# QoS-Aware Service Selection

**Abstract**

As the widespread use of the Internet, the number of web services that can provide similar functionality is increasing rapidly in recent years, web service selection has to be based on some non-functional attributes of the services such as the quality of service (QoS). In this chapter, we use a server switching service that is commonly used in Internet hosting environment to explain how an agent can use a performance model to evaluate services and select the most suitable services among a number of functionally similar services returned by the service discovery. The various criteria that can be used to assess QoS are introduced in this chapter, including mean response time, throughput, system utilisation and some others closely related to business such as revenue and operating costs. Service selection in the chosen case study depends on the quality and suitability of various switching policies, in another word, different switching policies can be selected depending on the QoS of the services and the run-time system state. Since the system performance can be evaluated using a analytic model, therefore, the QoS of services is assessed based on the output of the performance model.

## 1 Introduction

There are two key challenges in Semantic Web services. One is service advertisement and discovery, which has been discovered in last chapter. The second key challenge is service selection and composition, which has attracted extensive research in the literature [1, 2, 3, 4, 5, 6, 7, 8].

Web services are usually described by WSDL [9] and published by registering the service using UDDI [10]. Current approaches for service publication and registration rely on static description of web service interfaces. The static description is sufficient for providing some information such as service functionality, service URL and the service namespace. However, some other attributes such as QoS of a service can not be accurately described as it is runtime environment dependent. A web service might work well in one scenario, whereas it might be a bad choice for another scenario. Therefore, it is crucial to select the most suitable service among many functionally similar services.

The goal for service selection is to find the best set of services available at runtime, taking into consideration end-user preferences and the execution context [2]. It is a challenge task as it is very difficult to predict the QoS of a given web service. The challenge arises partly because you may not able to trust the other party who could claim arbitrary QoS properties to attract interested parties, and partly because you lack knowledge of the environment within which it is executing, especially in some runtime context where many factors could affect the performance of the service. Moreover, dynamic evaluation of service is usually required as the run-time system state is changing. In addition, all customer system environments are different, thus it is difficult for the service provider to test the

service for all scenarios. Therefore, it might be a good idea that the agents be able to evaluate the quality of a service in different customised environments using a performance model.

In this chapter, we use a server switching service usually used in Internet hosting centres to explain how an agent can use a performance model to evaluate and select the most suitable services among a number of functionally similar services returned by the service discovery. Service selection in the chosen case study depends on the suitability and quality of various switching policies, that is different switching policies can be selected depending on the QoS of the services and the run-time system state. Since the system performance can be evaluated using a analytic model, therefore, the QoS of services is assessed based on the output of the performance model.

# 2 Service Selection Procedure

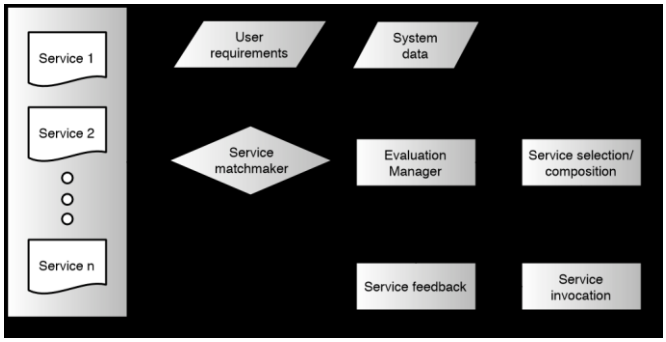


Figure 1: System diagram of service selection.

Figure 1 is an illustration of the service selection procedure. First, when a client sends a service request, the agent searches for services that can provide required capabilities in the registry and uses the matchmaker to match the user requirements (in terms of the functionality required) with all available services. The output from the service matchmaker is a number of functionally similar services. The agent needs to choose the most suitable service among those services based on some non-functional attributes such as the QoS of the services. As introduced earlier, it is very difficult to present QoS using static description in WSDL. Therefore, the performance evaluation manager can play a important role in the service selection process. Evaluation can be made throughput analytical model, simulation or the hybrid approach. The evaluation manager takes the system data such as system architecture configuration information, runtime workload demand and feeds the data into the performance model for evaluation. The main benefit of the use of performance model is that performance can be quickly evaluated without actual invocation of the services. Performance metrics of each model depends on the design of the model and the common metrics include mean response time, throughput and system utilisation. Some other performance metrics related closely to business include operational costs, system revenue.

Based on the performance evaluation results, the agent can choose the most suitable service and composite it when it is needed. This chapter focuses on service selection, and service composition will be introduced in the next chapter. When a service is selected and properly composited, it then can be called by the client. After service invocation, the user can give feedback of the service via a feedback (or recommendation) system. The *feedback* component in the framework is used for the purpose – to adjust the performance model and to dynamically adapt to user requirements.

In the next section, we use the server switching service as an example to explain how performance evaluation can be done and how the results can be used to assist web service selection.

# 3 Case Study – Selection of Switching Service

## 3.1 Server Switching in Internet Hosting Centres

Internet services are normally hosted in a commercial hosting environment that are run by Internet Service Providers (ISPs). Workload demand for Internet services is usually very bursty [11][12][13], thus it is difficult to predict the workload level at a certain point in time. Therefore, fixed server configurations for a service are far from satisfactory for an application when the workload level is high; whereas it is potentially a waste of resource while the workload is light for the remaining applications supported by the system. Therefore, it is desirable that server resources in a shared hosting environment can be *switched* between applications to accommodate workload variation.

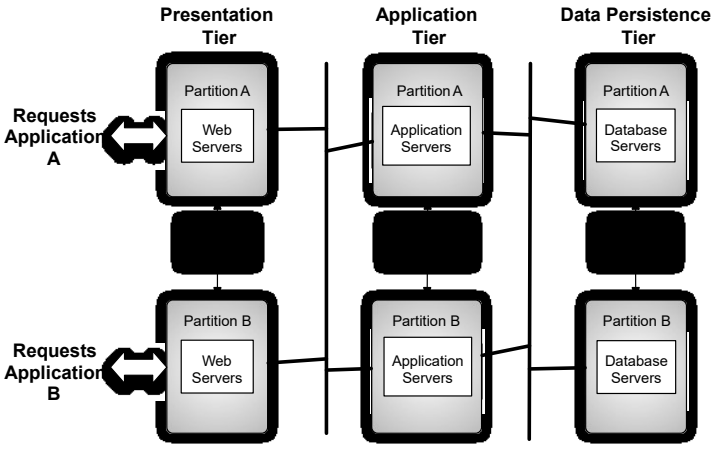


Figure 2: Illustration of server switching in a multi-tier architecture.

A server switching service is a service that can be employed by the ISPs to improve the Internet service and optimise the resource usage in the server centres. To employ a switching service, the ISPs need to assess the quality of the service, in another word, to assess the benefits of using the switching service. The quality of the switching service depends on the benefits it has brought to the ISPs,

thus, to assess the quality of a switching service, one needs to assess the improvement of the Internet services. There are a number of performance metrics to evaluate the quality of a switching service. From a request sender's perspective, mean response time is the main performance metric; from ISPs' perspective, some performance metrics include throughput, system utilisation and total generated revenue due certain period.

Figure 2 is an illustration of how server switching happens in a distributed e-Business environment. The diagram assumes the ISP hosts two different Internet services, both of which require a multi-tier system architecture. The typical system configuration includes the presentation tier, application tier and data persistence tier. In each tier, a cluster of servers is used for processing the requests. In Figure 2, the cluster of servers in each tier is partitioned into two pools, each of which is responsible for each Internet application. When there is a need, some portion of servers at the same tier can be switched between pools to adapt workload fluctuation.

## 3.2 Server Switching Procedure

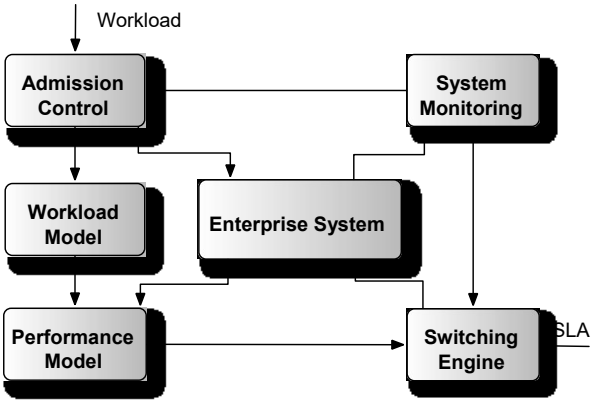


Figure 3: Server switching procedure.

Figure 3 shows how server switching works in a distributed e-Business environment. This diagram is a portion of Figure 1 and it corresponds to the *evaluation manager* component of Figure 1. In this diagram, there are some key components, including *admission control, workload model, performance model, system monitoring and the switching engine*. When requests arrive, they are controlled by the *admission control* component, based on the system information (e.g. system utilisation) from the *monitoring* component. The *workload model* takes as the input the allowed requests and builds a workload model based on the workload characteristics. The *performance model* then takes as input the output of the *workload model* and system architecture configuration and calculates the required performance metrics. These metrics combined with system information from the monitoring facilities are fed into the *switching engine*, which then computes the benefits and penalties of all possible switches before making the

final switching decision. In the following section, we show how to model the multi-tier Internet services using queueing network.

## 3.3 Modelling Multi-tier Internet Services

A multi-tiered Internet service can be modelled using a multi-class closed queueing network [14, 15]. Figure 4 shows a model for a typical configuration of such applications. In the model, C refers to the client; WS, AS and DS refer to the web server, application server and database server respectively. The queueing network is solved using the MVA (Mean Value Analysis) algorithm [16], which is based on Little's law [17] and the Arrival Theorem [16, 18] from standard queueing theory. In this section, we briefly describe how different performance metrics can be derived from the closed queueing network model. Table 1 summarises the notation used throughout this chapter.

Table 1: Notation used in this chapter

| Symbol | Description |
|---|---|
| $S_{ir}$ | Service time of job class-$r$ at station $i$ |
| $v_{ir}$ | Visiting ratio of job class-$r$ at station $i$ |
| $N$ | Number of service stations in QN |
| $K$ | Number of jobs in QN |
| $R$ | Number of job classes in QN |
| $K_{ir}$ | Number of class-$r$ job at station $i$ |
| $m_i$ | Number of servers at station $i$ |
| $\varphi_r$ | Revenue of each class-$r$ job |
| $\pi_i$ | Marginal probability at centre $i$ |
| $T$ | System response time |
| $D_r$ | Deadline for class-$r$ jobs |
| $E_r$ | Exit time for class-$r$ jobs |
| $P_r$ | Probability that class-$r$ job stays |
| $X_r$ | Class-$r$ throughput before switching |
| | Class-$r$ throughput after switching |
| $U_i$ | Utilisation at station $i$ |
| $t_s$ | Server switching time |
| $t_d$ | Switching decision interval time |

Consider a product form closed queueing network with $N$ load-independent service stations. $N = \{1,2,\cdots,N\}$ is the set of station indexes. Suppose there are $K$ customers and they are partitioned into $R$ classes according to their service request patterns; customers grouped in a class are assumed to be statistically identical. $R = \{1,2,\cdots,R\}$ is the set of class indexes. The service time, $S_{ir}$, in a multi-class closed queueing network is the average time spent by a class-$r$ job during a single visit to station[1] $i$. The service demand, denoted as $D_{ir}$, is the total service requirement, which is the average amount of time that a class-$r$ job spends in service at station $i$ during execution. This can be derived from the Service Demand

[1] the terms *station*, *centre* and *node* have the same meaning, and are used interchangeably.

Law [19] as $D_{ir} = S_{ir} \cdot v_{ir}$; here $v_{ir}$ is the visiting ratio of class-$r$ jobs to station $i$. $K_r$ is the total population of customers of class $r$. The total population of the network is thus defined as $K = \sum_r K_r$. The vector

$\tilde{K} = \{K_1, K_2, \cdots, K_R\}$ is used to represent the population of the network.

In modern enterprise systems, clusters of servers are commonly used in each application tier to improve server processing capability. Thus, when modelling

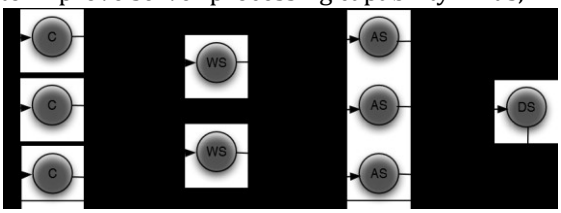


Figure 4: A model of a typical configuration of a cluster-based multi-tiered Internet service. C represents customer machines; WS, AS and DS represent web servers, application servers and database servers, respectively.

those applications, we need to consider both -/M/1-FCFS and -/M/m-FCFS in each station. Suppose there are $k$ jobs in the queueing network, for $i = 1, ..., N$ and $r = 1, ..., R$, the mean response time of a class-$r$ job at station $i$ can be computed as follows [20],

(1)

here, $(k - 1_r) = (k_1,...,k_r - 1,...,K_R)$ is the population vector with one class$r$ job less in the system. The mean system response is the sum of mean response time of each tier.

For the case of multi-server nodes ($m_i > 1$), it is necessary to compute the marginal probabilities. The marginal probability that there are $j$ jobs ($j = 1,...,(m_i - 1)$) at the station $i$, given that the network is in state $k$, is given by [20],

(2)

Applying Little's law [17], the throughput of class-$r$ jobs can be calculated,

(3)

Applying Little's Law again with the Force Flow Law [19], we derive the mean queue length $\overline{K_{ir}}$ for class-$r$ job at station $i$ as below,

$$\overline{K_{ir}}(k) = X_r(k) \cdot \overline{T_{ir}}(k) \cdot v_{ir} \tag{4}$$

The starting point of this equation is $K_{ir}(0,0...,0) = 0, \pi_i(0 \mid 0) = 1, \pi_i(j \mid 0) = 0$; after $K$ iterations, system response time, throughput and mean queue length in each tier can be computed.

In multiclass product form queueing networks, per-class station utilisation can be computed using the following equation [16],

[redacted] (5)

and the total station utilisation $U_i(k)$ is the sum of per-class station utilisation,

[redacted]

The above is the exact solution for multiclass product form queueing networks. The trade-offs between exact solutions and approximations are accuracy and speed. We use exact solutions to guide server switching decisions as a higher degree of accuracy is believed to be important here. However, a dedicated machine can be used for the switching system itself, to solve speed and storage issues and to reduce the interference with the servers themselves. In our model, job class switching is not permitted.

## 3.4 Model Parameterisation

Once a performance model is built, it can be parameterised. The parameterisation involves collection and manipulation of sample data. Sample data to be collected include service time $S_{ir}$ of each type of request, the visiting ratio $v_{ir}$. Since service demand $D_{ir} = S_{ir} \times v_{ir}$, so essentially, only service demand of each request needs to be collected. Service demand of each request is difficult to measure, however, according to the service demand law [19], $D_{ir} = U_i/X_{ir}$, here $U_i$ is the utilisation of service station $i$ and $X_{ir}$ is the throughput of job class $r$ at station $i$. Therefore, we can measure $U_i$ and $X_{ir}$ (through monitoring utility or system log) and calculate $D_{ir}$ using the service demand law. In a real test-bed, we could drive the system utilisation to a required level by sending a large number of requests that are of the same type, and measure the resulted throughput. The service demand of each request can then be computed based on the service demand law.

## 3.5 Bottleneck Identification of Multi-tier Architecture

Bottlenecks are a phenomenon where the performance or capacity of an entire system is severely limited by a single component. This component is sometimes called the *bottleneck point*. Formally, a bottleneck lies on a system's critical path and provides the lowest throughput [21]. It has been shown in [22] that multiclass models can exhibit multiple simultaneous bottlenecks. The dependency of the bottleneck set on the workload mix is therefore derived. In an enterprise system there are normally different classes of jobs and the class mix can change at run-time. This suggests that there might be several bottlenecks at the same time and bottlenecks can shift from tier to tier over time. Therefore, system designers need to study the best server configuration to avoid bottlenecks during system capacity planning and provisioning, and ideally provide schemes to support dynamic server allocation during run-time.

### 3.5.1 Identification Methods

In [23], it is shown that the bottleneck for a single class queueing network is the station $i$ with the largest service demand $S_i v_i$, under the assumption of the invariance of service time $S_i$ and visiting ratio $v_i$ and given routing frequencies. Considerable research exists [22][23][24][25][26] which studies bottleneck identification for multi-class closed product-form queueing networks as the population grows to infinity. For a finite population, the results in [27][28] can be used. In this paper we use the approach developed in [29], which uses convex polytopes for bottleneck identification in multi-class queueing networks. This method can compute the set of potential bottlenecks in a network with one thousand servers and fifty customer classes in just a few seconds.

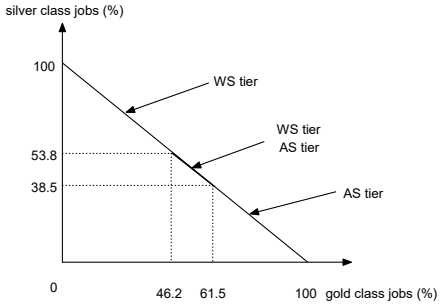


Figure 5: Bottleneck of the two-class queueing network in pool 1.

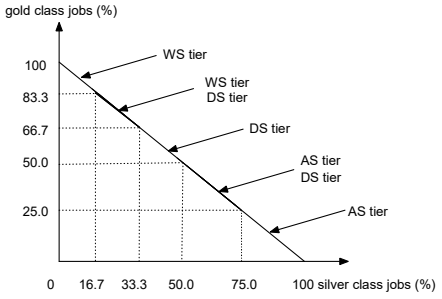


Figure 6: Bottleneck of the two-class queueing network in pool 2.

Fig. 5 and Fig. 6 are the bottleneck identification results using convex polytopes for our chosen configurations for pool 1 and pool 2. Fig. 5 shows that in pool 1, when the percentage of gold class jobs is less than 46.2%, the web server tier is the bottleneck; when it is between 46.2% and 61.5%, the system enters a *crossover points region*, where the bottleneck changes; when the percentage of gold class jobs in pool 1 exceeds 61.5%, the application server tier becomes the bottleneck.

Fig. 6 shows the bottleneck identification in pool 2. It is more complex and is a good example of multiple bottlenecks and bottleneck shifting. In this case, when the percentage of silver class jobs is less than 16.7%, the web server tier is the bottleneck; when it is between 16.7% and 33.3%, both the web server tier and the

database tier are in the *crossover* region; if the percentage of silver class jobs lies in the region 33.3% to 50.0%, the database tier becomes the bottleneck; when it is between 50.0% and 75.0%, the system enters another *crossover* region, where the application server tier and the database server tier dominate; and finally, if the percentage of silver class jobs exceeds 75.0%, the application server tier is the bottleneck in the system.

Fig. 7 and Fig. 8 provide a clear picture as to how the utilisations corresponding to the workload mix changes in both pools. The two figures can also be used to verify the results in Fig. 5 and Fig. 6.

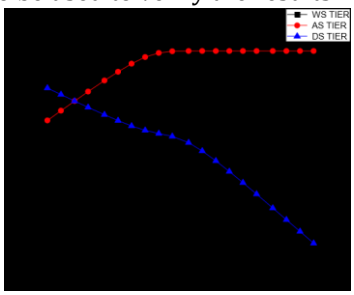


Figure 7: Utilisation in pool 1.

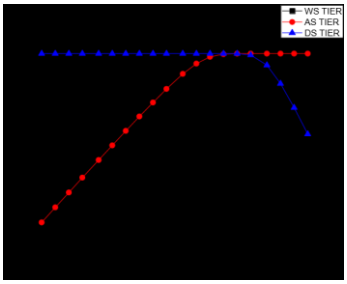


Figure 8: Utilisation in pool 2.

## 3.6 Server Switching for Revenue Maximisation

As previously highlighted, the workload in enterprise systems can vary significantly. It is therefore the case that one-time system configuration is no longer effective and it is desirable that servers be able to switch from one pool to another, depending on the load conditions. However, the server-switching operation is not cost-free, since during the period of switching the servers being switched cannot serve jobs. Therefore, a decision has to be made as to whether it is worth switching in terms of revenue maximisation.

### 3.6.1 Revenue Function

For a typical Internet service, a user normally issues a sequence of requests (referred to as a *session*) during new visit to the service site. Intuitively, a request contributes full revenue if it is processed before the deadline[2] $D_r$. When a request $r$ misses its deadline, it still waits for execution with a probability $P(T_r)$ and credit is still due for late, yet successful processing. As can be seen from Figure 9, when the response time $T_r < D_r$, then $P(T_r) = 1$; which means that the request contributes full revenue and the user will send another request. Suppose $E_r$ is some time point, at which the request is dropped from the system. It is assumed in this chapter that when $D_r \leq T_r \leq E_r$, the request will quit the system with probability $P(T_r)$, which follows a uniform distribution (refer to Figure 10). If $T_r \geq E_r$, then $P(T_r) = 0$, which means that the request quits the system without contributing any revenue. The following equation is used for calculating $P_r$,

[2] soft deadline in lieu of hard deadline is used in this chapter.

[redacted] (6)

The meaning of the above equation is that the longer the completion time of a job *r* exceeds its deadline, the more likely it is that the client will quit the system, thus approximating real-world client behaviour.

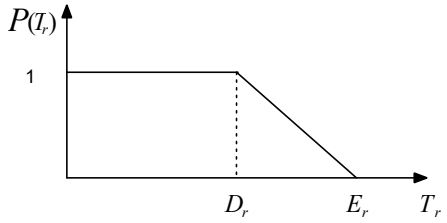


Figure 9: Illustration of the relationship between job response time and the probability that the customer will remain in the system.

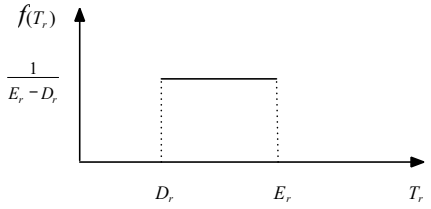


Figure 10: Illustration of the relationship between the probability density function and request response time.

### 3.6.2 Revenue Maximisation

Based on the revenue function, the revenue gained and lost by server switching can be calculated. Suppose some servers need to be switched from pool *i* to pool *j*. We use [redacted] to represent the revenue loss in pool *i*. From the time that switching happens, the service capacity offered by server pool *i* starts to degrade. From eq. 7, the revenue loss in pool *i* can be derived,

$$V_{loss}^{i} = \sum_{r=1}^{X_i^R} X_r(k)\varphi_r P(T_r) t_d - \sum_{r=1}^{X_{i^0}^R} X_r(k)\varphi_r P(T_r) t_d \quad (7)$$

The server switching itself takes time, during which neither pool *i* nor pool *j* can use the servers being switched. Only after switching time $t_s$, does pool *j* then benefit from the switched servers. During the switching decision interval time $t_d$, the revenue gain $V_{gain}^{j}$ can be calculated as below,

[redacted]

here, it is assumed the decision interval time $t_d > t_s$.

Our goal in this chapter is to maximise the ISP's total revenue contributed by both pool *i* and pool *j*. In other words, when we decide whether to switch servers, we need to compare the revenue gain and loss caused by server switching, and the switching is done only when [redacted]. In this chapter, we only consider switching servers between pools in the same tier (i.e., we switch web servers from pool *i* to the web server tier in pool *j*), although given proper configuration, the switching is also possible between tiers (i.e., switching web servers in pool *i* to the application tier in pool *j*).

## 3.7 Switching Policies

In this section, we describe two different server switching policies namely the *proportional switching policy (PSP)* and the *bottleneck-aware switching policy (BSP)*. In the real-world web service registry, there might be a large number of similar services in terms of the switching functionality, however, the service selection procedure discussed in this chapter is the same for each of the services.

### 3.7.1 Proportional Switching Policy

First, we consider a n¨aive policy called the proportional switching policy (PSP). The policy switches servers between pools based on the workload proportion in both pools. Performance criteria for server switching is computed using the queueing network model; if the performance of the new configuration is better than the current one, then server switching is done, otherwise the server configuration remains the same. Algorithm 1 describes how the policy operates.

**Input:** $N$, $m_i$, $R$, $K_{ir}$, $S_{ir}$, $v_{ir}$, $\varphi_r$ , $t_s$, $t_d$
**Output:** Server configuration
1. **for each** *i* **in** *N* **do**
2. $m_{1i}/m_{2i} = K_1/K_2$
3. **end for**
4. calculate $V_{loss}$ and $V_{gain}$ using eq. 7 and eq. 8;
5. **if** $V_{gain} > V_{loss}$ **then**
[redacted] 6. do switching according to the calculations;
7. ;
8. **else**
9. server configuration remains the same;
10. **end if**
11. return current configuration.

**Algorithm 1**: Proportional Switching Policy

Algorithm 1 is simple as it only considers the workload proportion. In fact, workload mix and revenue contribution from individual classes in different pools can also affect the total revenue. In the next section, we will introduce a new switching policy, which takes the above factors into account.

### 3.7.2 Bottleneck-aware Switching Policy

Here we describe a more sophisticated server switching policy called the bottleneck-aware switching policy (BSP), as described in Algorithm 2. BSP works in two phases: 1) *Bottleneck identification*. It first checks for bottleneck saturation in both pools. If both pools have bottlenecks at the same tier, two cases are considered: *a*) if both of them are saturated, then no server will be switched; *b*) if a bottleneck is saturated in one pool but not in the other, then the algorithm incrementally switches servers to the bottleneck tier and compares the

**Input:** $N_r$, $m_i$, $R$, $K_{ir}$, $S_{ir}$, $v_{ir}$, $\varphi_r$ , $t_s$, $t_d$
**Output:** new configuration
1. **while** bottleneck saturation found in one pool **do**
2. **if** found at same tier in the other pool **then**
3. return;
4. **else** switch servers to the bottleneck tier;
5.and;
6. **end if**
7. **end while**
8. search configurations using Algorithm 3
9. return current configuration.

**Algorithm 2**: The Bottleneck-aware Switching Policy

**Input:** $N_r$, $m_i$, $R$, $K_{ir}$, $S_{ir}$, $v_{ir}$, $\varphi_r$ , $t_s$, $t_d$
**Output:** best configuration
**Initialisation**: compute $U_i^1$,$U_i^2$

1. **while** [redacted] **do**
2. **if** [redacted] **then**
3. [redacted] ;
4. **while** [redacted] **do**
5. **if** [redacted] **then**
6. [redacted];
7. **whiledo**
8. **if** [redacted] **then**
9. [redacted];
10. compute $V_{loss}$ using eq. 7;
11. [redacted];
12. compute $V_{gain}$ using eq. 8;
13. **if** $V_{gain} > V_{loss}$ **then**
14. store current configuration;
15. **end if**
16. compute new $U_i^1$,$U_i^2$;
17. **end if**
18. **end** [redacted] **while**
19. similar steps for [redacted]
20. [redacted] ;
21. compute new $U_i^1$,$U_i^2$;
22. **end if**
23. **end while**
24. similar steps for [redacted]
25. [redacted] ;
26. compute new $U_i^1$,$U_i^2$;
27. **end if**
28. **end while**
29. similar steps for [redacted]
30. return best configuration.

**Algorithm 3**: The Configuration Search Algorithm new revenue with the value from the current configuration. If a potential switch will result in more revenue, then the configuration will be stored. The process continues until no bottleneck saturation in either pools or no more switching can be done from the other pool. Note that when bottleneck saturation is found, server switching in other tiers has little or no effect, thus it can be safely neglected. 2) *Local search*. If there is no bottleneck saturation in either of the pools, then the algorithm computes the server utilisation at all tiers in both pools and switches servers from low utilisation tiers to high utilisation tiers using a local search algorithm (Algorithm 3). In both algorithms,

superscripts represent pools and subscripts 0, 1, 2 represent the web tier, application tier and database tier respectively.

Algorithm 3 uses nested loops to search for possible server switches, starting from the web tier continuing to the database tier. It tries to explore as many possible switching configurations as possible. However, the algorithm will not guarantee that the best switching result (the global optimal) will be found, thus it is a best-effort algorithm. If we use $m_0$,$m_1$,$m_2$ to represent the total number of web servers, application servers and database servers in both pools respectively, in the worst case, the total number of searches made by Algorithm 3 will be $(m_0-2)\times(m_1-2)\times(m_2-2)$, therefore the time complexity is $O(m_0 \cdot m_1 \cdot m_2)$. For typical server configurations, $m_0$, $m_1$ and $m_2$ are not normally large, thus Algorithm 3 is feasible in practice. The time for each search iteration depends on the complexity of the underlying queueing network model, which in turn depends on the number of stations and the number of job classes (the dominant factor as shown in [25]). Enterprise systems are normally three-tiered ($N$ = 3), and the number of job classes is normally small, depending on the classification criteria. Therefore, solving such a multi-class closed queueing network model is very quick, thus the same applies for each iteration in the searching algorithm. As shown later in this chapter, for our configuration, the average runtime of the algorithm is less than 200 milli-seconds on a 2.2Ghz computer, which is considered acceptable.

For complex multi-class closed queueing network models, with thousands of stations and hundreds of job classes, the storage requirement for solving the models are very high. In our case, storage is also not an issue as the model is relatively simple. Moreover, as mentioned in section **??**, using a dedicated machine for the switching engine can increase the searching speed, and also relax the associated storage requirement.

## 3.8 Proactive and Reactive Switching

In our proposed switching system, two approaches to server switching can be used – proactive switching and reactive switching. Proactive switching is motivated by identifying similar workload patterns over time (hours, days, weeks etc). Most Internet services have cyclical patterns. For instance, for real-time financial applications, the peak load normally appears at the beginning and the end of the market, and the load is lower during the remainder of the opening hours; it is also the case that Monday and Friday are busier than other weekdays. Based on historical workload patterns, and by applying some workload prediction techniques such as those introduced in [30], the server switching engine can re-allocate resources before the expected heavy workload arrives, and also, can save the costs of server switching during a heavily loaded period. However, due to uncertainties, workload demand can have huge variation and predictive inaccuracies can be introduced by the workload predictor, which are then passed to the switching engine, stimulating inappropriate or wrong decisions. Therefore, proactive switching is not perfect and it can at best hope to improve the overall performance during long term periods.

Reactive switching is more dynamic, based on run-time system parameters and can respond to system state changes quickly. The run-time data is collected

via system monitoring tools, is reformatted, and is fed into the analytical model. The model then is solved and alternative switching decisions are compared. The proactive and reactive switching approaches can of course work together to optimise the overall system performance.

### 3.9 Admission Control

As described in the literature, admission control (AC) is necessary for busy Internet services in order to achieve the SLAs. When a system is overloaded, most ISPs simply reject less important requests. ISPs may give their customers compensation for the rejected requests, depending on the SLAs between themselves and their customers.

In this work, we also use a simple admission control scheme, in addition to the server switching policy, to maintain the number of concurrent jobs in the system at an appropriate level. When the workload is high, which in turn makes the overall system response time high, less important requests are rejected first. If requests in this category are rejected, but the overall response time still remains high, the AC scheme continues to reject jobs in the system, until the response time decreases to an acceptable level.

## 4 Performance Evaluation

### 4.1 Experimental Setup

We design and develop a simulator to evaluate the server switching approach in this chapter. Two applications are simulated, running on two logical pools (1 and 2). Each application has two classes of job (gold and silver), which represent the importance of these jobs. Both applications are multi-tiered and run on a cluster of servers. The service time $S_{ir}$ and the visiting ratio $v_{ir}$ are chosen based on realistic values or from those supplied in supporting literature.

Based on a real test-bed which we have access to, the application server switching takes less than five seconds and web server switching is relatively straightforward. Database server switching is more complex, however, it does not affect the switching policy itself. In this chapter, we assume switching cost for web servers, application servers and database servers is the same for simplicity. Experimental parameters used for our evaluation can be found in Table 2.

### 4.2 Evaluation Results

Experiments have been conducted for a number of different workload scenarios called *mixed workload*, *cross load*, *random load* and workload generated from real-world Internet traces. For each of these cases, we compare the results from

Table 2: Experimental parameters.

| | Pool 1 | | Pool 2 | |
|---|---|---|---|---|
| | Silver | Gold | Gold | Silver |

| | | | | | |
|---|---|---|---|---|---|
| Number of servers | WS | 4 | | 5 | |
| | AS | 10 | | 15 | |
| | DS | 2 | | 3 | |
| Service time(sec) | WS | 0.07 | 0.1 | 0.05 | 0.025 |
| | AS | 0.03125 | 0.1125 | 0.01 | 0.06 |
| | DS | 0.05 | 0.025 | 0.0375 | 0.025 |
| Visiting ratio | WS | 1.0 | 0.6 | 1.0 | 0.8 |
| | AS | 1.6 | 0.8 | 2.0 | 1.0 |
| | DS | 1.2 | 0.8 | 1.6 | 1.6 |
| Deadline (sec) | | 20 | 15 | 6 | 8 |
| Exit point (sec) | | 30 | 20 | 10 | 12 |
| Revenue unit | | 2 | 10 | 20 | 4 |

our proposed bottleneck-aware server switching policy (BSP) with those from the proportional server switching policy (PSP) and the non-switching policy (NSP).

### 4.2.1 Mixed Workload

As described in section 3.7.2, even if the total workload remains the same, system bottlenecks can shift among tiers depending on the workload mix. To study the system behaviour of different workload mixes, we choose a few key evaluation points illustrated in Figure 5 and Figure 6. Two sets of experiments are run: 1) keeping the workload mix constant in pool 1 and altering the workload mix in pool 2, as shown in Figure 11, 12 and 13; 2) keeping the workload mix in pool 2 constant and altering the workload mix in pool 1 as seen in Figure 14, 15, 16. The server switching time is set to 5 seconds and the switching decision is made every 30 seconds. We explain the impact of the workload mix on the total revenue for the NSP, and compare the results against the PSP and BSP policies.

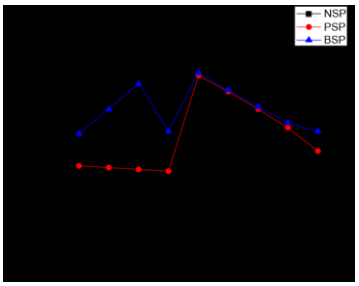


Figure 11: The ratio of silver class jobs to gold class jobs in pool 1 is (80:20).The percentage of silver class jobs in pool 2 ranges from 10% to 90%.

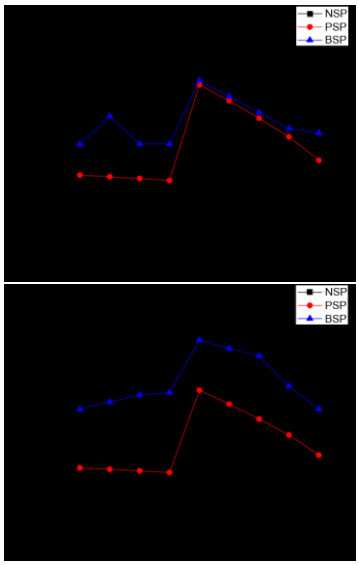


Figure 12: The ratio of silver class jobs to gold class jobs in pool 1 is (60:40). The percentage of silver class jobs in pool 2 ranges from 10% to 90%.

Figure 13: The ratio of silver class jobs to gold class jobs in pool 1 is (20:80). The percentage of silver class jobs in pool 2 ranges from 10% to 90%.

Figure 14: The ratio of gold class jobs to silver class jobs in pool 2 is (80:20).The percentage of gold class jobs in pool 1 ranges from 10% to 90%.

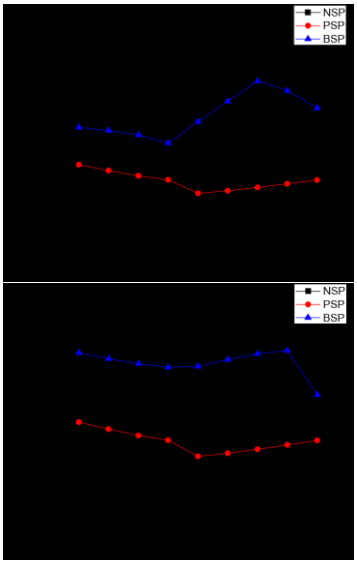


Figure 15: The ratio of gold class jobs to silver class jobs in pool 2 is (60:40). The percentage of gold class jobs in pool 1 ranges from 10% to 90%.

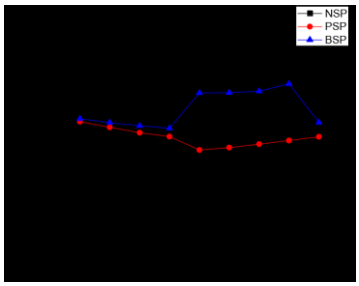


Figure 16: The ratio of gold class jobs to silver class jobs in pool 2 is (20:80). The percentage of gold class jobs in pool 1 ranges from 10% to 90%.

From Figure 11, 12 and 13, it can be seen that when the workload mix in pool 1 is constant, Figure 11, 12 and 13 show similar patterns. The total revenue from both pools from NSP and PSP decreases when the percentage of silver class jobs in pool 2 increases from 10% to 40%. This is understandable as silver class jobs

contribute less to the total revenue. When the percentage increases to 50%, there is a big increase in total revenue. Based on our observations, this is due to a lower response time in pool 2, which is less than $E_r$ for gold class jobs in pool 2. When the percentage of silver class jobs is over 50%, although the response time in pool 2 decreases, the total revenue again decreases due to the decreasing weight of gold class jobs. It can also be seen that Figure 11 has the highest revenue and Figure 13 has the lowest revenue among the three cases. This is due to the longer response time (within deadline) in pool 1 as a result of the percentage increases in gold class jobs in the pool. As we know, a longer response time results in less throughput, which then results in less revenue contribution.

In the second set of experiments, the workload mix in pool 2 is constant and the percentage of gold class jobs in pool 1 is altered. Figure 14, 15 and 16 also present similar patterns. The total revenue in all three cases decreases when the percentage of gold class jobs in pool 1 increases from 10% to 50%. The difference in revenue between BSP and the other two policies is smaller as the weight of gold class jobs increases. When the percentage is greater than 50%, the total revenue increases as the percentage of gold class jobs in pool 1 increases. We notice that the total revenue in Figure 16 is significantly higher than that in the other two cases. This is due to lower response time (below $E_r$) of both classes of jobs in pool 2, which can result in a significant increase in revenue.

In both sets of experiments, it can be seen that PSP and NSP have almost the same impact on total revenue for the one-time switching. The total revenue from NSP is always higher than those from the other two policies as the local search algorithm is employed in BSP and switching is done only when a better configuration is found.

### 4.2.2 Alternative Workload

In a web hosting centre, it is not uncommon that during certain periods the workload for one application is increasing while it is decreasing for another. This kind of crossover in workload can affect overall system performance. In this section, we conduct performance evaluation for two cases: 1) when the workload increases in pool 1 and decreases in pool 2; 2) when the workload increases in pool 2 and decreases in pool 1. In both cases, the workload mix for silver and gold class jobs in both pools is constant. The total number of concurrent users is set to a fixed number (200), which matches the value in section 4.2.1. During evaluation, admission control is applied when necessary. Both sets of experiments are run for 570 seconds, during which 19 switching decisions are made. Table 3 and Table 4 list the results for both sets of experiments.

In Table 3, we see that the workload in pool 1 increases by 10 each time from 10 to 200, while it decreases by 10 from 200 to 10 in pool 2. The total revenue from NSP is 88,093. If AC is not applied, the total revenue from PSP and BSP are 85,130 and 1,900,034, representing a -3.4% and a 115.7% improvement, respectively. When AC is applied, the total revenue from PSP and BSP are 85,130 and 211,947, representing a -3.4% and a 140.6% improvement, Table 3: Load in pool 1 increases while it decreases in pool 2.

| Workload | NSP | Without A/C | With A/C |
|---|---|---|---|

| (P1, P2) | | PSP | BSP | PSP | BSP |
|---|---|---|---|---|---|
| (20,190) | 2418 | 403 | 5916 | 403 | 5916 |
| (30,180) | 2429 | 2429 | 2569 | 2429 | 2569 |
| (40,170) | 2429 | 2429 | 6134 | 2429 | 6134 |
| (50,160) | 2425 | 2425 | 2619 | 2425 | 2619 |
| (60,150) | 2420 | 2420 | 7175 | 2420 | 7175 |
| (70,140) | 2415 | 2415 | 3458 | 2415 | 3385 |
| (80,130) | 2410 | 2410 | 15097 | 2410 | 15097 |
| (90,120) | 3827 | 3827 | 10389 | 3827 | 9288 |
| (100,110) | 3459 | 3459 | 11014 | 3459 | 3837 |
| (110,100) | 6374 | 6374 | 11872 | 6374 | 16510 |
| (120,90) | 5244 | 5526 | 11189 | 5526 | 16497 |
| (130,80) | 5557 | 6923 | 7963 | 6923 | 16233 |
| (140,70) | 4761 | 6255 | 13367 | 6255 | 16151 |
| (150,60) | 6735 | 3780 | 13408 | 3780 | 16038 |
| (160,50) | 6834 | 6905 | 13461 | 6905 | 15877 |
| (170,40) | 6944 | 6273 | 13532 | 6273 | 15639 |
| (180,30) | 7068 | 6478 | 13632 | 6478 | 15264 |
| (190,20) | 7201 | 7012 | 13752 | 7012 | 14604 |
| (200,10) | 7143 | 7387 | 13487 | 7387 | 13114 |
| Total revenue | 88093 | 85130 | 190034 | 85130 | 211947 |
| Improvement | | -3.4% | 115.7% | -3.4% | 140.6% |

respectively. The negative impact from PSP is reasonable as the PSP is a n¨aive switching policy, which simply allocates servers based on the workload proportion regardless of the performance results. Moreover, for each server switching, there is also a cost associated with it. Although during each run, the resulting revenue from PSP is higher than from NSP, in the long term the overall improvement could be negative (note that PSP does not switch servers in each run). In this set of experiments, there is also a performance improvement when admission control is applied.

In Table 4, the workload in pool 1 decreases by 10 each time step from 200 to 10, while it increases in steps of 10 from 10 to 200 in pool 2. The total revenue from NSP is 83,289. Without AC, the total revenue from PSP and BSP are 105,698 and 127,469, representing a 26.9% and a 53.0% performance improvement, respectively. When AC is applied, the new total revenues are 105,698 and 117,808, representing a 26.9% and a 41.4% improvement. Note that with AC, the total revenue from BSP is less than it is in the no AC case. This is reasonable for light load situation (such as the chosen workload in this case) because the AC works before the BPS and if the workload results in system bottleneck saturation, the AC simply rejects requests. However, the saturation for current configuration can be relaxed in another configuration that is returned by the BPS, and the rejected requests will result in loss of revenue. We believe when workload is high, due to switching cost, the overall revenue without AC will be less than it in the with AC case. To confirm this, we set the total number of users in both pools to

250. The total revenue from NSP is now 84,170. Without Table 4: Load in pool 1 decreases while it increases in pool 2

| Workload (P2, P1) | NSP | Without A/C | | With A/C | |
|---|---|---|---|---|---|
| | | PSP | BSP | PSP | BSP |
| (20,190) | 7201 | 6227 | 14492 | 6227 | 14492 |
| (30,180) | 7068 | 6862 | 11895 | 6862 | 11895 |
| (40,170) | 6944 | 5584 | 9865 | 5584 | 9865 |
| (50,160) | 6834 | 5576 | 14617 | 5576 | 14617 |
| (60,150) | 6735 | 5569 | 14787 | 5569 | 14787 |
| (70,140) | 4761 | 5560 | 14917 | 5560 | 14917 |
| (80,130) | 5557 | 5551 | 1790 | 5551 | 1790 |
| (90,120) | 5244 | 5540 | 4567 | 5540 | 4742 |
| (100,110) | 6374 | 5528 | 7562 | 5528 | 5238 |
| (110,100) | 3459 | 5515 | 9442 | 5515 | 3321 |
| (120,90) | 3827 | 5499 | 12455 | 5499 | 4233 |
| (130,80) | 2410 | 5482 | 586 | 5482 | 2028 |
| (140,70) | 2415 | 5461 | 1792 | 5461 | 7181 |
| (150,60) | 2420 | 5436 | 1346 | 5436 | 1346 |
| (160,50) | 2425 | 5405 | 1468 | 5405 | 1468 |
| (170,40) | 2429 | 5367 | 1469 | 5367 | 1469 |
| (180,30) | 2429 | 5314 | 1471 | 5314 | 1471 |
| (190,20) | 2418 | 5229 | 1474 | 5229 | 1474 |
| (200:10) | 2339 | 4993 | 1474 | 4993 | 1474 |
| Total revenue | 83289 | 105698 | 127469 | 105698 | 117808 |
| Improvement | | 26.9% | 53.0% | 26.9% | 41.4% |

AC, it is 127,918 using PSP and 158,487 from BSP, representing a 52.0% and a 88.3% performance improvement. With AC, the total revenue from PSP and BSP are 127,918 and 161,550, representing a 52.0% and a 115.7% performance improvement. In conclusion, BSP always outperforms PSP in terms of revenue contribution. The AC doesn't always improve performance, depending on the workload intensity and workload mix.

### 4.2.3 Random Workload

In this section, we consider a more representative workload scenario – the random workload. The number of users in pools 1 and 2 are uniformly distributed between 20 and 200. Moreover, the workload mix in each pool is also random. In section 4.2.1 and section 4.2.2, a thirty-second fixed switching decision interval is used. In this section the switching decision interval time is the same as the workload change interval time, which is also a random number uniformly distributed in a fixed range. Two cases are considered: 1) a short switching decision interval time uniformly distributed between 15 and 25 seconds; 2) a long switching decision interval time uniformly distributed between 25 to 55 seconds. In section 4.2.1 and section 4.2.2, a 5 second fixed server switching time is used;

we also alter the switching time (to 5, 10 and 15 seconds) and evaluate the performance impact of the switching cost on total revenue for the three different switching policies. We evaluate the performance of the three policies with and without the admission control scheme for each of the above cases. All the experiments run for approximately two hours, during which 1,000 switching decisions are made.

Table 5: Short decision interval for random load.

| | | Without A/C | | | With A/C | | |
|---|---|---|---|---|---|---|---|
| Switching time | Metrics | NSP | PSP | BSP | NSP | PSP | BSP |
| 5 sec | No. of switches | 0 | 130 | 20 | 0 | 145 | 15 |
| | Revenue (x1000) | 2340 | 2833 | 5692 | 2340 | 2813 | 5702 |
| | Improvement (%) | 0 | 21.1 | 143.3 | 0 | 20.2 | 143.7 |
| | Improvement over non-ac (%) | | | | 0 | -0.71 | 0.17 |
| 10 sec | No. of switches | 0 | 108 | 3 | 0 | 112 | 13 |
| | Revenue (x1000) | 2340 | 2886 | 4731 | 2340 | 2894 | 5684 |
| | Improvement (%) | 0 | 23.3 | 102.2 | 0 | 23.7 | 142.9 |
| | Improvement over non-ac (%) | | | | 0 | 0.27 | 20.2 |
| 15 sec | No. of switches | 0 | 101 | 3 | 0 | 106 | 3 |
| | Revenue (x1000) | 2340 | 2928 | 4730 | 2340 | 2937 | 4783 |
| | Improvement (%) | 0 | 25.2 | 102.1 | 0 | 25.5 | 104.4 |
| | Improvement over non-ac (%) | | | | 0 | 0.29 | 1.13 |

Tables 7 and 8 list the performance results for short and long switching decision intervals (thus switching decision interval time). As can be seen from Table 7, for different server switching times, both PSP and BSP perform better than NSP in terms of revenue contribution with and without AC. When no AC is applied, the improvements are 21.1% and 143.3%, 23.3% and 102.2%, 25.2% and 102.1% for the 5, 10 and 15 second switching times respectively. With AC, the improvement are 20.2% and 143.7%, 23.7% and 142.9%, 25.5% and 104.4%, for the three cases, respectively. Without AC, the numbers of switches are 130 and 20, 108 and 3, 101 and 3, for 5, 10 and 15 second switching times respectively. When AC is employed, the numbers are 145 and 15, 112 and 13, 106 and 3, respectively. As can be seen from both tables, the number of server switches decreases as the server switching time increases. This is because the increase in switching time makes server switching more costly, which results in fewer switches. PSP always implements more switches than BSP. Also, the total revenue from BSP decreases slightly whereas it increases using PSP as the server switching time increases. This is understandable since PSP makes switching decisions solely based on workload proportion, and it switches servers even though the performance improvement may be very small. BSP on the other hand tries to search for the best switching that results in more improvement at each switching step. We find that the configuration returned by BSP is usually much

further from the current configuration (that not found by PSP), thus each BSP switching step is more costly than that from PSP. On average, for each switching step, the ratio of the improvement over the cost from BSP is greater than that from PSP. Thus, BSP results in more revenue than the PSP policy. Due to the nature of the random load, servers may need to be switched back to their original pool. As the switching time increases, the number of switches for both policies decreases, therefore the total revenue increases from PSP but decreases from BSP. However, BSP consistently outperforms PSP in terms of revenue contribution for all cases, and the improvement from BSP over NSP is more than four times that of PSP.

From Table 7, it can also be seen that when AC is employed, there is a considerable improvement (20.2%) when the server switching time is 10 seconds. The improvement for the other two cases is less pronounced. The table also shows that when AC is employed, PSP results in more switches in each case compared with the no AC case. We believe this is a result of the workload mix change, which is caused by the AC.

Table 6: Long decision interval for random load.

| | | Without A/C | | | With A/C | | |
|---|---|---|---|---|---|---|---|
| Switching time | Metrics | NSP | PSP | BSP | NSP | PSP | BSP |
| 5 sec | No. of switches | 0 | 152 | 20 | 0 | 158 | 13 |
| | Revenue (x1000) | 4778 | 5702 | 11567 | 4778 | 5661 | 11579 |
| | Improvement (%) | 0 | 19.4 | 142.1 | 0 | 18.5 | 142.4 |
| | Improvement over non-ac (%) | | | | 0 | -0.73 | 0.11 |
| 10 sec | No. of switches | 0 | 134 | 20 | 0 | 82 | 15 |
| | Revenue (x1000) | 4778 | 5710 | 11557 | 4778 | 6399 | 11577 |
| | Improvement (%) | 0 | 19.5 | 141.9 | 0 | 33.9 | 142.3 |
| | Improvement over non-ac (%) | | | | 0 | 12.1 | 0.17 |
| 15 sec | No. of switches | 0 | 119 | 3 | 0 | 80 | 15 |
| | Revenue (x1000) | 4778 | 5832 | 9539 | 4778 | 6436 | 11566 |
| | Improvement (%) | 0 | 22.1 | 99.7 | 0 | 34.7 | 142.1 |
| | Improvement over non-ac (%) | | | | 0 | 10.4 | 21.2 |

Table 8 presents similar results to those seen in Table 7. Without AC, the number of switches for PSP increases from 130 to 152, 108 to 134, 101 to 119 for 5, 10 and 15 second switching times, respectively; the number from BSP drops to 3 for the 15 second case, this trend can also be seen in Table 7. This is reasonable as longer switching interval times result in potentially better configurations, thus more switches. With AC, the number of server switches for PSP increases from 145 to 158 for the 5 second case, but decreases from 112 to
82, 106 to 80 for the other two cases; the numbers of switches from BSP are 13, 15, 15 for 5, 10, 15 second switching times, respectively. We believe that the workload mix (more weight for gold class jobs) in the long switching decision

interval case will result in more potentially better configurations, and thus more switches.

The revenue improvement when using BSP is almost 142% for all the cases regardless of the use of AC (an exception is the 99.7% implement for the case when the server switching time is 15 seconds and no AC is employed). The reason for the latter decrease is the same as for the number of switches above. The total revenue improvement from PSP without AC are 19.4%, 19.5% and 22.1% for the three switching time cases. With AC, the improvements are 18.5%, 33.9% and 34.7%. The improvements are, however, much less than those from BSP regardless of the use of AC.

### 4.2.4 Workloads Generated from Internet Traces

The workloads used for our simulation are generated from real-world Internet traces [31]. Two Internet traces are used for the workloads in the two server pools in the experiments. The *EPA-HTTP* trace contains a day's worth of HTTP requests to the EPA WWW server located at Research Triangle Park, NC. The *SDSC-HTTP* trace contains a day's worth of HTTP requests to the SDSC WWW server located at the San Diego Supercomputer Centre in California. Workload characteristics (in terms of the number of requests in the systems) in both traces are extracted every five minutes. In this section, two switching decision intervals are considered: 1) a short switching decision interval – 30 seconds; 2) a long switching decision interval – 60 seconds. In section 4.2.2, a five-second fixed server switching time is used; we use different server switching times (5, 10 and 15 seconds) in this section and evaluate the performance impact of the switching cost on total revenue for the three different switching policies. We evaluate the performance of the three policies with and without the admission control scheme for each of the above cases.

Table 7: Short decision interval for workload from traces.

| | | Without A/C | | | With A/C | | |
|---|---|---|---|---|---|---|---|
| Switching time | Metrics | NSP | PSP | BSP | NSP | PSP | BSP |
| 5 sec | No. of switches | 0 | 18 | 5 | 0 | 18 | 7 |
| | Revenue (x1000) | 614 | 683.5 | 1374 | 614 | 683.2 | 1447 |
| | Improvement (%) | 0 | 11.3 | 123.7 | 0 | 11.3 | 135.6 |
| | Improvement over non-ac (%) | | | | 0 | 0 | 11.9 |
| 10 sec | No. of switches | 0 | 14 | 5 | 0 | 13 | 16 |
| | Revenue (x1000) | 614 | 715 | 1370 | 614 | 714.7 | 1370 |
| | Improvement (%) | 0 | 16.4 | 123.2 | 0 | 16.4 | 123.2 |
| | Improvement over non-ac (%) | | | | 0 | 0 | 0 |
| 15 sec | No. of switches | 0 | 13 | 16 | 0 | 13 | 24 |
| | Revenue (x1000) | 614 | 648.7 | 569.1 | 614 | 648.7 | 1250 |
| | Improvement (%) | 0 | 5.6 | -7.3 | 0 | 5.6 | 103.5 |

| | | | | |
|---|---|---|---|---|
| Improvement over non-ac (%) | 0 | 0 | 110.8 |

Tables 7 and 8 list the performance results for short and long switching decision intervals. As can be seen from both tables, for different server switching times, both PSP and BSP perform better than NSP in terms of revenue contribution with and without AC, except for the long interval case when the server switching time is 15 seconds. The improvement for PSP ranges from 5.6% to 16.4% whereas it ranges from 103.5% to 136% for BSP with one exception (-7.3%).

Table 8 shows that when the number of switches is the same (the number of switches for BSP is the same for different switching times), the longer the server switching time is, the less the performance improvement is. Table 8 also shows that the number of switches for PSP when the switching time is 5 and 10 seconds is the same; but when the switching time increases to 15 seconds, the number of switches decreases by 2 to 16, which results in a slight performance improvement. In Table 7, it can be seen that when the server switching time increases from 5 seconds to 10 seconds, the number of server switches for PSP drops from 18 to 14, which results in slight performance improvement. When the switching time is increased to 15 seconds, the number of switches only decreases by 1. Since the switching cost has increased by 50%, the total revenue is reduced. Results from both tables are intuitive. Server switching is not costfree, therefore, the performance improvement is closely related to how long a switching takes and the number of server switches. There is a trade-off between performance improvement and the number of server switches, and it depends on the decision interval and the server switching time. On the one hand, more switches results in more potential performance improvement, on the other hand, due to the switching costs involved, too many switches could result in less or negative improvement.

Table 8: Long decision interval for workload from traces.

| | | Without A/C | | | With A/C | | |
|---|---|---|---|---|---|---|---|
| Switching time | Metrics | NSP | PSP | BSP | NSP | PSP | BSP |
| 5 sec | No. of switches | 0 | 18 | 5 | 0 | 18 | 7 |
| | Revenue (x1000) | 1228 | 1369 | 2750 | 1228 | 1369 | 2899 |
| | Improvement (%) | 0 | 11.5 | 123.9 | 0 | 11.4 | 136.0 |
| | Improvement over non-ac (%) | | | | 0 | -0.1 | 12.1 |
| 10 sec | No. of switches | 0 18 | 5 | 0 | 18 | 7 | |
| | Revenue (x1000) | 1228 | 1367 | 2747 | 1228 | 1366 | 2894 |
| | Improvement (%) | 0 | 11.3 | 123.7 | 0 | 11.3 | 135.6 |
| | Improvement over non-ac (%) | | | | 0 | 0 | 11.9 |
| 15 sec | No. of switches | 0 | 16 | 5 | 0 | 16 | 7 |
| | Revenue (x1000) | 1228 | 1420 | 2744 | 1228 | 1420 | 2890 |
| | Improvement (%) | 0 | 15.6 | 123.4 | 0 | 15.6 | 135.3 |

| | | | |
|---|---|---|---|
| Improvement over non-ac (%) | 0 | 0 | 11.9 |

For the chosen workload, when AC is applied, there is no performance improvement for PSP, and the overall improvement for BSP is approximately 12%. The exception is the last case in Table 7, where the improvement is 103.5% with AC but is negative without AC.

In conclusion, for certain workload scenarios, there is a trade-off between performance improvement and the number of server switches for different server switching times and switching decision intervals. The number of server switches depends on workload characteristics. Admission control schemes do not always improve performance for all workload scenarios.

# 5 The Selection of Switching Services

After extensive performance evaluation of the switching services, the agent can then choose the most suitable service among all services. The goal of each server switching service in the given example is to maximise the total revenue from both server pools. The performance results show that the BSP service outperforms the PSP service in terms of revenue contribution to ISPs, therefore, it should be chosen in the given scenario.

# 6 Summary

In this chapter, we first explain the importance of web service selection as the number of functionally similar services is increasing, hence, the agent has to choose the best one among those services based on some non-functional attributes such as the QoS of each service. The challenge of web service selection arise due to a number of factors. One important factor is that it is very difficult to assess the QoS of a service, especially in the real-time environment, where changing system state can affect the quality of the service. Another important factor that could affect the assessment of QoS is that some dishonours service providers over claim the quality of their services to attract more clients. This issue is closely related to reputation of service providers and trust between them and the agents (or the end users). Some approaches such as the introduce of user feedback (or voting) mechanisms can help to resolve the trust-related issues.

After general discussion of the procedure of service selection, we then use a server switching service as an case study to describe the service selection procedure. The focus of case study is on performance modelling of the multitier Internet services and performance evaluation of various switching services employed in such as multi-tier architecture. As can be seen in this chapter, service selection after performance evaluation is straightforward, therefore, performance evaluation plays a very important role in web service selection. The development of performance models can be time-consuming, however, it is the service providers' responsibility to model their service and the targeted execution environments.

# References


[1] L. Vu, M. Hauswirth, and K. Aberer. QoS-Based Service Selection and Ranking with Trust and Reputation. Lecture Notes in Computer Science, 2005.

[2] M. Sun and F. Arbab. Qos-driven Service Selection and Composition. In *8th International Conference on Application of Concurrency to System Design*, 2008.

[3] D. A. D'Mello and V. S. Ananthanarayana. Quality Driven Web Service Selection and Ranking. In *Fifth International Conference on Information Technology (ITNG'08)*, 2008.

[4] Y. Gao, J. Na, B. Zhang, L. Yang, and Q. Gong. Optimal Web Services Selection Using Dynamic Programming. In *11th IEEE Symposium on Computers and Communications (ISCC'06)*.

[5] P. C. Xiong and Y. S. Fan. QoS-aware Web Service Selection by a Synthetic Weight. In *Fourth International Conference on Fuzzy Systems and Knowledge Discovery*, 2007.

[6] D. T. Tsesmetzis, I. G. Roussaki, I. V. Papaioannou, and M. e. Anagnostou. QoS awareness Support in Web Service Semantics. In *the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW'06)*, 2006.

[7] A. S. Ali, S. A. Ludwig, and O. F. Rana. A Cognitive Trust-Based Approach for Web Service Discovery and Selection. In *Third IEEE European Conference on Web Services*, 2005.

[8] S. Galizia, A. Gugliotta, and J. domingue. A Trust Based Methodology for Web Service Selection. In *International Conference on Semantic Computing (ICSC'07)*, 2007.

[9] WSDL. Web Serivce Description Language. In *http://www.w3.org/TR/ wsdl*.

[10] UDDI. Universal Description Discovery and Integretion. In *http:// www.uddi.org*, 2006.

[11] M. Arlitt and T. Jin. A Workload Characterization Study of the 1998 World Cup Web Site. *IEEE Network*, 14(3):30–37, 2000.

[12] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. *ACM SIGMETRICS Performance Evaluation Review*, 26(1):151–160, 1998.

[13] J. Y. Zhou and T. Yang. Selective Early Request Termination for Busy Internet Services. In *15th International Conference on World Wide Web, Edinburgh, Scotland*, 2006.

[14] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. Tantawi. An Analytical Model for Multi-tier Internet Services and its Applications. *ACM SIGMETRICS Performance Evaluation Review*, pages 291– 302, 2005.

[15] A. Zalewski and A. Ratkowski. Evaluation of Dependability of Multitier Internet Business Applications with Queueing Networks. In *International Conference on Dependability of Computer Systems ( DEPCOSRELCOMEX'06)*, 2006.

[16] M. Reiser and S. Lavenberg. Mean-value Analysis of Closed Multi-Chain Queueing Networks. *Journal of the Association for Computing Machinary*, 27:313–322, 1980.

[17] J. Little. A Proof of the Queueing Formula $L = \lambda W$. *Operations Research*, 9(3):383–387, May 1961.

[18] K. Sevcik and I. Mitrani. The Distribution of Queueing Network States at Input and Output Instants. *Journal of the Association for Computing Machinary*, 28(2), 1981.

[19] D. A. Menasce and V. A. F. Almeida. *Capacity Planning for Web Performance: metrics, models,and methods*. Prentice Hall PTR, 1998.

[20] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: modelling and performance evaluation with computer science applications*. Wiley, 2nd edition, 2006.

[21] J. Y. L. Boudec. *Rate Adaptation, Congestion Control and Fairness: A Tutorial*, Nov 2005.

[22] G. Balbo and G. Serazzi. Asymptotic Analysis of Multiclass Closed Queueing Networks: Multiple Bottlenecks. *Performance Evaluation*, 30(3):115– 152, 1997.

[23] P. J. Denning and J. P. Buzen. The Operational Analysis of Queueing Network Models. *ACM Computing Surveys*, 10(3):225–261, 1978.

[24] C. Knessl and C. Tier. Asymptotic Approximations and Bottleneck Analysis in Product Form Queueing Networks with Large Populations. *Performance Evaluation*, 33(4):219–248, 1998.

[25] M. Litoiu. A Performance Analysis Method for Autonomic Computing Systems. *ACM Transaction on Autonomous and Adaptive Systems*, 2(1):3, 2007.

[26] P. J. Schweitzer. A Fixed-point Approximation to Product-form Networks with Large Populations. In *2nd ORSA Telecommunication Conference*, 1992.

[27] D. L. Eager and K. C. Sevcik. Bound Hierarchies for Multiple-class Queueing Networks. *Journal of ACM*, 33(1):179–206, 1986.

[28] T. Kerola. The Composite Bound Method for Computing Throughput Bounds in Multiple Class Environments. *Performance Evaluation*, 6(1):1– 9, 1986.

[29] G. Casale and G. Serazzi. Bottlenecks Identification in Multiclass Queueing Networks Using Convex Polytopes. In *Modelling, Analysis, and Simulation of Comp. and Telecommunication Systems (MASCOTS)*, 2004.

[30] J. Rolia, X. Zhu, M. Arlitt, and A. Andrzejak. Statistical Service Assurances for Applications in Utility Grid Environments. Technical report, Technical Report HPL-2002-155, HP Labs, 2002.

[31] Internet Trace. Internet Traffic Archive Hosted at Lawrence Berkeley National Laboratory. In *http://ita.ee.lbl.gov/html/traces.html*, 2008.