtualization · Two-level control · Data centers

J. Xu (✉) · M. Zhao · J. Fortes
University of Florida, Gainesville, FL, USA
e-mail: jxu@acis.ufl.edu

M. Zhao
e-mail: ming@acis.ufl.edu

J. Fortes
e-mail: fortes@acis.ufl.edu

R. Carpenter · M. Yousif
Intel Corporation, Hillsboro, OR, USA

R. Carpenter
e-mail: robert.e.carpenter@intel.com

M. Yousif
e-mail: mazin.s.yousif@intel.com

1 Introduction

Today's data centers host a variety of business-critical applications such as web hosting, e-commerce sites and enterprise systems on shared hardware platforms. The need to manage multiple applications in a shared infrastructure creates the challenge (and also the opportunity) of on-demand resource provisioning and allocation in response to time-varying workloads. Ideally, data centers should provide a "pay-per-use" approach, i.e., hosted applications or application providers would deliver services to their customers using hardware resources provided by data centers that charge based on the resources consumed (instead of charging for resources needed to meet peak demands). To realize this in a cost-effective manner, 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., [2, 7, 15]), which provide strong isolation, security and customizability, can be dynamically created to serve as virtual containers. 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. 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 more 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 behavior of individual applications fail to capture the competition for shared resources by virtualized containers.

This paper presents a two-level resource management system that enables automatic and adaptive resource allocation in accordance with Service Level Agreements (SLA) specifying dynamic tradeoffs of service quality and cost. Resource management in a data center is decoupled at two levels: virtual containers and resource pools. 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. In each virtual container hosting an application, a local controller is responsible for determining the amount of resources needed by the application and making resource requests accordingly. A global controller responds to the local controllers' requests by dynamically allocating resources across multiple virtual containers hosted on the same physical resources. It controls the resource allocations in a way that maximizes the data center's profit.

Two fuzzy-logic-based approaches—fuzzy modeling and fuzzy prediction—are proposed for use by local controllers in automatically learning of runtime behavior of virtual containers under dynamically changing workloads. One of the advantages of fuzzy approaches is that they do not require prior knowledge or a mathematical model of the system being managed. They typically do not need time-consuming training, which makes them suitable for real-time control. Moreover, the approaches are robust with respect to noisy data and have the ability to adapt to changes very quickly. 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 accuracy of the fuzzy-logic-based approaches employed by the local controller and the efficiency of resource allocation determined by the global controller. The results show that the proposed system can effectively allocate resources to virtual containers under dynamically changing workloads, and significantly reduce resource consumption while still achieving the desired application performance.

The rest of this paper is organized as follows. Section 2 provides an overview of the two-level resource management system. Sections 3 and 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 Sect. 7 concludes the paper.

2 Two-level autonomic resource control

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

Application SLAs between an application provider and its customers state the quality of service providers promised to the clients. To achieve performance isolation and guarantee an application SLA 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 to the global controller. By doing so, the local controller minimizes leasing costs by avoiding over-provisioning for the application running on the container. *Resource SLA*s between application providers and the data center owner specify both the cost of rental resources and the penalty due in case the data center fails to deliver resources needed by application providers. The underlying assumption is that if the data center does not allocate enough physical resources requested by the local controller resulting in its application's SLA violation, the data center provider will be penalized. The global controller makes allocation decisions among competing requests, trying to avoid violations of resource SLAs.

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. The system handles two different types of optimizations independently. The local controller tries to minimize the resource consumed by the virtual container

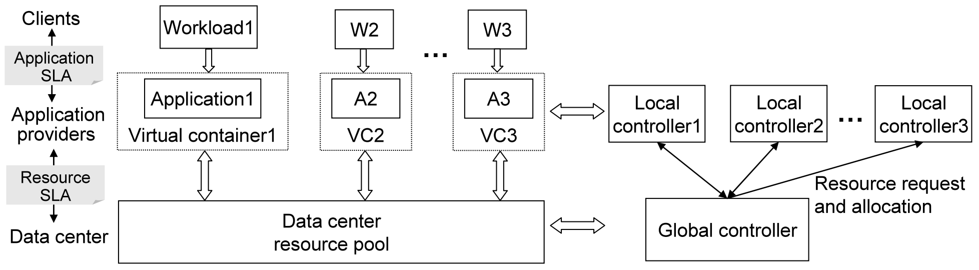


Fig. 1 "Pay-as-you-go" data center with virtual resource containers to host applications, and a two-level controller architecture to allocate physical resources to containers

to reduce the resource cost while still satisfying application 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 controller design is applicable to manage any type of resource such as CPU, memory and I/O bandwidth, while provisioning and managing CPU cycles are of particular interest in this paper and tested in more detail. 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 response to increased workload, and to reduce its resources when they are no longer needed. The main task of the local controller is to estimate 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 logic theory, as discussed next.

3.1 Background

Fuzzy logic [21] is a tool to deal with uncertain, imprecise, or qualitative decision-making problems. Unlike Boolean logic, where an element x either belongs or does not belong to a set A, in fuzzy logic the membership of x in a fuzzy set F has a degree value (called fuzzy value) in a continuous interval between 0 and 1 representing the extent to which x belongs to F. 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 (also called fuzzy rules) whose antecedents and consequents use fuzzy-logic statements to represent the knowledge or control strategies of the system. The collection of fuzzy rules is called a rule base. There are many approaches to the construction of fuzzy rules, for example, by capturing expert experience or system operator's control actions. The approach taken for the design of our system is to learn fuzzy rules using online monitoring information, making it a so-called self-organizing fuzzy system.

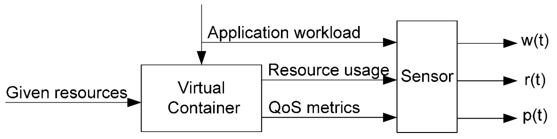


Fig. 2 The monitored runtime behavior of virtual container

The process of formulating the mapping from inputs to outputs using fuzzy rules is called the fuzzy inference (FIS) mechanism. Since fuzzy rules use fuzzy sets and their associated membership functions to describe system variables, 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 variable.

3.2 Virtual container run-time behavior

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 the data center resource pool to process the workload. The achieved QoS of the application depends on the amount of allocated resources and the incoming workload. The information about the application's workload, its received performance and its virtual container's resource consumption are monitored by the system sensors as Fig. 2 illustrates. The local controller adaptively adjusts the resources requested from the global controller, in order to achieve desired QoS with the minimum cost. Depending on what information is available from the system, two approaches are proposed for estimating resource needs: (1) fuzzy modeling to characterize the relationship between workload and resource use and (2) fuzzy prediction to determine a mapping from observations of recent resource usage to future resource needs.

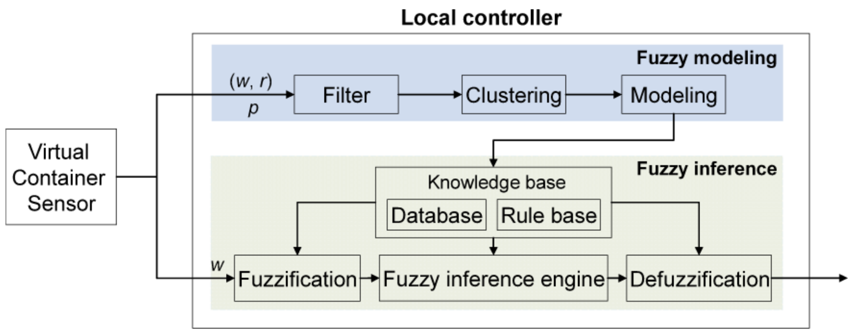


Fig. 3 Fuzzy modeling and inference functions in a local resource controller

3.3 Fuzzy-modeling approach

The first approach uses fuzzy logic to model the behavior of a virtual container by automatically learning the relationship between workload and the corresponding resource consumption when the desired QoS is achieved. It requires the system to periodically monitor the application workload and their resource usage, which are then used as an input-output data pair for generating fuzzy rules. Figure 3 illustrates the key functions for fuzzy modeling in the local controller. The data monitored by the sensors are first processed by the filtering and clustering functions. The modeling function constructs fuzzy IF-THEN rules using the produced data clusters and keeps them in a rule base. The fuzzy model's parameters determined by the calculated centers and ranges of data clusters are also stored in a database. Once the fuzzy model is built up, the fuzzy inference functions periodically process the fuzzy rules kept in the knowledge base to determine the resource needs based on the currently monitored workload. The rest of this section explains these functions in detail.

Data monitoring and filtering The sensors periodically measure the application workload $w(t)$, its performance $p(t)$, and the resource usage $r(t)$ 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 of static Web-content requests to dynamic ones. The performance metrics 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 utilization, used 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 consumption of its virtual container. This is sensible because in the envisioned data center a virtual container is dedicated to an application.

The measured workload and its resource usage compose the input-output data used for system modeling. A sequence of input-output data pairs $(w(t), r(t))$ produced by the sensors at constant time intervals (20 seconds in our experiments) is filtered based on the corresponding performance measurements $p(t)$. The filtering policy is such that a data pair measured at time t is kept or filtered out depending on whether the performance measured at the same time satisfies the SLA or not, respectively. Performance is satisfactory only if the resource capacity allocated to the virtual container at time t is sufficient for the given SLA. In this case, the monitored resource utilization represents the actual resource needs, and thus the data pair 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 that the monitored values are less than the desired resource demands and cannot be used in fuzzy modeling.

Data clustering and fuzzy rule construction The filtered data pairs are used for building fuzzy models which characterzing the relationship between the application workload and resource consumption. In order to avoid generating a large number of fuzzy rules, the data are clustered 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 density as 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 the data are within the radius of a cluster center. The variable *radius* represents a cluster center's range of influence in each of the data dimensions, assuming that the data fall within a unit hyperbox. Setting small radius values generally results 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 rule that describes system behavior. If n data clusters are formed, n rules can be produced in which the ith rule is expressed as:

IF input $w(t)$ is in cluster i,

THEN output $u(t)$ is in cluster i.

Each cluster specifies a fuzzy set with its membership functions determined by the 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 equals the center of cluster i and the weight of membership function σ_i equals the radius of that cluster.

The model described by the above fuzzy rules 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,

IF input $w(t)$ is in cluster i, THEN output $u(t) = aw + b$,

where the parameters a and b in the linear equations are estimated by the least-squares method.

Fuzzy inference Once the fuzzy model relating workload to resource consumption is learned from the selected workload to resource usage measurements, it can be used in a rule-based fuzzy inference module which, given the application's workload, produces the estimated application's resource demand for the virtual container.

The fuzzy inference module consists of four basic functions as illustrated in Fig. 3. The knowledge base includes a database consisting of membership functions of the fuzzy sets and a rule base where the fuzzy rules are specified. In the fuzzification function, the input $w(t)$ measured from the sensor is mapped to input fuzzy sets using the membership functions. A decision-making unit, called the fuzzy inference engine, infers from input fuzzy sets to output fuzzy sets according to the rules stored in the knowledge base. The defuzzification function aggregates the fuzzy outputs and converts them to a numeric output. The final output is the weighted average of all rule outputs with the weight of ith rule being the membership of the input in cluster i.

In summary, using fuzzy modeling and fuzzy inference shown in Fig. 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.

Adaptive modeling The discussion so far has only considered offline model learning from collected data. As the workload or system conditions change, 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 and keeps updating, so that up-to-date clusters are always provided for the modeling. Whenever the data clusters are updated, the parameters of the membership functions 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 become disqualified and are 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% during a time window T (set to 5 minutes in the prototype), the controller asks for an additional predefined percentage (10% is used in the prototype) of current resource allocation from the global controller to improve the application's performance back to the desired level.

3.4 Fuzzy-prediction approach

The fuzzy-modeling-based approach described above automatically builds a mapping from the application workload to the corresponding resource needs for the desired QoS. This approach is applicable only when the application workload can be characterized and monitored. However, data centers can host a variety of applications which are very different from each other so that there is no standard way of measuring workloads. In some cases it is hard to describe an application workload using a few metrics like request rate. The second proposed approach—fuzzy-prediction—only requires information about the resource usage (e.g., CPU utilization), which is easy to obtain by monitoring system-level metrics. The basic idea is to determine future resource needs on the basis of observations of past resource usage using fuzzy system.

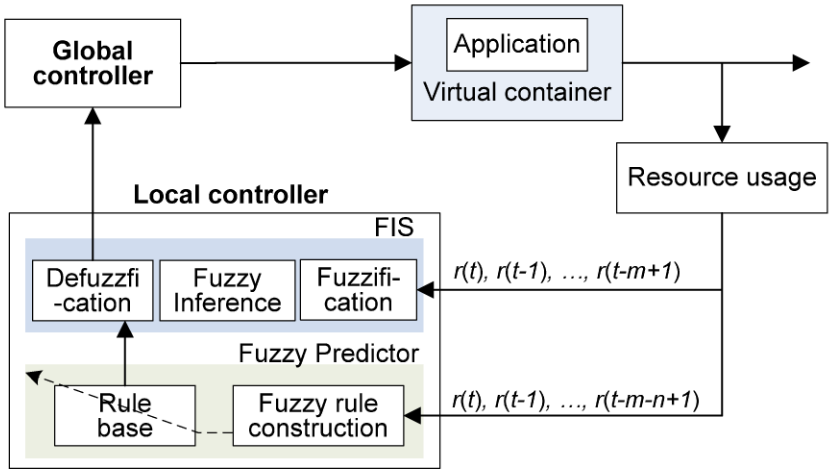


Fig. 4 Fuzzy prediction functions in a local controller

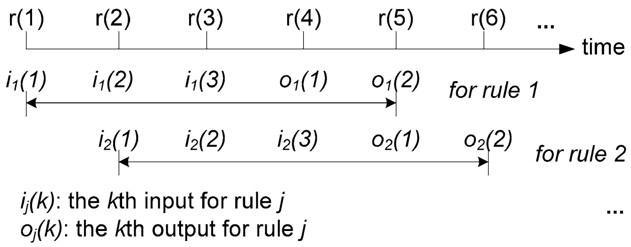

r(1) r(2) r(3) r(4) r(5) r(6) ...
time
i1(1) i1(2) i1(3) o1(1) o1(2) for rule 1
i2(1) i2(2) i2(3) o2(1) o2(2) for rule 2
ij(k): the kth input for rule j
oj(k): the kth output for rule j
...

Fig. 5 An example of three-input two-output data sequences for fuzzy rule construction

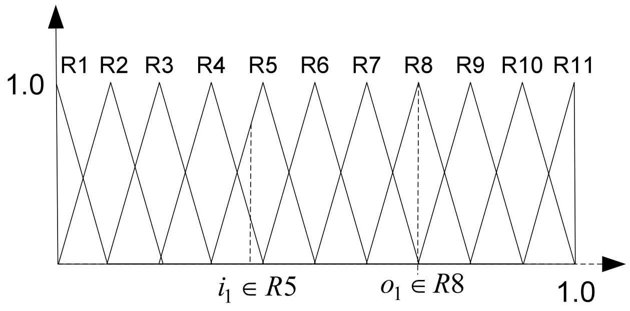


Fig. 6 An example of dividing the input/output space into 11 fuzzy domains and the corresponding membership functions

Fuzzy rule construction The fuzzy prediction system, illustrated in Fig. 4, has some components that are similar to those used in the fuzzy modeling approach. The fuzzy rules representing a mapping from input space to output space are generated from the monitored data and stored in the rule base. The fuzzy inference system processes the learned fuzzy rules to forecast future resource demands based on the current system observations. Let $r(t)$ $(t = 1, 2, 3, \ldots)$ be the measured resource usage at sampling time t. The problem addressed by the fuzzy prediction system can be formulated as: at time t, given the latest m measurements $r(t), r(t-1), \ldots, r(t-m+1)$ as the inputs, determine the resource needs at future times $r(t+1), r(t+2), \ldots, r(t+n)$ as the outputs (m and n are the number of inputs and outputs for a fuzzy rule, respectively).

A fuzzy rule is generated from an input-output data pair, whose components are subsequences of the successive resource usage measurements with the input subsequence preceding the output subsequences. Figure 5 shows an example of three-input two-output ($m = 3$ and $n = 2$ in this case) fuzzy rules. To translate input-output pairs into fuzzy rules, the first step is to divide the input and output spaces into subdomains representing by fuzzy sets. Assuming that the input and output ranges are normalized to [0, 1], each space is divided into $2N + 1$ domains, denoted by $R_1, R_2, \ldots, R_{2N+1}$, each assigned a fuzzy membership function. Figure 6 gives an example where the input/output space is divided into 11 fuzzy sets ($N = 5$ is used in our prototype) with triangular membership functions.

The next step is to assign a given data point to the fuzzy set with the highest membership degree using the membership functions described above. For example, input i_1 is considered to be R_5 and output o_1 is considered to be R_8 in Fig. 6. Finally, a fuzzy rule is constructed from a pair of input-output data as follows,

$$\text{Rule } i\text{: IF } i_1 \text{ is } R_{i_1} \text{ and } i_2 \text{ is } R_{i_2}, \ldots, \text{ and } i_m \text{ is } R_{i_m},$$
$$\text{THEN } o_1 \text{ is } R_{o_1}, \text{ and } o_2 \text{ is } R_{o_2}, \ldots, \text{ and } o_n \text{ is } R_{o_n}.$$

Therefore, every sequence of $m + n$ consecutive resource usage measurements can be used to generate a fuzzy rule which maps the input space $(i_1, i_2, \ldots, i_m)$ representing the previous system state to the output space $(o_1, o_2, \ldots, o_n)$ representing the more recent or current state.

Fuzzy rule update A fuzzy rule is generated at every sampling time and each rule with m-input and n-output represents a point in the $(m + n)$-dimensional rule space. If every rule is stored in the rule base the memory requirements will

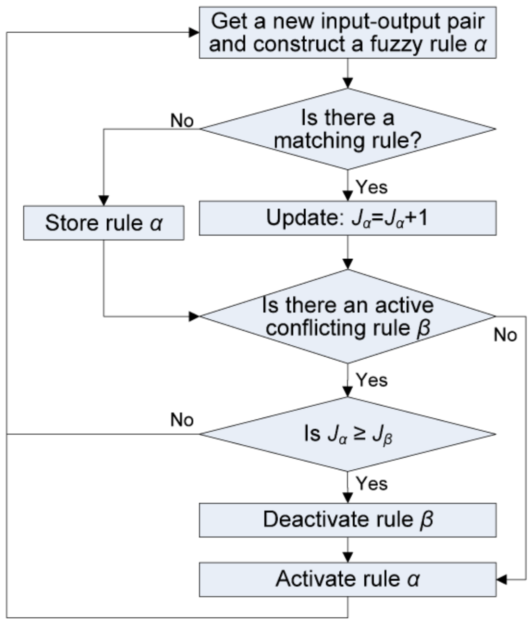

Get a new input-output pair and construct a fuzzy rule α
Is there a matching rule?
No
Yes
Store rule α
Update: Jα=Jα+1
Is there an active conflicting rule β
No
Yes
No
Is Jα ≥ Jβ
Yes
Deactivate rule β
Activate rule α

Fig. 7 The fuzzy-rule updating procedure

be excessive, and it is probable that there would be conflicting rules which have the same IF part but a different THEN part. The first problem is solved by partitioning input-output spaces into a finite number of domains as described above so that at most one rule is stored in the rule base for each domain. The number of rules increases as new input-output data are collected, but it never exceeds the maximum number of domains partitioned in the rule space.

To overcome the second problem, when updating the rule base a reliability index is computed for each rule as $J_i =$ the number of occurrences of rule i. Whenever a rule is generated, the system scans all the rules stored in the rule base. If there is a matching rule (i.e., a rule in the same domain), the value of J is increased by 1. Otherwise, the new rule is added to the rule base and J is initialized to 1. Figure 7 illustrates the procedure for updating rules. If there exist conflicting rules, which one takes effect is determined by the value of the reliability index. The rule with the highest reliability index is activated, indicating that the active fuzzy rule appears more frequently than the other conflicting rules. If conflicting rules have the same value of reliability index, the one that appeared most recently is activated.

Fuzzy inference Given the latest resource usage measurements as inputs, as Fig. 4 shows, the fuzzy inference engine processes the active fuzzy rules in the rule base to determine the outputs which consist of the future resource usage. Initially, there is no rule in the rule base. After the first $m + n$ measurements are obtained, the first rule is generated and stored in the rule base. Afterwards, at each sampling point, a rule is constructed and the rule base is updated following the updating procedure illustrated in Fig. 7. This updating procedure makes the proposed fuzzy prediction capable of self-learning the resource usage behavior of the managed virtual container.

Compared with the fuzzy-modeling approach, both methods essentially "learn" from the input and output history to build a mapping and can adaptively update the mapping when new data are available so that it can reflect system changes very quickly. No prior knowledge or mathematical model of the system is required and they both are a one-pass build-up procedure that does not need iterative time-consuming training. The difference between the two approaches is that the fuzzy modeling approach maps workload to resource consumption, while the fuzzy prediction maps the observations of recent resource usage to the future resource needs.

4 Global resource controller

Each individual local controller tries to minimize the resource cost by only requesting the resources necessary for meeting the application SLA. The global controller receives requests for physical resources from the local controllers and allocates the resources among them as required. It seeks to make allocations that maximize the data center's profit, which is the revenue received by allocating the physical resources among virtual containers minus the penalties due to violations of resource SLA.

The global controller makes allocation decision based on the received requests and currently available resources in the data center. If the requested resources are allocated, the application providers are charged for the resources they receive. Otherwise, the data center has to pay certain penalties for the unsatisfied requests. The resource price and penalties specified in the resource SLA are negotiated between the data center owner and application providers. To simplify the problem, it is assumed that the revenue linearly increases with the amount of allocated resources and is bounded by the point of requested amount. The penalty also has a linear relationship with the amount of unsatisfied resources (shown in Fig. 8).

Without loss of generality, this paper considers a single resource type and a single allocation period. Suppose that K virtual containers are concurrently active in the data center. Let req_k denote the resources requested from virtual container k, and alc_k be the amount of resources granted to it by the global controller. The data center receives revenue of rev_k for every allocated resource unit over an allocation period. But if the global controller cannot satisfy the request req_k, the data center pays a penalty of pnl_k per unit of unmet resource demand, according to the resource SLA. Each resource allocation decision made by the

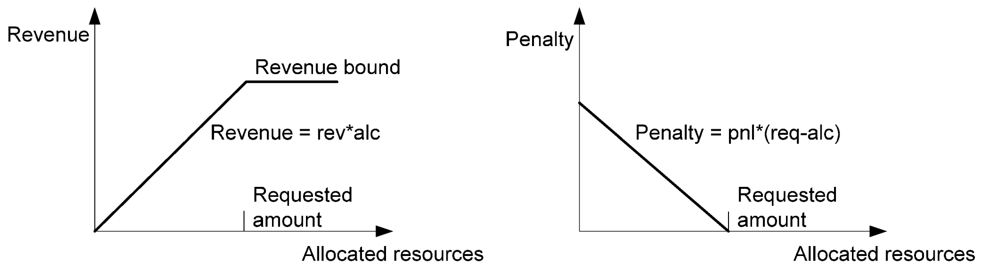


Fig. 8 The revenue and penalty function with respect to allocated resources

global controller is expressed as a resource allocation vector $(\mathrm{alc}_1, \mathrm{alc}_2, \ldots, \mathrm{alc}_K)$, and the total profit obtained by the data center for a time period is

$$\mathrm{profit}(\mathrm{alc}_1, \mathrm{alc}_2, \ldots, \mathrm{alc}_K) = \sum_{k=1}^{K} [\mathrm{rev}_k \mathrm{alc}_k - \mathrm{pnl}_k(\mathrm{req}_k - \mathrm{alc}_k)] \quad (1)$$

$$\text{s.t.} \quad 0 \le \mathrm{alc}_k \le \mathrm{req}_k, \quad C = \sum_{k=1}^{K} \mathrm{alc}_k \le A$$

where C is the total amount of resources allocated to the virtual containers, and A is the total resource capacity available in the data center. The profit equation can be rewritten as follows, for one allocation period t,

$$\mathrm{profit}(\mathrm{alc}_1, \mathrm{alc}_2, \ldots, \mathrm{alc}_K) = \sum_{k=1}^{K} (\mathrm{rev}_k + \mathrm{pnl}_k)\mathrm{alc}_k - \sum_{k=1}^{K} \mathrm{pnl}_k \mathrm{req}_k \quad (2)$$

$(\mathrm{rev}_k + \mathrm{pnl}_k)$ is defined as the *profit rate* for virtual container k. Assuming 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]).

To optimize profit over multiple time periods, the allocation decision has to be repeated. Equation (3) defines a cumulative profit which is the discounted sum under a discounting factor γ over a time horizon T. The factor models the fact that future profit is worth less than current profit because of the uncertainty in the future, for T allocation periods,

$$\sum_{t=1}^{T} \gamma^t \mathrm{profit}_t = \sum_{t=1}^{T} \sum_{k=1}^{K} \gamma^t [\mathrm{rev}_k \mathrm{alc}_{tk} - \mathrm{pnl}_k(\mathrm{req}_k - \mathrm{alc}_{tk})] \quad (3)$$

Based on the above profit model, a greedy strategy that maximizes the total profit for every period is still optimal because the allocation decision for current period does not affect the future periods.

5 Experimental evaluation

This section summarizes the experimental evaluation of the proposed two-level control system for dynamic resource allocation in a data center environment with time-varying workloads. Section 5.2 discusses the 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 (1) discussed in Sect. 4 when the global controller must allocate limited resources among several competing virtual containers.

5.1 Experimental setup

Data center testbed The testbed is deployed on a 16-CPU IBM x336 based cluster that provides virtual containers for applications, and several workload-generating clients. VMware ESX Server 3.0.1 is installed in each cluster node equipped with dual hyperthreaded Intel Xeon 3.2 GHz CPUs and 4 GB memory. Virtual machines are created on the servers and used as virtual containers to host applications. The clients are placed on VMware-Server-1.0.0-based virtual machines, hosted on another cluster of 32 dual-2.4 GHz hyperthreaded Intel Xeon nodes. Web-based workloads are generated by the clients and issued to the applications across a Gigabit Ethernet network.

Application and workload The Java Pet Store [26] was chosen to represent a typical e-business application. It 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. Synthetic HTTP workloads that mimic the key aspects of real-world workloads are created with various client sessions issued 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 are adjusted to vary a session's workload on the application: the user's 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) to static requests (e.g., browse static Web pages and view images) can be varied in order to change the workload characteristics. A Perl program was 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 [25] consist of about 1.3 million requests made to the World Cup Web site between April 30, 1998 and July 26, 1998. A real-time log replayer [24] is used to generate workloads using the trace.

Global/local 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 physical resources among its hosted virtual machines (e.g. setting the minimum, maximum and proportional resource shares of a virtual machine), and also provides 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 the ESX cluster 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 Local controller validation

The first set of experiments is to validate whether the local controller can accurately estimate resource needs using fuzzy-modeling and fuzzy-prediction approaches under dynamically changing workload.

5.2.1 Fuzzy-modeling approach

In the first experiment, the workload generator issues sessions to the Pet Store every 10 seconds. These sessions only contain requests for static Web content with a user think-time ranging from 0.1 to 1 second, and each session lasts around 1 minute. The generated sessions are divided into groups and the average user think-time of each group is decreasing, hence increasing the request rate among groups. This setup emulates the presence of bursts in real-world workloads. The entire experiment lasts for 4000 seconds.

Because the workloads used in this experiment consist of only static Web-content requests, the CPU usage is highly

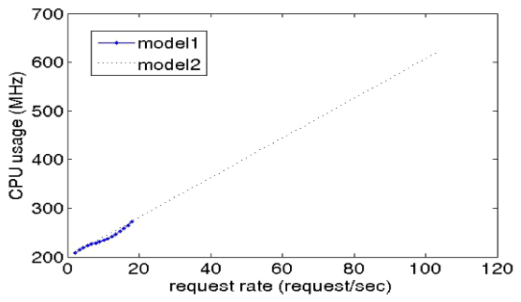


Fig. 9 Fuzzy models learned from the workload with static Web requests at the beginning (model1) and the end (model2) of the experiment

correlated with the request rate, which is then used as the only metric to characterize the workload. In this case, the input and output to fuzzy modeling are the request rate and CPU usage measurements. The first 50 pairs of data points collected from the sensors are used to initialize the learning of the fuzzy model. Afterwards, the model is continuously updated every 200 seconds (during which 10 new data points become available from the sensors). Figure 9 illustrates the models learned from the monitored data 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 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 (3.2 GHz) in order to obtain the ideal throughput for the same workload.

The throughputs from these two runs are compared in Fig. 10, 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. This confirms 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.

In the second experiment, the workloads are generated similarly to the previous one, except that requests for dynamic Web content are also considered. Every group of sessions differs not only in the request rate but also in the pro-

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

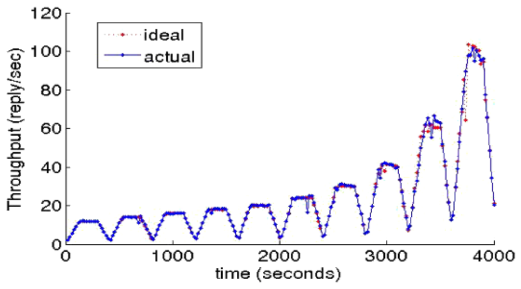


Fig. 10 Comparison of the throughput achieved by using local controller and the ideal throughput for the workload with static Web requests

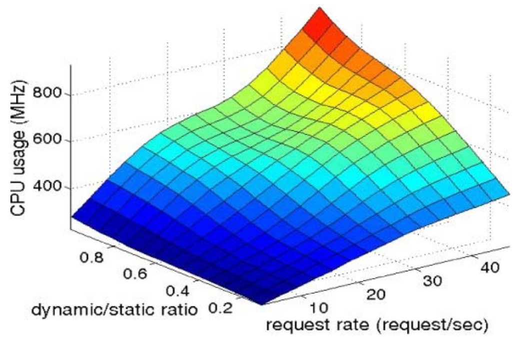


Fig. 11 The surface of the fuzzy model learned from the workload with dynamic Web requests

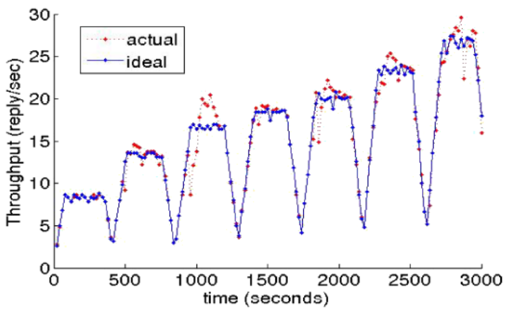


Fig. 12 Comparison of the throughput achieved by using local controller and the ideal throughput for the workload with dynamic Web requests

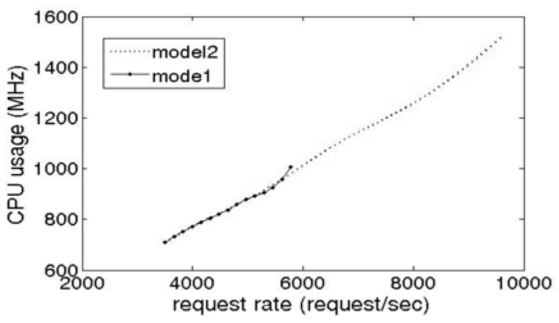


Fig. 13 Fuzzy model learned from the trace-based workload at the beginning (model1) and the end (model2) of the experiment

portion 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 local controller still uses request rate as the only metric for representing workload, the resulting model cannot truly describe 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 11 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 are well-suited for learning non-linear and complex relationships.

Figure 12 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 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.

In the third experiment, the 1998 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 13 illustrates the model learned at the beginning and the end of the experiment. Figure 14 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 effec-

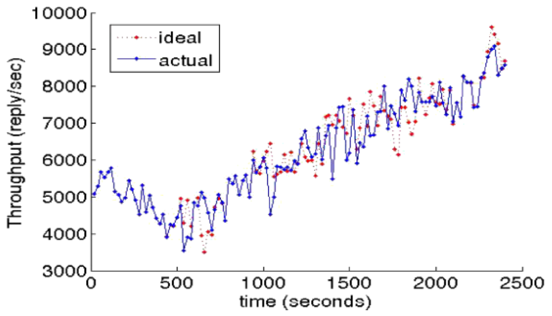


Fig. 14 Comparison of the throughput achieved by using local controller and the ideal throughput for the trace-based workload

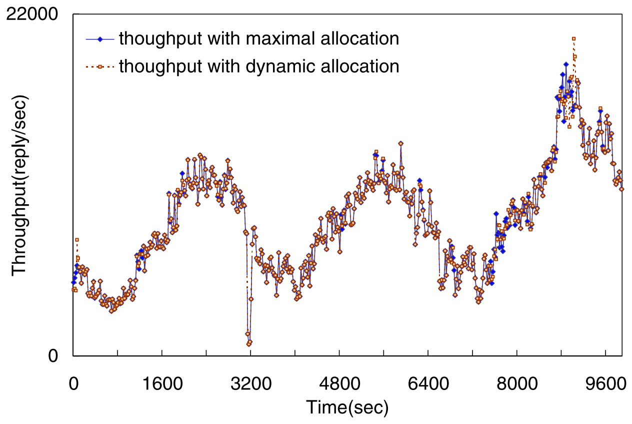


Fig. 15 Comparison of the throughput achieved by dynamic allocation using fuzzy prediction approach and the ideal throughput with maximal allocation

tiveness of the fuzzy-modeling approach under real-world workloads. The dynamic allocation uses less than 30% of the CPU cycles used by a static approach that allocates maximum CPU fraction based on the highest workload.

5.2.2 *Fuzzy-prediction approach*

Similar to the previous experiment, three days of web traces from World Cup 98 Web site are chosen to generate workload and played at 24X speedup to reduce experiment duration. During the experiment, only the CPU utilization of the virtual container is measured and fed to local controller every 20 seconds. The local controller applies the proposed fuzzy-prediction approach to estimate resource needs for each allocation interval (set to one minute in the experiment).

The first 50 measurements are used to initialize the fuzzy rules. Then the rule base is updated whenever the new CPU usage measurement is available (every 20 seconds). For every minute, the local controller estimates the CPU usage for the next minute based on the fuzzy rules learned from the pervious observations and then sends requests to the global controller. The global controller adjusts the CPU allocation

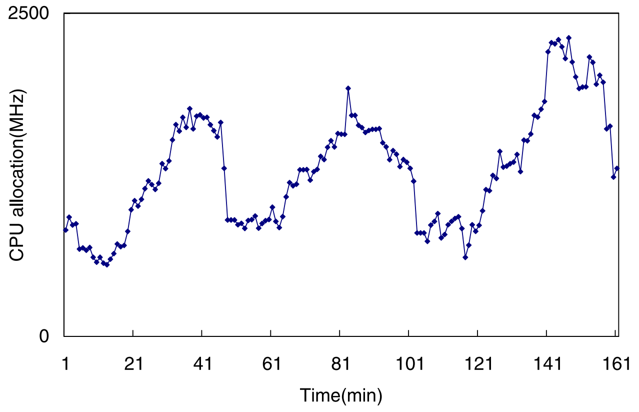


Fig. 16 The CPU allocated to the virtual container by using local and global controller

as required. Figure 15 shows the resulting throughput for the trace-workload by using fuzzy prediction and the throughput achieved by using maximal allocation (3.2 GHz). The results are very close and the differences between them are less than 1% on average. Figure 16 plots the CPU allocated to the virtual container during the experiment and about 44% resources can be saved using the dynamic allocation compared with maximal allocation.

Comparing to the fuzzy-modeling approach, fuzzy prediction can produce accurate short-term resource estimation for local controllers. Fuzzy modeling applies clustering techniques to provide concise data presentation, resulting in smaller rule base (less than ten fuzzy rules during experiments) than the one generated from the fuzzy-prediction approach (about several tens of fuzzy rules in the experiments).

5.3 Global controller validation

The last set of experiments investigates the allocation decisions of the global controller among multiple virtual containers. Two virtual containers (VC1, VC2) running on the same server node compete for the available CPU cycles (restricted to 1 GHz in the experiment). Both containers host the Java Pet Store application and two clients issue requests for static Web content to them. 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 local controllers of these two containers employ the fuzzy modeling approach to dynamically estimate their CPU demands for the workloads. The amounts of resources requested by the two local controllers during the experiment are plotted in Fig. 17. The local controller of VC1 requests around 500 MHz of CPU throughout the entire experiment; while VC2 increases its CPU request from about 200 MHz to more than 800 MHz as its workload grows.

When the CPU needed by VC2 goes beyond 500 MHz, the global controller responds to the resource shortage by

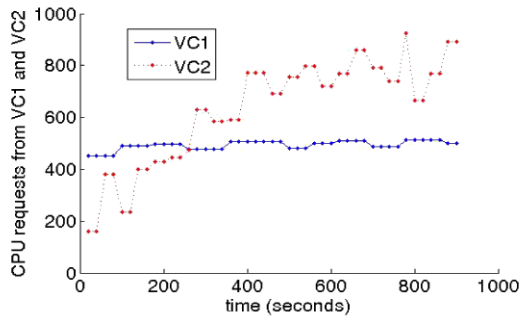


Fig. 17 CPU requests from two virtual containers (VC1, VC2) that share limited resources

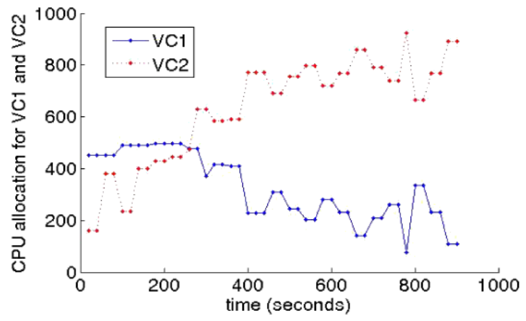


Fig. 18 CPU allocation that favors VC2

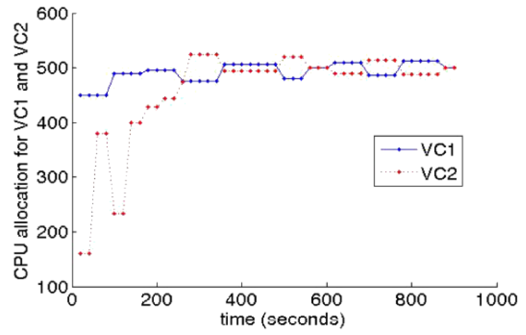


Fig. 19 CPU allocation that favors VC1

reducing the CPU allocation to VC1 or VC2 or both. The allocation policy of the global controller is to maximize its profit by applying the greedy algorithm discussed in Sect. 4. Two 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 18 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 a resource shortage when the global controller cannot allocate enough resources for both containers (Fig. 19). The greedy strategy to resource allocation using the simplified profit model can be theoretically proved to be the optimal solution to maximize the total profit.

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 [23] 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 [19, 22], 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, 20] 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.

Reinforcement learning (RL): Tesauro [16] proposed to use reinforcement learning for autonomic resource allocation. Compared with our fuzzy-logic-based approaches, both can automatically learn from past experiences without an explicit performance model. However, RL usually uses lookup table to store the information it obtained from training data. The size of table increases exponentially with the number of state variables, causing the scalability issue. Fuzzy rule based system keeps its knowledge more efficiently in the form of fuzzy rules and fuzzy membership functions. RL may also have a long training time due to the absence of domain knowledge or good heuristics, while the construction of fuzzy rule base in our approach does not require time-consuming training. In [17], the authors proposed to use a hybrid RL method combining RL and queuing models, in which RL trains offline on data collected while a queuing model policy controls the system to avoid performance degradation in live online training.

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 in this paper 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 "just enough" 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-logic-based approaches provide 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 can efficiently model the nonlinear system with dynamically changing operating conditions.
- The resource management process is automatic without any human intervention. The fuzzy rules are automatically learned from online monitoring data and the knowledge base is updated continuously as new data arrives, enabling the system to capture transient or unexpected workload changes.

7 Conclusions and future work

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. At the application level, to enable the local controller to accurately estimate the resource demands for different workloads, two fuzzy logic based methods—fuzzy modeling and fuzzy prediction—are proposed to guide resource allocation based on online measurements. Both approaches have an adaptive learning ability, thus requiring no prior knowledge about the system or the application. Specifically, the fuzzy modeling approach characterizes the mapping from the workloads and the corresponding resource requirements, while the fuzzy prediction builds a mapping from current resource usage to future resource needs. In this context, "adaptive" means that the mapping can be easily updated when new information is available in order to adapt to the system changes and reflect the most recent system conditions. The global controller at the resource-pool level tries to find the optimal resource allocation based on the proposed profit model, towards maximizing the total profit of the data center.

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.

Future papers will consider the cost of virtual machine migration (i.e., reassignment of a virtual machine to another physical host due to resource shortage in the current host). The profit model described in (3) does not include the migration cost (i.e., the cost is assumed to be zero). This assumption is reasonable in the scenario where the data center's physical resources reside in one location and they have adequate network bandwidth so that the migration could take place in less than a few seconds. However, in the case of scale-out data centers where the migrations may happen between different locations, the migration delay could be significant. The cost due to the resource overhead and reallocation delay should be incorporated into the profit model, which makes the profit optimization much harder. The global controller has to determine not only the quantity of physical resources allocated to each virtual container, but also the location of those virtual containers (i.e., which physical server is assigned to host the virtual container) at each allocation period. The global controller's allocation vector extends to $((\mathrm{alc}_{t1};\mathrm{loc}_{t1}),(\mathrm{alc}_{t2};\mathrm{loc}_{t2}),\ldots,(\mathrm{alc}_{tK};\mathrm{loc}_{tK}))$ in which alc_{tk} and loc_{tk} represent the amount of the physical resources assigned to the kth virtual container and its location at time t. If loc_k is changed from time t to $t+1$, it indicates that the virtual container k is moved from one host to another. The cost of resource overhead and potential performance loss caused by migration should be deducted from the data center's profit which is then redefined as, for one allocation period t,

$$\begin{aligned}&\mathrm{profit}_t((\mathrm{alc}_{t1};\mathrm{loc}_{t1}),(\mathrm{alc}_{t2};\mathrm{loc}_{t2}),\ldots,(\mathrm{alc}_{tK};\mathrm{loc}_{tK}))\\ &\quad=\sum_{k=1}^{K}[\mathrm{rev}_{tk}\mathrm{alc}_{tk}-\mathrm{pnl}_k(\mathrm{req}_k-\mathrm{alc}_k)\\ &\qquad-\mathrm{migr}(\mathrm{loc}_{tk}-\mathrm{loc}_{(t-1)k})]\end{aligned} \tag{4}$$

$$\mathrm{migr}(\mathrm{loc}_{tk}-\mathrm{loc}_{(t-1)k})=\begin{cases}0 & \text{if } \mathrm{loc}_{tk}=\mathrm{loc}_{(t-1)k}\\ \mathrm{migr} & \text{otherwise}\end{cases}$$

where migr is the cost of moving a virtual container from one host to another. Clearly, the assertion that an allocation decision for a given period won't affect the future profit is no longer valid and the global controller has to make better decisions than merely maximizing its profit for the current period. Considering the effect on future periods, the cumulative profit over a time horizon T defined in (3) is rewritten as follows, for T allocation periods,

$$\begin{aligned}\sum_{t=1}^{T}\gamma^t\mathrm{profit}_t&=\sum_{t=1}^{T}\sum_{k=1}^{K}\gamma^t[\mathrm{rev}_k\mathrm{alc}_{tk}-\mathrm{pnl}_k(\mathrm{req}_k-\mathrm{alc}_{tk})\\ &\qquad-\mathrm{migr}(\mathrm{loc}_{tk}-\mathrm{loc}_{(t-1)k})]\end{aligned} \tag{5}$$

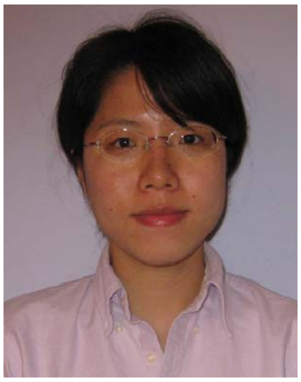
To maximize the profit using the above model is much more complicated than using (3) because each allocation decision will affect the future decisions and the resulting profit as well. Moreover, the above profit formula requires the information about the future resource requests alc_{tk} ($t=2,\ldots,T$), indicating that the local controllers have to provide n-step ahead forecast of resource needs. How to incorporate forecasting component into the local controllers and how to solve the optimization problem (5) in the global controller are the subject of ongoing work.

Acknowledgements This work was supported in part by an Intel grant (IT R Council), National Science Foundation grant CNS-0540304, the BellSouth Foundation, equipment awards from DURIP and IBM and software donations from VMware. 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.

References

1. Abdelzaher, T., Shin, K.G., Bhatti, N.: Performance guarantees for web server end-systems: a control-theoretical approach. In: IEEE Trans. Parallel Distrib. Syst. **13**(1) (2002)
2. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Proc. of the ACM Symposium on Operating Systems Principles (SOSP), October 2003
3. Bennani, M.N., Menascé, D.A.: Resource allocation for autonomic data centers using analytic performance models. In: Proc. of 2nd IEEE International Conference on Autonomic Computing (ICAC) (2005)
4. Chandra, A., Gong, W., Shenoy, P.: Dynamic resource allocation for shared data centers using online measurements. In: Proc. of IEEE International Workshop on Quality of Service (IWQoS), June 2003
5. Chiu, S.: Fuzzy model identification based on cluster estimation. J. Intell. Fuzzy Syst. **2**(3) (1994)
6. Diao, Y., Hellerstein, J.L., Parekh, S.: Using fuzzy control to maximize profits in service level management. IBM Syst. J. **41**(3) (2002)
7. Dike, J.: A user-mode port of the Linux kernel. In: Proc. of 4th Annual Linux Showcase & Conference (ALS 2000) (2000)
8. Doyle, R., Chase, J., Asad, O., Jin, W., Vahdat, A.: Model-based resource provisioning in a web service utility. In: Proc. of the 4th Conference on USENIX Symposium on Internet Technologies and Systems, March 2003
9. Liu, X., Zhu, X., Singhal, S., Arlitt, M.: Adaptive entitlement control of resource containers on shared servers. In: Proc. of 9th IFIP/IEEE International Symposium on Integrated Network Management, May 2005
10. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. Wiley, New York (1990)
11. Mosberger, D., Jin, T.: httperf: a tool for measuring web server performance. Perform. Eval. Rev. **26**(3) (1998)
12. Rolia, J., Cherkasova, L., McCarthy, C.: Configuring workload manager control parameters for resource pools. In: Proc. of 10th IEEE/IFIP Network Operations and Management Symposium (NOMS), April 2006
13. Sha, L., Liu, X., Lu, Y., Abdelzaher, T.: Queueing model based network server performance control. In: Proc. of the 23rd IEEE Real-Time Systems Symposium (RTSS'02) (2002)
14. Sugeno, M., Yasukawa, T.: A fuzzy-logic-based approach to qualitative modeling. IEEE Trans. Fuzzy Syst. **1**(1) (1993)
15. Sugerman, J., Venkitachalam, G., Lim, B.: Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In: Proc. of 2001 USENIX Annual Technical Conference, June 2001
16. Tesauro, G.: Online resource allocation using decompositional reinforcement learning. In: Proc. of the Twentieth National Conference on Artificial Intelligence (AAAI-05), July 2005
17. Tesauro, G., Jong, N., Das, R., Bennani, M.: On the use of hybrid reinforcement learning for autonomic resource allocation. Clust. Comput. **10**(3) (2007)
18. Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., Tantawi, A.: An analytical model for multi-tier internet services and its applications. In: Proc. of ACM Sigmetrics Conference (SIGMETRICS), Jun 2005
19. Wang, Z., Zhu, X., Singhal, S.: Utilization and SLO-based control for dynamic sizing of resource partitions. In: Proc. of 16th IFIP/IEEE Distributed Systems: Operations and Management (DSOM), October 2005
20. Xu, W., Zhu, X., Singhal, S., Wang, Z.: Predictive control for dynamic resource allocation in enterprise data centers. In: Proc. of 2006 IEEE/IFIP Network Operations & Management Symposium, April 2006

21. Zadeh, L.A.: Fuzzy sets. Inf. Control **8**(3), 338–353 (1965)
22. Zhu, X., Wang, Z., Singhal, S.: Utility-driven workload management using nested control design. In: Proc. of American Control Conference (ACC), June 2006
23. HP-UX Workload Manager, http://docs.hp.com/en/5990-8153/ch05s12.html
24. See http://www.cs.virginia.edu/~rz5b/software/software.htm
25. See http://ita.ee.lbl.gov/html/contrib/WorldCup.html
26. See https://blueprints.dev.java.net/petstore/



Jing Xu is a PhD candidate in the Department of Electrical and Computer Engineering and a member of the Advance Computing and Information Systems Laboratory at the University of Florida. She received the degree of B.E. from University of Science and Technology of China in 2002. Her research interests are in the areas of distributed/grid computing, autonomic computing, and virtualization.

Ming Zhao is a PhD candidate in the Department of Electrical and Computer Engineering and a member of the Advance Computing and Information Systems Laboratory at the University of Florida. He received the degrees of BE and ME from Tsinghua University. His research interests are in the areas of distributed computing, autonomic computing, and virtualization. He is a student member of IEEE and ACM.

José Fortes (S'80–M'83–SM'92–F'99) received the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, in 1984. From 1984 to 2001, he was on the faculty of the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN. In 2001, he joined both the Department of Electrical and Computer Engineering and the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, as a Professor and BellSouth Eminent Scholar. At the University of Florida he founded and directs the Advanced Computing and Information Systems (ACIS) laboratory and the NSF Industry-University Cooperative Center on Autonomic Computing. His current research interests are in the areas of distributed computing and autonomic computing. Dr. Fortes was a Distinguished Visitor of the IEEE Computer Society from 1991 to 1995.

Robert Carpenter joined Intel in 1997 after a 20-year career as an attorney, retiring as district attorney of Greene County, NY in 1996. For many years, he taught mathematics at Bard College and later law at Albany Law School. After serving as the Business Process Automation architect within IT, he was the IT architect for SOE [Service Oriented Enterprise] working closely with the platform groups sharing time between Beta SI and IT Research. Robert Carpenter was deceased on November 2nd, 2007.

Mazin Yousif is a Chief Systems Architect at Numonyx Corporation. Prior to that, Mazin was a Principal Engineer and Director in the Corporate Technology Group of Intel Corporation in Hillsboro, OR, where he led a team that focused on platform provisioning & virtualization to enable platform autonomics & Capacity on Demand (CoD) in a Scale-out Environment. Mazin was also one the principal architects defining the InfiniBand Architecture, during which he chaired the Management Working Group (MgtWG) in the InfiniBand Trade Association (IBTA). From 1993 to 1995, he was an assistant professor in the Computer Science Department at Louisiana Tech University. Mazin worked for IBM's xSeries Server Division in Research Triangle Park (RTP), NC, from 1995–2000. Mazin also served as Adjunct Professor at several universities including the Oregon Graduate Institute (OGI), Duke University and North Carolina State University.
Mazin finished his Master and Ph.D. degrees from the Pennsylvania State University in 1987 and 1992, respectively.
Mazin's research interests include Computer Architecture, Autonomic and Grid Computing, Clustered Architectures, workload characterization and workload-driven platform architectures He has published 50+ articles in his areas of research. Mazin chaired the program committee of several conferences and workshops and has been in the program committees of many others. Mazin is in the advisory board of the Journal of Pervasive Computing and Communications (JPCC), and is an editor in Cluster Computing, the Journal of Networks, Software Tools and Applications. He is also in the Advisory board of ERCIM. Mazin is an IEEE senior member.