A multi-model framework to implement self-managing control systems for QoS management

Tech-Report

Tharindu Patikirikorala, Alan Colman, Jun Han
Swinburne University of Technology
Victoria, Australia
tpatikirikorala, acolman, jhan@swin.edu.au

Liuping Wang
RMIT University, Melbourne,
Victoria, Australia
liuping.wang@rmit.edu.au

ABSTRACT

Many control theory based approaches have been proposed to provide QoS assurance in increasingly complex software systems. These approaches generally use single model based, fixed or adaptive control techniques for QoS management of such systems. With varying system dynamics and unpredictable environmental changes, however, it is difficult to design a single model or controller to achieve the desired QoS performance across all the operating regions of these systems. In this paper, we propose a multi-model framework to capture the multi-model nature of software systems and implement self-managing control systems for them. An extendable meta-model and a class library are introduced to implement such self-managing control systems. The proposed approach is also validated and compared to fixed and adaptive control schemes through a range of experiments.

Keywords

Reconfiguring control, feedback control, adaptive control, self-managing systems, quality of service, multi-model

1. INTRODUCTION

The increasing complexity and scale of the software systems demand effective and efficient mechanisms to manage them in a systematic way. E-commerce, banking, utility computing systems and many other businesses rely on software systems to deliver desired functional services, while maintaining the non-functional properties. Response time, throughput, availability and security are some of these non-functional (quality of service) attributes such systems must maintain at an acceptable level to avoid customer dissatisfaction. The existing traditional methods such as manual tuning have proven to be costly and error-prone, while fixed and ad-hoc threshold based polices depending on the peak demands is increasingly difficult to design due to lack of a formal/systematic design process [4, 8]. Thus, it is important to implement methodologies that address these limitations by integrating runtime decision making (self-managing) capabilities into software systems, with reduced or no human intervention.

Control theory has been looked at in the last few years as one such design technique to incorporate self-management into software systems to achieve performance quality objectives [11]. Use of control theory to control web server systems [12, 22], cache/storage systems [16, 23] , data centers/server clusters [8, 19] and multi-client class systems [17, 19, 22, 37] are such attempts. Despite these attempts, application of feedback control in real/production systems is not widely adopted in software engineering practice, unlike other engineering disciplines [18, 39]. In [3, 11, 15, 39], a number of challenges and limitations for application of control theory to QoS management are identified. They include the difficulties in developing accurate system models, requirements for requiring rigorous mathematical analysis, the inherent nonlinear system behavior/dynamics and the discrete/ discontinuous control inputs.

Because of such difficulties, approaches that attempt to model the system behavior usually focus on a particular operating region under certain operating conditions that can be characterized using a single linear model. However, such linearization is inherently problematic because the software systems have to work in the entire operating region with changing operating conditions and deal with un-modeled system dynamics/behaviors. For instance, e-commerce systems may face sudden intensive workloads when promotional offers are run or when referenced by a high-traffic site (the so called 'slash-dot' effect). The workloads may also vary dramatically depending on the time of day (e.g.: stock market applications) or the time of year (e.g. tax office sites). In addition to these conditions, a software system may change due to bug fixes and component failures/ replacements. With these concerns approximating a system as a single linear model is difficult and carries uncertainties, leading to issues in performance management and flexibility in control system design.

The highly nonlinear system behavior suggests that multiple models may be needed to effectively model and control such software systems. In this paper we refer to this need for multiple models to characterize the behavior of a software system as the *multi-model nature* of the system.

The need to characterize the behavior of software systems using multiple models is further motivated from experimental results in some of the existing literature. For instance, in [16] two linear models were implemented for cases where data is retrieved from, respectively, cache or disk. The authors pointed out those two models are drastically different, consequently designing a controller to satisfy both conditions was difficult. They further mentioned that a single fixed model cannot handle dynamic multiple concurrent workloads and different operating conditions or system configuration changes. Similar arguments are made in [24] for the case of cache server system. Lu et al. implemented a control theoretic approach to achieve absolute and relative delay guarantees for Apache web servers [22]. In their experiment for different workload conditions different models were implemented. However, they selected the model generated by one workload setting for the controller design. Although this design provided stability under other conditions, the performance achieved under other conditions were sacrificed. Similarly, [36, 40] discuss the bimodal (under-loaded and overloaded) characteristic in time varying workloads of data centers and the demand for intelligent switching control approach to provide control under these conditions.

To address the above discussed multi-model nature of the system, the possible solutions could be: (i) design multiple static models that can capture and cope with the system behavior in different operating region/conditions; (ii) develop models/algorithms that can adapt/learn at runtime; (iii) combination of the above. After capturing the behavior, multiple controllers or self-tuning controllers have to be designed and suitable controller has to be selected at runtime to provide control decisions. These solutions demand self-managing control systems which can dynamically reconfigure themselves and select the suitable model and controller in response to changing environmental and system conditions. Furthermore, such runtime

reconfigurations should achieve the desired performance objectives while maintaining the stability of the system and providing a generalized design process.

In this work, we propose an approach designing self-managing control systems for the performance management of software systems that have the multi-model behavior. Motivated by a control theoretic approach called Multi-Model Switching and Tuning (MMST) adaptive control, this paper makes the following major contributions:

- An extendable meta-model providing the basis for designing MMST-based self-managing control systems for software systems,
- A class library that implements the meta-model and can be used in research and real software systems,
- A comparative study on the performance of the proposed MMST-based self-managing control scheme with respect to existing fixed and adaptive control techniques.

The remainder of the paper is organized as follows. Section 2 provides the background and related work. Section 3 covers the details of MMST adaptive control systems. In section 4, a meta-model for designing self-managing control systems based on MMST is introduced, while the experiment results are presented in section 5. Finally, we conclude with a discussion and an outline of future work in section 6.

2. BACKGROUND AND RELATED WORK

Control system and design: Figure 1 shows a block diagram of a feedback control system. The system controlled by the controller is referred to as the target system. The target software system provides a set of performance metrics for properties of interest (e.g.: response time) referred to as outputs. Sensors monitor the outputs of the target system, while the control input (e.g.: resource allocation) can be adjusted through actuators to change the behavior of the system. The controller is the decision making unit of the control system. The main objective of the controller is to maintain the output of the system sufficiently close to the target operation, by adjusting the control input. This target operation is translated in control system terms as the set point signal, which gives the option for the control system designer to specify the goal/desired value of the output.

The control system design generally consists of two main steps. Firstly, a formal relationship between the control input and the output has to be constructed. In control theory this relationship is referred to as the behavioral *model* of the system. System identification (SID) is a method used to construct the model via the measurement of input and output data [21]. The model of the system is then utilized in the second step, which includes controller design, simulation, analysis and testing. The ability of the feedback controller is to achieve the operational goals, while reacting to unpredictable disturbances and un-modeled system dynamics. In addition, there are well established formal techniques and methodologies to design, develop and analyze the performance specifications (e.g.: stability, settling time and overshooting) of the control system.

Using the above general methodologies, various control systems such as fixed gain, adaptive and reconfiguring controllers can be designed. The following sections discuss the design, advantages and limitations of these control techniques, together with existing applications in the literature that utilized them for QoS management.

Fixed-gain control: The structure of a fixed-gain control scheme is similar to Figure 1. As mentioned, SID experiments are conducted offline, in order to build the model of the system. Typically, autoregressive exogenous input (ARX) models are used to describe the behavioral model of the system [21, 39]. The standard form of the ARX model is as follows:

ARX(n,m,d)

$$y(k) = \sum_{i=1}^{n} a_i y(k-i) + \sum_{j=1}^{m} b_j u(k-d-j) \qquad (1)$$

where, n,m – order of the model , a_i, b_j- parameters of the model,

d – delay (time intervals taken to observe a change of input in the output) and k- current sample instance.

The order, parameters and delay of the system model are determined using the input and output data gathered from SID experiments. Using the model created in the model identification phase, the controller is designed with the objective to reduce the variation between the desired set point and the actual output signal. The different variations of the Proportional Integral Derivative (PID) controller are widely used for this purpose due to their robustness against modeling errors, disturbance rejection capabilities and simplicity [6]. In these controllers there is a parameter called *gain* which can be adjusted to achieve the desired performance specifications.

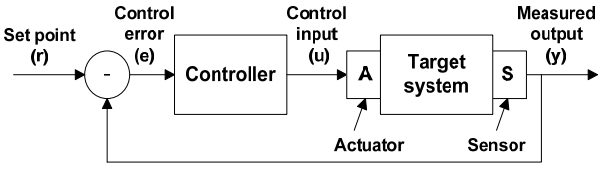


Figure 1. Block diagram of a control system

Fixed gain control designs have a number of advantages. The fixed gain controllers are useful to alter the control input to achieve performance goals with less human intervention. In addition, such controllers are relatively easy to design and formal techniques exist for stability analysis and tuning of the controller. They can deliver the desired performance up to some extent under varying workloads,

changing operating conditions. However, fixed-gain controllers have some limitations. Under dynamic and unpredictable variations, the performance of the fixed-gain controllers can degrade because the control algorithm and the gains remain unchanged at run time. In addition, if there are a number of dimensions across which operating conditions can vary, selecting a gain to satisfy all operating conditions can be difficult (as shown in [16, 23]). Thus, the single fixed gain controller alone cannot provide an effective solution to deal with the multi-model nature of software systems. However, most of the existing approaches (e.g.: [5, 9, 12, 22, 40]) have used fixed gain control in their design.

Adaptive control: Adaptive control addresses some of the limitations of fixed gain controllers by dynamically estimating the model parameters and adjusting the gains of the controller to achieve the high-level design objectives. As a consequence, changes in the system model due to varying conditions are captured and it is integrated into the controller design online. In this sense adaptive control captures the multi-model behavior of the software systems. However, a basic assumption of adaptive control is that the model parameters remain constant or vary slowly over time [2, 27]. Otherwise, it is well known in the literature that performance can degrade, causing large transient responses and temporal instabilities [25, 39]. Fast changing conditions can be seen in software systems, such as sudden workload spikes, 'slashdot'-effects, component failures and the garbage collection process. Adaptive control also has other limitations, such as computational cost due to online estimation and design. The start-up performance may not be satisfactory since it takes time to come up with the estimations. Furthermore, these methods require the input signal to contain sufficient range of frequencies to excite the system (*persistently exciting* condition) for fast and accurate model estimation [2, 16].

The Self-tuning Regulators (STR)[2, 12] are often been applied as an adaptive control scheme in software systems. There are two types of STR designs. The *indirect*-STR uses the estimations of the system model to derive the controller parameters. However, in *direct*-STR, controller parameters are derived directly without system model estimations by re-parameterization of the model with controller parameters [2, 7]. There are a number of approaches that have used STR in their controller design. Lu et al. used indirect-STR in the differential caching problem [24]. They point out that adaptive controller provides design flexibility, and due to online model estimation it can be ported to different technical environments without re-modeling overhead. Karlsson et al. built a direct-STR to regulate throughputs of different workloads in storage systems [16]. Besides, [17, 20, 31] have also used adaptive control in their designs.

Reconfiguring control: The adaptive control scheme provides more flexibility compared to the non-adaptive scheme by adjusting the controller parameters online. However, the controller algorithm and the organization of the components (e.g.: filters) in the loop stays fixed overtime [18]. For different operating conditions and disturbances different control algorithms or loop organization may provide better control [33]. The main idea of the reconfiguring control scheme is to change the control algorithms, models and control loop organization to deal with the changing operating regions of the system and environmental conditions. In addition, some of the requirements for adaptive control can be avoided (such as persistently exciting signal, slowly varying conditions) by combining multiple fixed gain controllers together and selecting a suitable one at runtime. Furthermore, fixed and adaptive control strategies can be combined together to improve performance under fast varying conditions [25, 28]. As such, this approach is useful in capturing the multi-model nature of the software system and selecting appropriate control loop configurations at runtime. However, there are tradeoffs between the design complexity and the runtime overhead on the system due to the design and evaluation of the multiple models and controllers [12]. In addition, to come up with proper reconfiguration schemes prior information about the system and environmental conditions may be required. *Chattering* is another issue that can occur in reconfiguring control. Chattering occurs when a system frequently changes between controllers or different loop configurations without providing desired control. This could lead to drastic performance degradations [18].

There are only a few approaches that uses this type of techniques. The *Self-Controlling* software approach proposed by the Kokar et al. discusses the concept of reconfiguration loop in [18]. Some architectural approaches were proposed in [10, 33] to realize reconfiguration control for software systems. However, these approaches concentrate on state management when the components of the loop are removed and replaced. In addition, the reconfiguration is triggered by using ad-hoc rules, which could cause design issues when there is a large number of environment and system conditions. Furthermore, they do not provide stability guarantees which are vital to the control system design.

The work in [16, 22, 36] identified the need for methodologies to capture the multi-model nature of the system behavior, but they opted to use either a single fixed gain controller or an adaptive controller in their design, which have limitations as discussed above.

In summary, the single model based fixed gain or adaptive control schemes do not sufficiently capture the multi-model behavior of software systems, and consequently cannot provide an adequate solution to their QoS performance management. There is a clear need for better ways to integrate multiple models and controllers into software systems and achieve the desired system behavior through their selection or reconfiguration depending on the operating conditions at runtime. Such integration should provide accurate, fast and robust decisions under varying conditions, without adversely affecting the stability of the system. In this work, we investigate a reconfiguring non-linear feedback control approach called the Multi-Model-Switching and Tuning adaptive control (MMST). The MMST adaptive control is a control scheme that has capabilities to integrate multiple

models and controllers to the control system. It is an extension of the adaptive control and reconfiguration control schemes, and has the potential to improve the performance of the systems showing multi-model behavior.

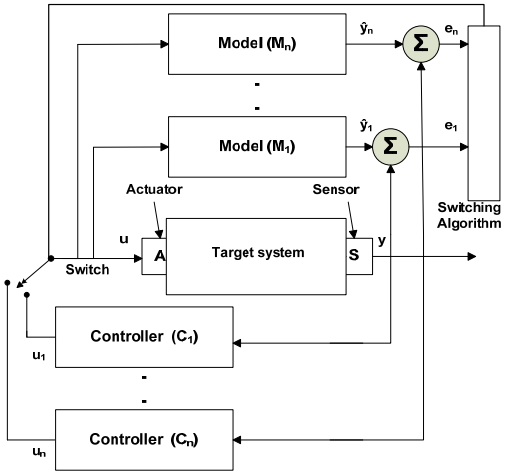


Figure 2. Block diagram of MMST adaptive control

3. ADAPTIVE CONTROL WITH MULTI MODEL SWITCHING AND TUNING

In this section, we provide an overview of the adaptive control approach with Multi Model Switching and Tuning (MMST) [29] which supports various multi-model schemes. The MMST adaptive control was proposed by Narendra and Balakrishnan to improve the transient response of adaptive control systems in the presence of model uncertainties [25]. It has been applied in domains such as robotic manipulators, process control and flight control systems [27]. Our work focuses on applying MMST to QoS performance management of software systems.

The MMST adaptive control is a concept inspired by biological systems [28]. Biological systems have the ability to select an appropriate action for a specific situation from a collection of behaviors. MMST uses the same concept by selecting the most suitable controller for the current environment that the system is in. Figure 2 shows the main components of MMST. The target system has control input u and measured output y. There are *n* number of models (M_1, M_2….M_n) representing different operating conditions that provide estimations for the system model, simultaneously. The estimates from these *n* models are denoted by $\hat{y}_1$, $\hat{y}_2$…. $\hat{y}_n$. Similarly, there may be *n* controllers, with each corresponding to a model.

Although, there are multiple controllers, only a single controller can be connected in the control loop to make control decisions at runtime. Thus, at the runtime the most appropriate model and controller for the system and environment condition have to be selected to make the control decisions. The responsibility of the switching algorithm is to select the appropriate model and corresponding controller based on some criteria that will improve the performance of the controlled system. There are multiple switching algorithms discussed in [28]. All of these algorithms are based on the prediction errors of the models ($e = y - \hat{y}$). The prediction error provides the indication that at the current instance, which model fits the current operating conditions of the system. Hence, integration of this model and the corresponding controller in the control loop should improve the performance of the control system [28]. The model evaluation and selection steps of the switching algorithms are summarized as follows:

Model evaluation:

$$J_i(k) = \alpha\, \mathbf{e}^2(k) + \beta \sum_{r=0}^{k} \mathbf{e}^2(r)\ , \forall \mathbf{i} = \mathbf{1}, \mathbf{2}..\mathbf{n} \qquad (2)$$

Model selection:

$$J_j(k) = Min\{J_i(k)\}\ , \quad \mathbf{i} = \mathbf{1}, \mathbf{2}..\mathbf{n} \qquad (3)$$

where α and $\beta \geq 0$ are parameters that should be carefully decided by the designer. If $\alpha > 0$, $\beta = 0$, only the instantaneous part is utilized. In this case switching may be frequent, leading to performance degradation [25]. If $\alpha = 0$, $\beta > 0$ the long term component is active, hence the switching may be infrequent, again possibly leading to performance degradation [25]. In the model evolution step prediction errors are used to calculate the *J* index of all the models with equation (2). In the model selection step the model that produced minimum *J* index is selected and then the corresponding controller is integrated into the control loop. In addition, T_{min} is another parameter called *waiting time period* which specifies the time that the control system has to wait before selecting the next controller to control the system [25, 29]. T_{min} is recommended to be set to a small value if the conditions are fast varying, in order to avoid instabilities.

Depending on the application characteristics, the designer can decide on the above parameters to achieve the desired performance. The index in equation (2) is most suitable in time-invariant environments (i.e. systems that does not depend on absolute time [21]). For time-varying environments long term error accumulation will affect the *J* indexes. In such conditions the performance can be improved by calculating the performance index in a finite window (T) as illustrated in equation (4).

$$\boldsymbol{J_i(k) = \alpha\, e^2(k) + \beta \sum_{r=k-T}^{k} e^2(r)\ , \forall i = 1, 2..n} \qquad (4)$$

The above discussion presents the general design approach of MMST. Going further, different types of multi-model schemes have been evaluated and formal stability proofs are provided in [14, 29]. These multi model schemes are as follows.

Type 1: *All adaptive models*- in this scheme all the models (1, 2…n) are represented by adaptive identification (estimation) algorithms [21]. The corresponding controller uses the parameter estimation to come up with the control input *u*. This scheme is computationally inefficient. In addition, if the environment remains unchanged for a long time, all the adaptive models will converge to the same parameter neighborhood which reduces the advantage of having multiple models. In addition, when a sudden disturbance occurs, models may not react to it rapidly without re-initialization due to the inherent characteristics of adaptive control [29].

Type 2: *All Fixed models*- in fixed models the above limitations are not seen. While fixed controllers are generally not regarded as an adaptive technique, but because of the switching capabilities this scheme can be considered as an adaptive control technique. However, fixed models can only represent a finite number of operating regions or conditions. Apart from that, fixed models should capture the dynamics of different operating and environment conditions assuming that there is always one of the models that closely approximate the system behavior. To satisfy this assumption and the stability requirements we may have to build a large number of fixed models.

Type 3: *One Adaptive model and one Fixed model* – here initially the fixed model may be selected most of the time since adaptive model takes time to converge at the startup. However, when the adaptive model converges, it will outperform the fixed model often. This scheme is simple and addresses some of the limitations in the above two schemes.

Type 4: *Two Adaptive models and n-2 Fixed models*– this scheme is regarded as one of the best schemes based on simulation results [28, 29]. From prior knowledge of the system's operating and environment conditions, *n-2* number of fixed models can be designed. Then, one of the adaptive models is run free of interference to capture the dynamics not captured by the fixed models. However, if a sudden disturbance occurs this model may not converge fast enough due to the delay of convergence. The other adaptive model is re-initialized with the parameters of the best model in the current instance. The main purpose of this model is that after re-initialization it may converge fast to the new system model so that the systems can respond faster to sudden disturbances. To achieve effective performance, the design of the fixed models has to be done after carefully analyzing the available prior knowledge on the system and its environment.

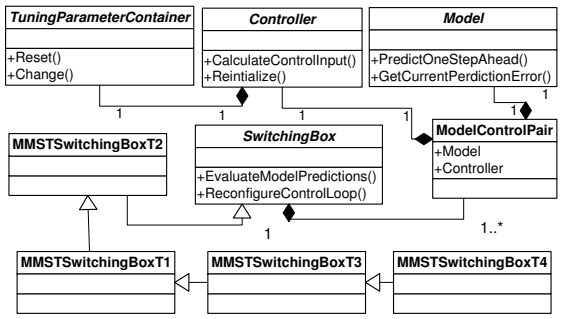


Figure 3. The meta-model

Interested readers are referred to [28, 29] for the details of the stability proofs of these multi-model schemes in guaranteeing that the system will not be unstable due to the switching and tuning behavior of these schemes. The above discussion provides the objectives, features and some limitations of different MMST adaptive control schemes. MMST is an *indirect* adaptive control scheme, since it depends on the model selection or estimation before providing control decisions. Furthermore, it is a form of reconfiguration control because MMST changes the models, controllers and components in the control system at runtime, depending on the changing conditions. More importantly it provides the ability to incorporate multi-model behavior of the software systems while providing formal stability proofs, which are two important factors in control system design.

The main questions arising from the above schemes are: (i) which schemes to use, (ii) how many fixed/adaptive models and how to identify them and (iii) how to configure the switching algorithm parameters. Answers to these questions are application or requirements dependent. At the design time, suitable models and controllers have to be formulated, depending on the available knowledge about the operating conditions, system behavior and physical analysis (However, precise/accurate knowledge is not required) [29]. Considering the characteristics of the software systems, scheme 2 and 4 maybe most suitable. For systems where some prior knowledge is available about the system and environment conditions, different fixed models can be approximated and integrated utilizing scheme 2. For instance, e-commerce systems that may face high workloads due to promotional offers can simulate these conditions together with another model to represent the normal operating conditions. Also, this would be useful in the cases of [16, 22] discussed in sections 1 and 2. If there is less/no prior knowledge or the systems that change often need the integration of adaptive models with fixed models as in scheme 4. Those could be systems that frequently change due to new features/bug fixes and other unknown environmental conditions. The number of fixed models depends on the knowledge of the conditions that can be used and simulated to design the system identification experiments. However, in the case there is little prior knowledge, fixed models can be uniformly placed in the model parameter space [26]. Since software systems are typically nonlinear and time-varying [12], the switching scheme in equation (4) may be most suitable by setting $\alpha=0$, $\beta = 1$ [28] to achieve predictable and consistent switching. Details of some of these recommendations are validated experimentally in section 5.

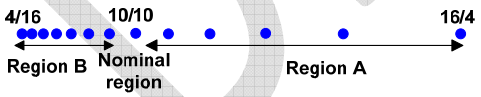


Figure 5. The operating points and regions

4. META-MODEL

In this section a meta-model to design and develop MMST based self-managing control systems for software systems is provided. Figure 3 shows the proposed meta-model at a high-level of abstraction. This meta-model can be used/extended in many ways to design self-managing control systems. Below, we first discuss the structure of the meta-model. Then we show how this meta-model can be used/extended to integrate different control algorithms and switching schemes, before presenting the realization of this meta-model as an executable class library.

The *SwitchingBox* class is an abstract class which is extended to implement the switching and selection algorithms. As shown in Figure 3, among many methods *EvaluateModelPredictions* and *ReconfigureControlLoop* methods implement various

switching algorithms and the reconfiguration techniques. The *SwitchingBox* class requires a collection of model-controller pairs representing the different operating conditions. As discussed in section 2, these models have to be constructed using SID experiments under desired operating conditions that need to be captured. Similarly, suitable controllers have to be designed. The *ModelControlPair* serves as a container for a model and the corresponding controller. The *Model* class is the super class of different models that can be integrated into this meta-model. For instance, first order, high-order and adaptive models are specializations of this class. It provides facilities to calculate the look-ahead predictions and prediction errors, which are vital for the switching schemes. The *Controller* class is the super class of all the controllers that need to be integrated to this meta-model. For instance, fixed-gain and STR controllers are the child classes of this class. These child classes must implement *CalculateControlInput* method depending on its control law. Finally, the *TuningParameterContainer* class abstract away different tuning parameters that need to be provided to the controller. The controllers that need different tuning parameters need to extend this class.

As shown in the bottom of Figure 3 MMSTSwichingBoxT2 class extends the *SwitchingBox* class and implements general functionalities which is either extended or overridden by *T2,T3* and *T4* classes to achieve required objectives.

Extendibility: The meta-model can be extended to integrate different types of controllers. For example, if linear quadratic controllers [12] or model predictive controllers need to be integrated into the system, the implementation of the controllers are done by extending the *Controller* class in the meta-model. Similarly, if nonlinear (e.g.: neural network) models are needed, the *Model* class can be extended. Even with these extensions, the rest of the meta-model does not need to be changed. Similarly, if new switching schemes (e.g.: gain scheduling [2]) have to be implemented the *SwitchingBox* class can be extended without affecting other implementations.

Implementation: The meta-model in Figure 3 is currently implemented using C#.Net as an extendable control library [1]. The implementation provides following off-the-shelf standard control algorithms

a. PID controller – which can be configured as 6 different types of controllers (algorithm [30])

b. Model predictive controller (MPC) (algorithm [35])

c. Indirect self-tuning regulator (algorithm [2])

d. Self-tuning PID controller (algorithm [12])

In addition, the first order, higher order and adaptive models (implementing the recursive least squares algorithm [2]) are also available. Furthermore, all four MMST schemes with configurable switching algorithms are provided. Any of the above controllers can be used in the design. For example, PID controllers and the model predictive controllers can be combined in a scheme depending on the requirements. This class library can be readily integrated into a software system implemented in any of the programming languages supported by the .Net framework. For systems implemented in other languages (e.g.: Java), this class library may be integrated as a web service component.

5. Validation

This section provides an introduction to a case study and a prototypical system designed simulating a real world application scenario. Depending on the characteristics of the case study we provide a systematic design process to design and integrate multiple models and controllers to capture the different operating regions/conditions and the multi-model behavior. Then, the class library implemented based on the meta-model discussed in section 4 is used to implement the self-managing control system for the prototypical system using the MMST type 2 scheme. Finally, we validate the proposed approach with a comparative study with two fixed gain controllers and an adaptive controller.

5.1 Case study

The software system in Figure 4 illustrates one of the common resource allocation problems in travel (flight) reservation systems. The server has to communicate with a 3rd party supplier to respond to client requests (e.g.: check/ book flight availabilities). However, the 3rd party supplier only provides a limited number of sessions (20 in the scenario discussed in this paper) to communicate with them. Assume that, two client classes (travel agents A and B) are interested in the services provided by this server. The objective of the server is to allocate the limited number of sessions provided by the 3rd party supplier between agents A and B, and maintain the response time of the requests at an acceptable level, under varying workloads. Additionally, there could be constraints/policies on a minimum number of resource units that has to be maintained for specific client classes to avoid starvation of resources.

A prototype of such a reservation system was developed implementing the architecture shown in Figure 4[1]. The services of the sever can be accessed by clients by connecting to the server socket. After connection is made the clients can send different messages invoking different service methods. When a message is received by the message queue, *time stamp-#1* is applied and then the request is classified according to the client class and put to the relevant client queue. The scheduler accesses these queues in a first come first serve (FIFO) fashion, and assigns these messages to a virtual application instance with client specific method pointers and virtually partitioned resources (e.g.: session handlers) to be sent to the 3rd party supplier. When the server receives the response from the 3rd party supplier, it is sent back to the client through the socket. The *time stamp-#2*, is applied before the response is written to the client socket.

In this work we design a relative guarantee feedback [22, 23] control system to allocate sessions depending on the varying workloads of A and B so that the response time ratio could be maintained within an acceptable range depending on the relative importance of the client. The response time for a single request is the time difference between time stamp-#1 and #2 measured in seconds. An average response time of the workloads over a 2 second sampling window is used as measured output to trade-off between monitory overhead and reaction to sudden disturbances. Let us denote the response time of workloads A and B as R_a and R_b respectively. Similarly, the session allocation between workloads A and B as S_a and S_b where, $S_a + S_b = 20$. To incorporate the relative importance between client classes in relative guarantee scheme the control input is the ratio of session allocations, represented by S_a/S_b and the output variable is the ratio of the average response time of the workloads, represented by R_a/R_b. For notational simplicity let us denote S_a/S_b and R_a/R_b as u and y respectively. The control objective of this control system is to maintain a constant response time ratio between the two workloads A and B. The costs, penalties and response time requirements can be used to decide this relative importance. In addition to the main control objective, the scheduler is constrained with an ad-hoc policy/constraint with the minimum number of sessions $S_a, S_b \geq 4$. (refer [22], for more details on the relative guarantee feedback scheme).

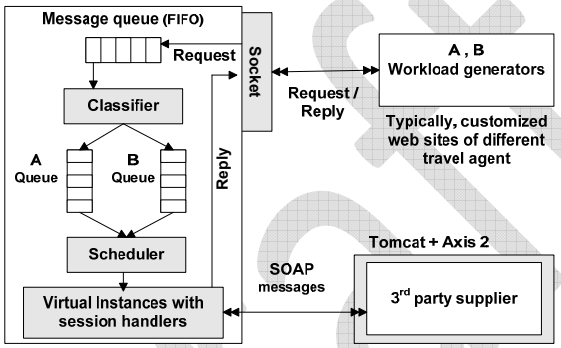


Figure 4. The target system

When the above requirements and policies are embedded in the design, the control input S_a/S_b, can only take certain discrete values. With $S_a = 20 - S_b$ and $S_a, S_b \geq 4$, the possible operating points of the system belong to (S_a/S_b = 4/16,5/15,6/14…….15/5,16/4). Assuming the point where both classes getting equal sessions (S_a/S_b =10/10) is the normal condition, we call the operating region where class A gets more resources as region A (S_a/S_b = {11/9,…15/5,16/4}). Similarly, when B gets more resources we say is in region B (S_a/S_b = 4/16,5/15,6/14...11/9). Figure 5 shows the operating points (S_a/S_b). Note that the operating points are unequally spaced. In region A, spacing increases towards the final operating points where as in region B spacing decreases towards the final operating points. Such highly discontinuous operating points could affect the linearity of the system, so that the performance of a linear controller may degrade drastically when it is operating away from the nominal operating region. In section 5.3 we show this concern experimentally. The main issue is due the fact that with the discontinuous operating points it is hard to decide the gains (parameters) for the fixed gain controller, which are required to achieve the desired performance when the controller is operating in region A and B. When the controller is optimized to operate

[1] The prototype system was deployed in a Intel Core(TM)2duo E8400 CPU@3.00 GHz and 2.99 GHz and 2 GB memory. To simulate two client workloads we used tailor made workload generators which was deployed in a machine with Core(TM)2duo E6550 CPU@2.33 GHz and 2.33 GHz and 3 GB memory. The target software system and the controllers were built in *C#.Net* and the 3rd party supplier component was designed using Java as a web service deployed in a Tomcat 5.5 with Axis 2 web service engine. The machines were connected via 1 Gps Ethernet.

in region A, performance in region B drastically degrades and vice versa. Thus, this case study system shows the multi-model behavior, leading to the need for integration of the adaptive control or multiple models and controllers.

5.2 Design process

As pointed out in the previous section, the above prototype system has two regions because of the discontinuous control inputs. Using this knowledge we can design two models to approximate the behavior/dynamics and represent those two regions.

Model Identification: Firstly, using the prototype system we have designed a SID experiment, simulating the workloads that are suitable to capture the dynamics of region A. The control inputs under region A, S_a/S_b = {10/10, 11/9,…15/5,16/4} are used to design a pseudo random signal [32] for the SID experiment. The nominal pint (10/10) was also added to this region without loss of generality, to reduce the modeling effort (i.e. to avoid representing a single operating point as a region/model). 30 requests per second (req/sec) and 10 req/sec was selected to simulate A and B workloads. Then the system output is monitored by applying a randomly selected control input from the aforementioned set for 4 consecutive sample periods. This experiment was carried for 600 sample periods and the gathered input-output (*u-y*) data was used to create the model for region A. To construct and validate the model, *u-y* data pairs were divided into two sets called the *estimation set* and *test set*. The data samples till 400 periods were included in the estimation set and were used to derive the model of the system. The rest of the data samples formed the test set, to validate the model. A 1st order ARX model (in equation (1)) was used to fit the data with sufficient accuracy (which we call *model-A*). Although higher order models provide better fit we opted to use a first order model for simplicity and avoid over-fitting [12]. A similar experiment was carried out in the B region as well. In this case the input signal was constructed by the control inputs of B region, S_a/S_b = {4/16,5/15,6/14...11/9}. The workloads of 30 req/sec and 10 req/sec were used to simulate B and A workloads respectively. Again, a 1st order ARX model was used to fit the data with sufficient accuracy (which we call *model-B*). Finally, another experiment was conducted to capture the dynamics of the system in the entire operating region with the set S_a/S_b = {4/16,5/15,6/14…….15/5,16/4}. For this experiment we used workloads of 20 req/sec to simulate both A and B and fitted another 1st order ARX model (which we call *model-AB*). The model parameters and structure of *model-A*, *model-B* and *model-AB* are shown in equation (5), (6) and (7) respectively.

Controller Design: After the models have been identified the next step is to construct the suitable controllers. The PI control algorithm was selected for this purpose. Two types of discrete PI control laws are illustrated in equation (8) and (9). K_p (propositional gain) and K_i (integral gain) are the tuning parameters of the PI controller, which have to be selected to achieve desired system performance and stability. The formal design methodology called the *pole-placement design* [12] was used to derive the gains of the controllers.

Equation (8) illustrates the *position/full-value form* of the PI control law, which is used widely to manage software systems (e.g.:[9, 12, 22, 32]). However, in the switching control systems this control law may cause performance issues. For instance, when the control is switched from one controller to another due to a large disturbance, the integral term (the second term) in equation (8) may have different values, which may lead to implementation of different/unsuitable control input by the new controller. As a consequence, there may be large transient output responses, called as bumpy transfers [38]. In contrast, the *velocity/incremental form* of the PI law illustrated in equation (9) is well known to implement *bumpless transfers* in the case of mode and controller switching systems [38]. In this work, the velocity form of PI control is implemented to achieve smooth/bumpless transitions under controller switches in the control system.

$$\boldsymbol{y(k+1) = 0.67\, y(k) + 0.13\, u(k)} \qquad (5)$$

$$\boldsymbol{y(k+1) = 0.65\, y(k) + 0.79\, u(k)} \qquad (6)$$

$$\boldsymbol{y(k+1) = 0.84\, y(k) + 0.59\, u(k)} \qquad (7)$$

$$\boldsymbol{u(k) = K_p e(k) + K_i \sum_{j=1}^{k} e(j)} \qquad (8)$$

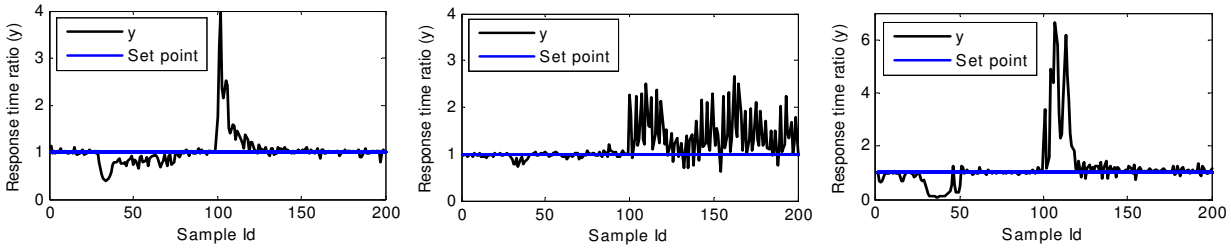
$$\boldsymbol{u(k) = u(k-1) + (K_p + K_i) e(k) - K_p e(k-1)} \qquad (9)$$

Typically, the control system designer would use the *model-AB* and the high-level performance specifications (e.g.: settling time, overshooting) to implement a single fixed gain controller. However, from the experimental results and observations in section 5.3, it is hard to satisfy the stability and performance of the system when they are operating away from the nominal region. An aggressive controller (with K_p=*1.15* , K_i=*0.61* called *controller-A*) is needed to achieve the performance objectives in

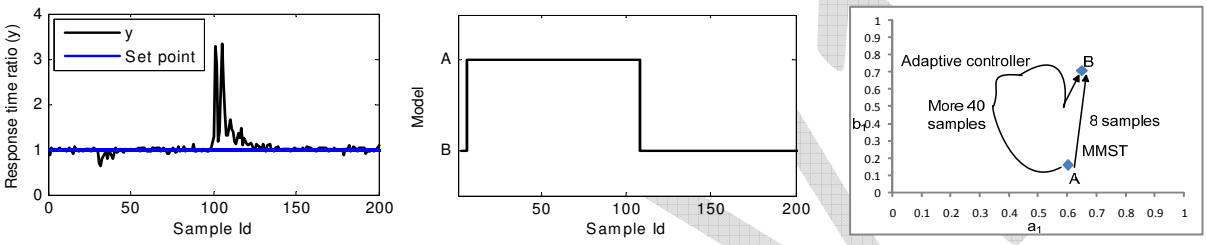
region A, whereas a comparatively less aggressive controller (with K_p=*0.47* , K_i =*0.11* called *controller-B*) is needed to provide control in region B. In this work we try to combine {*controller-A*, *model-A}* and *{controller-B, model-B}*, to improve the performance of this the control system by implementing a type 2 MMST scheme. These models (*A, B*) in equations (5), (6) and the corresponding controllers (*A, B*) were implemented as *ModelControlPair* instances in design. Afterwards, the *MMSTSwichingBoxT2* implementation from the control library was utilized as the self-managing control scheme because the knowledge about the operating conditions/ regions was available at the design time while avoiding the requirements/ overhead of adaptive control. The model evaluation algorithm with a finite window (T, T_{min} = 3) illustrated in equation (4) was used, because typically the operating conditions of the software systems are time-varying. α, β were set to 0 and 1 respectively. We set *controller-A* as the startup controller without loss of generality. Then the sensor and actuator of the prototype system was connected to the instance created out of the *MMSTSwichingBoxT2* class before the control library was compiled as a component in the prototypical system.

5.3 Experiment results

In this section we present the performance of the MMST type 2 scheme designed in section 5.2, with different workloads/operating conditions. In order to do a comparative performance analysis we also use the two fixed gain controllers designed in section 5.2 separately and a self-tuning regulator. In section 5.3.1 we demonstrate the performance of these controllers when the system is facing high workloads form A or B separately, fixing the set point at 1. Section 5.3.2 evaluates the performance differentiation qualities of the controllers when the importance of the client classes are differnt by fixing the set point at 1.5.

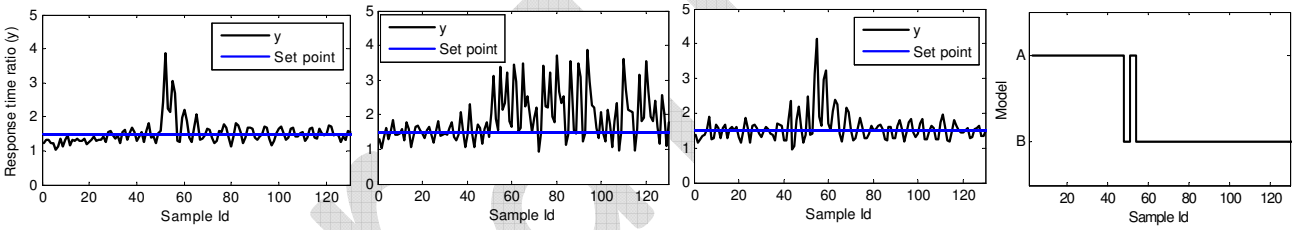


(a) Less aggressive (controller-B) (b) Aggressive (controller-A) (c) Adaptive controller (STR)



(d) MMST (e) Model switching in MMST (f) MMST vs Adaptive control

Figure 6. Performance of the controllers in the away from the nominal region



(a) Controller-B (b) Controller-A (c) MMST (d) Model switching (MMST)

Figure 7. Performance of the controllers when set point at 1.5

5.3.1 Away from the nominal region

In this experiment the set point is fixed at 1, indicating that both client classes are equally important. The experiment starts off in the nominal region with A and B sending 10 req/sec. To enable the control system to operate away from the nominal region, at the 30th sample *A* workload increases from 10 to 30 req/sec. This could be a scenario where travel agent *A* has advertised special travel plans/fares for a limited amount of time, so that there is a sudden increase of system workload. Then at the 80th sample the *A* workload reduces to the nominal request rate of 10 req/sec. Then at the 100th sample, the B workload increases to 30 req/sec from 10 req/sec.

Figures 6a) and b) show the performance of c*ontroller-A* and *controller-B* as they are deployed separately. Both controllers reject the high workload disturbances applied at the 30th and 100th samples. After the disturbance at the 30th sample aggressive *controller-A* settles down quickly with less overshooting compared to the less aggressive *controller-B.* The performance of *controller-A* is significantly better in region A. However, the steady state performance of *controller-A* is oscillatory and unstable compared to *controller-B* after the 100th sample. This is because of the discontinuous control inputs. The aggressive *controller-A* does not settle to a suitable operating point, leading to controller induced oscillations. *controller-B,* in contrast, settles down to a desired operating point in 32 samples (after the 100th) and achieves better steady state behavior. These

results indicate that the both controllers cannot provide effective control in the entire region. Hence, designing a single controller to operate in both regions has to sacrifice performance in both regions.

Figure 6c) shows the performance of the adaptive controller. For this experiment a minimum degree self-tuning regulator *without zero cancelation* was used placing two poles and zero at 0.6, 0.6 and 0.6 respectively with the forgetting factor of 0.94 (for implementation details see [2]). The disturbance rejection capability of the adaptive controller is significantly poor and inconsistent compared to the fixed gain controllers. This is because of the large sudden disturbances at the 30th and 100th sample instances. The model estimation process takes several time samples to converge to new model parameters which makes the system to have high transient responses. To settle down after 100th sample, it takes more than 40 samples and with large overshooting. However, the adaptive controller captures the multi-model nature of the prototype system, because after settling down there is no significant oscillatory behavior compared to *controller-A* in the region B.

Figure 6d) shows the performance of the MMST type 2 scheme which combines *controller-A* and *controller-B* together with their corresponding models representing regions A and B, respectively. It rejects the disturbance at the 30th and 100th samples and settles down with low steady state error in both regions. The settling times at the 30th sample is approximately 12, 12, 46 sample periods in MMST type 2, *controller-A* and *controller-B* respectively. The MMST type 2 settling time after disturbance at 30th sample is better than *controller-B* and similar to *controller-A*. At the 100th sample *controller-B* and MMST type 2 shows low steady state error, whereas *controller-A* shows significantly high steady state error. The overshooting of *controller-B is* significantly high than MMST type 2 as well at the 100th sample (4.1 and 3.3 respectively). The results, show that the MMST type 2 scheme which combines c*ontroller-A* and *controller-B* provides effective performance in both regions. This performance improvement can be explained by the model switching behavior shown in Figure 6e). The MMST type 2 switching algorithm selects the appropriate model and the controller depending on the operating conditions. Although, there is no model designed to represent the startup workload conditions (till 30th sample), at the 6th sample *model-A* is selected, identifying the model as close to the operating conditions. However, it did not lead to any instability. After the 30th sample the control system operates in region *A*, (represented by *model-A*), so no switching is required. Similarly, after the 100th sample when the control system is operating in region *B*, *model-B* is selected at the 108th sample. However, no chattering was observed during the switching because of the utilization of the velocity form of the PI controller, which provides bumpless transfers. Figure 6f) compares the of model convergence in the MMST and adaptive controllers. The MMST convergence to the neighborhood of the model parameters after 8 samples compared to the adaptive controller which takes over 40 samples after the disturbance at the 100th sample. Hence, significantly poor performance was observed in the adaptive controller.

5.3.2 Different importance levels

In this section the set point of the control system was fixed at 1.5, indicating the different importance of client classes A and B. The models/controllers were designed assuming the set point is 1, hence this experiment compares robustness of the self-managing control system under model uncertainties. A and B client classes start off by sending 10 req/sec and 20 req/sec workloads respectively till the 50th sample and afterwards B workload increases to 30 req/sec. The performance of the controllers is shown in Figure 7.

Figures 7a) and b) show the performance of *controller-A* and *Controller-B*. The settling time at the startup and after the disturbance at 50th sample is low in the case of *controller-B*, which is less aggressive. The *controller-A*, on the other hand provides much better disturbance rejection capabilities but the steady state error after the disturbance at the 50th sample is significantly high. Again the issue is due to the discontinuous operating points in the control system leading to controller induced oscillations. In contrast, the performance of the MMST type 2 scheme (see Figure 7c)) is much better than both the fixed controllers. It settles down in 5 samples compared to *controller-B* which take 28 samples to settle at the startup. However, *controller-B* settles down in 4 samples before the MMST scheme after the disturbance at the 50th sample. This is because of the chattering observed around that time period. Figure 7d) shows that frequent model switching occurs from the 50th to 57th samples, leading to oscillatory behavior in the system output. However, after selecting the closest model to the conditions, the performance of the MMST scheme was significantly better than *controller-A* and similar to *controller-B*.

The performance comparison of many other conditions (e.g.: nominal, overload) also indicated the significantly better performance of the MMST adaptive control. Further experiments were also conducted using a predictive controller in a gain scheduling switching setting extending the meta-model, which also produced similar results.

5.4 Nominal region

In this section, performance of the control system when it operates in the nominal region is discussed. The nominal region is where both classes gets same amount of resources. So that the workload conditions should be similar. We used the workload settings, where A and B start off by sending 10 requests/sec each till the 50th sample and afterwards both classes increase their workloads to 20 requests/sec simultaneously. The performance of the controllers is shown in Figure 8.

The performance of all three control systems is similar. Even at the disturbance after 50th sample, no overshooting is observed. This is because the operating point selected by all three control systems is 10/10 in the entire experiment. So that, when we

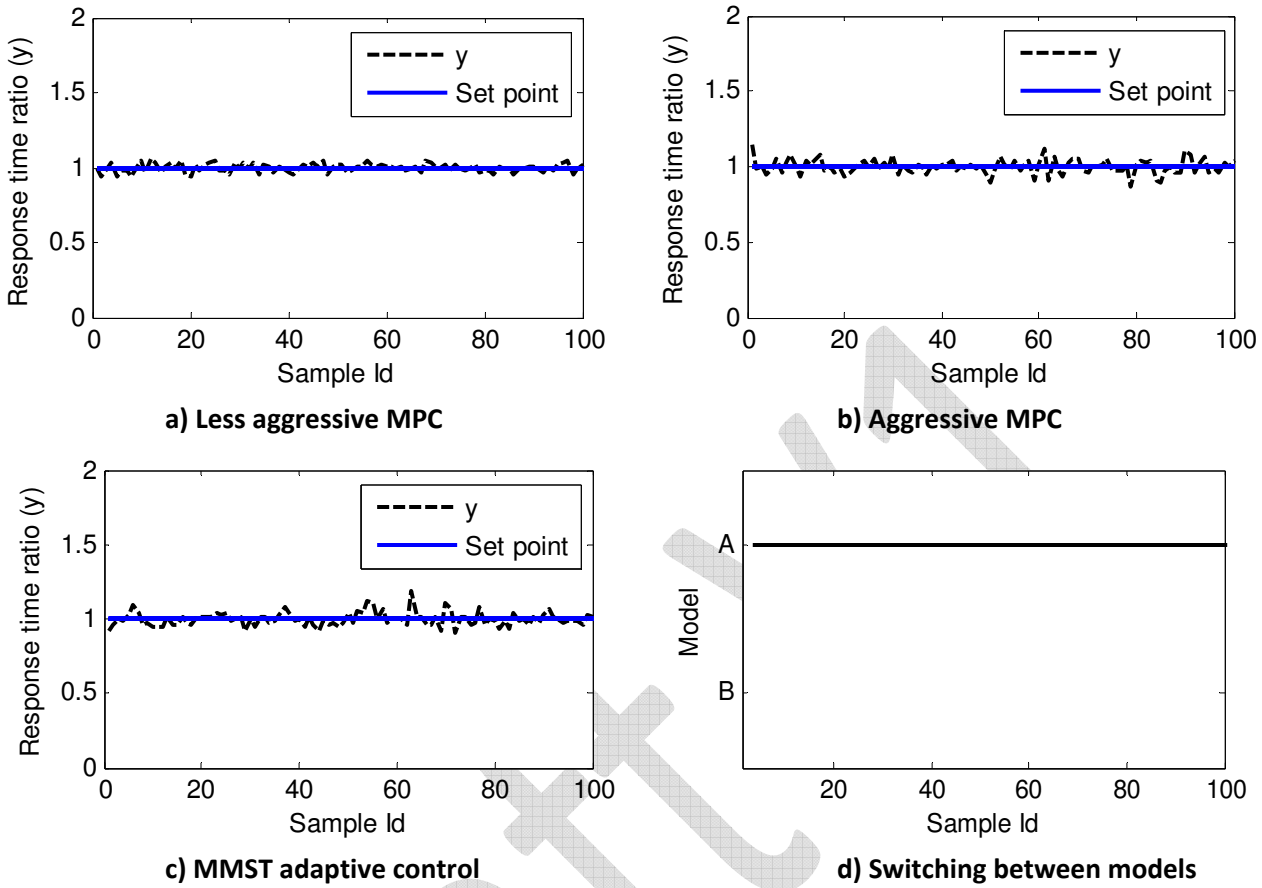


Figure 8: Performance of the controllers in the nominal region

increase the workload simultaneously there is no need to change the operating point. Model switching in figure 8 d) illustrates the same situation, where the *model-A* was utilized in the entire experiment, indicating *model-A* as the closest model.

5.5 Overloaded condition

When the system is overloaded exact relative value is hard to maintain by any of the controllers because of the highly nonlinear nature of that condition. The main observation would be oscillatory behavior in around the set point. To overload the system we utilized similar workloads to away nominal region, however workloads at 30th sample was 10 req/sec and 35 req/sec for B and A, while at 100th sample it is req/sec and 35 req/sec for A and B respectably. That is the sever is running overload of 5 req/sec. The performance of the control systems are shown in figure 9.

Similar results to section 5 can be observed in this experiment as well. The less aggressive controller takes more than 25 samples to reach the set point (Figure 9a)), where as it takes less than 10 samples for the aggressive controller after the overloaded workload condition at 30th sample(Figure 9b)),. However, steady state error is high compared to the results of section X in both cases because of the overload. Then at the 100th sample B workload goes in to overload causing high overshooting (greater than 5) in the less aggressive MPC. However, it settles down and maintains low steady state behavior. The aggressive controller has low overshooting at that region, but shows highly oscillatory behavior and high steady state error. This is because of the discontinuous control input in region B. MMST performance drastically better in this case (Figure 9c)). The settlting time is similar to aggressive MPC after 30th sample , and steady state behavior is similar to less aggressive controller after 100th sample. Overshooting and settling time is significantly better than less aggressive controller after the disturbance at 100th sample. This performance improvement is because of the appropriate model and controller selection by the MMST scheme. Figure 9d) shows that at the 108th sample the most sutiable controller for region B was selected and maintained improving the performance

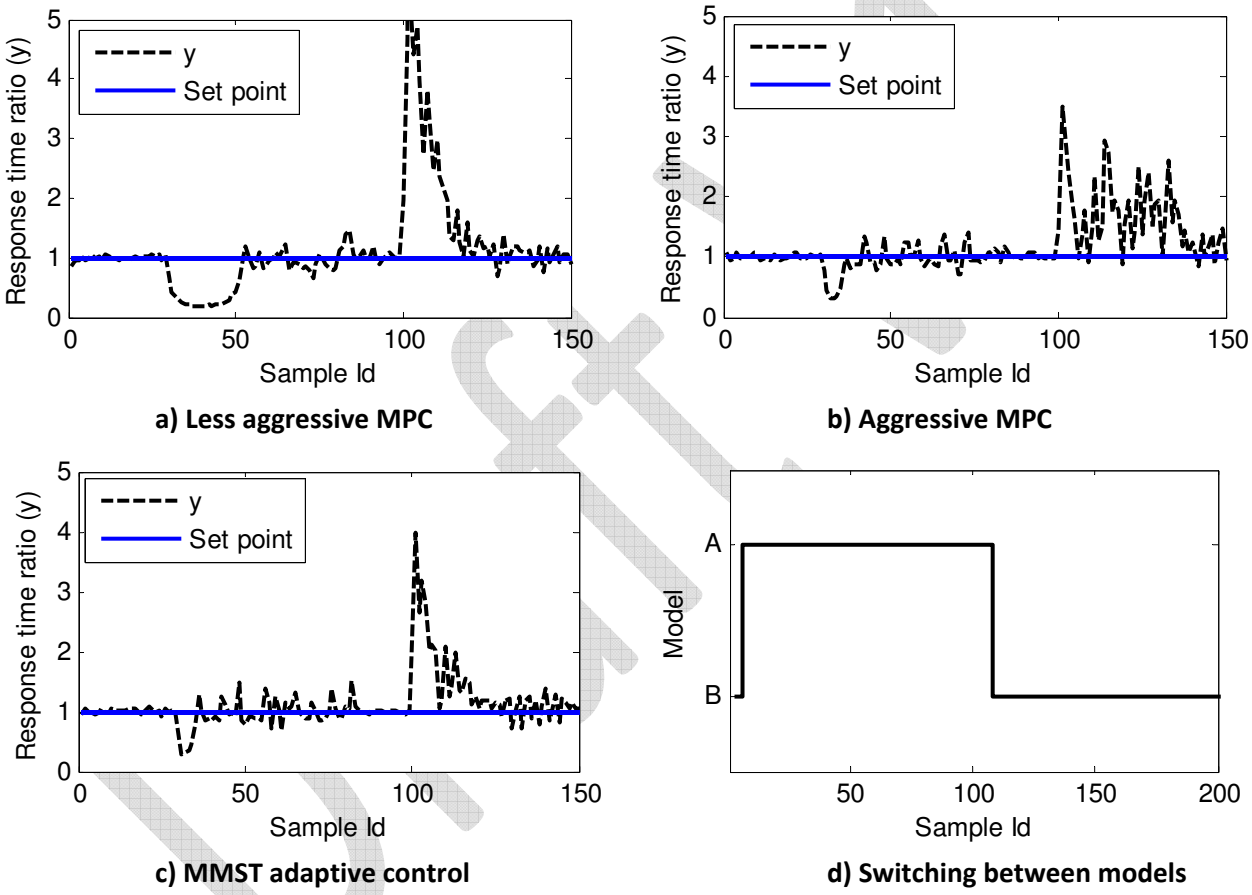


Figure 9: Performance of the controllers in the overloaded condition

after overload at 100th sample. In the beginning model-A is the most suitable hence it was maintained.

5.6 Set point at 0.6

In this section the set point is fixed at 0.6 indicating B is more important than A. Here, 10 and 20 requests/sec workloads are applied for *B* and *A* respectively, and then *A* increased its workload to 30 requests/sec. Figure 10 shows the experimental results. This experiment also validates the performance of control systems under high model uncertainty.

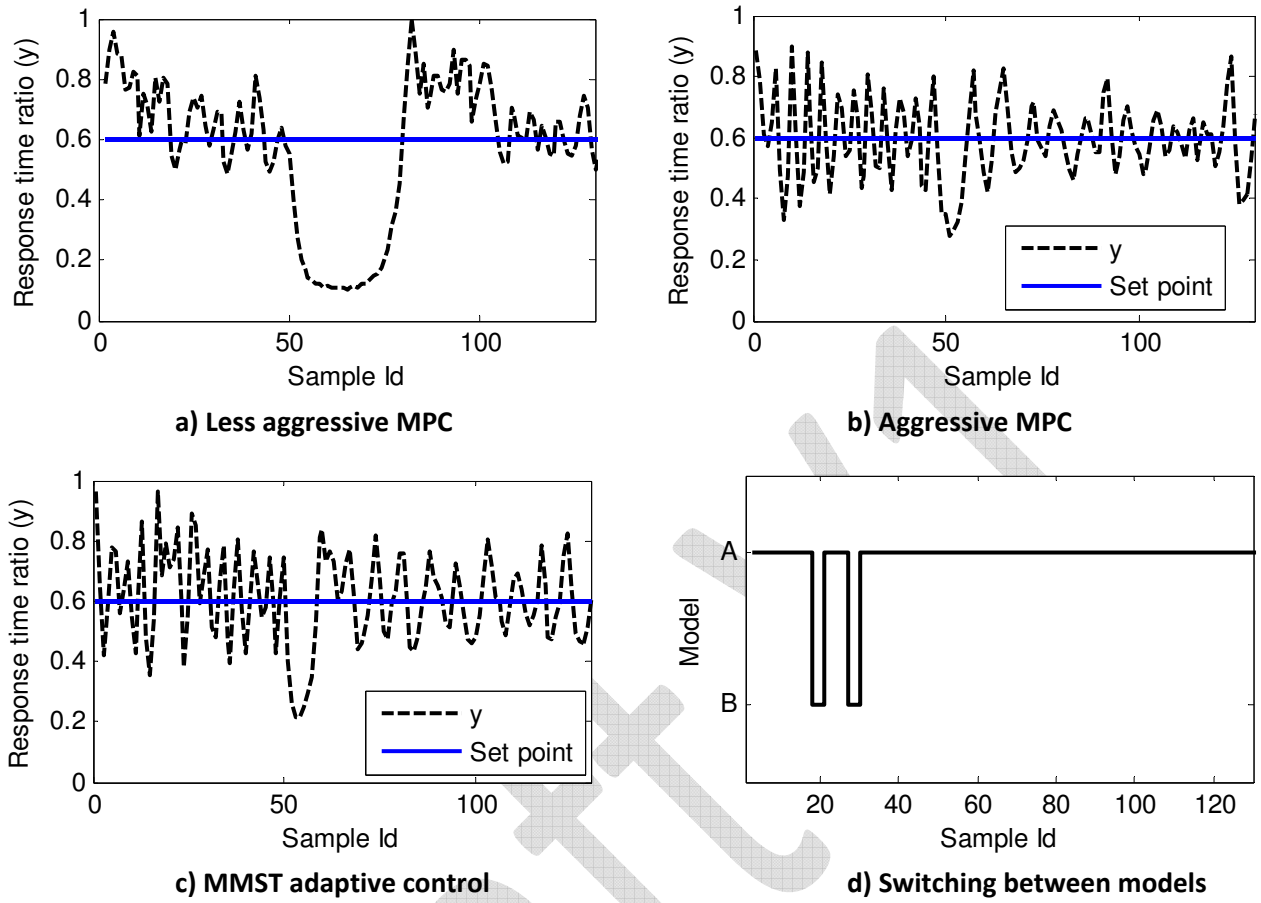


Figure 10: Performance of the controllers at set point = 0.6

Figure 10a) shows the performance of less aggressive controller. Initially, it takes time to settle down to the set point but after settling down less steady state error till 50th sample. Then after the disturbance at 50th sample it takes more than 110 samples to settle down to the set point with large overshooting. The aggressive controller in contrast shows better disturbance rejection capabilities with low overshooting and settling time. Then after the disturbance at 50th sample it provides satisfactory steady state performance. MMST control in contrast, illustrate similar performance to aggressive controller with less overshooting and settling time after the disturbance at 50th sample. But start up performance will be oscillatory. This is because the chattering observed around 20th to 35th sample. The initial oscillatory behavior of the aggressive controller causes this chattering, but it did not lead to instabilities in the system.

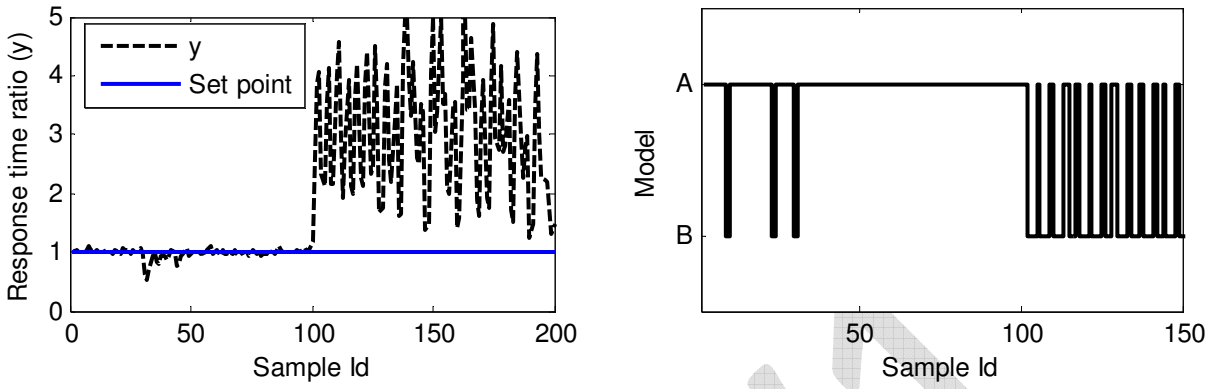
5.7 Other parameters

In this section, firstly the effect of low switching time/waiting time (T,T_{min}) will be illustrated, secondly the effect of startup controller will be presented. Finally, we set $\alpha = 1$ and $\beta = 0$ and shows the performance of the system. For all these experiments, the workload setting in section 5.3.1 will be used.

5.7.1 Effect of low T_{min}

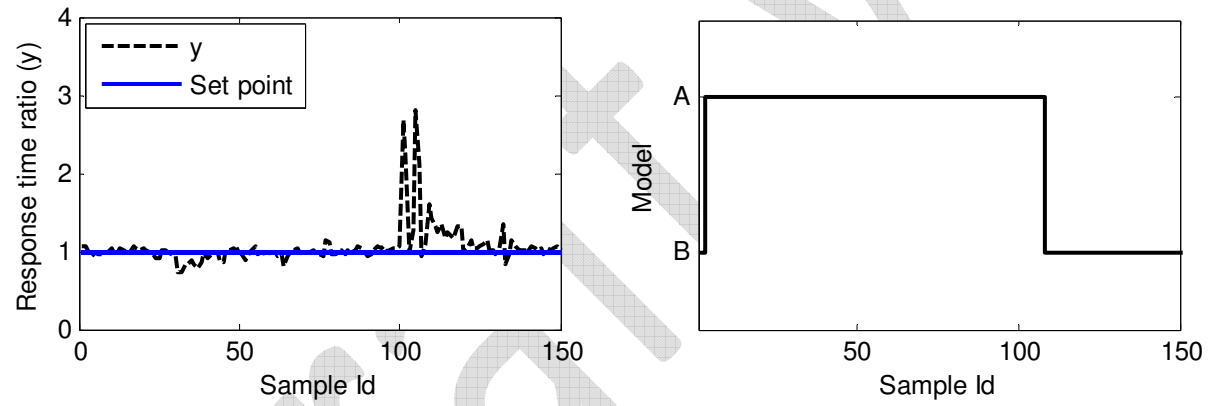
For this experiment T_{min} was set at 1, which means in every sample instance the *J* indexes are evaluated and model/controller is selected. Figure 11 shows the performance and switching behavior.

Compared to the results in Section 5.3.1 which used $T_{min}=3$, the performance under low T_{min} causes frequent chattering. The surprisingly the performance is better in region A (after 30th sample) compared to region B(after 100th sample). This is mainly because no chattering occurred during the time period of 30th sample to 100th. However, due to highly discontinuous operating points in region B, chattering occurs frequently, leading to high steady state error and oscillatory behavior. It is evident that *model-B* was used most of the time during 100th to 150th sample, however *model-A* was selected in-between as well. The high switching times used in section 5.3.1, used *model-B* after 10th sample because it accumulates the error for 3 samples according to the J index settings of equation (4).



a) MMST adaptive control **b) Switching between models**

Figure 11: Performance of MMST T_{min} =1



a) MMST adaptive control **b) Switching between models**

Figure 12: Performance of MMST with controller-B as the startup controller

The general recommendation is, if the system has fast varying dynamics it is better to have low T_{min} but if it is not that fast varying relatively high T_{min} can be used.

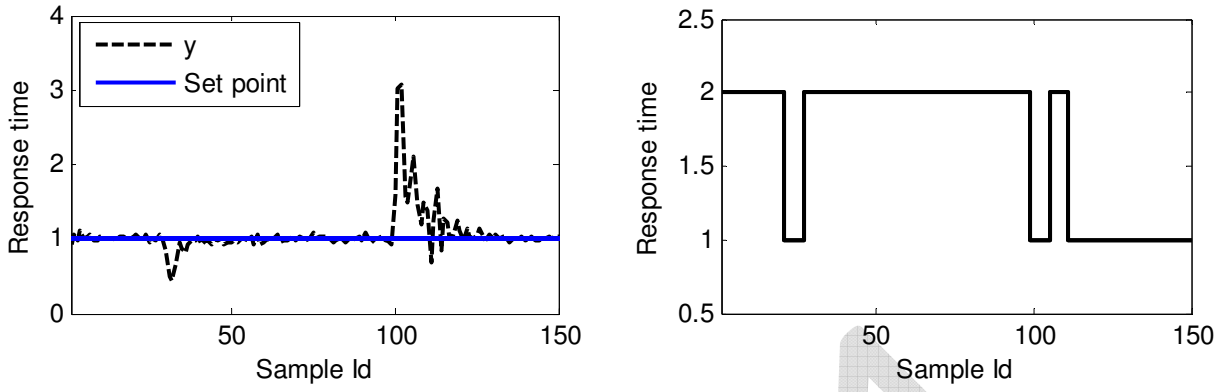
5.7.2 Effect of the startup controller

For the above experiments *model-A* and *controller-A* was used as startup controller. In this section we set model *model-B* and *controller-B* as the startup controller to check it has effect on switching behavior.

The performance and switching behavior is similar to the results of Section 5.3.1. At the 6th sample the *model-A* is selected and maintained without any difference to section 5.3.1. Hence, there is no effect of the startup controller for this case study.

5.7.3 When $\alpha = 1$ and $\beta = 0$

The output of the system doesn't show any noticeable difference compared to the performance in section 5.3.1. However, The figure 13b) shows the switching behavior. It is drastically different to the swathing behavior in section 5.3.1. Especially, at the 100th sample, chattering could be observed causing slight deviations in the output. In addition there is a inconsistent chattering shown around the 18th to 24th sample. Such inconsistencies are there because of the utilization of only the instantaneous component, which doesn't use the averaging of the error signals in the finite window. When the long term averaging component is added this effect can be effectively removed.



a) MMST adaptive control **b) Switching between models**

Figure 13: Performance of MMST When α = 1 and β = 0

Model predictive control in a gain scheduling setting

In this section, the above described meta-model and the control library was used to design a multi-model based self-managing control system for the prototype system. Instead of using multiple controllers like in the previous section, here a single predictive controller is used. Depending on the selected the model, the gain of this predictive controller is modified. This setting can be referred as gain scheduling, where the model prediction error is selected as the scheduling variable (for more details about gain scheduling[2, 12]). However, the main difference is the switching logic is precise mathematical formulations in the case of MMST, compared to ad-hoc logic in the case of gain scheduling. The control system integrates the same *model-A* and *model-B* designed in the previous sections. There many different ways to integrate the controllers. Here, we created a *ModelControlPair* collection with corresponding controllers with appropriate gains. Then, the *ReconfigureControlLoop* method of *MMSTSwichingBoxT2* class was overridden to configure the gain of the startup controller instead of switching between other controllers. Here, the gain of the startup predictive controller is changed to the gains of the controller corresponding to the selected model of the current switching instance. So that the controllers coupled with the *model-A and B* are just used to contain the gains, but not utilized at runtime. The final control system architecture is shown in Figure X.

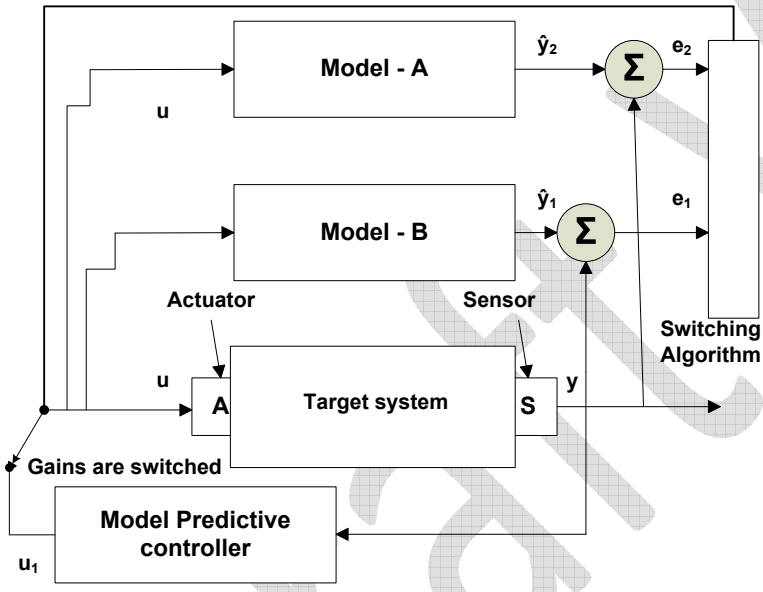
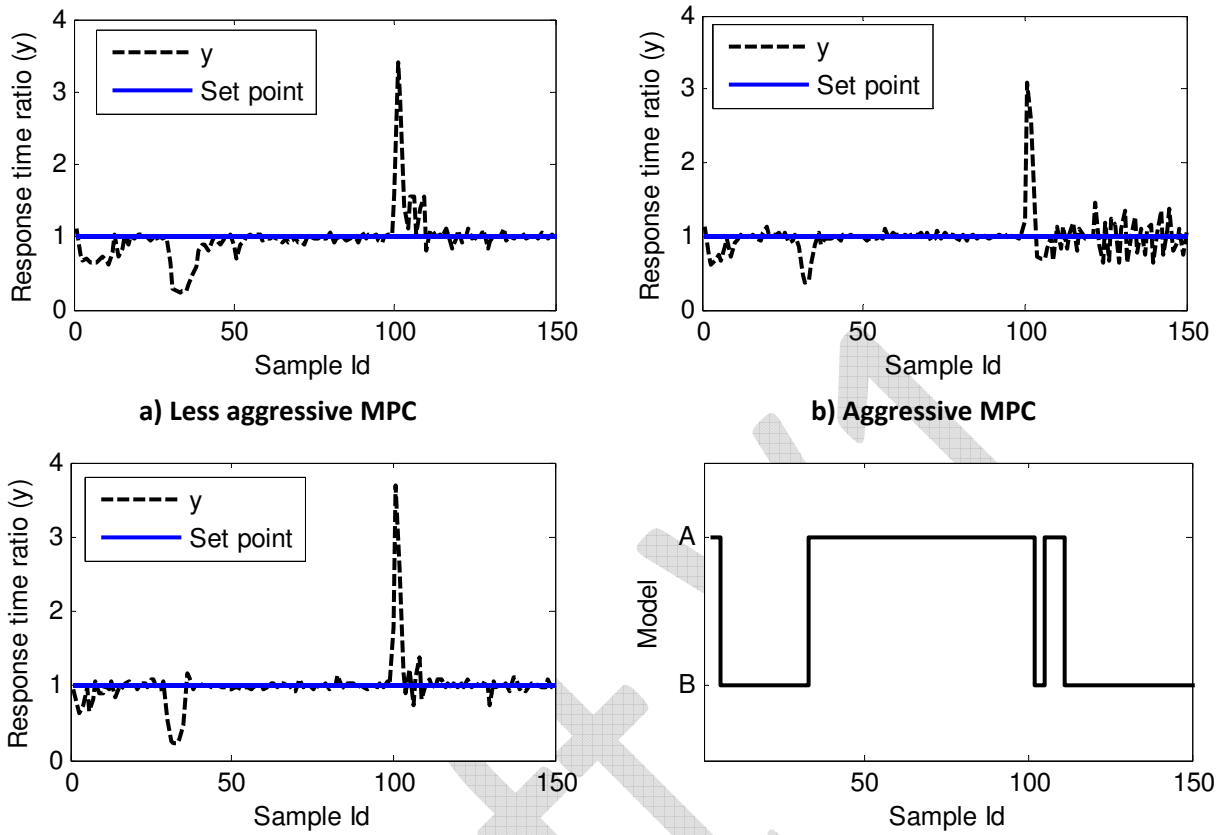


Figure 14: MPC based MMST control systems

This section is organized as follows. In section XXX, brief introduction to model predictive control (MPC) is provided. In the next section, the experimental settings and results are discussed.

MPC

The general idea behind MPC is to optimize the future behavior of the system outputs by computing the trajectory of the control inputs. Firstly, using the model of the system and the current available output data , the behavior of the system output is predicted over the $t + N_p$, where *t* is the current time sample and N_p is called the *prediction horizon*. Then objective of the predictive control is to maintain the future predicted output sufficiently close to the desired values (set point) subject to various constraints on input, output or combination of them that have to be maintained/optimized within the prediction horizon. Then MPC produces a sequence of control inputs ($\Delta u(k)$, $\Delta u(k+1|k)$, $\Delta u(k+2|k)$…. $\Delta u(k+N_c|k)$) **,** which would achieve these objectives (N_c is called as control horizon). However, only the first control input ($\Delta u(k)$) is used to calculate the final control input *u(k)* (= $u(k-1)+\Delta u(k)$) applied on the system and rest of the sequence is discard according to *receding horizon control principle* [35].This process is continued by sliding the prediction horizon one time step ahead and incorporating the feedback signals. This control process can be categorized as a self-optimizing technique according to the definition in [13]. For more details refer [35].



a) Less aggressive MPC

b) Aggressive MPC

c) MMST adaptive control

d) Switching between models

Figure 14: Performance of the controllers in away from the nominal region

Experimental results

MPC parameters : In MPC there are many tunable parameters, which could affect the gains of the controller. In this case, we fix many of the parameters and use weight on control effort parameter R_w as the tuning parameter. The prediction horizon Np = 10, control horizon Nc =5. The two MPC paired with *model -A* and *model -B* has gains of 8 and 0.5 respectively, where former is less aggressive compared to the later. The model incorporated with the controller was *model –AB* in equation XX. We implemented the control system shown in Figure XX , for the prototypical system utilizing the control library. The parameters of MMST is similar where T,T_{min} = 3, α=0 , β =1. For the comparison purposes, two linear MPC controllers with the setting of MPC disrobed above is used but with two different gains. The MPC with 8 is called as less aggressive controller and MPC with 0.5 is the aggressive controller.

Away nominal region

Using the control system designed an experiment was conducted with the workload settings of section 5.31, which makes the control system to operate in away nominal region. The point is set point at 1. Figure 14 shows the results.

When Figure 14a) and b) is compared, similar performance issues observed in section 5.31is evident. The less aggressive MPC takes more than 23 samples to settle down after the disturbance at 30th sample (region A), where as the aggressive controller settle down in 7 samples. However, the aggressive controller settles down quickly with high overshooting compared to less aggressive controller in region B (i.e. after the disturbance at 100th sample). In contrast, performance of MMST adaptive control shown in Figure 14) illustrates combined performance of above two controllers. At the 30th sample it settles down in 8 sample showing significantly better performance than less aggressive MPC and similar performance to aggressive controller. Then at after the disturbance at 100th sample MMST control provide significantly better steady state performance compared to the aggressive controller and similar performance to less aggressive controller. However, overshooting is bit higher than model linear controllers because of the chattering occurred during 100th to 111th samples. In between that time model A and B were switched back and forth leading to change in the gains of MPC. However, after the 111th sample the suitable and closest model to the

operating conditions was maintained without abrupt switching. Although chattering occurred the system stability was not affected.

Nominal region

The workloads settings of section 5.5 was utilized in these experiments with set point at 1. The performances of the controllers are shown in Figure 15.

The performance of all three control systems in Figure 15a), b) and c) are similar. The settling time at the 50th sample is 3 for aggressive controller and close to 8 in MMST and less aggressive controllers. The interesting thing to note here is the switching behavior. The control system operates with the model-B till the 57th sample and then move to the model-A. This indicates that the model-B is closer to the operating conditions of the system when the workloads are low in nominal region, but when it's high model-A is the closest. Figure 15d) indicates that the model switching happened without any chattering.

Different Importance levels (set points)

In this section we show the performance of the control system when the importance of the client classes are different. This is translated in to the control system by the set point signal (not equal to 1). In this experiment we fixed the set point at 1.5 and use the workload settings of 5.3.2. Figure 15 shows the performance of the control systems.

Less aggressive controller takes approximately 25 samples to settle down to the set point at the start up. In contrast, the aggressive controller and MMST settle down in 10 samples at the start up. Then at the 50th sample the disturbance make the less aggressive controller to have high overshooting compared to the aggressive controller. However, the aggressive controller shows highly oscillatory behavior in after the 50th sample, due to discontinue operating points it has to operate with. MMST control on the other hand, is similar to less aggressive MPC, showing less steady state error compared to aggressive MPC. This is because the at 57th sample the MMST control system switch to model-B smoothly, selecting the gains of less aggressive controller. Hence, MMST demonstrate better settling time at startup, because of using aggressive gains and then shows better steady state

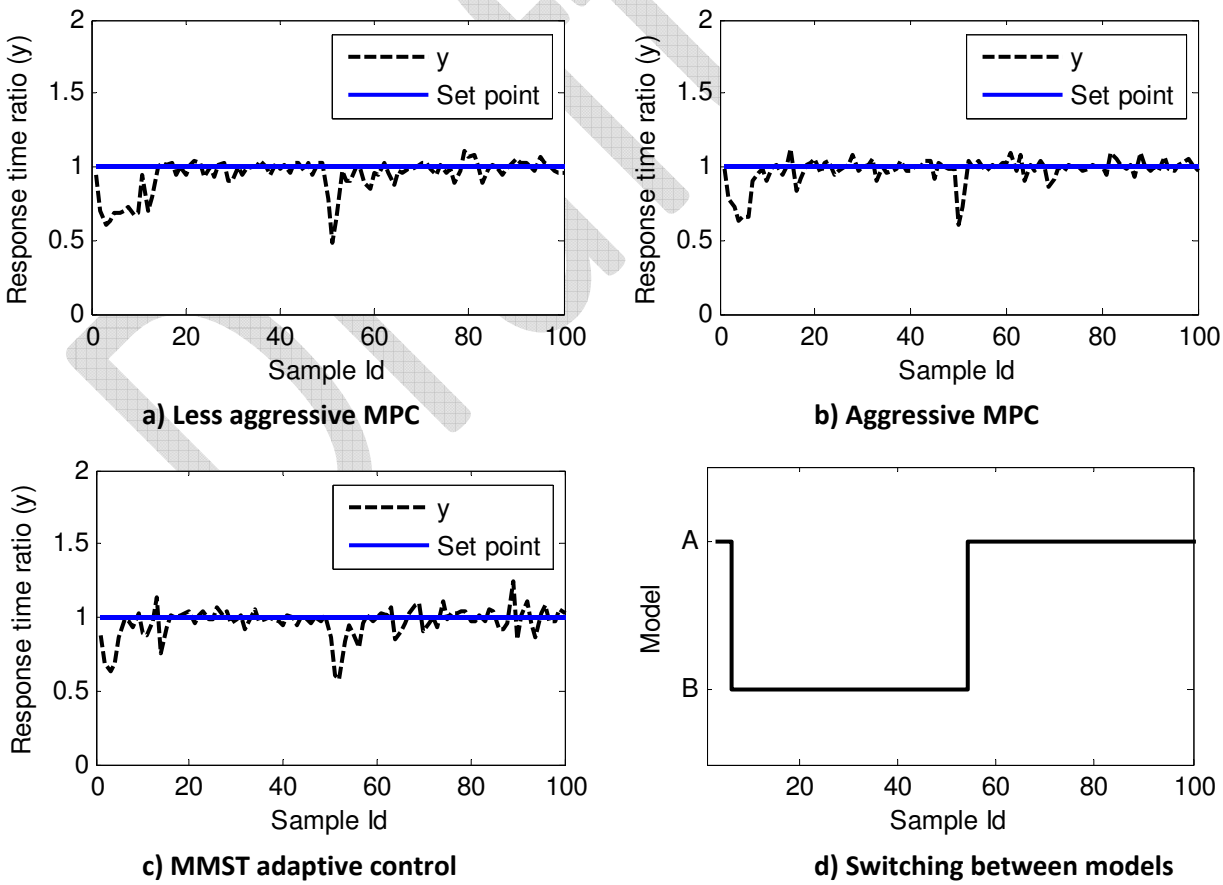


Figure 15: Performance of the controllers in the nominal region

behavior by selecting the gains of less aggressive controller appropriately after the high disturbance.

6. DISCUSSION

The experiment results under two different operating conditions indicate that the MMST (type 2) scheme is promising in handling the multi-model behavior of the software system by providing self-managing (reconfiguring) control at runtime. Depending on the different operating conditions the suitable controller can be selected from the available without any human intervention. In addition, the control system designer can combine the performance of the suitable controllers without concerning about the performance tradeoffs involved in designing a single controller. The MMST type 2 scheme integrates the fixed controllers into the control system and overcomes the limitations of adaptive control. Although short-term chattering was observed, it did not cause any instability. Furthermore, there are different control schemes (discussed in section 3), that can be utilized depending on the design requirements. The extendable meta-model and the control library available from [1] provides the opportunity to design MMST schemes for real systems or research purposes, without needing to develop them from scratch. The control library also provides the capability to design a control system combining different standard control algorithms (e.g.: PID with predictive control).

However, the MMST schemes have to be chosen only when a single fixed or adaptive controller cannot provide the effective performance in the entire operating regime. Since this is a reconfiguring control scheme limitations discussed in section 2 exist such as the performance overhead and chattering. In addition, the number of fixed models that have to be designed depend on the various operating conditions, which requires many SID experiments and priory knowledge. For the case study discussed in this paper we have used only two models, but according to the stability proofs many models may be needed to guarantee stability of the system in the case of type 2 scheme. The number of models/controllers required depends on the available priory knowledge and the system characteristics. If such uncertainties exist, the MMST type 4 is most suitable after approximating fixed models depending on the available knowledge.

Furthermore, MMST specific tuning parameters α, β and T_{min} have to be selected carefully. From our experiments having small value or 0 for α and β= 1 provides consistent switching. Higher values for α improves the performance but with variable switching performance. $\alpha > 0$, $\beta = 0$ may cause chattering, leading to performance degradation. T_{min} is recommended to be set as low as possible to avoid instabilities due to the use of unsuitable controller for a long time. However, very small T_{min} (=

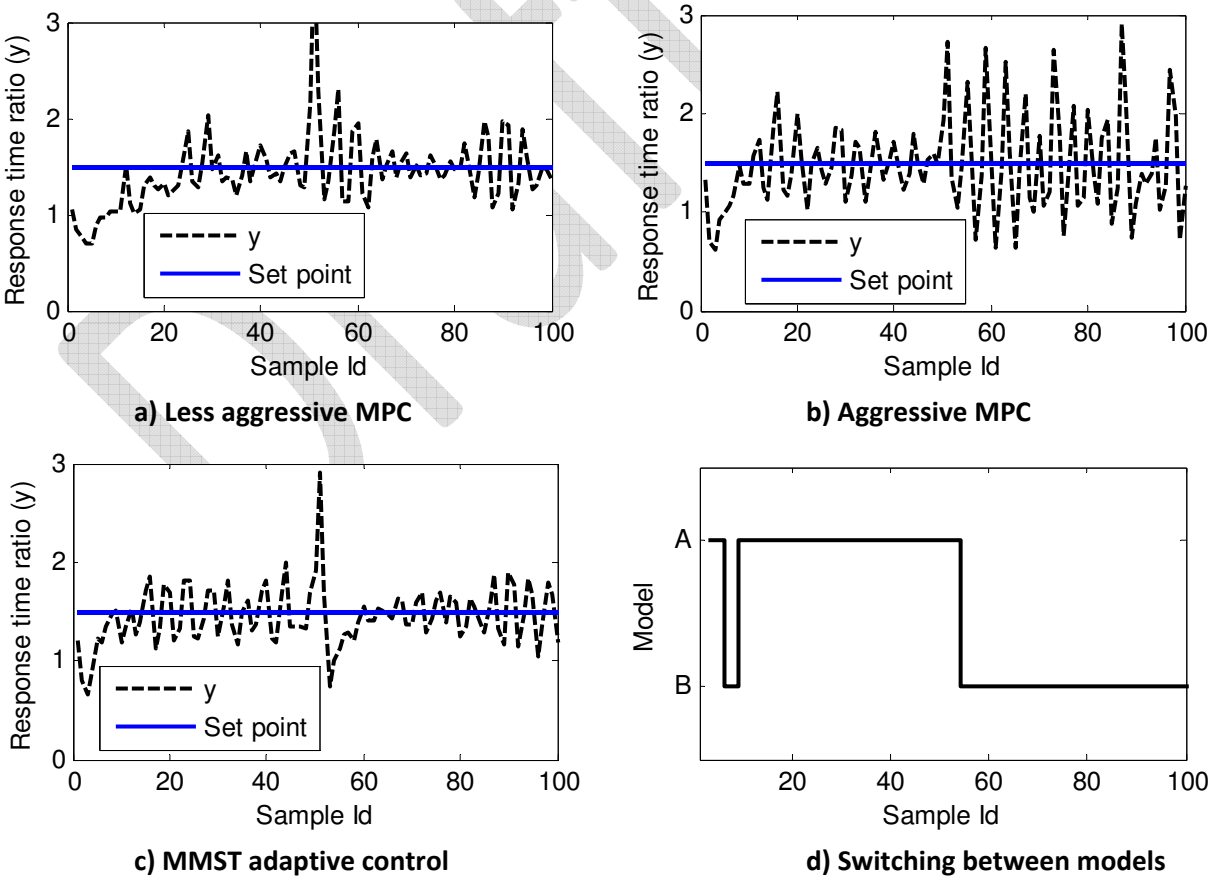


Figure XXX: Performance of the controllers in the nominal region

1) could cause chattering. When T_{min} =5 drastically less chattering was observed, but the performance of the system was affected. The startup controller does not affect the performance apart from the first few samples (for more details see [34]).

The proposed MMST schemes may be used effectively in applications that show multi-model nature. For example promotional, seasonal web sites and the systems that face sudden workload spikes in a predictable/known way can use the MMST type 2 scheme. Other MMST schemes (types 3 and 4) could be used when systems demonstrate unpredictable and unknown conditions.

As future work we intend to investigate runtime model learning mechanisms using the MMST-type 4 scheme, which would reduce the effort and prior knowledge required in modeling.

[1]"http://www.ict.swin.edu.au/personal/tpatikirikorala/Downloads.htm".

[2]Åstrom K. J. and B. Wittenmark., *Adaptive Control, 2nd ed*: Addison-Wesley Publishing Company, 1995.

[3]Brun Yuriy, Giovanna Marzo Serugendo, et al., "Engineering Self-Adaptive Systems through Feedback Loops," in *Software Engineering for Self-Adaptive Systems*: Springer-Verlag, 2009, pp. 48-70.

[4]Diao Yixin, J. L. Hellerstein, et al., "Managing Web server performance with AutoTune agents", *IBM Systems Journal,* vol. 42, pp. 136-149, 2003.

[5]Diao Yixin, Joseph L. Hellerstein, et al., "Using MIMO linear control for load balancing in computing systems", *American Control Conference,* vol. 3, pp. 2045- 2050, 2004.

[6]Dorf Richard C. and Robert H. Bishop, *Modern Control Systems*: Prentice-Hall, Inc., 2000.

[7]Dumont G.A. and M. Huzmezan, "Concepts, methods and techniques in adaptive control", *American Control Conference, 2002. Proceedings of the 2002,* vol. 2, pp. 1137- 1150, 2002.

[8]Dutreilh X., N. Rivierre, et al., "From Data Center Resource Allocation to Control Theory and Back", in *CLOUD 2010* 2010, pp. 410-417.

[9]Gandhi N., Tilbury D.M., et al., "MIMO control of an Apache web server: modeling and controller design", *American Control Conference,* vol. vol.6, pp. 4922- 4927, 2002.

[10]Goel Ashvin, David Steere, et al., "SWiFT: A Feedback Control and Dynamic Reconfiguration Toolkit", Oregon Graduate Institute School of Science and Engineering1998.

[11]Hellerstein Joseph L., "Self-Managing Systems: A Control Theory Foundation", *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks,* 2004.

[12]Hellerstein Joseph L., Yixin Diao, et al., *Feedback Control of Computing Systems*: John Wiley & Sons, 2004.

[13]IBM, "An Architectural Blueprint for Autonomic Computing", June 2006.

[14]K.S.Narendra and Cheng Xiang, "Adaptive control of discrete-time systems using multiple models", *Automatic Control, IEEE Transactions,* vol. 45, pp. pp.1669-1686, 2000.

[15]Karamanolis Christos, Magnus Karlsson, et al., "Designing controllable computer systems", *Proceedings of the 10th conference on Hot Topics in Operating Systems - Volume 10,* 2005.

[16]Karlsson Magnus, Christos Karamanolis, et al., "Triage: Performance differentiation for storage systems using adaptive control", *Trans. Storage,* vol. 1, pp. 457-480, 2005.

[17]Karlsson Magnus, Xiaoyun Zhu, et al., "An Adaptive Optimal Controller for Non-Intrusive Performance Differentiation in Computing Services", *Control and Automation,* 2005.

[18]Kokar M. M, Kenneth Baclawski, et al., "Control Theory-Based Foundations of Self-Controlling Software", *IEEE Intelligent Systems,* vol. 14, pp. 37-45, 1999.

[19]Kusic D. and N. Kandasamy, "Risk-Aware Limited Lookahead Control for Dynamic Resource Provisioning in Enterprise Computing Systems", in *IEEE International Conference on Autonomic Computing, 2006. ICAC '06.* , 2006, pp. 74-83.

[20]Liu X., Xiaoyun Zhu, et al., "Adaptive entitlement control of resource containers on shared servers", in *Integrated Network Management*: IEEE, 2005, pp. 163-176.

[21]Ljung Lennart, *System identification: theory for the user*: Prentice-Hall, Inc., 1997.

[22]Lu Chenyang, Ying Lu, et al., "Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers", *IEEE Trans. Parallel Distrib. Syst.,* vol. 17, pp. 1014-1027, 2006.

[23]Lu Ying, Tarek Abdelzaher, et al., "An Adaptive Control Framework for QoS Guarantees and its Application to Differentiated Caching Services", *Tenth IEEE International Workshop on Quality of Service,* pp. 22-32, 2002.

[24]Lu Ying, Tarek Abdelzaher, et al., "An Adaptive Control Framework for QoS Guarantees and its Application to Differentiated Caching Services", *IWQOS,* 2002.

[25]Narendra K. S. and J. Balakrishnan, "Improving transient response of adaptive control systems using multiple models and switching", in *Conference on Decision and Control, 1993*, 1993, pp. 1067-1072 vol.2.

[26]Narendra K. S., J. Balakrishnan, et al., "Adaptation and learning using multiple models, switching, and tuning", *Control Systems Magazine, IEEE,* vol. 15, pp. 37-51, 1995.

[27]Narendra K. S. and Osvaldo A. Driollet, "Adaptive Control using Multiple Models, Switching, and Tuning", *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000.,* pp. pp.159 - 164, 2000.

[28]Narendra K. S., Osvaldo A. Driollet, et al., "Adaptive control using multiple models, switching, and tuning", *International journal of adaptive control and signal processing* pp. pp.1-16, 2003.

[29]Narendra K.S. and J. Balakrishnan, "Adaptive control using multiple models", *IEEE transactions on automatic control,* 1997.

[30]Ogata Katsuhiko, *Modern Control Engineering*: Prentice Hall PTR, 2001.

[31]Padala Pradeep, Kai-Yuan Hou, et al., "Automated control of multiple virtualized resources", *Proceedings of the 4th ACM European conference on Computer systems,* pp. 13-26, 2009.

[32]Parekh S., N. Gandhi, et al., "Using Control Theory to Achieve Service Level Objectives In Performance Management", *Real-Time Syst.,* vol. 23, pp. 127-141, 2002.

[33]Solomon Bogdan, Dan Ionescu, et al., "A real-time adaptive control of autonomic computing environments", *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research,* 2007.

[34]Tharindu Patikirikorala, Alan Colman, et al., "Tech-Report : Multi-model driven framework to implement self-managing control systems for QoS management", Swinburne University of Technology 2010

[35]Wang Liuping, *Model Predictive Control System Design and Implementation Using MATLAB*: Springer Publishing Company, Incorporated, 2009.

[36]Wang Zhikui, Xiaoyun Zhu, et al., "Utilization vs. SLO-Based Control for Dynamic Sizing of Resource Partitions", *16th IFIP/IEEE Distributed Systems: Operations and Management (DSOM 2005),* 2005.

[37]Woodside M., Tao Zheng, et al., "Service System Resource Management Based on a Tracked Layered Performance Model", *Proceedings of the 2006 IEEE International Conference on Autonomic Computing,* 2006.

[38]Youbin Peng, D. Vrancic, et al., "Anti-windup, bumpless, and conditioned transfer techniques for PID controllers", *Control Systems Magazine, IEEE,* vol. 16, pp. 48-57, 1996.

[39]Zhu Xiaoyun, Mustafa Uysal, et al., "What does control theory bring to systems research?", *SIGOPS Oper. Syst. Rev.,* vol. 43, pp. 62-69, 2009.

[40]Zhu Xiaoyun, Zhikui Wang, et al., "Utility-Driven Workload Management using Nested Control Design", Hewlett Packard LaboratoriesApri1 06 2006.