Application of Fuzzy Control Theory in Resource Management of a Consolidated Server

Sho Niboshi and Hitoshi Oi
Computer Architecture and Operating Systems Group,
The University of Aizu
Aizu-Wakamatsu, JAPAN
Email: {nibo,oi}@oslab.biz

***Abstract*—A virtualized system incorporates multiple systems into a single physical computer as virtual domains. A lot of data centers and server systems have been organized using virtualization technology to merge several computer systems. On the shared system, resource manager is the key affecting the performance. However, the resource management in current systems does not provide accurate resource allocation, because it only utilizes information from virtual machines and disregards the state of running applications.**

The paper demonstrates the CPU resource controller taking the state of application as inputs to produce the minimum resource retaining application performance in acceptable level. In particular, it employs two-layered controller. The first layer controller makes resource request based on the relationship between the state and resource demand of each application, modeled by fuzzy control theory. This approach is efficient to represent resource allocation model since fuzzy control theory deals imprecise and uncertain problems. The second layer controller adjusts the requests to the system capacity and builds the layout of resource capacity based on the relative Quality of Service performances between applications. For the separation of resource, common resource controller imposes a hard limit on the amount of resource a given domain can consume. The controller allocates resource with most effective capacity configuration. Under certain specified conditions, the controller does not set the capacities and allows domains to use the free time if the resource is idle. This results in eliminating unused resources and achieves relative high resource usage.

Finally, the resource controller is evaluated with a virtualized system, and its advantages over conventional resource allocation methods are shown.

I. Introduction

Virtualization is one of the technologies to merge multiple computer systems into a single physical computer, and the technique is widely used to reorganize data centers and server systems with a small number of computers. Generally speaking, on average, servers utilize a small fraction of their total capacity, with Windows servers running at 8% to 12% of the capacity and UNIX servers at 25% to 30% of their capacity [1] [2]. In such situation, a virtualized system allows multiple services to be hosted by a single computer, sharing the unused resources of existing platform, thereby providing better resource utilization.

On this shared system, resource allocation plays an important role in determining the system performance, because it controls resource assignment to each domain. However, conventional resource allocation methods have some issues. One of them is to perform the allocation without accounting for the state of running applications. Although virtualization is typically dedicated to one or a few specific applications and desirable to ensure that the application can achieve its performance goal, conventional approaches to provisioning resources to virtual domains only utilize the state of the domain instead of application itself. The allocation without considering applications leads to un-balanced performance distribution of the consolidated system because if applications have different performance characteristics, the performance on one application is higher than the performance on other applications, despite same allocation amounts.

This paper describes an adaptive resource controller which maximizes the performance of virtualized systems through the effective resource allocation of virtual components. The methodology, called AARM (Application-Aware Resource Management) for the sake of brevity, takes application state, e.g. Quality of Service (QoS) and resource utilization, into account to identify resource demand.

Because the relationship between an application's state and its required resources is still complex and empirical, to represent the relationship fuzzy control theory was chosen for the control logic. The control theory keeps a simple and easy to understand form, also makes it easy to modify the rule if you want to change the target of control. A rule-based controller also has the advantage of producing output quickly. These two advantages are useful to model the above relation.

The AARM controller is tested on a virtualized system having two domains: one is a mail server and the other is a Java application server (Java server).

While the performance of a computer system is affected by many factors, such as memory and I/O of disks and network, this paper focuses on the CPU, as it can be considered the bottleneck resource for practical purposes.

This paper is organized as follows, Section II presents background information. In Section III, the method of the AARM is described and Section IV models the AARM controller for the target system. Section V presents the experimental results obtained with the AARM controller. Section VI gives an overview of related work and the paper is summarized in Section VII.

Annual International Conference on Cloud Computing and Virtualization (CCV 2010).
Edited by Prof. Gagan Agrawal.
Copyright © CCV 2010 & GSTF.
ISBN: 978-981-08-5864-3.
doi:10.5176/978-981-08-5837-7_204

II. Background

A. Xen

Xen hypervisor [3], called a Virtual Machine Monitor (VMM), is a software layer between computer hardware and the operating systems running on Virtual Machines (VMs). Xen composes two types of virtual domains, Domain 0 and Guest Domain. Xen refers to Domain 0 (*dom0*) as a controller domain having elevated privileges. This domain typically has two objectives. First, the domain works as driver domain. Since Xen hypervisor has no device drivers, *dom0* must handle devices and provide an interface to hardware for other domains. This is necessary in order to create a single point of access to hardware, as access from multiple operating systems is not supported. Another task on *dom0* is the management of other virtual machines, such as starting and stopping VM. While this is actually implemented by the VMM software, *dom0* provides its interface tools. Guest domain (*domU*) is an unprivileged domain. One specific feature of Xen VMM is its use of paravirtualization, which uses modified versions of the OS to achieve low overhead. Since the OS in *domU* is running on VMM instead of common hardware, the kernel must be modified to allow it to run on VMM.

The VMM is responsible for assigning CPU time to running guests. Credit scheduler is the default scheduler [4] for Xen hypervisor and it operates in work-conserving mode by default. This means that the scheduler guarantees that there is no idle CPU as long as the system has a runnable domain. The configurable parameter *cap* forces the scheduler to be non-work-conserving. In this mode, domains can use CPU time up to their credit limit only, and will not exceed it even if a CPU is idle. Because of these differences, work-conserving mode typically has a higher CPU utilization than non-work-conserving mode. This paper uses non-work-conserving mode to control the allocation of CPU time for each domain. However, work-conserving mode is also be used to achieve high CPU utilization. The AARM controller selects fittest mode under the given circumstances.

B. Fuzzy Control Theory

Fuzzy control theory [5] [6] is a control theory to treat problems with vague, unclear, or complex values and relationships, and it is a useful tool to model non-linear relationships. One of the advantages of the theory is that it does not require complex mathematical formulas to characterize a relationship. Instead of a mathematical model, it is described by fuzzy rules, which are nothing more than simple rules of thumb based on human knowledge and experiences. Essential elements of this theory are fuzzy rules and membership functions.

The fuzzy rule is designed with the form "`IF <condition> THEN <actions>`", which is formulated in linguistic terms describing how the control system should behave. Since the information is often obtained from system experts, linguistic terms are important to represent information that is non-describable in mathematical values and formulas. As fuzzy rules are used to describe the behavior of the system, fuzzy sets are used to describe the state of the system. The members of a fuzzy set are usually formulated in linguistic terms, and have a degree of membership in the set, given by a membership function associated with the set.

III. Resource Controller

Main steps of the AARM are, 1) computes resource size for domains, 2) adjusts them to the system. These steps are executed by the controller in each layer. The first layer controller monitors the target applications, collects the application states and calculates the minimal resource size retaining the performance in the acceptable level. In our design, the computation is based on the collected application states. Second layer controller normalizes the requested resource percentages if the total requested is over the system capacity (e.g. 100%), and then assigns the normalized percentages to each domain until the end of the next controller interval.

A. QoS Controller

The QoS controller lives in each domain and dedicates the first step of resource allocation. The controller collects the application information and determines the resource needed to keep the application's performance around the acceptable level. The determination is produced through a fuzzy control model which characterizes the relationship between the current state and a resource demand. The relation is strongly dependent on the types of applications, because the effect of allocated resource size on the performance of an application may be completely different from one applications type to another. Fuzzy control theory helps to deal the model in simple way, adapting the issue to the type of application, thus making better decision of resource requirements. After computing resource requirements, the QoS controller sends it to the arbitration controller.

B. Arbitration Controller

The arbitration controller is placed on *dom0* and works for second step of resource allocation. The role is to receive requests from QoS controllers and perform the actual resource allocation to them. It adjusts the allocation requests to the capacity of the system and the resource capacity layout.

Since QoS controllers work independent of each other, their requests might exceed the capacity of the system. Thus, one of the jobs is to check if the total of the requests gets over the maximum capacity (e.g. 100% CPU time), and adjust the requests according to a ratio of requests.

Another job is select capacity layout. Commonly, the resource separation is performed by creating resource capacity which the domain cannot utilize resources over the amount. For example, if the system wants to distribute 50% of resource to each of two domains, it makes capacity of 50% to domains, then both domain can consume 50% of resources at maximum. The arbitration controller designs three types of capacity layout, which are Cap_{all}, Cap_{some} and Cap_{no}. Cap_{all} gives hard limits of resource consuming for all domains, Cap_{some}

Component	Specification
CPU	Athlon64 X2 (Dual Core) 2000 MHz
Memory	4GB
HDD	SATA 7200 rpm $\times 2$
OS	CentOS 5.2 (kernel 2.6.18)
VMM	Xen hypervisor 3.2.2

TABLE I
EVALUATION ENVIRONMENT. TABLE SHOWS THE SPECIFICATION OF THE TARGET SYSTEM.

has the capacity to one domain, and all domain have no resource limits in Cap_{no}. In most of cases, Cap_{no} gives better resource utilization than the other layouts, but it is difficult to provide isolated resource. To achieve both high resource utilization and resource separation, the arbitration controller selects efficient capacity layout from these three according to the current applications' states.

IV. MODELING OF AARM CONTROLLER

The AARM controller is evaluated on a virtualized system. The test environment is composed of three nodes: a target virtualized system, a mail client and a mail sink. The virtualized system is the target system, which includes the AARM controller and two virtual domains: one for the mail server application; and another running the Java server and client applications. The other two nodes (a mail client and a mail sink) are dedicated to evaluate the mail server. The system specification is shown in Table I. It has a dual-core CPU, 4GB main memory and two SATA hard disks. Xen Hypervisor is used for constructing the virtualized environment and CentOS runs on each domain. Each domain has two virtual CPUs which are not pinned into a specific physical CPU. Main memory is distributed through the domains, with 1GB to the mail server, 512MB to the Java server, and the remaining for *dom0*. One SATA hard disk is shared with all domains, and another disk is occupied by the mail server for user's message directories and mail spool directory.

In this system, SPECmail2001 and SPECjbb2005 are deployed to generate workloads. SPECmail2001 [7] is a standardized mail server benchmark published by Standard Performance Evaluation Corporation (SPEC). This benchmark is designed to measure the quality of a system acting as mail server based on the SMTP and POP3 protocols. SPECjbb2005 [8] is also a benchmark software developed by SPEC. SPECjbb2005 implements a 3-tier system modeling a wholesale company. The whole database tables and data records are held in the memory, and these are implemented as Java objects, thus SPECjbb2005 performs no I/O traffic to disk or network I/O. SPECjbb2005 (running locally) emulates and generates transaction traffic in the Java server. For the sake of simplicity, three levels of workloads are defined for both mail and Java servers: *wLow*, *wMedium* and *wHigh*. The definition of *wMedium* is a workload to utilize almost 50% of CPU time in average, and *wLow* is used to indicate a slightly lower workload than *wMedium*. The *wHigh* level represents a slightly higher workload than *wMedium*. The three workload levels are used for all experiments presented in the paper.

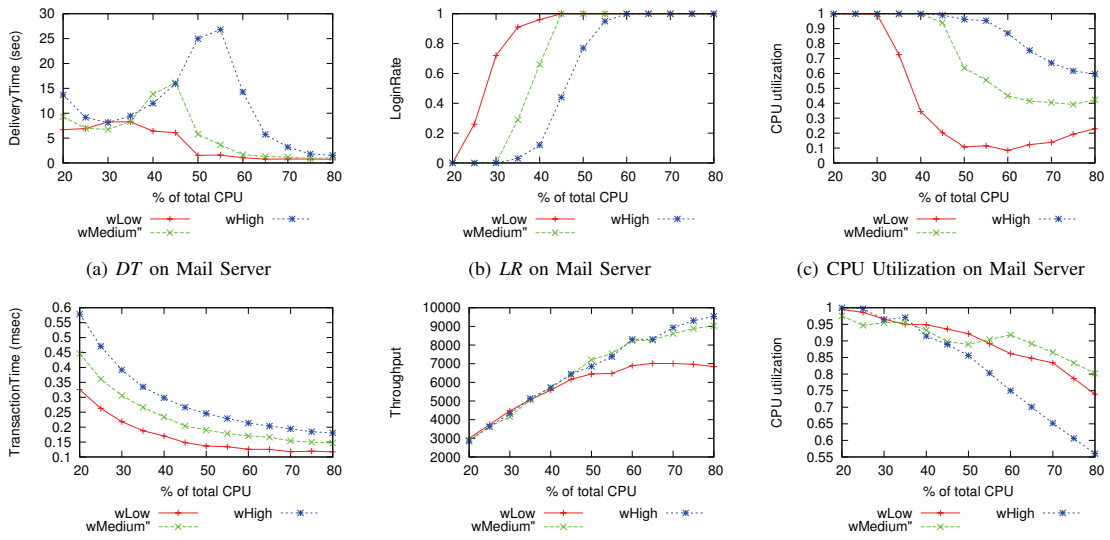
A. QoS indicator

The AARM controller computes resource size based on QoS indicators. In this subsection, we select the indicators identifying the application states on the domains of the target system.

On mail server, the performance is defined by two QoS indicators, *DeliveryTime* (*DT*) on SMTP server and *LoginRate* (*LR*) on POP3 server. *DT* is an average time that SMTP server takes to complete a message delivery. Figure 1 (a) shows *DT* under the varying the amount of allocated CPU time with defined three workloads (*wLow*, *wMedium* and *wHigh*). As expected, the maximum resource amount (80% of total CPU) provides the lowest *DT* through the experiment, and as given CPU time decreases, the message delivery takes long. However, once it peaks at the half of maximum allocation size, *DT* goes down even through the resource decreases. *LR* is the success rate of opening POP3 connection used by 'LOGIN' command in a second. Figure 1 (b) shows the relationship between *LR* and CPU capacity under the three workloads. The relation is simple. When the domain gets high CPU time, *LR* performs high success rate. As the amount of capacity decreases, the success rate drastically falls and it reaches to 0 under the low CPU allocations. From Figure 1 (a) and (b), we see that when *LR* starts its dropping from 1.0, SMTP server reduces *DT* than the peak. In the other words, in low CPU allocation, SMTP server improves its performance, while POP3 drops the service quality. When the system provides high CPU time to the mail server, both SMTP and POP3 server service incoming requests without problem. However, the mail server with low CPU provision forces POP3 server to reduce acceptable POP3 connections because the server does not have enough CPU time to run a lot of listen processes. As the number of accepted requests become small, its total required resource is decreased. Therefore SMTP server can utilize the left CPU time and it processes message deliveries in regular time. In fact, when *LR* falls to 0, *DT* increases again.

The performance on the Java server is defined by *TransactionTime* (*TT*). *TT* is an average time that the Java server takes to complete one transaction. Figure 1 (d) shows the relationship between the amount of allocated CPU time and *TT*. *TT* is completely depends on the CPU time and has monotonic decrease for the increasing resource.

In addition, the controller designs to use CPU resource utilization on the domain to produce more accurate resource decision in both mail and the Java server. This information helps to determine minimal resource maintaining the application performance in an acceptable level. For instance, the controller aims to maintain not only enough low utilization including safety margin to meet good performance, but also enough high to keep high resource utilization. Figure 1 (c) and (f) show the utilization of given CPU time in the mail and Java servers. We see that over-provisioning of CPU time just leads to the waste of this important system resource and



(a) *DT* on Mail Server (b) *LR* on Mail Server (c) CPU Utilization on Mail Server

(d) *TT* on Java Server (e) *Throughput* on Java Server (f) CPU Utilization on Java Server

Fig. 1. Performance effect for the amount of resource. Figures show the application states when CPU allocation varies from 20% to 80% in steps of 5% on each workload.

this phenomena is in the mail server (Figure 1 (c)) than in the Java server (Figure 1 (f)).

B. Modeling QoS Controller

In control interval i (which is 3 seconds), QoS controller computes a resource request for the next period $i+1$ by using a following formula.

$$Request_{i+1} = Consumption_i + \Delta_{cpu} \quad (1)$$

The request is the sum of current resource consumption with the output (Δ_{cpu}) of the fuzzy model.

In this control model, the output is represented by five linguistic values in the fuzzy model, which are NS = *NegativeSmall*, ZE = *Zero*, PS = *PositiveSmall*, PM = *PositiveMedium* and PL = *PositiveLarge*. Figure 2 (a) shows the membership functions of Δ_{cpu} and the resource request is computed by these functions.

The definition of linguistic values and membership functions for inputs data, i.e. QoS of application and the relative utilization of a domain, is processed through finding inflecion points in Figure 1. The figures show the relation between the allocated resource and the application state (Section IV-A). The inflecion points in these figures can be assumed that the greatly-changed points correspond to the boundaries between fuzzy memberships. In the paper, three linguistic values are used to represent the state of inputs, both QoS and utilization, which are *Low*, *Medium*, *High*.

In the mail server, two QoS indicators, *DT* and *LR*, and resource utilization are chosen to identify the application state. Figure 1 (a) illustrating the relationship between allocated resource and *DT* shows two inflecion points. First point places at $DT = 5$ and second point is at $DT = 15$. Figure 1 (b) also shows two changes of *LR* where *LR* equals 0.9 and 0.15. From the found inflecion points, the membership functions of *DT* and *LR* can be defined as Figure 2 (b) and (c) for each. Each figure illustrates three states (*Low*, *Medium* and *High*) and their membership functions. For example, Figure 2 (b) depicts the memberships of *DT* and it describes that the range of *Medium* state is 5 to 15. If *DT* is 10 then the value is completely *Medium*, however if *DT* is 7.5 the value includes two states, *Low* and *Medium*. The relation between resource utilization and allocated resource size in the mail server is illustrated in Figure 1 (c). The figure contains of two points changing the state at utilization equals 0.6 and 0.9. Assuming these points as the membership boundaries, the membership functions are defined as Figure 2 (d).

The acceptable level of performance in the mail server can be represented by the requirement: the *DT* is *Low* and *LR* is *High*. The fuzzy rule to satisfy the requirement is shown in Table II (a) and (b). *DT* behaves in different ways in the left and the right of the peak (Section IV-A). The controller also needs to produce different reaction according to the situation. For instance, Rule II (a) is reaction when *LR* = *High* (*DT* is in the right of the peak), otherwise Rule II (b) (*DT* is in the left of the peak) is used. In Rule II (a), the controller observes *DT* and utilization to make a resource request. The rule leads the amount of resource to satisfy the above requirement, *DT* is *Low* and *DT* is *High*, with *Medium* utilization which keeps both high resource usage and enough resource margin. Rule II (b) always produces largest resource allocation, because when the rule is required the resource on the mail server is completely under-provisioning and difficult to serve the incoming requests.

The Java server has one QoS indicator, *TT*, and resource

utilization to identify its state. Figure 1 (d) shows the relation between *TT* and the amount of resource allocation, but this figure does not have any inflecion points indicating the boundaries of the memberships. The Java server has *Throughput* QoS indicator which is the number of completed transactions per second. The parameter strongly depends on *TT* so the membership of *TT* is defined through the indicator. Figure 1 (e) illustrates *Throughput* under the amount of resource varying. Combining the two figures (Figure 1 (d) and (e)), the membership functions of *TT* are defined as Figure 2 (e). The definition of the membership functions for utilization is also produced by using Figure 1 (f). The figure shows the utilization varying and it has a inflecion points around utilization equals to 0.9 and the membership functions are defined as Figure 2 (f). Similar to the mail server, the controller regulates *TT* to *Low* with *Medium* utilization, thus the fuzzy rule can be described as Table II (c).

C. Modeling Arbitration Controller

The arbitration controller plays two roles. The first role is recalculating the resource size if the total requested size exceeds the capacity of the system. In this case, the controller calculates new resource allocations based on the relative request rate. For instance, following equation is used for the calculation on domain k in a virtualized system with n domains.

$$Allocation_k = Request_k / \sum_{j=1}^{n} Request_j \qquad (2)$$

The latter role is selecting the capacity layout (Cap_{all}, Cap_{some} or Cap_{no}) according to the state transition diagram illustrated in Figure 3. The state diagram starts at Cap_{no} with no resource restriction. From this state, the capacity layout is changed to Cap_{some} if the controller detects losing the balance of applications' QoS ($Cap_1 \neq Cap_2$). In this layout, the domain having higher QoS is set a resource capacity. The amount of the capacity corresponds to the output of the arbitration controller which is the size adjusted system capacity. Domain having lower QoS metric is not required to set the resource restriction, the domain can perform work-conversing. If the QoS balance is lost again in this layout, the layout is changed to Cap_{all} that both domain have the resource capacity. Furthermore, these capacities are cancelled when the controller detects that system can provides enough resource, which is case of the total of domain requests is under the system capacity ($Request_{total} <= 1$)

V. Experiment Results

The AARM controller aims to maintain the performance on each application with minimal allocated resources and keep the balance between the application performances. The experiments test how the controller achieves these two goals under dynamic workloads. In this paper, we experiment with the AARM controller in the virtualized system with two domain, mail and Java server, and two workload levels, which are *wMedium* and *wHigh*, are used. The experiments are designed

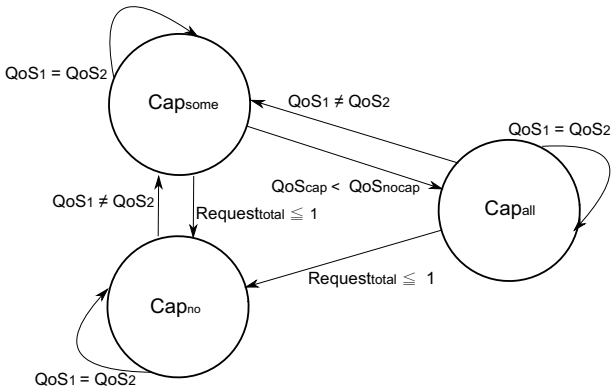


Fig. 3. Transition diagram of capacity layout. Figure shows the transition diagram of the layout with two domains which have QoS_1 and QoS_2 as QoS metric for each. The controller has three types of the capacity layouts; making capacity to no domains (Cap_{no}), to some domains (Cap_{some}) and to all domains (Cap_{all}).

(a) Mail Workload Increase (*MWI*)

Intervals (sec)	Mail	Java
1–399	*wMedium*	*wMedium*
400–1199	*wHigh*	*wMedium*
1200–1599	*wMedium*	*wMedium*

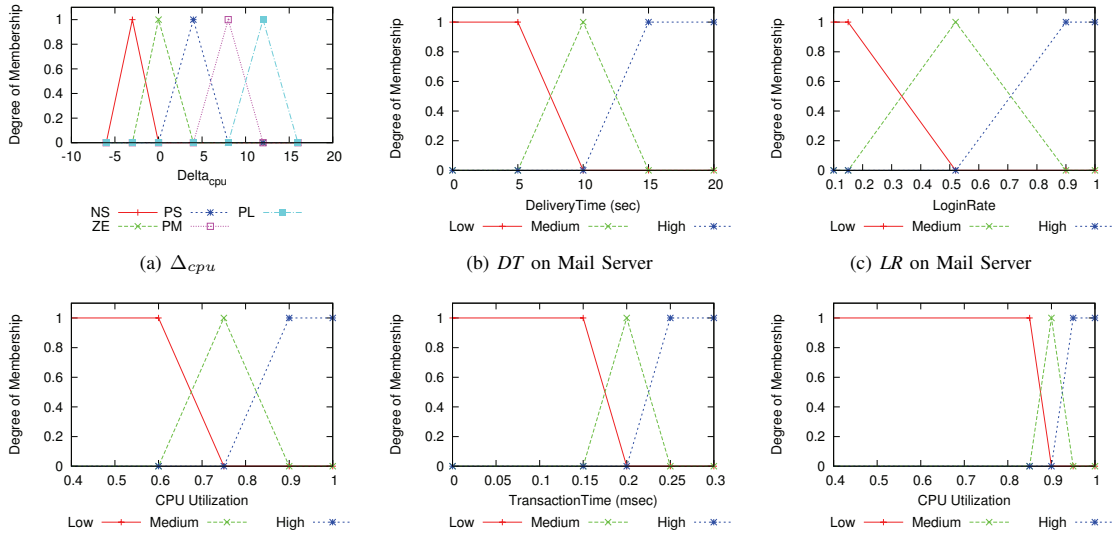
(b) Java Workload Increase (*JWI*)

Intervals (sec)	Mail	Java
1–399	*wMedium*	*wMedium*
400–1199	*wMedium*	*wHigh*
1200–1599	*wMedium*	*wMedium*

TABLE III
WORKLOAD IN THREE INTERVALS OF EACH SCENARIO. TABLES SHOW THE WORKLOAD LEVEL TO DOMAINS IN EACH TIME INTERVAL.

to generate the traffics containing of sudden workload changes, scenario of mail workload increase (*MWI*) and Java workload increase (*JWI*) as shown in Table III. Experiments have three time intervals. At first time interval, 1 second to 399 seconds, both workloads start to stress the servers under *wMedium* workload level. At the end of this interval, either mail or Java workload suddenly increases to *wHigh* and it continues during the second interval (400 seconds to 1199 seconds). After this interval, 1200 seconds to 1599 seconds, the increased workload changes again and it becomes *wMedium*.

Results are shown in average measurement values in Table IV for each interval. The metrics of *cap*, *cons* and *utz* evaluate resource allocation; *cap* indicates the capacity which the resource controller sets for each domain, *cons* means CPU consumption the domain utilizes and *utz* shows CPU utilization on the virtualized system. These values are represented by the percentage of total CPU time. *DT*, *LR* and *TT* are used to measure the application performances. For comparison, the experiment has two results, experiment with the AARM controller (*WRC*) and with Xen default scheduler (*DEF*). Both of them are tested under the same workload varying and *DEF* uses credit scheduler by default (cap is 0 and weight is 255



(a) Δ_{cpu} (b) *DT* on Mail Server (c) *LR* on Mail Server

(d) CPU Utilization on Mail Server (e) *TT* on Java Server (f) CPU Utilization on Java Server

Fig. 2. Membership functions. Figures show membership functions for each indicator. These functions are defined through inflecion points which indicate boundaries of the membership of indicators.

(a) Right Fuzzy Rule on Mail Server

Δ_{cpu}		*DT*		
		Low	*Medium*	*High*
Utilization	*Low*	NS	ZE	PS
	Medium	ZE	PS	PM
	High	PS	PM	PL

(b) Left Fuzzy Rule on Mail Server

TT	*Low*	*Medium*	*High*
Δ_{cpu}	PL	PL	–

(c) Fuzzy Rule for Java server

Δ_{cpu}		*TT*		
		Low	*Medium*	*High*
Utilization	*Low*	NS	ZE	PS
	Medium	ZE	PS	PM
	High	PS	PM	PL

TABLE II
FUZZY RULES. TABLE (A) AND (B) SHOW FUZZY RULES ON THE MAIL SERVER, AND TABLE (C) SHOWS A FUZZY RULE ON THE JAVA SERVER.

in all domains).

Value	Description
cap	The amount of capacity set to a domain. This parameter specifies the limit of CPU time the domain can consume. Setting of non-capacity is calculated as 100% capacity.
cons	CPU resource consumption on a domain
utz	CPU resource utilization per a virtualized system
DT	SMTP message delivery time (seconds) on mail server
LR	POP3 successful login connection rate on mail server
TT	Time (milliseconds) to complete a transaction on Java server

TABLE IV
MEASUREMENT VALUES. EXPERIMENTS' RESULTS ARE REPORTED BY THE METRICS OF THIS TABLE.

The control interval of resource allocation is selected as 3 seconds, because mail workload generates frequency changing traffic and the small interval is required to react fast to the workload changes. The measurement metrics are captured in each control interval.

A. Mail Workload Increase Scenario

In this experiment, the workload to the mail server is increased during the middle of the duration (Table III (a)). Table V (a) shows the result of the experiment with *WRC*. The experiment begins with workloads in *wMedium*. In the first interval, both the mail and the Java server still keep their performances to high and the *cons* of domains are equal because of the same levels of workloads. The *cap* of the mail server, however, is higher than *cap* of the Java server. The mail workload tends to generate unstable and highly variable load for the server, thus the value is higher than the Java server to allow a safety margin for the sudden load. At the end of this interval, the workload to the mail server suddenly increases. In this time, the controller detects the changed traffic through the decreased the performance of the mail server and it allocates extra CPU time to this domain. Actually, while the consumption rate between two domains was even in the first interval, it gives high priority to the critical domain (the mail server) in the second interval. This control behavior attempts to balance the performance between applications. The *cap* of the Java server is decreased because of low resource priority and *cap* of the mail server is also decreased because few selections for Cap_{no} (*cap* = 100) layout. Under the high workload, Cap_{no} layout tends to affect another domain's performance a lot, so the resource controller tries to isolate each domain's

resource to reduce this effect. At the third interval, the risen workload on mail server drops to *wMedium* level. The ratio of both *cons* and *cap* on domains get back to the value in the first interval. However, since the effects of *wHigh* workload remains at the beginning of this interval, the performance of the mail server slightly worse than the result in the first interval.

Table V (b) shows the results of experiment with *DEF*. At the first interval, the performance balance is maintained, and it performs almost same results as *WRC* case. However, when the mail workload increases at the second interval, the system loses the balancing of the application performances; although mail server runs with low resource and has a serious problem difficulty to serves the incoming messages, Java system still keeps high performance with underloaded resource. *DEF* cannot detects the critical application, therefore, it still distributes the same amount of resource to each domain. The stacked messages on mail server remain until the next interval and make this server to be busy.

Resource allocation of *WRC* and *DEF* affect the QoS metrics measured on each domain. In the second interval, *DT* has huge difference between the two cases. *DEF* takes 38.13 seconds for message delivery and is significant long compared to 18.9 in *WRC*. Also, *LR* in *DEF* 0.97 is lower rate than *WRC* (1.00). *TT* in the Java server, by contrast, takes only 0.17 milliseconds in *DEF* and is fast in comparison with *WRC* (2.0 milliseconds). From these differences, we can say *WRC* provides better performance than *DEF* in this experiment. Although QoS indicators have significant differences between two cases, *cons* (54.8 on *WRC* and 52.3 on *DEF*) does not make such big difference. This means that if the load on the mail server is relatively low, incoming messages are small or have few recipients, then Java server can execute high CPU usage, and otherwise mail server obtains extra resource. The control reaction reduces the unused resource. Actually, *WRC* records its *utz* to enough high closed to *DEF* which has no resource limits.

The experiment shows that the *WRC* results in better performance than *DEF* and also balances between application performances when mail workload increases. However, the controller should gives few more resource to the mail server because *DT* is classified in *High* and *TT* is in *Medium* in the defined fuzzy members.

B. Java Workload Increase Scenario

In this experiment, in order to test the resource controller in the case of increased Java workload, the Java workload increases to *wHigh* at the second interval (Table III (b)).

The result of *WRC* is shown in Table V (c). At first interval, the workloads to both servers are same as the previous experiment and produce same results as shown in the table. At the end of this interval, the workload to the Java server increases in this experiment. The resource controller detects risen *TT* and considers more CPU time is needed for the Java server to adjust its increased workload. Simultaneously, it decreases the amount of resource for the mail server giving low priority with a consistent workload. The allocation results show that *cons* on the Java server is increased and on mail server is decreased rather than the previous interval.

The result of *DEF* is also shown in Table V (d). At second interval, the increased workload to the Java server causes the performance down on this domain. For instance, QoS indicator *TT* increases from the first interval. *DT* on the mail server is also increased with similar *cons* on this domain. This because, the Java server tries to utilize the resource more and more and this makes the mail server to difficult to obtain large amount of CPU time when the variable load become high and thus the *DT* drops.

In this experiment, *WRC* balances the performance by giving additional resource to the Java server. Compared with both results, *DEF* reduces the rise in the *DT* to 3.3, although *WRC* increases *DT* to 4.9 in the mail server. In the Java server, *DEF* obtains higher QoS indicator *TT* (0.22) than *WRC* (0.21), but it is subtle distinction. From these reasons, describing the overall performance, *DEF* results better performance than *WRC*. By compared with the results of *cons*, *DEF* also remains consistently higher than *WRC* throughout the whole intervals and in both domains.

When Java workload increases, *WRC* can balance the application performance, but the results is worse than *DEF*. Furthermore, the controller should give higher priority to the Java server than the result of *WRC*, because *DT* is in *Low* and *TT* is in *Medium* of the defined fuzzy members.

VI. Related Work

Resource management on a virtualized system lately been attracting the attention of many researchers.

[9] deals resource partitioning with HP-UX Process Resource Management (PRM). The design of the control is based on the modeling of input-output relationship between CPU allocation and the mean response time (MRT), and it aims to regulate MRT to a reference value. To identify this relation, they observe static relation from experiments and found it demonstrates bimodal behavior of MRT under both overload and underload region. To build linear equation identifying the CPU entitlement to regulate MRT to the reference, they use inverse MRT (1/MRT) instead of MRT itself. The first controller they consider is use CPU entitlement as the input of the nonlinear static model to maintain the relative utilization at a target value. The controller behaves increase CPU allocation quickly when the system is overload, and becomes conservative when the system is in underload condition. This controller can maintain the utilization at around the target and achieves higher throughput and small response time. Authors also propose the controller regulating MRT at reference value. They established ARX model representing the relation between CPU entitlement and inverse MRT based on the analysis of the previous experiments. The model incorporates the relatively utilization into the control model and then it shows stable of the system is maintained under the dynamical workload.

[10] designs a two-layered controller that guarantees application-level QoS of two multi-tier applications in a vir-

(a) Result of *WRC in* MWI

	Mail				Java			Total
Intervals	*cap*	*cons*	*DT*	*LR*	*cap*	*cons*	*TT*	*utz*
1–399	83.5	45.0	3.5	1.00	95.1	46.4	0.17	91.4
400–1199	88.6	54.8	18.9	1.00	69.3	40.5	0.20	95.3
1200–1599	83.2	46.4	4.7	1.00	93.8	45.7	0.17	92.2

(b) Result of *DEF* in *MWI*

	Mail			Java		Total
Intervals	*cons*	*DT*	*LR*	*cons*	*TT*	*utz*
1–399	45.6	2.6	1.00	46.7	0.17	92.3
400–1199	52.3	38.1	0.96	45.3	0.17	97.6
1200–1599	47.1	14.3	0.99	46.5	0.17	93.6

(c) Result of *WRC* in *JWI*

	Mail				Java			Total
Intervals	*cap*	*cons*	*DT*	*LR*	*cap*	*cons*	*TT*	*utz*
1–399	83.3	44.8	3.3	1.00	94.8	46.1	0.17	90.9
400–1199	70.2	43.9	4.9	1.00	87.7	49.7	0.21	93.6
1200–1599	84.2	45.0	3.1	1.00	96.7	46.5	0.17	91.4

(d) Result of *DEF* in *JWI*

	Mail			Java		Total
Intervals	*cons*	*DT*	*LR*	*cons*	*TT*	*utz*
1–399	45.3	2.5	1.00	46.9	0.17	92.2
400–1199	45.7	3.3	1.00	50.0	0.22	95.7
1200–1599	45.3	2.3	1.00	47.4	0.17	92.6

TABLE V
EXPERIMENT RESULTS IN SCENARIO *MWI* AND *JWI*.

tualized system. The first layer controller computes resource requests to meet a specific level of resource utilization on each domain. Second layer controller serves arbitration when the sum of requests submitted by first layer controller exceeds the physical computer's capacity. In that case, the controller decides resource allocation based on QoS differentiation metric. Finally, they evaluated the controller model with two tier implementation of RUBiS.

[11] presents a methodology to allocate an appropriate amount of memory to a domain. The memory balancer (*MEB*) uses Least Recently Used (LRU) histogram to track memory accesses for each virtual domain. *MEB* identifies the relationship between memory size and page miss rate which can be derives from a page miss ratio curves in LRU histogram. Therefore, the balancer produces minimum memory size yield page miss rate no larger than a certain rate.

VII. CONCLUSION

The paper proposed and evaluated the effective resource controller in a virtualized system. The controller deployed two layered controlling. The first layer controller aims to maintain the application performance at an acceptable level. To achieve the goal, it observes and collects the application states, and produces the their desired amount of resources based on these states. The paper modeled the relationship between the resource demand and their states by fuzzy control theory. The second layer controller's objective is to maintain a high CPU utilization. This controller adjusts the requests to the system capacity and builds the layout of resource capacity based on the QoS performance balance of applications. This approach reduces the amount of unused CPU time and achieves high CPU utilization.

The evaluation showed the controller performance; keeps the performances balancing between applications while maintains the CPU utilization closed to the Xen default scheduler having no resource limits.

However, this resulted its drawback which it is difficult to react delicate operations.

As future work, consideration of the tuning of parameters, such as the number of average and control interval, is required as extended research topic. Also, to test the details of the control reaction, the variety of workloads and its scenario should be increased. In this paper only the average measurement values are used for measurement of the performance, but real time performances, responsiveness and stability, are important factors to determine the controller performance. Furthermore, the paper focuses on provisioning of CPU time as a primary component of computer resources, however, modern server applications utilize other components, such as, memory, disk and network I/O. In addition to the difficultly of virtualizing itself, the improvement, especially in their access speed, of I/O devices is at a much lower pace than CPUs. In this current situation, the virtualized system that utilizes I/O devices will become more limited by the bottleneck component, and it is desirable to provide more enhanced allocation method for the resource by resource manager as one of the methods to improve the critical resource.

REFERENCES

[1] B. Day, S. Yates, L. Koetzle, and T. Powell, "Identifying server consolidation cost savings," *Forrester Research*, October 2005.

[2] J. Daniels, "Server virtualization architecture and implementation," in *Crossroads*, vol. 16. ACM, September 2009, pp. 8–12.

[3] P. Barham *et al.*, "Xen and the art of virtualization," in *ACM Symposium on Operating Systems Principles*, 2003, pp. 164–177.

[4] D. Chisnall, *The Definitive Guide to the Xen Hypervisor*. Prentice Hall, 2008.

[5] J. Jantzen, "Design of fuzzy controllers," Technical University of Denmark, Tech. Rep. 98-E-864, August 1998.

[6] M. G. Simöes, "Introduction to fuzzy control," `http://inside.mines.edu/~msimoes/tutorials/Introduction_fuzzy_logic/Intro_Fuzzy_Logic.pdf`, 2008.

[7] Standard Performance Evaluation Corporation, "Specmail2001," `http://www.spec.org/mail2001`.

[8] ——, "Specjbb2005," `http://www.spec.org/jbb2005`.

[9] Z. Wang, X. Zhu, and S. Singhal, "Utilization and slo-based control for dynamic sizing of resource partitions," in *IFIP/IEEE Distributed Systems: Operations and Management*, October 2005, pp. 133–144.

[10] P. Padala *et al.*, "Adaptive control of virtualized resources in utility computing environments," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3, June 2007, pp. 289–302.

[11] W. Zhao and Z. Wang, "Dynamic memory balancing for virtual machines," in *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2009, pp. 21–30.