A nonlinear feedback control approach for differentiated performance management in Autonomic systems

(Technical 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
Management of Quality of Service (QoS) performance and resources at runtime is a much researched area. In particular, when there are different classes requiring different levels of QoS guarantees, implementing such differentiated services in software systems becomes increasingly difficult due to the lack of formal/systematic design process [9, 11].

The increasing complexity and scale of software systems demand effective and efficient mechanisms to manage them in a systematic way. The e-commerce systems, banking systems, utility computing environments, and many other businesses rely on software systems to deliver desired functional services, while maintaining the quality of service (QoS) properties. Response time, throughput, availability, and security are some of these QoS attributes that must be maintained at an acceptable level to avoid customer dissatisfaction. The existing traditional methods such as manual tuning via human intervention have proven to be costly and error-prone, while fixed/ad-hoc/threshold based policies depending on the peak demands are increasingly difficult to design due to lack of formal/systematic design process [10, 25].

Thus, it is important to implement methodologies that address the above limitations by integrating autonomic (runtime) decision making capabilities into software systems to achieve QoS objectives, with reduced or no human intervention.

Among many QoS and resource management requirements, providing different levels of QoS depending on the services requested, the content delivered or the client group is a much researched area in shared resource environments such as web servers [7, 24, 25], databases/storage systems [19, 20] and server clusters [21]. For instance, in information dissemination systems, disaster warning messages must have high priority compared to hourly weather information. Premium clients must be given higher priority compared to basic clients in product based software systems or else in stock trading applications trading requests should be given high priority compared to stock information requests. We refer to these different QoS levels as QoS classes. Absolute and relative guarantee schemes are often utilized to implement differentiated QoS management in software systems which have different QoS classes (e.g., [2, 10, 15, 19, 20, 24, 25, 29]). The absolute guarantee scheme, tries to maintain the QoS attribute of classes around a specified threshold (or absolute value). However, as argued in [7, 17] it is hard to specify and maintain absolute values for many QoS parameters because of the workloads and characteristics/dynamics of the systems. In addition, when the available resources are not sufficient, absolute guarantee for all classes cannot be achieved, requiring complex admission control policies to be imposed under overloaded conditions. In contrast, relative guarantee provides the advantage of incorporating additional matrices such as price, cost and importance of the class into runtime decision making, independent of the request rates (workloads) (so called controllability property) [10]. For instance, if the response time of the premium class should be maintained twice as the basic class, we can specify this as a response time ratio between client classes to be maintained at the ratio of 2. The limitation is that the absolute value of QoS classes may not be maintained especially, in the overloaded conditions, but there is guarantee that higher priority classes will get better or no worse performance compared to low priority classes (so called consistency property) [10, 25].

The relative guarantee schemes and autonomic resource management utilizing linear feedback control theoretic approaches can be found in [7, 15, 25, 29]. However, formulation of the relative guarantee scheme needs to incorporate ratio of the QoS attributes and resource sharing highly nonlinear behavior in the performance. As a consequence, linear model based feedback controllers tend to fail in achieving performance isolation/differentiation, flexibility in design and effective disturbance rejection capabilities under nonlinear/un-modeled dynamics. Thus, considering nonlinearities in the autonomic control system design and implementation may achieve the aforementioned requirements more effectively.

In this work, we introduce an approach for the autonomic runtime management of resource provisioning and performance in relative guarantee control scheme utilizing a nonlinear modeling and control technique called the Hammerstein model. The approach provides performance isolation/differentiation between different client classes while incorporating polices/constraints on resource requirements. The proposed approach also:

- Provides a way to incorporate nonlinearity into feedback control based design without requiring prior knowledge of the workload, precise modeling of system dynamics and disturbance, but achieves better performance management,
- Preserves the design and numerical properties of classical linear control system design, providing a systematic and formal design process for runtime performance management,
- Achieves a non-intrusive integration of the new autonomic management components, without the need to modify the system.

In addition, we have evaluated the approach by conducting a range of experiments and comparing to a linear model based control design showing significant improvement in performance management in the case of processing delays in a web server/service.
2. BACKGROUND
In this section we provide formulation of relative guarantee scheme, followed by background to linear feedback system design. Then using an example, nonlinearities in relative guarantee scheme will be discussed. Afterwards, related work will and Hammerstein-Wiener block oriented nonlinear model will be presented.

2.1 Relative guarantee scheme
In relative guarantee scheme, QoS attribute of QoS classes is maintained proportional to the differentiation factor specified/derived from the business/system design requirements. Let q\_i, p\_i be the QoS attribute under interest and the differentiation factor of class i out of n classes respectively. Between the pair of classes i and j, the constraint of q\_i / q\_j = p\_i / p\_j (i,j = 1, …….,n) is maintained at runtime by the relative guarantee control scheme. For instance, if p\_i / p\_j = \(\frac{1}{2}\), means QoS attribute of class j has to be maintained twice as of class i.

The main challenge of applying the above scheme in software system is managing above differentiation ratio while dynamic resource allocation [7]. In addition, workloads of these systems vary overtime predictable/unpredictable manner and the behavior/ dynamics of the software systems are typically nonlinear making it difficult to provide QoS guarantees. To address some of these issues C. Lu al proposed dynamic propositional resource share allocation approach to achieve aforementioned relative guarantee in multiple QoS class systems, in particular for web servers [7, 24]. Depending on the workloads the proportions of the resource share ratio (s\_i / s\_j) is manipulated at runtime to achieve the relative guarantee of QoS classes. Then the s\_i / s\_j ratio is used to calculate the individual resources for a particular QoS class using an algorithm.

2.2 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 target system. The target software systems provides a set of performance metrics for properties of interest (e.g.: response time, CPU) referred to as outputs. Sensors monitor the outputs of the target system, while 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 output. For the above discussed relative guarantee control system q\_i / q\_j, s\_i / s\_j and p\_i / p\_j are the measured output, control input and set point.

![Figure 1. Block diagram of a control system](image)

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 referred to as behavioral model of the system. System identification (SID) is a method used to construct this model via the measurement of input and output data [23]. 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 dynamics. In addition, there are well established formal techniques and methodologies to design, develop and analyze performance specifications (e.g: stability, settling time and overshooting) of control systems.

2.3 Scenario/Example
The software system in Figure 2 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, normally 50-200 in large scale systems) to communicate with them. Assume that, two clients (travel agents A and B) are interested in the services provided by this server. The objectives 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. Fast responses to the requests are one of the main objectives of the travel reservation web applications to avoid customer dissatisfaction (e.g: flight search requests). Additionally, there could be constraints/policies on a minimum number of resource units that has to be maintained for specific client class to avoid starvation of resources. A prototype of such a reservation system was developed implementing the architecture shown in Figure 2. 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-\#l is applied and then the request is classified according to the client class and put to the relevant client queue. The scheduler access these queues in first come first serve (FIFO) fashion, and assigns these messages to a virtual application instance with travel agent 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*-h2, is applied before the response is written to the client socket.

In this scenario response time is the QoS attribute of interest. The end-to-end response time consist of three main delay components: communication delay, connection delay and processing delay [24]. Here, we refer to processing delay as the response time, which is the main controllable delay component disregarding communication and connection delay due to network specific issues and different connection scheduling designs of different operating systems. The response time for a single request is the time difference between time stamp-h1 and h2 and it is measured by seconds. However, we refer to the average response time of the workloads in a 2 second sampling window as the response time. Let us denote the response time of workload A and B as $R_a$ and $R_b$ respectively. The session share between workload A and B as $S_a$ and $S_b$ where, $S_a + S_b = 20$. In addition to the main control objective, the scheduler is constrained with an ad-hoc policy/constraint with minimum number of sessions $S_a, S_b \geq 4$. Such ad-hoc policy/constraints are often implemented however depends on the requirements of the application. The session scheduling is the main performance bottleneck in this system to achieve the objectives, under varying conditions. Hence, we selected session allocation as the control input. According to relative guarantee and proportional resource allocation scheme the control input is the ratio of session allocations, represented by $S_a/S_b$ and the output variable is the ratio of average response time of the workloads, represented by $R_b/R_a$. However, these variables are inversely related to each other. This is because when the number of sessions for a workload increases the average response time decreases as a consequence of increase in resources to handle the requests. This implies, $S_a/S_b \propto R_b/R_a$. To incorporate this concern we modified the output to be $R_b/R_a$ to create more linear relationship\(^1\). For notational simplicity let’s denote $S_a/S_b$ and $R_b/R_a$ as $u$ and $y$ respectively. The control objective of this control system is to maintain a constant response time ratio $p_a/p_b$ between two workloads A and B depending on the importance of the travel agent. The costs, penalties and response time requirements can be used to decide this relative importance.

![Figure 2. The target system](image)

### 2.4 Classification of nonlinearity

When the above requirements and policies are embedded in design the control input $S_a/S_b$, can only take certain discrete values. Let $S_a = 20$ - $S_b$, $S_a, S_b \geq 4$ then $(S_a/S_b = 4/16, 5/15, 6/14, \ldots, 15/5, 16/4)$. Figure 3 shows the operating points that the controller can choose. These operating points are unequally spaced. We call region A and B as when A or B workloads gets more resources respectively. In region of A, spacing increases towards the end of the operating points where as in region of B spacing decreases towards the end of the operating points. The nonlinearities caused by restricted operating points exhibit the characteristics of static input nonlinearities, because these restrictions do not vary with time. Consequently, such static input nonlinearities may affect the performance of a linear controller when it is operating away from the nominal operating region. For instance, if an aggressive controller is used in region A controller will perform better because of the large steps, but in region B the same controller will create controller induced oscillations making the control system unstable.

Then if we consider the output $y (R_b/R_a)$ of the system the similar behavior to the input can be observed, however it cannot be predetermined. This is because $R_a$ and $R_b$ can take large range of values, causing $R_a/R_b$ ratio to have a large set of values. For instance, if $R_a = 400$ (sec) and $R_b = 1000$ (sec) then $R_a/R_b = 2.5$. Similarly, if $R_a = 1000$ (sec) and $R_b = 400$ (sec) then $R_b/R_a = 0.4$. As a consequence if workload disturbance of A class increases causing $R_a$ to increase while $R_b$ remaining at steady state value, then the output $y$ is to decay in a higher rate. In contrast, if the workload disturbance of B class increases causing $R_b$ to increase while $R_a$ remaining at steady state, then the output $y$ is to increase in a higher rate. Such nonlinearities could be characterized as *output nonlinearities* which could lead to performance issues when linear controller operates away from the nominal region. However, such output nonlinearities depends on system characteristics and workload conditions which may not be entirely characterized as static nonlinearity.

---

\(^1\) This conversion is required because typically negative feedback controllers are used. Not required for positive feedback controllers.
With the above characterized nonlinearities the performance of the linear control system may degrade, causing difficulties in design and autonomic control. In addition, depending on the different and changing requirements, workloads and the operating conditions, the control system should have the capability to operate in the entire operating region instead of limiting the region system was linearized.

2.5 Related work
There are different existing approaches to implement autonomic performance management in software systems. There are many threshold and heuristic policy based approaches applied in to complex software systems [5, 6, 17, 22] for resource provisioning. They are useful techniques because of their ability to handle complex policies/constraints and relatively easy to design/develop. However, they suffer from lack of well-founded design process to decide important design parameters (eg: thresholds) and the assumptions (on workload distributions, arrival rates).

The feedback control has been used to incorporate self-management capabilities into software systems in the last decade. Use of control theory to control web server systems [14, 24], cache and storage systems [19, 26] data centers/server clusters [21] and QoS management in multi-client class systems [20, 24] are such attempts. However, many existing literature mainly concerned with designing an inherently nonlinear software system with linear models. Such linear feedback control approaches are insufficient to capture the input and output nonlinearities discussed in previous section. Further, the relative guarantee design scheme and linear model based feedback control are discussed in cases of connection delay [7, 15, 24, 26], database processing delays [29] in web servers. However, additional policies on minimum resource constraint are not embedded in design and the nonlinear characteristics in input and output is not addressed. In addition, [25] discusses about the nonlinearities in the input-output relationship and possible issues, but goes on to use linear feedback control disregarding the nonlinearities.

In this work, a nonlinear modeling and control approach to capture the above characterized input and output nonlinearities will be investigated, in particular for a case of relative processing delay guarantees in multiple QoS class web service with limited amount of resources.

2.6 Hammerstein and Wiener block model
There are many nonlinear modeling techniques such as physical (first principle) models, neural networks, nonlinear-ARMA models and Volterra models that have been explored in control theory literature [27]. However, the model construction and controller design process becomes complex with such nonlinear models due to number of nonlinear operators and parameters [27]. In contrast, block-oriented nonlinear modeling techniques conceptualize the target system as
individual components (blocks) with defined inputs and outputs, which provide effective way to integrate static (non-time varying) nonlinearities in design. In addition when it comes to controller design, the block-oriented modeling preserves the design and numerical properties of linear controller design process [27]. Hammerstein and Wiener block structure is well known in block-oriented nonlinear control approach in literature to capture nonlinearities in the input and output [4, 8, 18, 30]. As shown in Figure 4 Hammerstein-Wiener block structure model has a linear block surrounded by two nonlinear blocks. The entire model can be divided in to two segments called Hammerstein and Wiener. The Hammerstein model has a nonlinear component preceding a linear component. In contrast, the Wiener model has a linear component following a nonlinear component. When these two schemes are combined together it is often called Hammerstein-Wiener model. The nonlinear blocks are assumed to model/capture the static nonlinearities in the system, while linear block captures the rest of the dynamics of the system. \( u \) and \( y \) denote the input and output, while intermediate variables \( v \) and \( w \) are not measurable.

![Figure 6. Mapping range of u to v](image)

In this work we investigate the applicability of Hammerstein model to capture the above characterized input nonlinearities in relative guarantee control system (Hammerstein-Wiener block structure modeling and control is outside the scope of this paper, details and experimental results can be found in [7]). Then we design Hammerstein model based control system for autonomic QoS performance and resource management for multiple QoS class systems extending the work in [7, 24, 25]. Section 4 provides the model estimation and controller design, followed by the experimental results on different operating conditions. Finally we conclude with a discussion and future work.

3. Approach

As shown in Figure 4, Hammerstein model is a block-oriented model, which has nonlinear component followed by a linear component. The nonlinear component is used to represent static input nonlinearities in the system. \( y \) and \( u \) are the output and control input of the target system respectively. \( v \) is the intermediate variable which is not measurable. In the model estimation, imaginary nonlinear and linear component have to be represented by formal relationships with respective variables. After the estimation of the (static) nonlinearity we can design a compensator framework to effectively remove the nonlinearity from the control system. Typically, inverse function of the estimated nonlinear component is derived to design the compensator framework. The final control system with the compensator framework is shown in Figure 5. The main difference compared to linear control system architecture (in Figure 1) is that with the integration of the compensator framework, the controller is operating with the intermediate variable (\( v \)) of the Hammerstein model. Even though, the nonlinearities are incorporated in the control system design we can still use linear controller and available formal design methodologies, which is an added advantage of proposed approach. In next section 3.1, 3.2 and 3.3 utilizing the scenario and the relative guarantee control scheme discussed in section 2, nonlinear, linear blocks of the Hammerstein model is estimated followed by control system design will be presented respectively.

3.1 Nonlinear block Design

The nonlinear component can be represented by many different nonlinear functions including piecewise linear, polynomials or nonlinear-Autoregressive Moving Average Models (ARMA) [27]. One of the necessary/useful condition of selecting the function is to have inverse function for it [27]. There are different types of system identification and estimation algorithms to derive parameters for different functions from input and output data (eg: [8, 12, 13]). However, if the nonlinearity is known at the design time the nonlinear function selection and its parameter estimation process becomes easier. For the scenario discussed in section 2.3, and as classified in section 2.4, the discontinuous operating points shown in Figure 3 may induce static input nonlinearity. Hence, if we give equally spaced operating points a linear controller may provide better performance in the entire operating region. After this physical analysis, assuming that the input nonlinearity is known, a method to estimate the nonlinearity is proposed avoiding the complexity of SID experiment based estimation. Let us assume that un-measurable intermediate variable \( v \) takes the values of \(-6, -5, -1, 0, 1, 2, 6\). The objective of selecting these points is to provide more equally spaced operating points by removing existing discontinuities. The range/values and the gap in-between values selected do not affect the design process as long as they are equally spaced. In the case of negative feedback control it is recommended to distribute the operating points either side of zero to improve performance. The next step is to estimate the nonlinear component using a formal relationship (model) between \( v \) and \( u \). To represent the input nonlinear component, polynomial function was selected, which
provides the convenience to construct its inverse function [27]. The goal is to derive a polynomial function \( f \) of order \( p \), to approximate \( v \) using the operating points of original system \( (u) \). As shown in Figure 6, range of \( v \) \((-6,0,5,6)\) was mapped to the range of \( u \) \((4/16,1,16/4)\) as data points. These data points were used to approximate \( f \) by curve fitting \((\text{cftool command in Matlab})\). A polynomial of degree 4 was sufficient for this approximation. It is always recommended to approximate a good fit while keeping the order of the polynomial as low as possible to avoid computational overhead. The model and its parameters are shown in Eq (1). Then the inverse function \( f' \) was approximated by interchanging the \( x \) and \( y \) parameters of the \text{cftool}. The model \( f' \) was also approximated with sufficient fit by a polynomial of order 4 (see Eq (2)). Afterwards, \( f' \) model (Eq (2)) was implemented as a software component and integrated to the system as a non-intrusive adaptor/compensator preceding the actuator.

\[
v = f(u) = -0.1584u^4 + 1.696u^3 - 6.959u^2 + 14.63 - 9.17 \quad (1)
\]

\[
u = f^{-1}(v) = 0.00038v^4 + 0.0034v^3 + 0.017v^2 + 0.186v + 1.006 \quad (2)
\]

\[
y(k) = 0.92y(k-1) + 0.24v(k) \quad (3)
\]

### 3.2 Linear Model design

The next step is to derive the linear component of the Hammerstein model to capture the rest of the dynamics in the system. For this purpose a SID experiment was conducted to derive the linear component. Typically, SID experiment is conducted to capture the dynamics of the system in a target operating region/set point. For instance, set points \((\text{e.g.: } p_1/p_2 = 0.5, 1, 2, 3)\) depends on the importance of the travel agent. Assuming nominal working condition is when both travel agents are equal (indicated by \( p_1/p_2 = 1 \)), a SID experiment is conducted to gather input-output data from the system. Specially designed (persistently exciting [14, 23]) control input signal is applied into the system and the output is observed for certain period of time. Typically, pseudo random signals, sinusoidal signals are used. For this prototype system, we used a pseudo random signal. After integration of the compensator system operates with transformed variable \( v \). The values of \( v \) was formed as a set \([-6,-5,\ldots,0,5,6] \) and in every fifth sampling instance we selected a random value from this set and passed it through the compensator to be actuated in the system. Likewise, we conducted this experiment and observed \( y \) for 600 samples. During the entire experiment, we applied 20 requests/sec workloads for A and B client classes. Then gathered \( v-y \) data pairs were divided into two sets called the estimation set and test set. The data samples between 1 to 400 periods were included in the estimation set. The rest of the data samples formulate the test set, to evaluate the model. There are different types of LTI models that could represent a system model. For instance, Autoregressive with exogenous terms (ARX), Finite Impulse Response [14] models. For this particular prototype system, the data in the estimation set was fit to an ARX model. The standard structure of the ARX model is as follows:

\[
\text{ARX}(n, m, d) \quad y(k) = \sum_{i=1}^{n} a_i y(k - i) + \sum_{j=1}^{m} b_j u(k - d - j)
\]

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), \( k \) - current sample instance.

The ARX(n, m, d) represents a relationship where output of the current sampling instance can be determined by output and input of the previous time samples. In the case of Hammerstein model \( u \) terms in the above standard model has to be replaced by \( v \). Given the estimation data set, the least square regression algorithm [23] was used to fit the gathered data to ARX model. We used different model structures and then evaluated the quality of the model by predictions with the test set. The goodness of fit \((R^2[14])\) statistic for both ARX(1,1,0) and ARX(2,2,0) was over 0.9, ARX(2,2,0) model having the highest \( R^2 \) value. Other higher order models did not improve the \( R^2 \) value that much. Hence, we can say that this system is fit in to second order ARX model. However, due to simplicity of design and sufficient accuracy we used ARX (1,1,0) model to represent the linear component of Hammerstein model. The model and the parameters are shown in Eq (3).

### 3.3 Controller design

In this section we provide the details of constructing the controller using the models constructed in section 4.1 and 4.2. Even though, we considered the nonlinearity of the system by integrating the compensator we do not have to design a complex nonlinear (eg: fuzzy, neural network based [14]) controller for this system. Assuming the system is linear and the model is represented by Eq (3), we can design a linear controller using existing linear controller design techniques. There are many types of controllers used for autonomic decision making including Propositional Integral (PI) controllers [14, 24], self-tuning controllers [19, 26] and predictive controllers [21]. For this particular system we designed a PI controller, which is one of the widely adopted controllers in industry due to their robustness, disturbance rejection capabilities and simplicity [14, 28]. The control law of the PI controller is shown in Eq (4), which calculates the control input \( u \) that should be applied in the system at current sample instance \( k \). First term in Eq (4) is the propositional component and the second term is the
integral component of the PI controller. \( e(k) \) indicates the control error, \( e(k) = y(k) - r \), where \( r \) is the set point. \( K_p \) (propositional gain) and \( K_i \) (integral gain) are called as gains of the PI controller. They are the tuning parameters of the controller to tune the performance.

\[
 u(k) = K_p e(k) + K_i \sum_{j=1}^{k} e(j) 
\]  

Apart from the main control objective of maintaining output sufficiently close to the set point, there are several other performance specific metrics considered in control system design. The stability, settling time, overshooting and steady state error are some of these matrices (interested readers refer \([14, 28]\)). Theses performance metrics are competing with each other, so that the optimal values of each of these metrics may not be achieved. The gains must be selected to tune the performance of the controller to achieve these performance metrics to an acceptable level. So that, calculating gains is an important step of controller design process. There are several well established formal techniques to aid the designer to select the gains of the controller while analyzing the close loop performance. The pole-placement design and root locus design are such techniques \([14, 28]\). Both these techniques could be used to decide the gains however, we utilize pole-placement design because root locus design allows only one gain to be altered at a time \([14]\). For more details of pole-placement design refer \([14, 28]\).

The above description provides the general concerns involved in design of a PI controller. For the case of Hammerstein model based controller design the control law has to be modified by replacing \( u(k) \) by \( v(k) \) in Eq (4), however the design process remains the same. Then at runtime, \( v(k) \) is calculated by the PI controller and passed through the compensator to be converted to \( u(k) \), before reaching the actuator of the original system as shown in Figure 5.

A common nonlinearity found in many physical systems is the phenomenon called windup \([3]\). The windup occurs because of the limited range of the actuation, which could cause temporal instabilities and large transient responses when large disturbances are encountered \([3]\). For instance, in the prototype system available sessions for communication are limited. These limits have to be implemented in the controller to avoid violation of these hard constraints. The actuator limits of original system is \( u_{\max}=4 \) (16/4) and \( u_{\min}=0.25 \) (4/16). In the Hammerstein controller the limits has to be converted using Eq (1) to \( v_{\max} = f(u_{\max}) \) and \( v_{\min} = f(u_{\min}) \).

Finally, the calculated ratio \( u = S_a/S_b \) by the controller, has to be decoded back to individual session allocations. An algorithm was proposed in \([24]\) for this calculation, given the ratios from the controllers in a multiple QoS class systems. Although our scenario describe only 2 client class case, the above modeling and controller design approach can be used for multiple QoS class system similar to \([7, 24, 25]\). There each adjacent QoS class i and i-1 is controlled by a controller designed using the above approach. However, with the integration of the compensators the Hammerstein model based controller operates with the intermediate variable require extending the proposed algorithm in \([24]\) as listed in table 1.

4. EXPERIMENTAL RESULTS

In this section we present the performance of the nonlinear Hammerstein model based controller designed using the approach in section 3, with different workload/operating conditions. In order to do a comparative performance analysis we introduce linear model similar to the approaches in \([24, 25, 29]\). Using the design process discussed in section 3.2 and 3.3, a linear model (see Eq (5)) and two controllers were designed. The control system in this case is same as Figure 1.

\[
y(k) = 0.84 y(k-1) + 0.59 u(k) 
\]
To show the performance issues due to nonlinearities, using pole-placement methodology [14] two linear (called as aggressive and less aggressive) controllers were designed placing poles of the characteristic equation at \((0.5, 0.5)\) and \((0.7, 0.7)\) respectively. Similarly, Hammerstein model based controller was designed placing poles of the characteristic equation at \((0.5, 0.5)\) for the fair comparison. The gains of the controllers are listed in Table 2. We opted to use analytic (synthetic) workloads for these experiments instead of trace base workloads (workloads recorded from a real system) because it is easy to understand the behavior of the control system when it faces the workloads with interested characteristics. Such types of characteristics are hard to find/accessed in trace base workloads. The section 4.1 compares the designed controllers in nominal operating region. Then section 4.2 demonstrates the performance issues of linear controllers when they operate way from nominal operating region, compared to Hammerstein controller. In 4.1 and 4.2 QoS classes were considered having similar QoS requirements (or equally important), by setting set point at 1. Finally, in section 4.3, the performance is analyzed when the QoS requirements/importance of the QoS classes are different maintaining set point signal at 1.5 and 0.6. To statistically compare the results of these experiments Sum of Square Errors (SSE) operator will be used, which indicate how effective the controllers are in achieving the control objective/set point.

### 4.1 In the nominal operating region

In this section the performance of the controllers are compared in the nominal operating region. The linear models in Eq(3) and Eq (4) where derived under workloads assuming both QoS classes are equally important. This can be achieved by fixing set point = 1. Hence, the desired nominal operating region is when the workload conditions of both client classes are 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 7.

Due to the disturbance at 50th sample high deviations can be observed in less aggressive linear controller compared to aggressive linear and Hammerstein controllers. This is because the less aggressiveness of that controller in rejecting workload disturbances. However, the steady state behavior is satisfactory, because they achieve the set point with low deviations in both linear and Hammerstein controllers. Other thing to note in both linear controllers is the performance at the start up (sample 1 to 12). Both linear model based controllers has large deviations from the set point compared to the Hammerstein controller. The disturbance due to JIT compilation at the start up is common in C#/Net/Java implementations. So that, the performance of linear controllers degrades due to such un-modeled dynamics. Investigation of the control/decision signal indicated that the existing input nonlinearities affect the settling time at the startup. However, nonlinearity compensated Hammerstein controller provides significantly better performance at the startup showing better disturbance rejection.

However, overall (especially in steady state) the performance of the linear controllers is satisfactory in the nominal region. This shows that

<table>
<thead>
<tr>
<th>Controller</th>
<th>(K_p)</th>
<th>(K_i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less aggressive linear controller</td>
<td>0.59</td>
<td>0.15</td>
</tr>
<tr>
<td>aggressive linear controller</td>
<td>1.00</td>
<td>0.42</td>
</tr>
<tr>
<td>Hammerstein controller</td>
<td>2.79</td>
<td>1.04</td>
</tr>
</tbody>
</table>
A properly tuned linear model based controller can achieve the desired performance in nominal region. The nonlinear Hammerstein model based controller performs equally well in steady state and significantly better at the startup compared to the linear controllers. The SSE calculations for this experiment is shown in Figure 12a), which indicates the better performance of Hammerstein controller with low SSE.

4.2 Away from nominal operating region

To enable the control system to operate away from the nominal region the workloads was increased for A and B to their highest capacity alternatively. Till 30th sample workloads of 10 requests/sec was applied for A and B. Then, at 30th sample A workload increases from 10 requests/sec to 30. 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 workload. Afterwards, at 80th sample the A workload reduces to 10 requests/sec. Then at 100th sample, B workload increases to 30 requests/sec from 10 requests/sec.

The performance comparison of two linear and Hammerstein model based controllers are shown in 8a) , 8b) and 8c) respectively. First let’s analyze the performance of the linear controller in the region A (between 30th and 80th sample). For aggressive and less aggressive controllers settling times are approximately 7, 16 sample periods respectively. This shows that when the gains of the controller increase the settling time after a disturbance decreases. Then it settles down and reaches the steady state, achieving the set point with small error. In contrast, after 100th sample when the linear controllers are operating in region B, the steady state performance is drastically different. The aggressive controller after the high workload disturbance at 100th sample moves to high oscillatory/unstable behavior with large steady state error. However, the less aggressive controller even with large overshooting around 100th sample shows satisfactory steady state performance. This is an indication that the aggressive controller is too aggressive to operate in region B creating controller induced oscillations. However, the same controller operates much better in region A compared to less aggressive controller. From these experimental results the aggressive controller demonstrates the best performance in region A, while less aggressive controller demonstrate the best steady state performance in region B. Thus, we can say that a linear model fails to achieve effective performance under this condition. In addition, they fail to operate in the entire operating region under changing conditions. Further, the discriminative behavior in different regions leads to model uncertainties becoming an additional performance/design metric. Consequently, leads to loss of flexibility in control system design.

From results of Figure 8c), the Hammerstein controller provides much better performance compared to both linear controllers demonstrating better steady state, overshooting and setting time performance in both region A and B. The similar performance can be achieved with even high aggressive controller. As a consequence, the flexibility in design is established. In Figure 12b), the SSE operator is
significantly low (7) for the Hammerstein controller compared to both linear controllers. The above performance degradations in linear controllers are due to nonlinearities induced by the discontinuous control input. The discontinuous control input imposes difficulties for the linear controller to settle down to a suitable operating point when it operates in region B. Even when there is small change in the output due to noisy workloads, un-modeled dynamics, the controller varies the control input in large steps leading to controller-induced oscillations in the case of aggressive controller. In Figure 10, after the 100th sample period the actuation (u) of linear controller, oscillates significantly compared to the Hammerstein model based control signal. Less, oscillatory control signal is seen in Hammerstein controller because it operates with equally spaced (transformed) control variable (v) reducing the existing nonlinearity in the original system. The Hammerstein model control input v, has a symmetric behavior around the nominal operating region which is transformed to the original actuation (u) using the integrated nonlinear compensator. These behaviors in the control signals generated by the linear and Hammerstein controllers explain the steady state performance after 100th sample.

4.3 Different QoS importance/requirements

In the above sections performance of the relative guarantee scheme was presented when both client classes are equally important, or has similar requirements. In this section we provide performance of the above designed controller in the case of different QoS importance/requirements which needs effective performance differentiation. This objective is translated to relative guarantee control system using the set point signal/value. For instance, if A client class is important than B, we can set the set point (p_A/p_B) to greater than 1 (e.g: p_A/p_B = 1.5, 2). If the set point is 2, the controller will try to maintain the response time of B twice as much as of A. Similarly, we can give priority to B by having set point less than 1 (e.g: p_A/p_B = 0.7, 0.5). The set point depends on the requirements of the system.

Firstly, performance of the linear controllers and nonlinear controller is compared when the set point p_A/p_B = 1.5. For this case 10 and 20 requests/sec was applied for A and B respectively, till 50th sample. Afterwards B increased its workload to 30 requests/sec. Figure 9 shows the performance of the controllers. The performance of both linear controllers is similar to what was observed in the previous section. The aggressive controller shows highly oscillatory steady state behavior after the disturbance at 50th sample but less overshooting compared to the less aggressive controller. Although, less aggressive controller takes time to settle down at the start up, it shows better steady state behavior. In contrast, Hammerstein controller provides significantly better steady state behavior with less overshooting and settling time before and after disturbance at 50th sample, compared to both linear controllers. In addition, it shows better settling time at the startup. The aggressive linear controller shows oscillatory behavior because the controller is operating with the inputs of region B. The SSE calculations (see Figure 12c) are 13, 18 and 8 for less aggressive, aggressive linear controllers and the Hammerstein controller respectively, indicating the better performance.
Secondly, the set point \((p_a/p_b)\) was fixed at 0.6 in the following experiment. 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 11 shows the experimental results. The performance comparison with SSE values in Figure 11d) shows Hammerstein controller performs much better compared to both linear controllers. The performance of the aggressive linear control shows better disturbance rejection at the startup and at the 50th sample with low settling time and overshooting compared to less aggressive linear controller. Although the less aggressive linear controller shows low disturbance rejection qualities, it provides better steady state error/performance. In contrast, Hammerstein model provides much better disturbance rejection capabilities with steady state behavior.

4.4 Overloaded condition
In this section we compare the performance of the controllers in the overloaded conditions. Under overloaded conditions system shows highly nonlinear characteristics because the available resource capacity cannot handle workload. This may cause high steady state errors. To overload the system, workload settings listed in Table 3 is used.

<table>
<thead>
<tr>
<th>Sample</th>
<th>A workload</th>
<th>B workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

For this case as well, set points are set to 1, indicating equal importance of the client classes. Figure X provides the performance comparison.
The performance of the less aggressive controller shows high settling time at the disturbance around at the high workload disturbance at the $30^{th}$ sample. It takes around 30 samples to settle down with high overshooting. Similar, behavior is seen at the $100^{th}$ sample when B workloads are increased. It takes around 45 samples to settle down. The aggressive controller on the other hand settles down with less settling time and overshooting under the disturbance at $30^{th}$ and $100^{th}$ samples. In contrast, the Hammerstein controller shows better disturbance rejection capabilities with significantly low settling time and overshooting under the both disturbances. At the steady state aggressive controller and hammerstein controller demonstrate similar behavior. When the control signal is analyzed the actuator saturation can be seen indicating the inadequacy of available resources.

5. DISCUSSION

From the experimental results in the section 4, we can conclude that linear models sufficiently capture the dynamics of a software system in the nominal operating region. However, nonlinear Hammerstein model captures the dynamics, in particular the static input nonlinearities of the system in the entire operating region. As shown in section 4 linear model based control provides poor performance and less flexibility in control system design when the system operates away from the nominal operating region. In contrast, Hammerstein model based design and control provides significantly better performance and design flexibility. In particular, the performance will not be discriminative for a particular client class consequently, better performance isolation when both client classes are equally important. In addition, the designer does not have to tradeoff between QoS classes while tuning the aggressiveness of the linear controllers. Furthermore, the comparison of startup performance indicates the disturbance due to un-modeled dynamics like JIT compilation and garbage collection can be rejected effectively by the Hammerstein controller compared to linear controllers. Similarly, effective performance differentiation can be achieved when the control system has different QoS requirements/importance/set points. Figure 12 justifies the better performance statistically, showing significantly low SSE in the case of Hammerstein model for all the above experiments.

Such performance improvements were obtained because, the modeling approach described in section 3 enables the linear controller to operate on (equally spaced) transformed control variables, instead of the discontinuous operating points in the original system. Consequently, the nonlinearity induced in the original system could be compensated with the addition of the non-intrusive compensator without needing any modification to managed software system (this is a vital requirement in autonomic computing). The proposed design process is a black-box modeling technique which does not require any previous knowledge on the workloads, system component behavior. Further, proposed nonlinear modeling technique preserves the numerical/design properties of linear control system design, enabling use of a linear controller providing generic/formal design process instead of other expensive nonlinear control approaches.

Both the design/implementation effort required and the runtime computational overhead are important concerns in autonomic computing [31]. In this paper Hammerstein model was designed assuming static input nonlinearity is known. In such cases design of the Hammerstein model becomes simple because only additional step required is deriving the nonlinear function and its inverse after physical analysis. If the input nonlinearity is not known, special SID experiments and validations has to be carried out to characterize the nonlinear and linear components. There has been work on Hammerstein model identification (eg: [8, 12, 13]), which could be used in such situations. A implementation of Hammerstein SID can also be found in Matlab [1]. While the approach we discussed introduces new non-intrusive adapter/compensator to the system, there is computational overhead imposed compared to an original control system. If we integrate a polynomial with order $N$, it introduced additional $N(N+3)/2$ multiplication and addition operations (however, negligible in current operating environments).

The proposed approach is also applicable in other cases where discontinuous operating points exists. For instance, to optimize power consumption in CPUs by adjusting the operating voltage/frequency using dynamic voltage scaling technology. The possible voltage transitions in [16] have discontinuous operating points, which could utilize our approach. These cases would also exist in many multi-tenant system where relative generate control is required, such as three tier e-commerce systems with limited storage/database connection pools, web servers with limited number of process threads, data centers with server clusters and CPU allocations.

As future work, this technique will be tested on scalable multi-tenant cloud computing environments (eg: Microsoft Azure).
6. Differentiated QoS management for Three QoS classes case

When there is more than two client classes are considered in relative guarantee control scheme the control system design becomes a multi-input-multi-output (MIMO) problem. There are different ways to approach this problem. A multi controller based approach is presented in [7, 24, 25]. There each adjacent QoS class i and i-1 is controlled by a controller designed using the above approach. That is for a system with N client classes there will be N-1 controllers. It is easy to model and design this technique because the model and the controller constructed for two client class case can be used in all the controllers. However, the theoretical limitation is it disregards the interactions between the inputs and outputs in the MIMO case. To incorporate such interactions between inputs and outputs MIMO modeling and control system design. Even in the linear modeling case this is a complex design process compared to SISO implementation. However, in the case of Hammerstein model interactions between input nonlinearities has to be approximated as well. This requires investigation of highly complex nonlinear SID technique. Due to these issues in approximating the system as MIMO model, in this section modeling and control system design utilizing the first approach will be presented, keeping second approach as future work. The following proposed technique can be extended to more than two client classes, however we discuss a scenario where we have three QoS (client) classes with resource constraints.

Scenario: There are three agents who are interested in the services provided by the server system discussed above. Let’s call them A, B and C agents. The response time is the QoS attribute of interest. The objective of the software system is to maintain the response time of the agent workloads in $P_a P_b P_c$ differentiation levels while allocating the available 20 sessions available. Let us denote the response time of workload $A, B$ and $C$ as $R_a, R_b$ and $R_c$ respectively. The session share between workload $A, B$ and $C$ as $S_a, S_b$ and $S_c$ where, $S_a + S_b + S_c = 20$. In addition, the scheduler is constrained with an ad-hoc policy/constraint with minimum number of sessions $S_a, S_b, S_c \geq 4$.

Unlike two client class case due to the constraints and contention in QoS classes for resources, it is hard to formulate the possible control input range. As mentioned earlier our objective is to design a controller to control each consecutive QoS class. Hence, we assume that all other classes maintain their workloads around the minimum range which can be satisfied by the minimum resource allocation. Then we define the control input range for two client classes maintaining the minimum resource allocation at all times. Form the above requirements keeping $S_a = 4, S_b S_c$ can 4 to 12. If $S_a = 20 - S_b - S_c$ where $S_b = 4, 5, 6 \ldots 12$. Then $S_b S_c = \{4, 12, 5, 11, \ldots 11, 5, 12, 4\}$. This range also shows the unequally spaced operating points, which would lead to the similar performance issues discussed in the previous section. To address this issue we use the same design methodology by estimating a Hammerstein block structure model.

Nonlinear and linear block estimation

First step is to select a suitable range for the intermediate input variable (v) in Hammerstein block. We selected v to be range of {-2,-1.5,...0,...1.5,2}, which has equally spaced points. To create the function between v and u, we mapped range of v{-2,-1.5,...0,...1.5 ,2} to values of u {4/12,5/11,....11/5,12/4}. Again we estimated f(u) with a 4th order polynomial using curve fitting with a fit of 0.99. Afterwards, we created the inverse of that function which is required to design the nonlinear compensator. Both functions are as follows.

$$v = f(u) = -0.112u^4 + 0.965u^3 - 3.319u^2 + 6.198u - 3.724$$ (1)

$$u = f^{-1}(v) = 0.012v^4 + 0.0450v^3 + 0.118v^2 + 0.485v + 1.001$$ (2)

Next step is to approximate the linear component of the Hammerstein model after integrating the nonlinearity compensating component into the system. A SID experiment was conducted using the transformed variable (v) as the input signal. Then the output was observed for 600 samples. Using the similar process presented in section XX. The model was fit in to first order ARX model with sufficient accuracy. The structure of the model is as follows:

$$y(k) = 0.92 y(k-1) + 0.24 v(k)$$ (3)

Controller design

We used PI controller for this implementations as well. Using the model of the linear controller was designed. However due to multiple controller integration and the compensator framework the structure of the control systems is complex. For system with n number of QoS classes n-1 controllers and commentators will be there as shown in figure XX.
For the above three client class case we implemented the above control system design with 2 controllers and 2 nonlinear compensators. The inputs and outputs and the set point is listed in table 3.

<table>
<thead>
<tr>
<th>Controller</th>
<th>output</th>
<th>input</th>
<th>Set point</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R_a/R_b</td>
<td>S_a/S_b</td>
<td>P_a/P_b</td>
</tr>
<tr>
<td>2</td>
<td>R_b/R_a</td>
<td>S_b/S_a</td>
<td>P_b/P_a</td>
</tr>
</tbody>
</table>

The input generated by the controller will be fed in to the aforementioned algorithm to calculate the individual resource share. PI controllers where designed as the controllers using the pole placement design.

**Experimental results**

This section provides experimental results conducted in different conditions. Firstly, we cover the experiments in the nominal region and when the system is overloaded in the same condition. Secondly, the performance in the away from nominal region is covered. Finally, the experiments are conducted in different set points simulating different importance of the client classes. The syntactic workloads where used in this case as well. To do a comparative study, again we design two linear PI controllers called as aggressive and less aggressive placing the poles at 0.5 0.5 and 0.7 0.7 respectively. Hammerstein controller was designed placing the poles at 0.5 0.5. The gains are listed in table 4.

<table>
<thead>
<tr>
<th>Controller</th>
<th>K_p</th>
<th>K_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less aggressive linear controller</td>
<td>0.52</td>
<td>0.18</td>
</tr>
<tr>
<td>aggressive linear controller</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>Hammerstein controller</td>
<td>1.6</td>
<td>0.62</td>
</tr>
</tbody>
</table>

**6.1 Nominal region**

Since the above models were conducted assuming both client classes are equally important the target operation is at 1, where p_a:p_b:p_c = 1:1:1. All three client classes start off with sending 5 requests/sec till the 30th sample, then they all increase their workloads to 10 requests/sec afterwards. Figure XX shows the performance of the controllers.
The less aggressive linear controller shows low disturbance rejection capabilities compared to the aggressive and Hammerstein controllers. Around the 30th sample when the workloads are increased for all client classes it takes approximately 20 samples (from 30th to 50th) to settle down showing low disturbance rejection capabilities. The both aggressive linear controllers show better disturbance rejection and steady state performance. However, all linear controllers take many samples at the start up to settle down. In contrast, both Hammerstein controllers show steady state performance, disturbance rejection capabilities and significantly better startup performance.

Another experiment was conducted to test the performance under the overloaded condition in the nominal region. Similar to the above workload were used however to at the 30th sample workloads of all client classes were increased to 15 requests/sec afterwards. Figure X shows the performance of the controllers.

After the 30th sample, all controllers show oscillatory behavior around the set point, because under overload conditions it is hard to guarantee required performance, while settling to single operating points. Less aggressive controller shows better performance in this case because it does not overreact to the small control errors. However, both aggressive controllers show large deviations from the set point because of overreacting control errors. Although there are oscillatory behavior in both Hammerstein controllers when the system is overloaded, the performance is significantly better than the aggressive controller.

### 6.2 Away from nominal region

In this section keeping the differentiation factors as $p_a:p_b:p_c = 1:1:1$, the system is forced to operate away from the nominal region by applying high workload disturbance for all three client classes interchangeably. Table 5 shows the workload intensity with respect to sample time. The performance of the controllers are shown in Figure XX.

<table>
<thead>
<tr>
<th>Sample</th>
<th>A workload</th>
<th>B workload</th>
<th>C workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>10</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>80</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>100</td>
<td>10</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>150</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>170</td>
<td>30</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>
Firstly, let’s compare the performance of the linear controllers. The settling time and overshooting of the less aggressive controller is high at the 30th and 100th and 170th samples due to disturbance compared to the aggressive linear controller. But the steady state performance of the aggressive linear controller is poor because of the high steady state error. In contrast, the Hammerstein controllers provide much better steady state performance compared to aggressive linear controllers and better steady state performance competed to less aggressive controllers.

6.3 Different importance/set points
In this section we test the performance of the designed controllers when the importance between the client classes is different. We conducted three experiments with different differentiation factors among classes.

Firstly, we set the differentiation factors $p_a:p_b:p_c = 1 : 1.5 : 2.25$. This is translated to the two controllers as set points with $p_a/p_b = 1.5$ and $p_b/p_c = 1.5$. For instance, if $R_a = 0.4$ (sec), $R_b$ has to be 0.6 (sec). If $R_a = 0.6$ (sec) then $R_b$ should be around 0.9 (sec). However, since the control system guarantee minimum resources ($S_a,S_b,S_c \geq 4$) allocation, the workload setting should be intensive enough to increase the response times of client classes to provide these guarantees. Hence, For this experiment 10, 10, 15 requests/sec workloads were applied for A, B and C client classes respectively. Figure XX shows the performance of the controllers.

The performance of the controller handling A, B client classes maintains the required differentiation level in all linear and Hammerstein cases. But, the controller handling B, C client classes shows highly oscillatory behavior. This is because to increase the response time to the required level, the resource for class C has to be shirked. However, because of that the response time increases rapidly moving the output higher than the required value. As a consequence, the controller has to allocate more resources. Such behavior causes the oscillatory behavior in the controllers. The required value that has to be maintained by C is located highly nonlinear region of the input-output curve of the system, hence it is hard to be achieve required level. Nonetheless, less aggressive controller takes more time to settle down at the startup compared to aggressive and less aggressive controllers. In addition the steady state performance of the aggressive controller is poor compared to Hammerstein controller.

Secondly, we compare the performance of the controllers when differentiation factors $p_a:p_b:p_c = 1 : 0.6 : 0.36$. This is translated to the two controllers as set points with $p_a/p_b = 0.6$ and $p_b/p_c = 0.6$. For this experiment 20, 10, 10 requests/sec workloads were applied for A, B and C client classes respectively. Figure XX shows the performance of the controllers.
The oscillatory behavior is because of the same reason explained in the previous paragraph. The performance of the less aggressive linear controllers has very high overshooting and settling time at the start up. It is close to 60 sample periods. The less aggressive controller also takes 30 samples to settle down, but maintains the required levels on average. The Hammerstein controller settles down in less than 10 samples but shows oscillatory behavior similar to linear controllers. But the steady state behavior is comparatively better than aggressive linear controller and the start up settling time is significantly better than less aggressive controller.

Finally, another experiment was conducted by setting differentiation factors between client classes in $p_a : p_b : p_c = 1 : 1 : 1.5$. In this case, the importance between A and B is equal, while the importance of the client class is 1.5 compared to both classes. Advantage of this setting is that we can test the performance of the control system instead of operating in the highly nonlinear region of input-output curve of the system. Workloads of 10, 1, 20 requests/sec were applied for A, B and C client classes respectively for this experiment. The performance is compared in Figure XX.

![Figure X. Performance of the controllers in the with overloaded conditions in nominal region](image)
7. REFERENCES


