A Study on Managing the Performance and Resources of a Business Process Engine using Nonlinear and Switching Control Systems

Tharindu Patikirikorala 1, Alan Colman1, Jun Han1, Liuping Wang2

1Swinburne University of Technology, Victoria, Australia
2RMIT University, Melbourne, Australia

1. Introduction

This technical report presents the experiment results of the a case study conducted on a business process server applying the nonlinear and switching control system design approaches given in [1, 2]. Here, we introduce the business scenarios and objectives of the businesses process server before we go on to the details of the system architectures and experimental results.

WSO2 BPS WSO21 is one of the leading open-source enterprise software platform providers, giving services to companies such as ebay2. From their product suite, WSO2 Stratos business process server (BPS) 3 is a multi-tenanted Platform as a Service (PaaS) middleware environment, which provides out of the box support to deploy software workflows for multiple clients (or tenants) using a single instance. WSO2 BPS already provides data, security and execution isolation, however performance isolation is not looked at so far [3]. In this work we combine and extend the WSO2 BPS to manage performance of multiple customers while sharing the resources effectively. The details are provided in Section 2

2. Management of WSO2 BPS

WSO2 Stratos Business Process Server4 is a multi-tenanted workflow engine which executes business processes compliant with WS-BPEL standard, and is built on top of WSO2 Carbon platform5. WSO2 Stratos BPS also supports data and execution isolation for multiple tenants [3]. Figure 1 shows its high level architecture. Tenant administrators and authorized users can manage and monitor the business process deployments and business process instances via the graphical administrative console. 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 Manger is used to provision, manage tenants including tenant subscriptions and billing.

1http://wso2.com/
2http://www.ebay.com/
3http://wso2.com/cloud/stratos/
4http://wso2.com/products/business-process-server/
5http://wso2.com/products/carbon/
Business process artefacts 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-tenant data architecture pattern defined in [4]. WSO2 Stratos BPS uses Apache ODE 6 as its BPEL execution run-time. ODE-Axis2 Integration Layer provides three main services: i) BPEL process and process instance management, ii) tenant-aware request dispatching, and iii) communication with partner services defined in a BPEL process. Integration Layer is also responsible to expose a BPEL process as an Axis2 7 Web Service. In the current multi-tenant BPS instance, a single ODE process engine is shared by multiple tenants. Therefore, a workload of a tenant may adversely affect the performance of other tenants. Consequently, a mechanism is required to manage the performance, resources and workloads of different tenants in a single BPS instance, which is the focus of this study.

Figure 1: Block diagram of WSO2 BPS

2.1. Implementation

In order to enable the performance and resource management in the BPS many modifications had to be done. In particular, tenant-aware queues, response time monitors/sensors for each tenant and resource partition scheduler were integrated. Figure 1 shows the architecture of the BPS after these modifications, which corresponds to the standard multi-class system architecture. 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) from 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

6http://ode.apache.org/
7http://axis.apache.org/axis2/java/core/
tenant information and puts it into the message queue corresponding to the 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 shares for each tenant. Here, the scheduler takes in to account the process instance shares \((S_i, i = 0, 1, \ldots n - 1)\) and current usage to schedule the requests from each tenant’s queue to be sent to ODE runtime. In addition, the average response time of requests in a 2 seconds sample window is calculated by the sensor for each tenant \((R_i, i = 0, 1, \ldots n - 1)\) and sent to the management system. The response time of a request includes the waiting time in the tenant’s queue and execution time in the ODE runtime.

2.2. Experiment setup

Here we consider a BPS with two tenants \((n = 2)\). The BPS and database was deployed in a virtual machine (VM) with two 2.67 GHz CPUs and 3 GB memory. We used the LoanProcess\(^8\) 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 \(S_{\text{total}}\) was set to 20. Although higher \(S_{\text{total}}\) increases the throughput, the response time was significantly affected as well (e.g., \(S_{\text{total}} = 30\) increased response time around 100 ms). In addition, \(S_{1,\text{min}}, S_{2,\text{min}} = 4\).

2.3. Hammerstein-Wiener control system design

This section gives the design details of the control system, including the two compensators and controller.

Firstly, to design the pre-input compensator the possible operating points for \(u\) were calculated as \(\frac{4}{10}, \ldots, 1, \ldots, \frac{16}{2}\). Then, the points of the intermediate variable \(v\) were selected as values \(-6, -5, \ldots -1, 0, 1 \ldots 5, 6\) by setting \(\delta v = 1, v_{\text{min}} = -6\) and \(v_{\text{max}} = 6\). Following the design process in [2], a fourth order polynomial was used in the estimation the inverse input nonlinear function (see equation (1)) with a goodness of fit of 0.9998. Figure 2a shows the model fit. This function was then implemented as a software component/compensator and integrated to the control system of the BPS.

\[
u = f^{-1}(v) = 0.0003828 \times v^4 + 0.003445 \times v^3 + 0.01722 \times v^2
+ 0.1857 \times v + 1.006
\] (1)

With the integration of the input static nonlinear compensator, the next step is to design the output nonlinear compensator. Following the design process in [2], a sinusoidal signal was designed with possible values of \(v\) and 40 requests/sec workloads were applied for each tenant to gather output data for 500 sample periods. Data samples between the 1st to 350th samples were included in the estimation set and the rest were used as the test set. 1st order ARX model and 4th order polynomial was sufficient to represent the system as a Wiener model with a \(R^2\) fit of 0.86. Then, after computing the \(w\) data, the output inverse nonlinear function was represented by the equation (2) close to 0.97 fit (see Figure 2c for the model fit).

\[
w = g^{-1}(y) = 7.48\log(y) - 0.08
\] (2)

\(^8\)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/
For the second SID experiment, a pseudo random input signal and 35 requests/sec workload for each tenant were used. The estimated linear model is given in equation (3) with 0.81 model fit.

\[ w(t + 1) = 0.79w(t) + 0.58v(t). \]  

(3)

The final step is to implement the Hammerstein-Wiener control system (called as HWCS) using the linear model and pole placement design method. The finalized parameters after placing poles at 0.7 are \( K_p = 0.47 \), \( K_i = 0.16 \) and \( v(0) = 0 \).

2.4. MMST-T2 control system design

In this section we present the design details of MMST-T2 control system design based on the approach presented in [1]. The idea is to design two models and controllers to represent the region 0 and 1 and then implement the switching scheme of MMST-T2 control system. Firstly, to capture the behaviour of the system when tenant_0 gets more resources the operating points of \( \frac{10}{16}, \frac{11}{16}, \ldots, \frac{15}{16} \) were selected to design a pseudo random signal. A high workload for tenant_0 was applied keeping the workload of tenant_1 at a low rate. Gathered data samples from this experiment was used to estimate the model for region 0 with \( R^2 \) fit of 93% (see equation (4)). A similar experiment was conducted with \( \frac{3}{16}, \frac{5}{16}, \ldots, \frac{15}{16} \) to estimate the model for region 1 as shown in equation (5). Afterwards, two controllers were designed to provide control in each region. An aggressive controller with \( K_p = 1, K_i = 0.44 \) and \( u(0) = 1 \) and less aggressive controller with \( K_p = 0.22, K_i = 0.11 \) and \( u(0) = 1 \) were used for region 0 and 1 respectively. In addition, the configuration parameters of MMST-T2 were set as \( \alpha = 0, \beta = 1, \) and \( T_{\text{min}} = 3 \).

\[ y(t + 1) = 0.84y(t) + 0.05u(t). \]  

(4)

\[ y(t + 1) = 0.71y(t) + 0.77u(t). \]  

(5)

2.5. Experiment Results

This section compares the performance and resource management capabilities of the control system designed in Section 2.3 under different workloads and settings.

In order to compare the management provided by the HWCS we also implemented a linear control system (called as LCS), using the linear model in equation (6). The same data used in the second SID experiment was used to construct this model with a fit of 0.67. Furthermore, similar to HWCS implementation poles were placed at 0.7 and a controller was designed with \( K_p \) and \( K_i \) 0.64 and 0.25 respectively and \( u(0) = 1 \).

\[ y(t + 1) = 0.72y(t) + 0.36u(t). \]  

(6)
2.5.1. 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 \( P_1 \) is fixed at 1, where both classes are treated equally. Further, the workload settings are such that both tenants require more than \( S_{\text{min}} \); process instances to cater the demand. Otherwise the performance isolation is automatically implemented due to the hybrid resource management. Under these conditions, the expected behavior is to adjust the process instance partitions so that the tenant with high workload rate gets more resources. The output and control signals of the LCS and HWCS are shown in Figure 3.

In this case study also we see the performance issues observed in [2, 1] for the case of LCS. The settling time and overshooting due to the disturbance at the 20th sample is significantly high compared to HWCS and MMST control systems. This is because of the output nonlinearity. Then, when LCS operates in the region where the input nonlinearity is severe, oscillatory behavior is observed at the output after the disturbance of tenant 1 at the 120th sample.

In contrast, HWCS and MMST provide much better control in this experiment setting with no instabilities. In the case of HWCS the compensators have reduced the impact of the non-
linearities providing better performance than LCS. In addition, MMST has effectively selected the appropriate controller for the particular region avoiding the instabilities observed in a single model and controller based LCS. The switching behaviour is shown in Figure 3g.

Table 1 summarizes the statistics for all the control systems. It is evident that HWCS has provided better performance when all the statistics are compared to the other two control system, while MMST has outperformed LCS significantly. However, when MMST is compared with HWCS, although chattering is not observed combined performance of two controllers in MMST has shown temporal instabilities, when the operating regions are changes abruptly.

2.5.2. Different priority levels between tenants

In this case we assess the performance of the above designed controllers in the case of different differentiation factors which needs effective performance differentiation when the system is running under the full capacity. The performance of the control systems is compared in this section when the set point is $p_1^0 = 1.5$. For this case 25 and 55 requests/sec were applied for tenant0 and tenant1 respectively. Table 1 shows the results of the control systems.

The performance of LCS is similar to what was observed in the Section 2.5.1. In particular, due to high workload of tenant1, LCS has to operate in the region where input nonlinearity is severe. Consequently, LCS produces highly oscillatory outputs and unstable behavior in the system showing larger SSE among the control systems. In contrast, the nonlinearity compensated HWCS provides significantly better steady state behavior compared to LCS with the lowest SSE. In the case of MMST, the SSE statistics are higher than HWCS, because of the chattering. The model1 and corresponding controller was selected most of the time, however switching to the other model lead to the temporal instabilities. The performance of MMST is therefore, significantly poor for this case compared to HWCS. The performance of MMST is therefore, significantly poor for this case compared to HWCS. This behavior was also observed in the simulation studies as well.

2.5.3. Overloaded condition

In this case we assess the performance of the above designed control systems in the case of persistent overload of a single tenant. The tenant0 sends 25 requests/sec workload, while tenant1 sends 150 requests/sec workloads for. Therefore, tenant1 has overloaded the system. We fixed the set point at $p_1^0 = 3$, providing better performance to tenant0.

The results under this condition is also similar to Section 2.5.1. HWCS outperforms both other control systems providing much stable and satisfactory performance. MMST also provide better performance, however chattering was observed for short periods of time leading to temporal instabilities.

Table 1: The Statistics of LCS, HWCS and MMST

| Section | LCS | | | HWCS | | | | MMST | | |
|---------|-----|---|---|-----|---|---|-----|---|---|
| SSE | MIN | MAX | SSE | MIN | MAX | SSE | MIN | MAX |
| 2.5.1 | 601.8 | 0.45 | 6.48 | 12.41 | 0.52 | 3.42 | 18.19 | 0.46 | 3.57 |
| 2.5.2 | 997.74 | 1.2 | 7.8 | 16.81 | 0.98 | 2.88 | 85.57 | 0.66 | 4.31 |
| 2.5.3 | 561.52 | 1.43 | 8.88 | 17.13 | 1.71 | 4 | 45.22 | 1.41 | 4.62 |

