Dynamic Performance Management in Multi-tenanted Business Process Servers Using Nonlinear Control

Tharindu Patikirikorala[1], Indika Kumara[1], Alan Colman[1], Jun Han[1], Liuping Wang[2], Denis Weerasiri[3], and Waruna Ranasinghe[3]

[1] Swinburne University of Technology, Australia
[2] Royal Melbourne Institute of Technology, Australia
[3] WSO2 Inc, Palo Alto, CA

Abstract. The methodologies to develop multi-tenanted architectures have been investigated in the recent literature due to the popularity of cloud computing. A number of challenges need to be overcome if multi-tenanted architectures are to be effective and efficient. Among the challenges is the management of performance properties while effectively sharing the limited resources between the tenants. This work presents an approach to design such a management system for a multi-tenanted business process server. This approach not only enables performance to be maintained at different levels for different tenants depending on their priorities, but also autonomously detects the overloads of aggressive tenants and dynamically changes the control objectives to safeguard the business operations of other tenants. The novelty of the proposed approach is the use of the nonlinear feedback control. The experiment results indicate that the proposed nonlinear control approach achieves the objectives much better compared to the existing fixed and linear control techniques.

1 Introduction

With the popularity of the cloud computing, multi-tenanted architectures are becoming important to realize economies of scale in a flexible manner. In such architectures, pools of resources are shared by multiple tenants/customers thereby raising many challenges for the practitioners. These challenges include enabling *tenant-specific* customizations, data isolation, security isolation and performance management while effectively achieving high resource utilization and sharing. However, these objectives are often competing with each other.

Researchers from SAP present three implementation and deployment options for multi-tenanted software systems in [12] with different levels of trade-offs. 1) *Shared infrastructure*: The tenant specific application instance is deployed in individual virtual machine (VM) instances. This option enables customizations to be provided as per tenant requirements, data and performance isolation [7]. The main drawbacks of this approach are the under-utilization of resources and requirement of a significantly large number of VMs to cater a larger number

C. Liu et al. (Eds.): ICSOC 2012, LNCS 7636, pp. 206–221, 2012.

of tenants [7]. 2) *Shared Middleware*: In this option, the application instances of many tenants share a single middleware instance with the view of having a dedicated middleware instance for each tenant. Consequently, higher resource utilization, tenant consolidation and customizability can be achieved compared to the first option. However, effective performance and resource management techniques have to be provided by the multi-tenanted middleware provider. 3) *Shared application*: In previous two options the application is unaware of the tenants. In this option, tenant management techniques have to be integrated and, in the ideal case, any application instance can serve requests of any tenant. This option enables higher resource utilization and consolidation, but the monolithic application makes customizability and runtime isolation more problematic.

Although the shared infrastructure mechanism can be enabled by the well established virtualization technology to share hardware resources, it is evident that the shared middleware option provides many advantages to the stake-holders. However, as mentioned the issues related to multi-tenancy such as i) data, security and execution isolation, ii) performance and resource management, iii) overload detection and protection, iv) scaling the tenant application instance have to be considered in the development of the multi-tenanted middleware compared to the traditional middleware.

This paper presents a middleware platform which takes into account the first three requirements[1]. Work in [14] has implemented a multi-tenanted Business Process Server (BPS) which enables business process deployments for multiple tenants with the capabilities of data, security and execution isolation. Here, we extend the same BPS and focus on the implementation of performance and overload management. To maintain the performance properties of tenants at acceptable levels depending on the business objectives or priorities in a multi-tenanted environment effective resource management is required at runtime [12,4]. In order to achieve these control objectives this work uses a relative management scheme [9], which also provides facilities to adjust the tenant priority levels at runtime. However, the novelty of our approach compared to the existing approaches (e.g., [9,13]) is that we take into account the nonlinear dynamics of the system and design compensators to reduce the impact of the nonlinearities on the management system, which enables the control system to operate in a wider operating range. Furthermore, the management system is also equipped with a novel overload management mechanism, to detect and react efficiently by degrading the priority of the aggressive tenant workloads in order to safeguard the business operations of the rest of the tenants. The experiment results conducted by deploying the BPS and control system in a VM environment and under instantaneous and real-world workload settings indicate that the proposed nonlinear control methodology achieves the management objectives significantly better compared to the existing fixed resource partitioning and linear control schemes.

[1] Scaling includes migration of the specific application to another middleware instance or starting new middleware instances [12]. This is out of the scope of this paper.

2 Background

WSO2 BPS: WSO2 Stratos Business Process Server[2] is a multi-tenanted workflow engine which executes business processes compliant with WS-BPEL standard, and is built on top of WSO2 Carbon platform[3]. WSO2 Stratos BPS also supports data and execution isolation [14]. Figure 1a) shows its high level architecture. A user of a tenant can consume a business process via a business process endpoint, which is a standard web service endpoint. *WSO2 Stratos Identity Server (IS)* provides security services such as authentication and authorization of tenants and users. *WSO2 Stratos Manager* is used to provision and manage tenants including tenant subscriptions and billing. Business process artifacts for each tenant are kept in *WSO2 Stratos Governance Registry* which is a multi-tenanted governance tool that follows the shared database and shared schema multi-tenanted data architecture pattern defined in [2]. WSO2 Stratos BPS uses Apache ODE[4] as its BPEL execution run-time environment. The ODE-Axis2 Integration Layer encapsulates the tenant-ware business process executions. In the current multi-tenanted BPS instance, a single ODE process engine is shared by multiple tenants. Therefore, a workload of each tenant is treated equally, which does not maintain the performance at required priorities or does not detect and avoid interference between tenants under the overload situations.

Relative Guarantee Scheme: Maintaining absolute values for the performance properties at runtime is difficult due to the workload characteristics of software systems [9]. As a result, maintaining the performance properties of multiple classes (or in other words tenants) at different priorities levels using relative performance management scheme has been identified as a promising alternative [20,9]. In the relative management scheme, the performance properties of the classes are maintained proportional to the performance differentiation factors derived from the business or system design requirements. Let Q_i, P_i be the actual performance property value of interest and the specified differentiation factor respectively of a class i =0,... n-1, out of n number of classes. Between the pair of classes i and j, the objective of the relative management scheme is to maintain $\frac{Q_j}{Q_i} = \frac{P_j}{P_i}$ (i $= 0 \ldots n-1, i \neq j$) at runtime under varying workload conditions. For instance, $\frac{P_1}{P_0} = 2$ means that the performance attribute of class_1 has to be maintained twice as of class_0. However, the main challenge to maintain the performance differentiation ratio in a shared resource environment is to compute the dynamic resource allocation ratio $\frac{S_j}{S_i}$ (where S_i and S_j, are the resource caps for i and j classes respectively). Lu et al. in [9] proposed a linear feedback control algorithm to automate relative performance and resource management.

[2] http://wso2.com/products/business-process-server/
[3] http://wso2.com/products/carbon/
[4] http://ode.apache.org/

3 Related Work

Much of the work related to multi-tenanted systems has been done in the last few years including work on maturity levels [2], data isolation [18,19], enabling customizations [11] and tenant placement [3,6].

A survey of the application of control engineering approaches to build self-adaptive software system can be found in [15]. The relative performance management scheme combined with linear model based feedback control is applied to manage the connection delay in web servers [9,10] and database processing delay in database servers [13]. In addition, Ying et al. analyze the nonlinearities and the related issues of relative management scheme in [10]. These existing control approaches use linear models to represent the inherently nonlinear behavior of the software system. Such models are insufficient to capture the nonlinearities exist in the relative management scheme as shown in this paper.

Feedback control has been also applied for the case of multi-tenanted software system in [8]. Their work is based on adjusting the thread priorities of different tenants depending on their level of importance. Consequently, this approach cannot provide finer grain performance differentiations depending on the business requirements because they rely on thread priority scheduling of the underlying environment. SPIN [7] implements a performance isolation technique by detecting aggressive tenants and reacting to reduce the negative effect on other tenants. However, this work does not consider typical performance variables such as response time in the management solution and the solution depends on highly stochastic workload rates, which are hard to measure or predict accurately.

In this paper we focus on the performance management of a multi-tenanted BPS. In particular, compared to general linear control approaches, we integrate a nonlinear control mechanism proposed in [16] to implement a relative management scheme and a novel overload detection method to achieve the performance objectives of the multi-tenanted BPS. To the best of our knowledge this is the first approach that provides data, security and execution isolation and performance management together in a multi-tenanted BPS.

4 Problem Overview and Analysis

In this section we present an overview of the performance management problem of a BPS serving multiple tenants.

4.1 Assumptions

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

1) The number of tenants (say n) placed in a single BPS instance is known at the design time.

2) The number of concurrent process instances (or worker threads) is considered as the shared and main bottlenecked resource as mentioned in [12]. This is because the hardware resources such as CPU and memory cannot be controlled in the granularity of the tenants at the middleware level.

3) When a tenant has overloaded the BPS, a portion of the workload will be rejected to avoid instabilities due to unbounded growth of the queues. Alternatively, tenant migration or scaling out can be done, which are out of the scope of this paper.

4) BPS profiling has been done, and the number of concurrent process instances that could exist in the BPS is determined (say S_{total}) based on the response time requirements of the tenants. This property is important to constrain the total resources and concurrent number of process worker threads affecting the response time of the BPS.

4.2 Performance Management Problem Definition

The main control objective is to share the available S_{total} process instances in an effective way to maintain the response time at levels that are acceptable or depending on the tenant's priority levels, with low interference between 0, 1, ..., n tenants. In addition, the unpredictable overload situations of an aggressive tenant have to be automatically detected and resource allocation decisions have to be made to protect the performance of other tenants.

To achieve these performance objectives we adopt a hybrid of fixed and flexible resource management mechanisms as recommended in [4] to increase resource sharing and utilization. That is, some amount of process instances (say, $S_{i,min}$, where $i = 0, 1, \ldots n$) need to be reserved for each tenant during the entire period of operations in order to avoid starvation of resources, however under unpredictable workloads the discretionary resources $s_{total} - \sum_{i=0}^{n-1} s_{i,min}$ are shared between the tenants depending on the demands. To enable the flexible resource partitioning $s_{total} - \sum_{i=0}^{n-1} s_{i,min} > 0$. With these settings, the control objectives are now specified based on the relative performance and resource management scheme as follows.

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

$$S_i(k) \geq S_{i,min}, \qquad \sum_{i=0}^{n-1} S_i(k)) = S_{total} \tag{1}$$

Example: Let us consider a BPS instance with two tenants, 0 and 1. $R_0(k)$ and $R_1(k)$ are the response times of two tenants respectively at the k^{th} sample. Furthermore, the process instance caps are $S_0(k)$ and $S_1(k)$ where $S_0(k) + S_1(k) = S_{total} = 20$ and $S_{0,min}, S_{1,min} = 4$. According to [9] the manipulated variable is $u = \frac{S_0(k)}{S_1(k)}$ and the controlled variable is $y = \frac{R_1(k)}{R_0(k)}$.

4.3 Analysis of Nonlinearity

Input nonlinearity: When the above requirements are embedded in the design, the manipulated variable $u = \frac{S_0(k)}{S_1(k)}$ can only take certain discrete values. By choosing $S_0 = 4, 5, \ldots, 16$, with $S_{total} = 20$ the manipulated variable $u = \frac{S_0}{S_1}$ takes value at $\frac{4}{16}, \frac{5}{15}, \ldots, \frac{15}{5}, \frac{16}{4}$. When these data points are plotted, one of the observations is that the operating points that the controller can take are unequally spaced. If we take the nominal operating point as when both tenants get equal number of process instances, $\frac{10}{10}$, the spacing increases towards the right side and spacing decreases towards the left side of the nominal point. Such, unequally spaced operating points exhibit the characteristics of *static input nonlinearities*, which could adversely affect the management of a linear controller.

Output nonlinearity: The controlled variable y ($\frac{R_1}{R_0}$) of the system exhibits the similar behavior to the u because of the division operation, however it cannot be predetermined. This is because R_0 and R_1 can take a large range of values, causing the $\frac{R_1}{R_0}$ ratio to have a large set of values. For example, when R_1 increases the $\frac{R_1}{R_0}$ increases at a high rate. In contrast, when R_0 increases the $\frac{R_1}{R_0}$ decays at a high rate. The divider operator in the output creates an asymmetric behavior leading to this nonlinearity. Such output nonlinear behavior may also cause performance degradation in a linear controller.

From this analysis it is evident that the above nonlinearities have to be taken in to account in the design of relative performance management system.

5 Approach

This section presents the methodology to design the management system that takes into account the requirements formulated in Sections 4.

Firstly, a set of major modifications was done to WSO2 BPS, in order to enable the performance and resource management capabilities. In particular, tenant-aware queues, response time monitors/sensors for each tenant and resource partition scheduler were integrated. Figure 1a) shows the architecture of the BPS after these modifications. The *Tomcat transport layer* receives the requests from the users of tenants, and forwards the requests to *Axis2 message handler chain*. Upon processing the request in the handler chain, an Axis2 message context is created, and the information about the tenant (so-called tenant domain) in the request is used to identify the corresponding BPEL process. When *ODE-Axis2 Integration Layer* receives an Axis2 message context, the message context is classified based on the available tenant information and put to the message queue corresponding to that tenant. The thread that processed the request waits until a notification of the result is available, in order to send back the response to the client. The management system informs the *Scheduler* via the actuator about the process instance caps for each tenant. The scheduler is implemented based on the algorithm specified in [9]. In addition, the average response time of requests is calculated in a 2 seconds sample window by the sensor for each tenant (R_i, $i = 0, 1, \ldots n$) and sent to the management system.

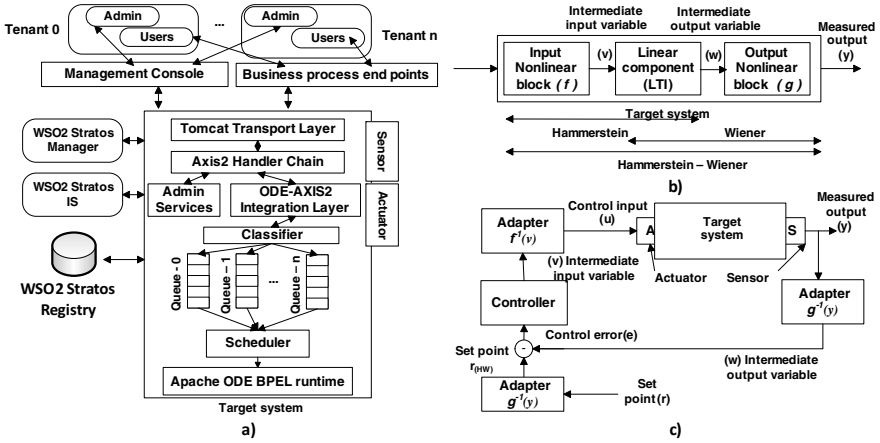


Fig. 1. Block diagram of a) WSO2 BPS, b) Hammerstein-Wiener model and c) Proposed control system

According to the analysis in Section 4.3, the control system designer must consider the aforementioned dominant input and output nonlinear dynamics of the system. A possible way to compensate these nonlinearities is by estimating the existing nonlinearities and then design compensators. Hammerstein-Wiener block structure model is well known in control literature to characterize input and output nonlinearities [16]. As shown in Figure 1b), the Hammerstein-Wiener model has a linear block surrounded by two nonlinear blocks. The entire model can be divided in to two segments called Hammerstein and Wiener block structures. The Hammerstein model has a nonlinear component preceding a linear component. In contrast, the Wiener model has a linear component followed by a nonlinear component. The nonlinear blocks capture the static nonlinearities in the system, while the linear block captures the rest of the dynamics of the system. u and y denote the input and output of the Hammerstein-Wiener model respectively. The intermediate variables v and w are, in general, not measurable.

In this work we use Hammerstein-Wiener model to formally estimate the nonlinear dynamics of the system, followed by their inverse functions to design compensators, which will effectively remove or reduce the impact of nonlinearities on the control system. The novelty of the proposed management system design compared to the linear control system architecture is the integration of the pre-input and post-output compensator as shown in Figure 1c). In this work, the approach proposed in [16] is utilized to design a Hammerstein-Wiener control system. The following subsections present the design details of the control system briefly together with the new overload management method.

5.1 Input Nonlinear Block Design

From the example analyzed in Section 4.3, the discontinuous operating points (u) may induce significant static input nonlinearity. If we implement equally spaced

operating points, a linear controller may provide better performance in the entire operating region. Such a conversion mechanism can be implemented by using the input nonlinear component of Hammerstein-Wiener model, transforming the input (u) into the intermediate input (v) with equal spaces.

Firstly, in order to compute the possible operating points (range of u) for the controller in the original system, we can utilize equation (1) and calculate a point for each $s_0 \in \{s_{0,min}, s_{total} - \sum_{j=0}^{n-1}(s_{j,min})\}$. Here, we have computed the operating points for a controller of the first pair of tenants, assuming that the rest of the tenants are guaranteed the required minimum allocation $S_{i,min}$. Let us represent these $p > 1$ number of operating points as $u = \{u_1, u_2, \ldots u_p\}$. Then, select an arbitrary $v_{min} \leq v \leq v_{max}$ for the intermediate input variable v[5]. With δv defined as $\delta v = \frac{v_{max} - v_{min}}{p-1}$, the intermediate input variable takes its own operating points as $v_1 = v_{min}, v_2 = v_1 + \delta v, \ldots, v_{l+1} = v_l + \delta v, \ldots, v_p = v_{max}$. Thirdly, map the individual values u_l and v_l in the u and v sets to create data pairs, where $l = 1, 2 \ldots, p$. These data pairs can be used to approximate a polynomial $v = f^{-1}(u)$ of order m using curve fitting. After estimation of the f^{-1} function, it is used to implement the pre-input compensator and then integrated to the BPS as shown in Figure 1c). The same compensation is done for each controller, managing the class pair $i-1$ and i according to the relative management control scheme (see, [9]).

5.2 Output Nonlinear Block Design

The output nonlinearity described in Section 4.3 cannot be estimated using the similar methodology in Section 5.1 because the output nonlinearity cannot be analytically defined. In [16], a method is proposed to estimate the output nonlinearity of the relative management scheme based on the Wiener model. We can use the same estimation approach because after the integration of the pre-input compensator the system can be treated as a Wiener model.

Firstly, the possible values of v are applied as a sinusoidal input signal to gather output data, after applying suitable workloads for the first pair of tenants for a sufficient amount of time. The gathered data is then input to the *nlhw* command in the *Matlab: system identification toolbox*, which provides algorithms to estimate the Wiener model. Afterwards, following the procedure in [16] data for w is computed and then $(w-y)$ data pairs are used to estimate the coefficients in the inverse output nonlinearity with a suitable function using the least squares approach. In this work we use a *log* function. Upon the estimation of the inverse output nonlinearity, the $w = g^{-1}(y)$ function is implemented as a component and integrated to each control system managing class pair $i-1$ and i as shown in Figure 1c).

[5] The simulation results in [17] indicate the values selected for v_{min} and v_{max} has less/no effect on the performance of the control system

5.3 Linear Model Design

The next step is to derive the linear component of the Hammerstein-Wiener model to capture the rest of the dynamics in the system. For this purpose a linear autoregressive exogenous input (ARX) model can be estimated similar to the existing work [9,13] by conducting a system identification [5] experiment. However, the transformed variables v and w have to be used in this experiment because of the integration of the compensators into the system.

5.4 Controller Design

By integrating the nonlinear compensators, the system can be assumed to be linear. Consequently, even though the nonlinearities are explicitly considered we can still use a linear controller and the well-established formal design methodologies, which is an added advantage of this proposed nonlinear control approach. We implement a Propositional Integral (PI) controller, which is one of the widely adopted controllers due to their robustness, disturbance rejection capabilities and simplicity [15]. The control equation of the PI controller is shown in equation (2) for the case of Hammerstein-Wiener model. The controller calculates v(k) for $k \geq 0$, given $v(0)$, which is then converted to $u(k)$ using $f^{-1}(v(k))$. $e(k)$ represents the *control error*, computed by $g^{-1}(y) - g^{-1}(r)$, where r is the set point of the original system. K_p (propositional gain) and K_i (integral gain) are called gains of the PI controller. Upon calculation u(k),the algorithm presented in [9] can be used to compute the individual process instance caps $(S_i(k), i = 0, 1 \ldots, n-1)$ implemented in the BPS.

$$v(k) = v(k-1) + (K_p + K_i)e(k) - K_p e(k-1) \tag{2}$$

5.5 Overload Detection and Adaptation

A typical behavior observed during a persistent overload caused by a single tenant is the unbounded growth of the request queue of that tenant [1]. As a consequence, the average response time of that tenant will increase substantially leading to system instabilities or failures. In such situations, if a single queue was implemented without the tenant-aware queuing, the requests of less aggressive tenants will also be rejected while significantly degrading their performance. Therefore, tenant-aware queuing and admission control mechanisms have to be implemented to reject a portion of the workloads in order to limit the response time and to maintain the performance isolation. Furthermore, because of the unpredictable overloads of different tenants, the management system should self-adapt at runtime after detecting the overloads.

An approach to detect server overloads by setting threshold on the queue length was proposed in [1] for a single queue based web server. In this work, we use a similar approach, however for multiple queues. We set queue limits $(q_{len,i}, i\epsilon\ 0, 1, \ldots n-1)$ for each tenant as a configuration parameter, where the incoming requests will be rejected with a SOAP fault message when the queue limit is reached. With the effective resource management the portion of

workload rejected can be reduced avoiding the overloads of one tenant affecting the others. To detect the overload, we set a threshold on the queue $q_{Thresh,i} (\leq q_{len,i}, i\epsilon\ 0, 1, \ldots n-1)$ for each tenant. Then, the queue length $(q_i(k))$ signal of each tenant is evaluated against the $q_{Thresh,i}$. If $q_i(k)$ stays above the threshold for T_{window} number of consecutive time samples, we say that the tenant i has overloaded the system. Similarly, to detect the end of an overload, $q_i(k)$ must stay below the threshold for T_{window} number of consecutive samples.

When an overload is detected, the system self-adapts by triggering a change to the specified priority levels of the tenants in the control system. When there is no overload by any tenant we assume that all the tenants placed in the BPS have equal priorities. Then, when an overload is detected, say, by tenant i, less priority is given to the requests of tenant i compared to other tenants. This can be implemented using the relative performance management scheme simply by adjusting the differentiation factor P_i dynamically (see Section 2). Afterwards, these differentiation settings can be implemented as simple rules in the management system. As a result, under overload situations our approach maintains the response times of each tenant according to the priority levels specified in the rules, with no human interventions.

6 Experimentation

This section provides the details of the experiments. A BPS with two tenants $(n = 2)$ is considered. The BPS and database was deployed in a VM with two 2.67 GHz CPUs and 3 GB memory. We used the LoanProcess[6] as the deployed business process for each tenant, which invokes three partner services sequentially. The workload generators and partner web services were deployed in two VMs each with a 2.67 GHz CPU and 2 GB memory. After initial profiling the maximum concurrent process instances S_{total} was set to 20. Although higher S_{total} increases the throughput, the response time was significantly affected as well (e.g., $S_{total} = 30$ increased response time around 100 ms). In addition, $S_{1,min}, S_{2,min} = 4$. Furthermore, BPS can handle 75 to 80 requests/sec workload without any overload.

6.1 System Modeling and Controller Design

This section gives the design details of the control system. Firstly, to design the pre-input compensator the possible operating points for u were calculated as $\frac{4}{16}, \ldots, 1, \ldots, \frac{16}{14}$. Then, the points of the intermediate variable v were selected as values $-6, -5, \ldots -1, 0, 1 \ldots 5, 6$ by setting $\delta v = 1$, $v_{min} = -6$ and $v_{max} = 6$. Following the design process in Section 5.1, a fourth order polynomial was used to represent the inverse input nonlinear function (see equation (3)).

$$u = f^{-1}(v) = 0.0003828 * v^4 + 0.003445 * v^3 + 0.01722 * v^2 + 0.1857 * v + 1.006 \quad (3)$$

[6] It is sample BPEL process available at `https://svn.wso2.org/repos/wso2/branches/carbon/3.2.0/products/bps/2.1.2/modules/samples/product/src/main/resources/bpel/2.0/LoanProcess/`

Similarly, following the design process in Section 5.2, a sinusoidal signal was designed with the possible values of v and then 40 requests/sec workloads were applied for each tenant to gather output data for 500 sample periods. Subsequently, the output inverse nonlinear function was represented by the equation (4). For the linear model estimation an experiment with a pseudo random input signal and 35 requests/sec workload for each tenant were used. The estimated ARX model is given in equation (5).

The final step is to implement the *Hammerstein-Wiener control system* (called as HWCS) using the ARX model and pole-placement design method [5]. The finalized parameters after placing poles at 0.7 are $K_p = 0.47$, $K_i = 0.16$ and $v(0) = 0$. In order to compare the management provided by the HWCS we also implemented a *linear control system* (called as LCS), with the same setting used in HWCS implementation. The parameters of LCS are $K_p = 0.64$, $K_i = 0.25$ and $u(0) = 1$.

$$w = g^{-1}(y) = 7.48log(y) - 0.08 \tag{4}$$

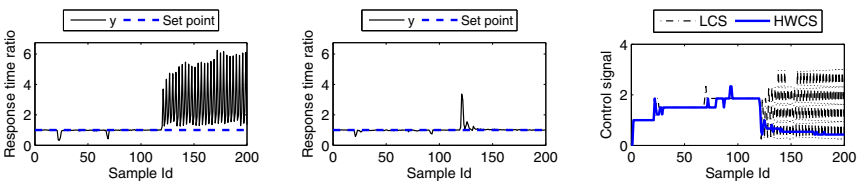
$$w(t+1) = \quad 0.79w(t) + 0.58v(t) \tag{5}$$

6.2 Experiment Results

This section compares the management capabilities of LCS and HWCS. Due to the variability of the operating conditions, each experiment was conducted 10 times and the average statistics of Sum of Square Error (SSE) are compared in Table 1.

6.2.1 High Workload Separately

This experiment compares the performance of LCS and HWCS when the total workload from two tenants is under the system capacity, however each tenant increases its workload to a high level requiring more resources than the other at separate time periods. Till the 20th sample workloads of 25 requests/sec was applied for tenant_0 and tenant_1. Then, at the 20th sample tenant_0 workload increases to 60 requests/sec. This could be a scenario where a high resource demand for tenant_0, while tenant_1 is at a lower workload rate. Afterwards, at the 90th sample tenant_0 workload reduces to 25 requests/sec. Then, at the 120th sample, tenant_1 workload increases to 60 requests/sec from 25 requests/sec. The set point ($\frac{P_1}{P_0}$) is fixed at 1, where both classes are treated equally. The output and control signals of the LCS and HWCS are shown in Figure 2.



(a) LCS output (y) (b) HWCS output (y) (c) Control signals (u)

Fig. 2. Results of the LCS and HWCS under high workloads separately

First, let us analyze the performance of the control systems in region where tenant_0 gets more resources (between 20th and 90th samples). The settling times and overshooting of the LCS is higher than HWCS. This is because of the output nonlinearity, the variations in the output signal are damped out so that the linear controller takes time to reject the workload disturbance and adjust the resource caps. In contrast, the output nonlinearity compensated HWCS rejects the disturbance much efficiently compared to LCS. Then, after settling down, both control systems provide similar steady state behavior, achieving the set point with small errors. At the 120th sample when tenant_1 increases its workload demanding more resources. LCS shows a highly oscillatory/unstable behavior with large steady state errors (see Figure 2(a)). The control signal (resource allocation decisions) of LCS illustrated in Figure 2(c), shows a highly oscillatory behavior, which led to this unstable behavior at the output. The reason for this behavior is the issue of input nonlinearity discussed in Section 4.3. The smaller gaps between the operating points when tenant_1 requires more resources affect the LCS under noisy workload conditions making the LCS to jump between several resource allocation points without settling down. This is an indication that the LCS cannot provide effective performance and resource management when the workload of tenant_1 is high. However, the performance in this region can be improved by reducing the aggressiveness (gains) of the controller. This adversely affects the performance when the workload of tenant_0 is high. Consequently, LCS fails to achieve effective performance management in the entire operating region under aforementioned nonlinearities. In contrast, the pre-input compensator of the HWCS reduces the impact of input nonlinearity by maintaining the control signal at a steady state (see Figure 2(c)), providing highly satisfactory steady state performance after the disturbance at the 120th sample without affecting the stability. The statistics in Table 1 show significant improvements in the case of HWCS compared to LCS.

6.2.2 Different Priority Levels between Tenants

This section compares the performance differentiation capabilities of the control systems when the set point is $\frac{P_1}{P_0} = 1.5$, making tenant_0 more important than tenant_1. For this case 25 and 55 requests/sec were applied for tenant_0 and tenant_1 respectively. Table 1 shows the results of two control systems.

The performance of LCS is similar to what was observed in the previous section. In particular, due to high workload of tenant_1, LCS has to operate in the region where the input nonlinearity is severe. Consequently, LCS produces highly oscillatory outputs and unstable behavior in the system. In contrast, the nonlinearity compensated HWCS provides significantly better steady state behavior compared to LCS. The statistics from Table 1 shows a significant reduction in SSE statistics for the case of HWCS.

6.2.3 Overload Detection and Adaptation

In this case we evaluate the overload detection and adaptation capabilities of the proposed control approach. The parameters related to overload detection, i.e. $q_{len,i}$, $q_{Thresh,i}$ and T_{window} were set to 30, 20 and 4 respectively. As the

Table 1. The SSE statistics of LCS and HWCS

Section	LCS	HWCS
6.2.1	602.43	12.41
6.2.1	994.71	16.71

Table 2. The total number of requests rejected from each tenant

Tenant	FCS	LCS	HWCS
0	4262	2296	1968
1	4012	5095	1859

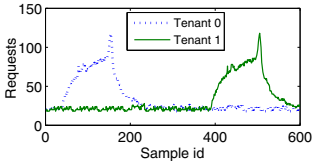
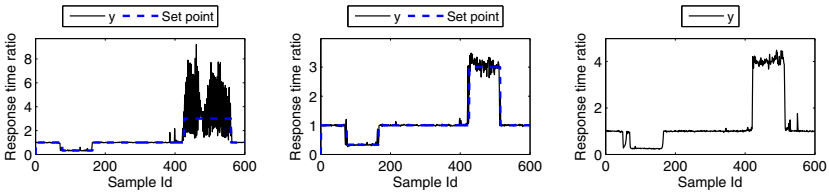


Fig. 3. Workload settings extracted from 1998 world cup website workload traces



(a) LCS output (b) HWCS output (c) FCS output

Fig. 4. Results of the control systems under persistent overloading by tenants

adaptive rules, we set $P_0 : P_1 = 1{:}3$ when tenant_1 overloads the system, while $P_0 : P_1 = 3{:}1$ when tenant_0 overloads the system. If both tenants are running below the system capacity or overload the system at the same time $P_0 : P_1$ set to $1 : 1$. In this experiment the workload settings extracted from 1998 football world cup workload traces[7] were used. However, the workload rates have to be scaled to fit the requirements of this experiment. Figure 3 shows the workload rates applied on the BPS for two tenants, which overloads the system in separate time periods. In addition to LCS and HWCS, we also provide the results of a fixed partition controller (FCS) that sets $S_0(k) : S_1(k) = 10{:}10$. Figure 4 shows the results of these control systems.

In order to analyze the overload detection capabilities let us investigate the behavior of the set point signal implemented by the management systems (see Figures 4(a), 4(b)). Initially, the set point is at 1 indicating both workloads are running below the system capacity, therefore both tenants are equally treated. However, when tenant_0 overloads the system, the proposed control solution has detected the overload around the 60th sample and dynamically changed the set point to be $\frac{1}{3}$. Consequently, the overload of the aggressive tenant_0 has not degraded the performance of tenant_1 during the overload, maintaining its response time approximately 3 times less than tenant_0's. Similarly, the control system has detected the overload of tenant_1 when it overloads the system, and has changed the set point to 3, giving high priority to tenant_0. Therefore, it is evident that the proposed overload detection and adaptation mechanism can implement effective performance management and isolation under heavy overload and varying workload conditions.

[7] `http://ita.ee.lbl.gov/html/contrib/WorldCup.html`

When LCS is compared with HWCS, it is clear that the nonlinearities have affected the performance of LCS when tenant_1 overloads the system and demands more resources after the 420th sample. Thereby leading to oscillations at the output and affecting the performance of the less aggressive tenant at that time period. The total requests rejected during the experiment listed in Table 2 indicates significantly larger workload rejections for tenant_1 and moderate workload rejections for tenant_0, in the case of LCS compared to HWCS. When FCS is compared to HWCS, it is evident that the response time of the overloaded tenant has significantly degraded compared to HWCS. For instance, the response time ratio is around $\frac{1}{4}$ and 4 when tenant_0 and tenant_1 overload the system respectively. Further, the request rejection statistics in Table 2 indicate significantly larger rejections compared to HWCS as well. The reason for this observation is that the resources are not shared or utilized efficiently in FCS, where some resources reserved for one tenant are wasted, while the other requires more resources than are allocated. Therefore, the proposed overload detection and self-adaptive priority adjustment mechanism coupled with HWCS achieves the performance and resource management objectives of the multi-tenanted BPS significantly better compared to the existing fixed partition and linear control methods.

7 Conclusions and Discussion

In this work we have implemented a BPS that can manage performance and resource under unpredictable workload conditions of different tenants. The priority levels of the performance variables are enforced by using the relative performance management scheme, while resource management was done using a hybrid of resource reservation and flexible resource partitioning scheme. In order to automate the management a nonlinear control technique was presented based on the Hammerstein-Wiener model. From the experiment results in Section 6.2, it is evident that the input and output nonlinearities in the relative management scheme significantly affects the performance of a linear control mechanism, consequently leading to instabilities in the system. In contrast, Hammerstein-Wiener model based nonlinear control system, which compensates the nonlinearities, improves the management capabilities compared to linear control. In addition, the proposed overload detection and self-adaptive mechanism shows accurate detection and stable adaptation under unpredictable overloads of different tenants.

Threats to Validity: Although the approach presented in Section 5 is generalized for system with n tenants, the experiment presented in this paper limits the number of tenants to 2. This is because only two tenants could be placed to effectively share the resources depending on the resource availability of the VM and number of the process threads that can run concurrently in the BPS without affecting the response time (see [14] for profiling results). In addition, the number of process instances used was limited to 20 to maintain the performance isolation with the increase of concurrent process worker threads. Further evaluation results are presented for the cases of more than two tenants and large number of resources based on a shared application multi-tenanted system in the technical report [17]. Furthermore, the proposed approach is validated using a

business process executes in short time periods. However, some real world scenarios may have long running business processes (may be days) which may affect the experiment results.

References

1. Abdelzaher, T.F., Bhatti, N.: Web content adaptation to improve server overload behavior. Comput. Netw. 31(11-16), 1563–1577 (1999)
2. Chong, F., Carraro, G.: Architecture strategies for catching the long tail. MSDN (2006)
3. Fehling, C., Leymann, F., Mietzner, R.: A framework for optimized distribution of tenants in cloud applications. In: International Conference on Cloud Computing (CLOUD), pp. 252–259 (2010)
4. Guo, C.J., Sun, W., Huang, Y., Wang, Z.H., Gao, B.: A framework for native multi-tenancy application development and management. In: Conference on Enterprise Computing, E-Commerce, and E-Services, pp. 551–558 (2007)
5. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. John Wiley & Sons (2004)
6. Kwok, T., Mohindra, A.: Resource Calculations with Constraints, and Placement of Tenants and Instances for Multi-tenant SaaS Applications. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 633–648. Springer, Heidelberg (2008)
7. Li, X.H., Liu, T.C., Li, Y., Chen, Y.: SPIN: Service Performance Isolation Infrastructure in Multi-tenancy Environment. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 649–663. Springer, Heidelberg (2008)
8. Lin, H., Sun, K., Zhao, S., Han, Y.: Feedback-control-based performance regulation for multi-tenant applications. In: International Conference on Parallel and Distributed Systems, pp. 134–141 (2009)
9. Lu, C., Lu, Y., Abdelzaher, T.F., Stankovic, J.A., Son, S.H.: Feedback control architecture and design methodology for service delay guarantees in web servers. IEEE Trans. Parallel Distrib. Syst. 17, 1014–1027 (2006)
10. Lu, Y., Abdelzaher, T., Lu, C., Sha, L., Liu, X.: Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers. In: IEEE Real-Time and Embedded Technology and Applications Symposium, p. 208 (2003)
11. Mietzner, R., Unger, T., Titze, R., Leymann, F.: Combining different multi-tenancy patterns in service-oriented applications. In: Enterprise Distributed Object Computing Conference, pp. 131–140 (2009)
12. Momm, C., Krebs, R.: A qualitative discussion of different approaches for implementing multi-tenant saas offerings. In: Software Engineering (Workshops), pp. 139–150 (2011)
13. Pan, W., Mu, D., Wu, H., Yao, L.: Feedback control-based qos guarantees in web application servers. In: IEEE International Conference on High Performance Computing and Communications, pp. 328–334 (2008)
14. Pathirage, M., Perera, S., Kumara, I., Weerawarana, S.: A multi-tenant architecture for business process executions. In: IEEE International Conference on Web Services, pp. 121–128 (2011)
15. Patikirikorala, T., Colman, A., Han, J., Wang, L.: A systematic survey on the design of self-adaptive software systems using control engineering approaches. In: Symposium on Software Engineering for Adaptive and Self-Managing Systems (2012)

16. Patikirikorala, T., Wang, L., Colman, A., Han, J.: Hammerstein-wiener nonlinear model based predictive control for relative qos performance and resource management of software systems. Control Engineering Practice 20(1), 49–61 (2011)
17. Patikirikorala, T., Wang, L., Colman, A., Han, J.: A nonlinear feedback control approach for differentiated performance management in autonomic systems. Technical report (2011)
18. Wang, Z.H., Guo, C.J., Gao, B., Sun, W., Zhang, Z., An, W.H.: A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing. In: Conference on e-Business Engineering, pp. 94–101 (2008)
19. Weissman, C.D., Bobrowski, S.: The design of the force.com multitenant internet application development platform. In: International Conference on Management of Data, pp. 889–896 (2009)
20. Zhou, X., Wei, J., Xu, C.-Z.: Quality-of-service differentiation on the internet: A taxonomy. Journal of Network and Computer Applications 30(1), 354–383 (2007)