4M-Switch : Multi-Mode-Multi-Model Supervisory Control Framework for Performance Differentiation in Virtual Machine Environments

Tharindu Patikirikorala
Swinburne University of Technology
Melbourne
Email: tpatikirikorala@swin.edu.au

Alan Colman
Swinburne University of Technology
Melbourne
Email: acolman@swin.edu.au

Jun Han
Swinburne University of Technology
Melbourne
Email: jhan@swin.edu.au

***Abstract*—When resources are shared in a virtual machine environment, providing different performance levels to different customer applications is a challenging task. In order to sustain stability, the control solution not only has to take into account the time-based dynamics, but also has to adapt to various operating modes. This paper proposes the *4M-Switch* supervisory control system design framework, which takes into account the possible operating modes and dimension changes of the VM environment at design time and then adapts the control solution to achieve required management goals when changes occur at runtime. 4M-Switch utilizes a piece-wise linear modeling approach to present the behavior of the system using multiple models and simple switching logic to change the controller parameters to mitigate the effects of nonlinearities. The experiment results conducted under a range of conditions show that 4M-Switch approach effectively adapts the control solution and provides significantly more stable performance differentiation compared to the existing approaches.**

I. Introduction

Compared to the traditional approach of having dedicated hardware resources for each customer application, the virtualization technology has enabled data centers to obtain economic benefits by sharing pools of hardware resources between software applications of multiple customers. However, managing physical hardware resources of a server between multiple virtual machines (VMs) to maintain the performance variables at different levels under unpredictable workload disturbances is a challenging task to automate. This is because, firstly, such virtualized servers often need to provide services to multiple VMs (i.e., multiple applications) with different performance objectives that may change overtime. Secondly, the workloads for these applications (VMs) may vary overtime in an unpredictable fashion. Thirdly, VMs may be in maintenance or idle for short/long periods of time changing the dimension of the control problem abruptly.

In order to automate the differentiated performance and resource management tasks in VM environments, the feedback control mechanisms have been looked at in the last few years [13]. However, these control approaches assume that the time-based state/performance variable signals of the control problem remain in a certain region of the state space. In other words, it is assumed that the system always remains in a single or limited set of operating modes which can be characterized by a single behavioral model. This means that these approaches have disregarded many other viable operating regions/modes, which could invalidate the proposed control solution due to mathematical ill-conditioning leading to significant failures of the control solution. In dynamic VM environments the operating mode and problem dimension may change because of the i) idling applications, ii) applications under maintenance and iii) VM migrations. In most of these operating modes, time-based dynamics are also shown due to workload variations and other disturbances faced by active VMs, requiring dynamic performance differentiation and resource management. In order to design a safe and comprehensive control solution, these different modes and time-based dynamics have to be considered together in the design.

Furthermore, the performance properties (e.g., response time) of a VM are nonlinearly related to the shared resource as shown by many existing works (e.g., [19], [14], [9], [17]). In addition, the performance differentiation schemes impose significant nonlinearities on the management system [7]. Consequently, the existing linear feedback control methods [6], [7], [10], [9] based on a single dynamic model typically fail to achieve effective performance differentiation objectives under changing workload conditions and control objectives.

Many physical systems show a hybrid of discrete-event (e.g., application maintenance event) and time-based (e.g., workload variations) dynamics, so-called *hybrid systems* [15], [16]. As opposed to representing the system just with time-based dynamics, this paper treats the VM environments as hybrid systems and proposes a new control framework called Multi-Mode-Multi-Model switching supervisory control (4M-Switch), which explicitly captures the different operating modes and dimension changes of the control problem and adapt the control system dynamically to suite the operating mode, system conditions and differentiation requirements. In order to tackle the limitations of a single linear model based control, a technique is proposed to present the behavior of the system using multiple models and then a control solution is implemented with a configuring controller to achieve the performance differentiation objectives. The novelty of 4M-Switch is the combination of hybrid and multi-model adaptive feedback control to achieve control objectives of a VM environment for the first time in literature. We show, with a range of experiments conducted in a VM environment, that the 4M-Switch approach results in performance and resource management that is much more comprehensive, stable and safe compared to the existing approaches.

The structure of this paper is as follows. Section II describes the background, followed by related work in Section III. The details of the VM environment used in this work is then presented in Section IV. The problem overview and the approach is covered in Sections V and VI respectively. Experimental results and comparative analysis are provided in Section VII.

II. Background

This section provides an overview of hybrid systems and relative differentiation schemes.

Hybrid control system: Figure 2(a) shows the architecture of a hybrid control system. The *generator* converts the time and event based data from the sensors to *plant symbols* when the special conditions are met. Depending on the plant symbols, the controller makes control decisions. The controller operates with the plant model, typically described by a finite automaton. In each state, the system is treated as a discrete or continuous time system [16] and control policies are implemented to come up with the control decisions. Finally, the controller decisions are converted to control inputs by the actuator.

Relative performance differentiation scheme: In the performance differentiation scheme, the performance attributes of the classes are maintained proportional to the performance differentiation factors derived from the business or system design requirements. Let P_i, Q_i be the specified differentiation factor and the actual performance attribute of interest respectively of a class i (i= $0, \ \ldots \ n-1$), out of n number of classes. Between the pair of classes i and j, the management objective is to maintain $\frac{Q_j}{Q_i} = \frac{P_j}{P_i}$ (i = $0 \ldots n-1, i \neq j$) at runtime under varying workload conditions. For instance, $\frac{P_1}{P_0} = 2$ means that the performance attribute of class_1 has to be maintained twice as of class_0. Lu et al. in [6] proposed a dynamic propositional resource share allocation approach to achieve the aforementioned relative performance differentiation scheme. Depending on the workloads the resource share ratio $\frac{S_j}{S_i}$ (where, S_i and S_j are the amount of resources share for i and j classes respectively) is manipulated at runtime to achieve the performance differentiation objectives of all the classes under changing workload conditions.

III. Related work

In order to manage performance in VM environments, many control engineering approaches have been proposed in the past few years. A survey of such approaches can be found in [13]. The relative performance differentiation scheme with feedback control has been utilized to manage web servers [6], [7], [10], storage systems [8] and data centers [9]. These existing approaches utilize a single linear model or adaptive model with a feedback controller, limiting the operating range of the controller to a single operating mode and a narrow region that can be linearized. As shown by [7], [9], the relative performance differentiation scheme inherits significant nonlinear behavior due to incorporating the ratio of performance attributes (e.g., response time, CPU utilization). The linear approaches face issues under these nonlinearities (see [14], [12]), while adaptive approaches [9], [8] fail when they face fast varying workload conditions as shown by [12].

All these approaches also assume that the dimension of the control problem remains constant over time (i.e., constant number of classes or VMs). However, in VM environments, VMs can go into idle or maintenance mode or get migrated, making certain state variables (e.g., response time) zero, which could lead to mathematical ill-conditioning of the control solution due to divide by zero. This means that the mathematical ill-conditioning and changing dimension of the control problem have been disregarded by existing approaches.

A discrete event system design approach to implement the hybrid control systems (as in Figure 2(a)) was proposed in [16], [15] where they use state-space analysis methods to achieve the control objectives. There is little work that uses hybrid control to manage performance in software environments. The work that exists [5], [1] focuses on the issue of discrete inputs available in software systems to provide fine grained control using state-space exploration techniques. As shown in [4], such exploratory techniques impose high computational load and time complexities to come up with the decisions. In [3], a CPU utilization control mechanism is presented by dividing the state space in to three fixed regions with one being the desired region for normal operations. However, they do not use any feedback controller for set point tracking, and if the control objectives or defined operating regions change, then their mechanism requires significant redesign.

In this paper we do not utilize complex state exploration techniques, instead the hybrid control concept is used to represent and identify the modes based on state variable data and then to implement different control policies depending on the state. This is to resolve the aforementioned mathematical ill-conditioning issues of the existing approaches. The finer grained performance differentiation is then achieved using feedback control. To handle the nonlinear dynamics we demonstrate a new gain-scheduling [2] approach based on multiple models.

IV. System Description

Figure 1 illustrates the architecture of the virtual machine environment under consideration in this paper. It consists of three physical machines (M_1, M_2 and M_3)[1]. Server (M_1) is running on CentOS and acts as the shared resource infrastructure. It is also equipped with Xen 2.6 hypervisor to manage virtual machines (VMs). Using the Xen hypervisor, n VMs can be deployed inside the server machine. Each VM also runs on CentOS and has Apache Httpd 2.2 server installed, in order to deploy customer software applications. To represent the client application, the RUBiS benchmark was used. As shown in Figure 1, client simulators (so-called workload generators) were deployed in M_2, while the database of RUBiS benchmark

[1] M_1: CPU is a Intel Core(2) Duo 2.33 GHZ processor with 4 GB memory, M_2: CPU is a Intel Core(2) Duo 2.33 GHZ processor with 2 GB memory and M_3: CPU is a Intel Core(2) Duo 2.33 and 2.99 GHZ processor with 3 GB memory.

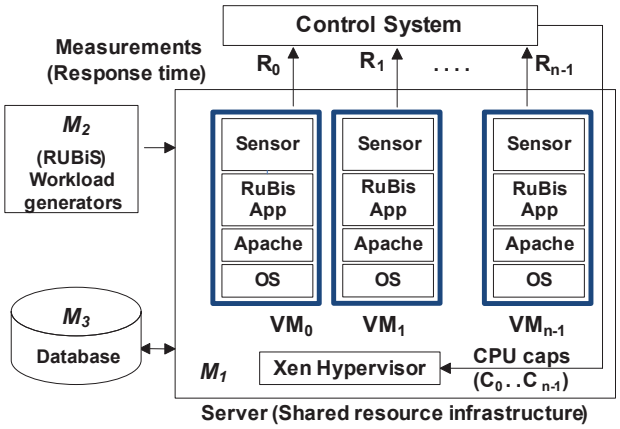


Fig. 1. Virtual machine environment

was deployed in M_3. All three machines were connected using a network switch, creating an isolated network between the three machines.

The management objective of this system is to control the average response time of each customer application. In order to measure the response time (R_0, R_1, ..., R_{n-1}) a background (sensor) program was implemented in Java and deployed to intercept the incoming and outgoing requests and compute the response time of each application (i.e., VM) in regular intervals (20 sec). The CPU capacity allocated to each VM (C_0, C_1, ..., C_{n-1}) by the hypervisor is the *manipulated resource*. Xen Hypervisor provides a credit-based scheduling mechanism, which enables users to adjust the CPU capacity of each VM dynamically. Using this functionality, an actuator was implemented in Java and deployed inside the server. In order to isolate the VMs from the hypervisor (Dom-0), the hypervisor was pinned on to a single core of the CPU of M_1, while VMs were pinned to the other CPU core[2]. Each VM was allocated with 1 GB of dedicated memory.

V. Problem Overview and Analysis

In this section, we present an overview of the performance differentiation problem of a VM environment with multiple VMs.

A. Assumptions

For the purposes of this paper we have made a number of working assumptions:

1) The CPU capacity is considered as the shared and main bottlenecked resource that has to be managed depending on the workload conditions[3].

2) Performance differentiation is practical only if there is more than one VM in the server. In the case of there is only one VM the designer specifies the CPU allocation policies for that VM after considering the number of VMs that could be migrated to the server.

3) In the case where there are two or more active VMs, we assume that each VM deployed in the server is guaranteed a fixed minimum limit of CPU capacity ($C_{i,min}$, where $i = 0, 1, \ldots n-1$) during the entire operation. This is to avoid starvation of resources under heavy workloads of other VMs.

4) When a migration takes place, we assume that hypervisor registers/deregisters the particular VM and connects/disconnects its sensors to the control system.

B. Management Problem Definition

The main control objective is to share the available CPU capacity in an effective way to maintain the response times of VMs depending on the specified differentiation levels. In addition, the control system should be able to self-adapt when the operating mode changes at runtime to avoid failures and mathematical ill-conditioning. Furthermore, the nonlinear dynamics have to be tackled effectively to handle changing control objectives.

As discussed above, we adopt the relative performance management scheme combined with a feedback controller to solve the management problem described previously. Therefore, the control objectives are now specified based on the relative performance differentiation scheme.

Control objectives: According to the business requirements let us assume the differentiation factors for n VMs are determined statically or dynamically as $P_i(k), i = 0 \ldots n-1$. Then, the control objective according to the relative performance differentiation control scheme becomes maintaining the response time ratio of $(i-1)^{th}$ and $(i)^{th}$ VMs $\frac{R_i(k)}{R_{i-1}(k)}$, around $\frac{P_i(k)}{P_{i-1}(k)}$ while computing the CPU caps $C_i(k), i = 0, 1 \ldots n-1$. Furthermore, the management system should honor the following constraints, related to total resource availability and per VM CPU reservations at all times.

$$C_0(k) \geq C_{0,min}, C_1(k) \geq C_{1,min}, \ldots, C_{n-1}(k) \geq C_{n-1,min}$$
$$C_0(k) + C_1(k) + \ldots + C_{n-1}(k) = C_{total}(k) \quad (1)$$

Example: Let us consider the VM environment described in Section IV having two VMs (VM_0 and VM_1), with the possibility of VM_2 to be active or migrated at anytime. $R_0(k)$ and $R_1(k)$ are the response times of two existing VMs respectively at the k^{th} sample. Furthermore, the CPU caps are $C_0(k)$ and $C_1(k)$ where $C_0(k)+C_1(k) \leq C_{total} = 100\%$ and $C_{0,min}, C_{1,min} = 20\%$. According to the definition in Section II, the control input is the ratio of CPU caps, represented by $\frac{C_0(k)}{C_1(k)}$ and the output variable is the ratio of average response time of the workloads, represented by $\frac{R_1(k)}{R_0(k)}$. For notational simplicity let us denote $\frac{C_0(k)}{C_1(k)}$ and $\frac{R_1(k)}{R_0(k)}$ as $u(k)$ and $y(k)$ respectively. The control objective of this control system is to maintain the response time ratio $y(k)$ of VM_0 and VM_1 around $\frac{P_1(k)}{P_0(k)}$ (i.e. the set point) depending on the performance differentiation factors of the VMs $P_0(k), P_1(k)$. When VM_2

[2]In a server where there are more than two cores the same approach can be used by pining VMs to share multiple cores

[3]In the system studied in this paper, the RUBiS benchmark has been modified to simulate CPU bound operations.

is added, the above differentiation problem changes to maintaining variables $\frac{R_1(k)}{R_0(k)}$ and $\frac{R_2(k)}{R_1(k)}$ around $\frac{P_1(k)}{P_0(k)}$ and $\frac{P_2(k)}{P_1(k)}$ respectively. See [6], [7] for formulation for n number of VMs.

C. Analysis of Operating Modes

To indicate the different operating modes and different control policies needed, in this section the following limited number of cases will be considered from all possible permutations.

Case 1: when $R_0(k)$, $R_1(k) > 0$. Here, the system is in normal operating mode, which means performance differentiation can be implemented safely.

Case 2: when $R_0(k)$, $R_1(k) = 0$. The applications in both VMs are idling or under maintenance simultaneously. The performance differentiation is not viable in this mode. Therefore, suitable CPU allocations policies have to be implemented by the control solution.

Case 3: when $R_0(k) = 0$, $R_1(k) > 0$ or $R_0(k) > 0$, $R_1(k) = 0$. This means one of the VMs is idling or under maintenance. These two modes have to be identified by the control solution and other CPU allocation policies have to be implemented in order to avoid mathematical ill-conditioning and maintain the integrity of the control system.

Case 4: when $R_0(k)$, $R_1(k) > 0$ and VM_2 has been added to the system making $R_2(k) > 0$. In this mode, the control solution has to be reconfigured with new state variables, set points and controllers to manage performance dedifferentiation and resource allocation between three VMs.

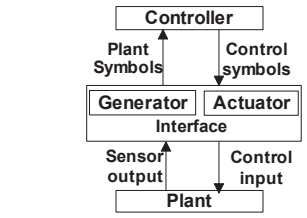
The existing literature assumes that the system is in normal operating mode (Case 1) during the entire operation, which is clearly invalid when the above cases are investigated. Therefore, control solution should self-adapt based on the state variable data and number of active VMs and then provide appropriate CPU allocation decisions to sustain the system stability and continuity of the control solution.

D. Analysis of nonlinearity

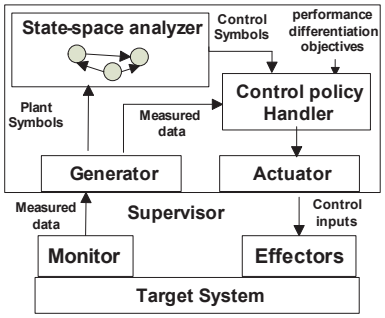
The output $y(k)$ ($\frac{R_1}{R_0}$) of the VM environment exhibits highly nonlinear behavior because of the division operation. For instance, if $R_0 = 0.4$ (sec) and $R_1 = 1$ (sec) then $\frac{R_1}{R_0} = 2.5$. If $R_0 = 1$ (sec) and $R_1 = 0.4$ (sec) then $\frac{R_1}{R_0} = 0.4$. Therefore, when R_1 increases $\frac{R_1}{R_0}$ increases at a high rate. In contrast, when R_0 increases $\frac{R_1}{R_0}$ decays at a high rate. This means that behavior of the output is significantly different in $y \geq 1$ region compared to y < 1 region. However, in a dynamic VM environment, the control objectives and workload conditions can vary overtime forcing the control system to operate in these highly nonlinear regions. Such nonlinear behavior may also cause performance degradation in a linear controller as shown in [14]. Hence, considering such nonlinear behavior at design time is also a requirement for a control solution.

VI. 4M-Switch Framework

Figure 2(b) illustrates the high-level architecture of the 4M-Switch supervisory control system. The VM environment under study is the target system and *Supervisor* component is the decision making unit. The supervisor operates similar to hybrid control system introduced in Section II. The following subsections cover the design details of its subcomponents.



(a) Hybrid system



(b) 4M-Switch

Fig. 2. Architecture of hybrid control and 4M-Switch

A. Generator

Upon receiving the state variable and other event data from the target system, the data is analyzed by the generator to detect the special events that may have occurred in the target system or its environment. These event occur when the state variable data has crossed some predefined threshold in a certain direction [15]. This threshold that divides the state space into two regions is called as *hyperplane* [16]. The hyperplane is formally defined by a function of ith discrete time-based state variable $x_i(k)$ (i.e., $h(x_i(k))$, where k is the time sample). The generator is defined by a set of such functions. The generated event is then represented as a plant symbol as follows,

$$E_i(k) = \begin{cases} e_i & \text{if } h(x_i(k)) \leq 0 \text{ and } h(x_i(k-1)) > 0. \\ \hat{e}_i & \text{if } h(x_i(k)) > 0 \text{ and } h(x_i(k-1)) \leq 0. \\ null & \text{otherwise.} \end{cases} \quad (2)$$

Equation (2) produces a plant symbol ($E_i(k)$) when $h(x_i(k))$ crosses zero form positive to negative side or vise versa. In all other cases a *null* symbol is returned. In the VM environments, the response time (R_i) or CPU caps of VMs can be used as the state variable to trigger events when the operating region changes. For instance, $h(R_i(k)) = a - R_i(k)$ can be used to generate plant symbols R_i and $\hat{R}_i$ denoting idle and active regions respectively when the response time crosses a limit from either of the directions (i.e. $a = 0$ or small value). At least these regions have to be incorporated

in the design to adapt the control system safely and avoid mathematical ill-conditioning. The above function is sufficient, in the cases where $R_i(k)$ can also be used to detect the maintenance activities and migration of the VM_i based on the assumptions in Section V-A. In other cases, alternative functions have to be implemented based on the state variables.

B. State-space Analyzer

The state-space analyzer consists of a deterministic finite automaton defined as $(S,\ E,\ R,\ \delta,\ \phi)$, where S, E, R are set of states or operating modes, plant symbols and control symbols respectively. $\delta : S \times E \rightarrow S$ is the state transition function, while $\phi : S \rightarrow R$ is the output function. Given the initial or previous mode and the plant symbol, the automaton reaches the current operating mode using the δ function. Finally, using the current operating mode, the output function ϕ generates the control symbol.

At this design stage, all possible events have to be listed and possible unique operating modes have to be identified. For instance, if only VM_0 and VM_1 are deployed in the physical machine and we use R_i and $\hat{R}_i$ plant symbols, where $i = 0, 1$ four unique operating modes can be defined. They are $\hat{R_0}\_\hat{R_1}$, $R_0\_\hat{R_1}$, $\hat{R_0}\_R_1$ and $R_0\_R_1$. The same approach can be used for VM environment with n VMs. Although all n VMs may not be involved at runtime at once, the design can be done for n VMs. This approach helps us to implement a comprehensive control solution considering all possible operating modes at the design time. As the control symbol a unique identifier can be used representing each state (i.e., ϕ).

C. Control Policy Handler

Given the control symbol, this component evaluates the most suitable control policy for system's operating mode. The control policies could be *static* (i.e. constant values or decisions based on If-Then rules) or *dynamic* (i.e., decisions generated by feedback control systems). The static control policies are most suitable when the performance differentiation is not possible or heuristic based logic needs to be implemented using historical data. As an example for the Case 2 in Section V-C, if the designer wants to allocate minimum CPU limit for each VM, this can be implemented as a static control policy.

However, in the normal operating mode under unpredictable workload variations resource allocation decisions cannot be predetermined, which means that the static control policies cannot achieve the required differentiation objectives or become significantly complex. In such modes, effective control can be implemented by using a feedback control system as shown by the existing literature [6], [9], [14]. The control policies composed of feedback control systems can be classified as dynamic control policies. In addition to executing the required control policy, control policy handler is responsible for (i) state management, (ii) computing required variables for the control solution and (iii) specifying control objectives when control systems switch from one to another subsequent to an operating mode change in the system.

One of the major issues of switching control system is large transient responses during and short after a switch has occurred [18]. In this proposed control technique, when mode switching takes place, a control system could be switched abruptly as well leading to such large transient responses. Therefore, in the design of feedback control solutions (i.e. dynamic policies) necessary techniques have to be incorporated to avoid such transient responses also called as *bumpless transfers* [18]. In the following sections we present feedback control system design details for such dynamic policies.

D. Dynamic Policy Design

In this section, we use the same control system architecture that has been utilized by the existing works [6], [10], [14] for performance differentiation between n VMs. However, the analysis in Section V-D revealed that there are significant nonlinearities in the relative performance differentiation system. Due to consideration of ratio, when the output is in region $(y = \frac{R_1}{R_0}) \geq 1$, the system behaves differently to that in the region $(y = \frac{R_1}{R_0}) < 1$. As a consequence, if the control objective (set point) is changed dynamically between these regions a controller designed with a single model fails to achieve the performance differentiation goals (see [12] for experimental evaluations). This section shows a design methodology to implement dynamic policies using a control system with two models one representing $y \geq 1$ region and other representing $y < 1$ region.

1) Model Identification: In order to model the system behavior, system identification is used [2]. Select two VMs. From their input set, select a subset of CPU ratios and workloads that would force the system to operate in $y > 1$ output region. Using this input subset design a pseudo random signal and apply it as the input of the system while sending a constant workloads for those two VMs. Then, the gathered data samples $(y(k) - u(k))$ are used to estimate an autoregressive exogenous input (ARX) model [2] with a sufficient accuracy (say $model_1$). Following the same process derive an ARX model by maintaining the system output in $y < 1$ region (say $model_2$).

2) Controller Design: To make dynamic control decisions, here we use a Propositional Integral (PI) controller [2]. The control equation of the PI controller is shown in equation (3). The controller calculates $u(k)$ for $k > 0$, given $u(0)$. $e(k)$ represents the *control error*, computed by $y(k) - r(k)$, where $r(k)$ is the set point of the system. K_p (propositional gain) and K_i (integral gain) are called gains of the PI controller decided by the designer using formal methods [2].

As mentioned, the existing works use a single linear model to represent the system, consequently gains of the controller remains static over the entire operation. However, in this work we use two models therefore gains have to be calculated at runtime or switched at runtime depending on a switching logic. The second approach is preferred because as the model parameters are known at the design time we can pre-calculate the gains and store them with the control policies. Therefore, it is a gain-scheduling technique [2], [13].

$$u(k) = u(k-1) + (K_p + K_i)e(k) - K_p e(k-1) \quad (3)$$

In order to switch between gains we use a simple If-Then rule as follows.
If $\frac{P_i(k)}{p_{i-1}(k)} \geq 1$ *Then use gains computed from* $model_1$,
Else use gains computed from $model_2$.

The above PI control system is then used to manage i and $i-1$ VMs. It adapts to the workload variations and adjusts the CPU caps to achieve the control goals, as well as handles the aforementioned nonlinear behavior and adapts the control parameters accordingly. It also implements bumpless transfers by using the velocity form of the PI algorithm. The *velocity/incremental form* of the PI law illustrated in equation (3) is an established approach for implementing *bumpless transfers* in the case of mode and controller switching systems [18]. In addition, if a switching occurs due to changes made to the performance differentiation factors (i.e. set point) by the designer/user, we use the same controller with all state data. The only change is the gains of the controller. The user requirements typically change infrequently, hence more stable switching can be achieved compared to using stochastic variables like workload rates.

E. Actuator

This component communicates the control decisions generated by the control policy handler to the effectors.

VII. Experimentation

This section first covers the design and implementation details of the 4M-Switch control system before presenting the experiment results. Here, the environment described in Section IV will be setup with maximum of three VMs ($n = 3$). The control objective is to provide performance differentiation between these VMs. However, the operating modes could change abruptly due to idling applications, VMs under maintenance or VM migrations[4]. If application is idling we specify a constant CPU of 10% to that specific VM, which we assume is sufficient to expect sudden incoming workloads. This percentage could be adjusted depending on the requirement and environment. $C_{i,min}$ is set at 20%, when performance differentiation is implemented between VMs $i = 0, 1, 2$.

A. Control system Implementation

The state space variables $R_0(k)$, $R_1(k)$ and $R_2(k)$ are the main drivers of operating modes. When idling and VMs migrated in and out these variables can be used to define the states. Now, using these variables we specify the generator functions as follows $h(R_i(k)) = 0.001 - R_i(k)$, which generates R_i, $\hat{R}_i$ and $null$ symbols, where i = 0,1,2 (see equation 2). From the experiments we observed that response time of the applications drops below 0.001 (sec) when there is no or only few requests. Consequently, we assume that if the applications response time is below 0.001 (sec) it is idling. Otherwise the application is in active mode.

The next step is to implement the finite automaton. The operating modes or states were selected as Init, R0, R1, R2, R0_R1, R0_R2, R1_R2, and R0_R1_R2. Each R_i symbol represents that the corresponding VM is in the system or in active mode in that particular state. The *init* mode represents when none of the VMs exist in the system or all VMs are inactive.

The control policies have to be then determined for each of these modes. In this implementation, we used both static and dynamic policies. For *init* state we allocated all VMs equal CPU caps (33%), which is a static policy. Similarly, in R0, R1 and R2 modes, we allocated 80% of CPU cap for the active VM, leaving 10% for each inactive VM. In all other modes we can implement performance differentiation using dynamic policies in an adaptive manner. As the dynamic policy, the aforementioned relative differentiation feedback control system can be implemented.

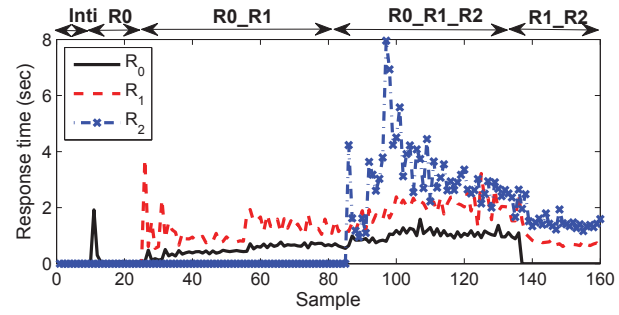
Following the design methodology in Section VI-D2, the control system is designed as follows. To represent the behavior of the system in y ≥ 1 and $y < 1$ regions, two experiments were conducted maintaining the output in each region. Without loss of generality, VM_0 and VM_1 were selected for this experiment. For the first region, a pseudo random input signal composed with the points in region where VM_0 gets more CPU were used while simulating 100 users for both VMs simultaneously. Gathered input-output data was then used to derive an ARX model. A similar experiment was conducted in the $y < 1$ region as well to derive the second model. Using the model parameters and pole-placement design methodology [2], gains for PI controller were then computed for each region. The controller gains for region ($y \geq 1$) are ($K_p = 0.05$, $K_i = 0.06$) and for region $y < 1$ gains are ($K_p = 0.15$, $K_i = 0.20$).

The name of the operating mode was used as the control symbol for simplicity. Based on the control symbol, we implemented additional logic in the control policy handler to compute the required feedback variables, control objectives and structure of the control system. For instance, we can manage ($\frac{R_1(k)}{R_0(k)}$) by adjusting $\frac{C_0(k)}{C_1(k)}$ using a dynamic policy in state R0_R1. Similarly, in R0_R2 mode, we can manage ($\frac{R_2(k)}{R_0(k)}$) by adjusting $\frac{C_0(k)}{C_2(k)}$. In R0_R1_R2 mode on the other hand, two control systems can be used in tandem to manage performance differentiation between (VM_0, VM_1) and (VM_1, VM_2) pairs [6]. Subsequent to a control symbol change, control policy handler executes the corresponding control policy or control system and continues to do so till the next control symbol change.

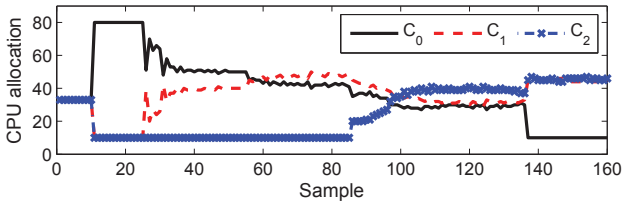
B. Experiment Results

In this section, we present experiments conducted to investigate the performance differentiation capability of 4M-Switch. The first one investigates the behavior under operating mode transitions and workload variations, while the second

[4]In this environment we simulate VM migrations by workloads, i.e. if workload increases from zero to some value, we assume that as VM being migrated to the system.



(a) 4M-Switch output



(b) 4M-Switch control signal

Fig. 3. Performance under operating mode transitions

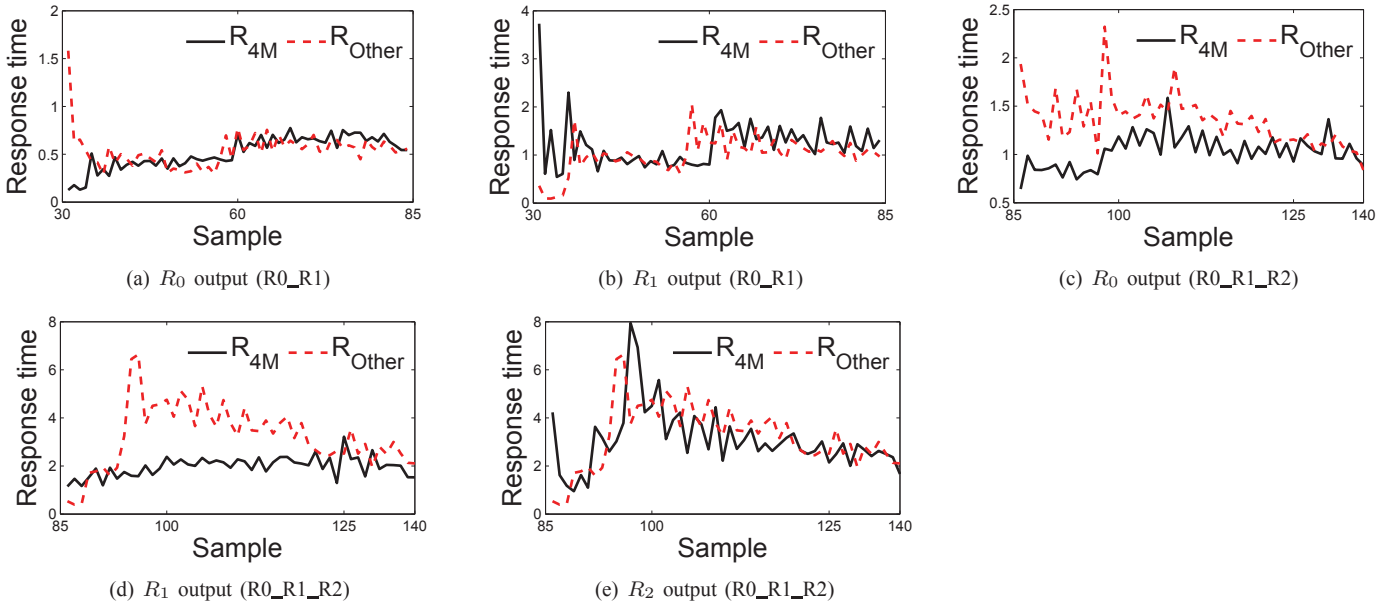
changes the performance differentiation factors investigating the issues under nonlinearities. Lastly, 4M-Switch control system is forced to operate under frequent transitions between operating modes in short time intervals to examine the issues of chattering. More experiment results can be found in [11], covering the performance management capabilities of this approach in large scale environments.

1) Operating Mode Transitions: We set performance differentiation factors of $VM_0 : VM_1 : VM_2 = 1 : 2 : 4$ and start with no workload assuming all three VMs are idling. At the 10th sample VM_0 receives a workload from 100 users. At the 30 to 35th samples workload of VM_1 increases to 100 users gradually. In order to investigate the disturbance rejection capabilities, at the 60th sample workload of VM_1 again increases by 50 users. At the 85th to 100th samples VM_2 workload gradually increases to 200 users. Finally, VM_0 goes in to idle mode at the 140th sample. This experiment simulates several operating mode changes, which necessitate the 4M-Switch to autonomously implement required control policies. As existing approaches do not have this ability to reconfigure, in order to compare the performance we use the same control system implemented by the 4M-Switch in each mode separately and execute them in corresponding sample periods as if that control system faced the workload at that point of time and thereafter till the next mode change. Figures 3 and 4 show the outputs of the control systems.

During the entire experiment the response time of VM_0 has been maintained lower than other two VMs under operating mode and significant workload changes, which is the specified performance differentiation objective. When a new VM is added to the environment (i.e. operating mode change), the generator has detected the change and fired an event to change the operating mode of the state-space analyzer (at 10th, 30th, 85th and 140th samples in Figure 3). Subsequently, the control policies have changed autonomously implementing appropriate control system safely with no instabilities. Although there are response time spikes at the beginning of the mode change due to sudden workload encountered by the applications, the response time settles down soon after because of the efficient CPU allocations implemented by 4M-Switch (see Figure 3(b)). When VM_0 is removed from the environment at the 140th sample performance differentiation between VM_1 and VM_2 has been implemented by the 4M-Switch showing adaptability and continuity of operation, with no instabilities which could have arisen from the aforementioned mathematical ill-conditioning issues of the existing works. In addition, even under operating mode switches, 4M-Switch has managed to implement bump-less transfers with no significant long term effects on the VM environment. Figure 4 shows a comparative performance of 4M-Switch and other control systems which operated individually in R0_R1 and R0_R1_R2 operating modes between 30th to 85th and 85th to 140th samples respectively. The individual control systems show significant performance degradation at the start up compared to 4M-Switch. This is because they were executed manually at the start of each mode, as opposed to automated transitions implemented by 4M-Switch. In fact, 4M-Switch shows smooth transactions for the highest priority class in all comparisons indicating that abrupt control system switches do not affect the performance of the control system.

2) Changes to Differentiation Objectives: The differentiation factors which drive the differentiation schemes could change over time because of differentiation requirement of the migrated VM. In this experiment, differentiation factors are changed at runtime, which will also enable us to investigate the issues of having a controller tuned to operate in a narrow region against the multi-model gain-scheduling control approach taken by 4M-Switch. Till the 60th sample performance differentiation ratio between VM_0 and VM_1 is 1:2, which changes to 2:1 soon after. These requirements force the control system to operate in $(y \geq 1)$ and $(y < 1)$ regions respectively. At the start VM_0 and VM_1 receive workloads from 100 users, however at the 15th sample workload of VM_1 increases by another 100 users. At the 45th sample VM_1 workload drops to 100 users again. At the 75th sample workload of VM_0 increases to 200 users. To compare the performance of 4M-Switch, here we execute controllers tuned for $y \geq 1$ (say controller-1) region and $y < 1$ (say controller-2) region separately. Figure 5 shows the outputs of the control systems.

When the control systems are operating in $y \geq 1$ region, controller-1 and 4M-Switch show similar performance. This is because 4M-Switch is also running the controller-1 during that period. Both control systems have settled down efficiently after the disturbance at 15th and 45th samples. In contrast, controller-2 has shown unstable behavior and high steady state errors. This is because in region $y \geq 1$, output changes at a high rate due to aforementioned nonlinearities (see Section V-D) generating high controller errors, which has been treated



(a) R_0 output (R0_R1) (b) R_1 output (R0_R1) (c) R_0 output (R0_R1_R2)

(d) R_1 output (R0_R1_R2) (e) R_2 output (R0_R1_R2)

Fig. 4. Comparison between outputs of 4M-Switch and other control systems in (R0_R1) and (R0_R1_R2) modes

aggressively by the high gains applied in controller-2. It is evident that controller tuned for $y < 1$ region cannot provide stable control in region $y \geq 1$ (see Figure 5(c)). Similarly, controller-2 and 4M-Switch has settled down under workload and performance differentiation objective variations compared to controller-1 after the 60th sample (see Figures 5(f), 5(d)). Controller-1 has smaller gains compared to controller-2, which means that controller-1 is not aggressive enough to handle the smaller output variations in $y < 1$ region. Compared to both controller-1 and controller-2, 4M-Switch has smoothly transitioned to $y < 1$ region and handled workload disturbances efficiently establishing most stable and safe control. It is therefore evident that the multi-model based gain-scheduling approach taken by 4M-Switch shows better control compared to the existing single model and controller based approaches.

3) Frequent Operating Mode Switches: In this experiment, we simulate sudden workload bursts and operating mode transitions in short time intervals, which could lead to significant performance issues in switching control systems. The experiment starts with workloads from 100 users for VM_0 and VM_1. In 25th sample VM_0 workload increases to 200 users from 100 users. At the 30th sample, the application in VM_1 suddenly becomes inactive. It becomes active again with 100 users at the 35th sample. VM_2 receives workload from 125 users at the 45th sample which again goes to idle mode at the 60th sample. Soon after at the 70th sample workload of VM_1 becomes zero. The performance differentiation factors are $VM_0 : VM_1 : VM_2 = 1 : 2 : 4$. Figure 6 shows the outputs of the system.

It is evident that 4M-Switch has provided stable performance differentiation, under sudden workload bursts and operating mode switches. The response time of the highest priority class has remained lower than other active VMs with much

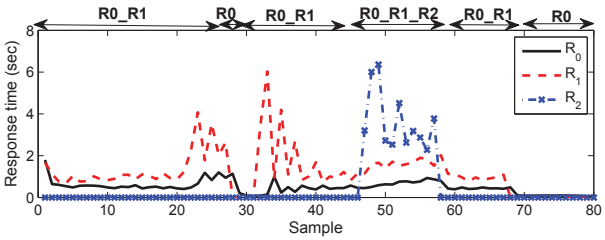
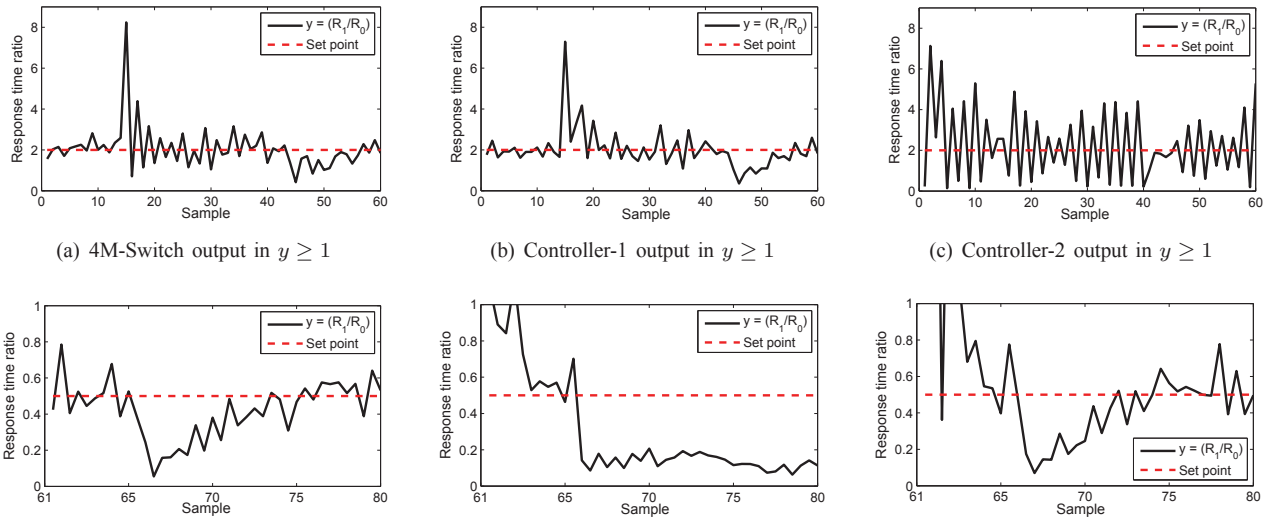


Fig. 6. Performance of 4M-Switch under frequent operating mode switching

lower priority. Although the response time at 25th, 30th, 45th samples is high because of the large disturbances (see Figure 6), no instabilities have been detected due to switching of the control systems. This indicates successful bump-less transfers and state management implemented by the 4M-Switch.

VIII. Conclusions

In this paper we have proposed the 4M-Switch supervisory control system design methodology to achieve performance differentiation and resource management objectives in VM environments. This new approach combines the capability of hybrid system design to capture the operating mode changes and feedback control mechanism to achieve the fine grained performance differentiation objectives. With this design methodology, a complex control system implementation can be divided into manageable operating modes and then corresponding control polices can be designed to achieve the required management objectives at the design time. At runtime, automatic switching between the modes and control systems provide self-adaptability with less or no human intervention. In addition, this paper provides a performance evaluation of 4M-Switch supervisory control system under a range



(a) 4M-Switch output in $y \geq 1$ (b) Controller-1 output in $y \geq 1$ (c) Controller-2 output in $y \geq 1$

(d) 4M-Switch output in $y < 1$ (e) Controller-1 output in $y < 1$ (f) Controller-2 output in $y < 1$

Fig. 5. Performance of 4M-Switch , controller-1 and controller-2 under changes in differentiation factors

of conditions, which has shown significant improvements in relation to performance, system stability and safety compared to the existing approaches.

REFERENCES

[1] ABDELWAHED, S., NEEMA, S., LOYALL, J., AND SHAPIRO, R. A hybrid control design for QoS management. In *24th IEEE Real-Time Systems Symposium* (2003), pp. 366 – 369.

[2] HELLERSTEIN, J. L., DIAO, Y., PAREKH, S., AND TILBURY, D. M. *Feedback Control of Computing Systems*. John Wiley and Sons, 2004.

[3] KOUTSOUKOS, X., TEKUMALLA, R., NATARAJAN, B., AND LU, C. Hybrid supervisory utilization control of real-time systems. In *Real Time and Embedded Technology and Applications Symposium* (2005), RTAS'05, pp. 12–21.

[4] KUSIC, D., AND KANDASAMY, N. Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems. In *IEEE International Conference on Autonomic Computing* (2006), pp. 74 – 83.

[5] KUSIC, D., KEPHART, J., HANSON, J., KANDASAMY, N., AND JIANG, G. Power and performance management of virtualized computing environments via lookahead control. In *International Conference on Autonomic Computing* (2008), pp. 3 –12.

[6] LU, C., LU, Y., ABDELZAHER, T., STANKOVIC, J., AND SON, S. Feedback control architecture and design methodology for service delay guarantees in web servers. *IEEE Transactions on Parallel and Distributed Systems 17*, 9 (2006), 1014 –1027.

[7] LU, Y., ABDELZAHER, T., LU, C., SHA, L., AND LIU, X. Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers. In *IEEE Real-Time and Embedded Technology and Applications Symposium* (may 2003), pp. 208 – 217.

[8] LU, Y., ABDELZAHER, T., LU, C., AND TAO, G. An adaptive control framework for QoS guarantees and its application to differentiated caching. In *IEEE International Workshop on Quality of Service* (2002), pp. 23 – 32.

[9] PADALA, P. *Automated Management of Virtualized Data Centers*. PhD thesis, University of Michigan, 2010.

[10] PAN, W., MU, D., WU, H., AND YAO, L. Feedback control-based QoS guarantees in web application servers. In *International Conference on High Performance Computing and Communications* (2008), pp. 328 – 334.

[11] PATIKIRIKORALA, T., COLMAN, A., AND HAN, J. Multi-mode-multi-model supervisory control framework for performance differentiation in virtual machine environments. Tech. rep., 2014. http://www.ict.swin.edu.au/personal/tpatikirikorala/Research.htm.

[12] PATIKIRIKORALA, T., COLMAN, A., HAN, J., AND WANG, L. A multi-model framework to implement self-managing control systems for QoS management. In *International symposium on Software engineering for adaptive and self-managing systems* (2011), pp. 218–227.

[13] PATIKIRIKORALA, T., COLMAN, A., HAN, J., AND WANG, L. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Symposium on Software Engineering for Adaptive and Self-Managing Systems* (2012), pp. 33–42.

[14] PATIKIRIKORALA, T., WANG, L., COLMAN, A., AND HAN, J. Hammerstein-Wiener nonlinear model based predictive control for relative QoS performance and resource management of software systems. *Control Engineering Practice 20*, 1 (2011), 49 – 61.

[15] STIVER, J., ANTSAKLIS, P., AND LEMMON, M. Interface and controller design for hybrid control systems. In *Hybrid Systems II*, P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, Eds., vol. 999 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1995, pp. 462–492.

[16] STIVER, J., ANTSAKLIS, P., AND LEMMON, M. A logical {DES} approach to the design of hybrid control systems. *Mathematical and Computer Modelling 23* (1996), 55 – 76.

[17] WANG, Z., ZHU, X., AND SINGHAL, S. Utilization and slo-based control for dynamic sizing of resource partitions. In *Distributed Systems, Operations and Management* (2005), pp. 133–144.

[18] YOUBIN, P., VRANCIC, D., AND HANUS, R. Anti-windup, bumpless, and conditioned transfer techniques for PID controllers. *Control Systems Magazine, IEEE 16*, 4 (1996), 48–57.

[19] ZHU, X., WANG, Z., AND SINGHAL, S. Utility-driven workload management using nested control design. In *American Control Conference* (june 2006), p. 6 pp.