

Performance Prediction of Service-Oriented Applications based on an Enterprise Service Bus

Yan Liu¹, Ian Gorton², Liming Zhu¹

¹National ICT Australia,

² Pacific Northwest National Laboratory

Jenny.liu@nicta.com.au; liming.zhu@nicta.com.au; ian.gorton@pnl.gov

Abstract

An Enterprise Service Bus (ESB) is a standards-based integration platform that combines messaging, web services, data transformation, and intelligent routing in a highly distributed environment. The ESB has been adopted as a key component of SOA infrastructures. For SOA implementations with large number of users, services, or traffic, maintaining the necessary performance levels of applications integrated using an ESB presents a substantial challenge, both to the architects who design the infrastructure as well as to IT professionals who are responsible for administration. In this paper, we develop a performance model for analyzing and predicting the runtime performance of service applications composed on a COTS ESB platform. Our approach utilizes benchmarking techniques to measure primitive performance overheads of service routing activities in the ESB. The performance characteristics of the ESB and services running on the ESB are modeled in a queuing network, which facilitates the performance prediction of service oriented applications. This model is validated by an example ESB based service application modeled from real world loan broking business application.

1. Introduction

Service Oriented Architecture (SOA) is a technology as well as a paradigm of designing a software system to provide services to either end-user applications or to other services distributed in a network. SOA raises the level of abstraction by masking the complexity of underlying technology implementations while presenting composable services to users. A well constructed, standards-based SOA can empower a business environment with a flexible infrastructure and processing environment.

In SOA, a service encapsulates reusable business functionalities with platform-independent interface contracts. A service can be dynamically located and invoked. The communication between services can be simply point-to-point, and also multi-point to multi-point in a complex SOA. This means not only that the service interfaces be specified according to SOA principles, but

also that the invocations to services are independent of the service location and the communication protocol involved. This requires a service routing capability, which is amongst the many capabilities of the Enterprise Service Bus (ESB) including communication, integration, security and message processing [11]. An ESB is a standards-based integration platform that combines messaging, web services, data transformation, and intelligent routing in a highly distributed environment. An ESB enables SOA by supporting services, messages and event-based interactions in a heterogeneous environment.

The performance of an SOA-based application is often critically important to a business. If the SOA doesn't perform well, then the applications that leverage the composition capability of SOA have limited benefit. In SOAs, service applications can be considered as a composition of individual services. Each individual service exposed is implemented by component hosted in some node in the distributed environment. It can be seen that the SOA performance problem falls into two broad categories: ensuring sufficient performance of individual services as well as of the composite services [6]. Individual services provide service interfaces that encapsulate existing systems, ensuring their performance necessitates managing the performance of the components, applications, and systems that lie beneath the services abstraction. Well-established capacity planning methods, techniques and tools can be leveraged to manage the performance of individual services, such as logging-based instrumentation [3], or simulating the load on service interfaces by load testing in a similar way in simulating traditional web application performance [1].

Dealing with the performance of services integrated by ESBs falls into the second category of SOA performance issues, and it is far more complex than atomic services. Services registered to an ESB can be uniquely identified by their endpoints. An ESB composes services in a loosely coupled way by routing and transforming messages among services with different protocols. Some services expect high volumes of traffic from a specific consuming application, while others expect high reuse, which implies that a service is used by different

consumers. This makes the workload characterization of composite services complicated.

Moreover, although ESBs mask the complexities of composing applications and services, they also introduce performance overheads incurred by message brokering and transformation between heterogeneous services and applications. Therefore, analyzing and predicting the performance of ESB-based applications requires both the understanding of atomic services and the performance characteristics of ESBs. One challenging issue is that these performance characteristics of ESBs are hard to measure by instrumentation or monitoring due to the nature of highly distributed architecture in SOA. This necessitates a systematic approach that facilitates estimating performance characteristics of ESBs and analyzing the performance relationship between the ESB and the services it composes.

In this paper we present a combined performance modeling and benchmark approach to address the above problems. A queueing network performance model is devised to represent the performance critical ESB infrastructure components and the services running on the ESB. The performance characteristics of these ESB components are estimated by benchmarking results.

The structure of this paper is as follows. Section 2 introduces a conceptual architecture of an ESB and the key components involved in this architecture. The performance modeling issues are identified at the architecture level. Section 3 presents an ESB-based application modeled from a real world loan processing application. Section 4 describes our modeling-based approach to analyzing and predicting the performance of the loan application. Section 6 discusses the related work. This paper concludes at section 7.

2. Understanding the ESB architecture

In order to model the performance of an ESB, we first need to understand the architecture of the ESB and identify key components of the ESB that have critical performance impact.

The capabilities of an ESB are implemented by middleware technologies such as web services, message brokering, security management and so on. Despite the diversities in ESB implementation, there are common components in the architecture of an ESB. An abstract architecture of an ESB is illustrated in Figure 1.

The core functionality of an ESB is supporting disparate applications and services to communicate using different protocols as shown in the protocol stack of Figure 1. The ESB message brokering provides content-based routing for dispatching a request message from one service to another service or application registered on the ESB. A service or application is connected to the service bus through an ESB ‘gateway’.

A gateway (also called a proxy in some ESB implementations) can be configured with a routing table to route a request to the appropriate business service. The routing table entry defines a set of actions, including routing to a service, call out a service, composing messages with input parameters for invoking services, transforming output parameters to response messages, and validating the message content. At runtime the routing table guides the message flow. Each action is executed by checking the predefined expression with variables, operators and variable values associated with an action. The gateway also marshals a request from the message bus to the service.

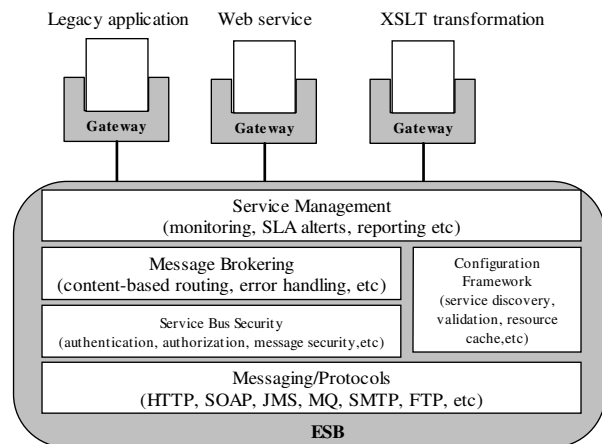


Figure 1 High level ESB architecture

Figure 2 shows an example using content-based routing to connect an application to two services depending on some condition configured in the routing table encapsulated in *RouteNode1*. This routing configuration is attached to the ESB gateway *LoanGateway1*. An ESB implementation normally provides comprehensive services and utilities such as service monitoring, reporting and message security. Details of these services can be found from individual ESB implementations [12][13].

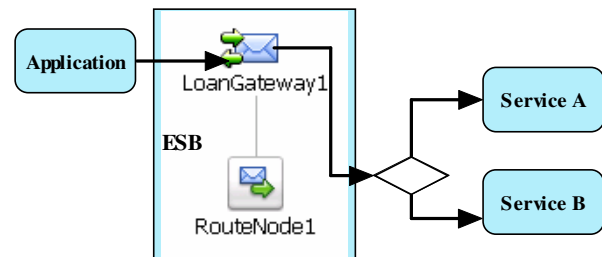


Figure 2 Example of ESB content-based routing

Based on the above understanding, we can see that the end-to-end responsiveness of a service request depends on the performance characteristics of the ESB routing and the destination services. As we discussed in section 1, the performance characteristics of individual services can be

obtained as they are atomic services. If these services are also composed then the activity of obtaining their performance characteristics becomes recursive.

The performance characteristics of an ESB are mainly determined by its capacity for content-based routing and message transformation. Extra ESB infrastructure functions such as message flow monitoring and message security management can also incur additional overhead. In this paper, we focus on the performance overhead of ESB routing and transformation. This can simplify the performance analysis approach and establish a base-line model, which can be further extended to incorporate more complex ESB features such as monitoring and security management. In the rest of this paper, we propose a modeling-based approach that maps the ESB architecture to an analytical model that is solved to analyze and predict the performance of the ESB-based loan processing application.

3. A real world example of ESB-based loan application

In this paper, we use an ESB integrated application modeled from a real world loan processing application as an example to illustrate our modeling approach. We consider this loan application because its scenarios are representative. They illustrate the basic requirements for SOA design and integration technologies such as Web Services and ESB. The architecture design is discussed in detail in [7] (see chapter 9). Different versions of the loan application were implemented on several ESB platforms including Mule [12] and BEA AquaLogic Service Bus (ASB) [13]. Therefore, analyzing the performance of this loan application can help us to identify research issues involved in performance modeling of ESB-based applications, and show insight in to their solutions.

In this paper we use the COTS BEA ASB implementation of the loan application. The basic business logic is that a primary mortgage company uses BEA ASB to route loan applications to appropriate business services. It includes three scenarios shown in Figure 3:

1. The messages for every loan application through the BEA ASB are validated (**loanValidation**). If the application is:
 - Incomplete, it is written to an error directory and an error message is returned to the client.
 - Complete, it is routed to an appropriate business service for review.
 - Approved, the service returns a message indicating whether the loan is accepted or rejected.
2. An application containing a request for a rate less than 5% requires management approval and is routed to an appropriate business service (**managerLoan-**

ReviewService) for processing. All other loan applications are routed to another business service (**NormalLoan**) for processing.

3. Loan applications with a principal request of greater than US\$ 25 million are candidates for sale to a secondary loan company. The applicant's credit rating information is retrieved by making a callout to a Web Service (**CreditRatingService**). The credit rating information is added to the loan application, then the application is forwarded to the secondary mortgage company Web Service to be processed (**LoanSaleProcessor**). Loan applications with a principal request equal to or less than US\$ 25 million are routed to the **NormalLoan** business service for processing.

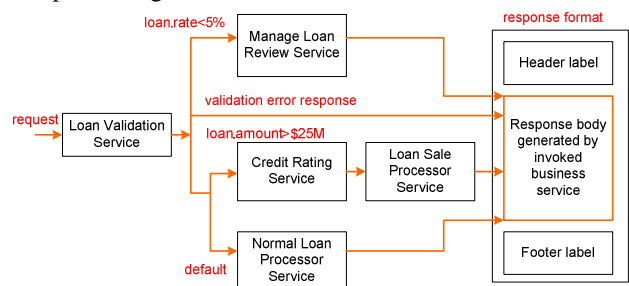


Figure 3 Scenarios of load application

In the ASB's implementation of this application, each scenario has a gateway in the ESB (called a proxy service in BEA ASB) for routing the loan request according to different conditions. The interactions between the loan application, business services and proxy services are shown in the sequence diagram of Figure 4.

For performance testing, the loan application is deployed on three workstations: one for the client loan application; one for the ASB server and one for hosting all the business services. These three workstations are identical with Dual Intel Xeon 3.00GHz CPUs and with 3G RAM, running Windows XP.

4. Performance modeling approach

The overall performance modeling approach follows the general capacity planning process [1]. This includes:

- Mapping the application components or services at the software architecture level to analytical model elements
- Characterizing the workload pattern for the individual components as input for the performance model
- Calibrating the performance model by populating the parameter values
- Validating the performance model and predicting the performance

We describe our modeling approach by illustrating each step in the context of the loan application using the

ESB. This approach is general and hence can be applied to other ESB-based applications.

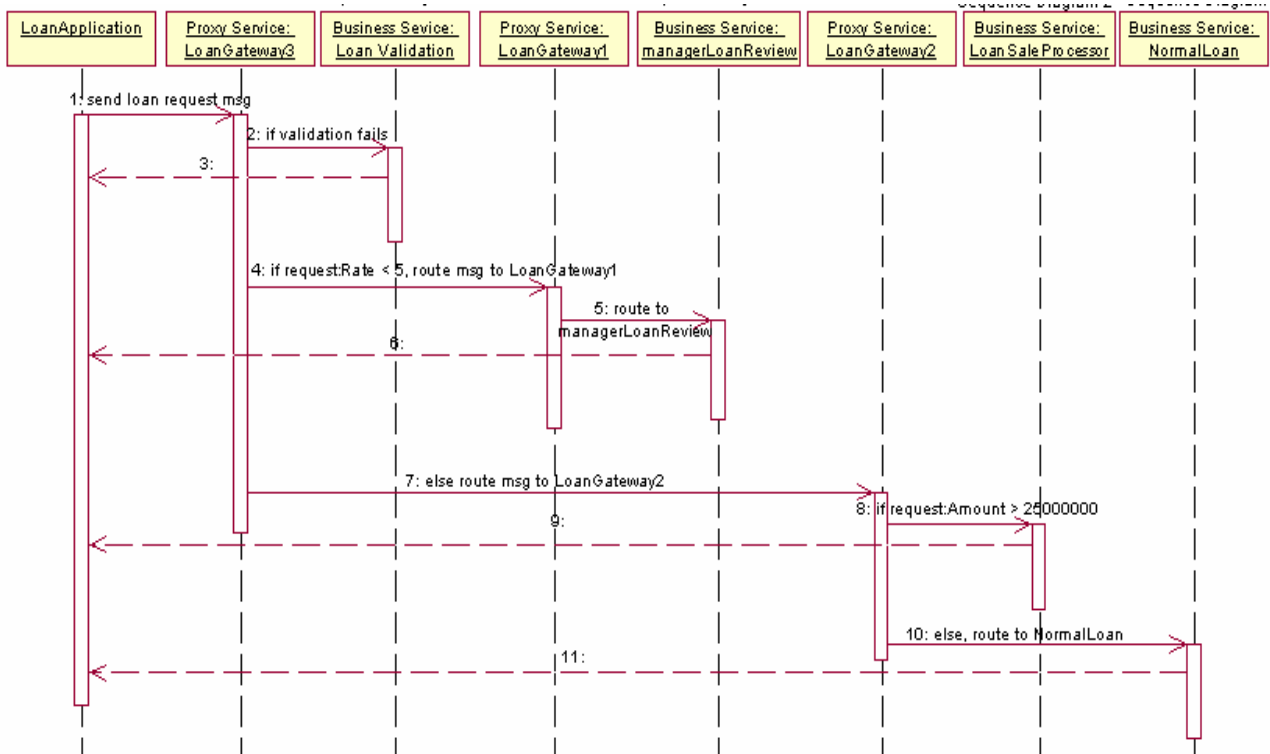


Figure 4 Interactions of the load application

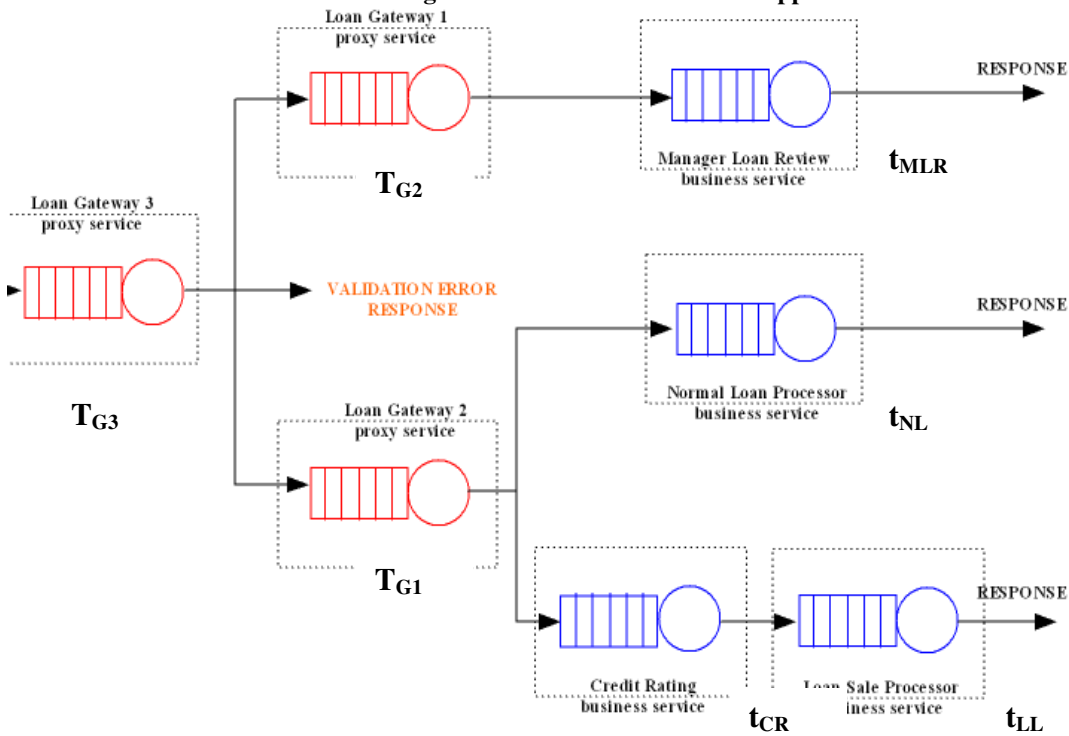


Figure 5 Queuing network model of the loan application

4.1 The performance model

In the interaction diagram of the loan application (see Figure 4), there are three proxy services for each scenario, and five business services, namely *loanValidation*, *manageLoanReview*, *normalLoan*, *creditRating* and *loanSaleProcess*. Note that the validation of messages is simple and executed by a proxy service without any business service involved in this scenario. These proxy and business services form a queuing network as illustrated in Figure 5.

In this model each service is mapped to a load-independent resource (represented by a queue notation in Figure 5) that serves arriving requests in the analytical queuing network model. The assumption that each service is load-independent simplifies the modeling complexity as well as reduces the effort of collecting parameter values to represent the load-dependent behavior of a service. This assumption is later validated by the accuracy of the modeling results in this section.

The connections between proxy services and business services can be of different types of networks, such as dedicated network channels, Ethernet, or WAN. There can be firewalls and also load balancing proxies installed along the requests invocation paths of business services. The service delay introduced by network protocols and software can be represented by a system level model (see Chapter 8 in [10]) and combined into the performance model in Figure 5 by applying component level modeling methods (see Chapter 9 in [10]). Details of modeling network protocols and software are out of the scope of this paper.

4.2 Workload characterization

The overall throughput of an ESB depends on the workload imposed to it. The ASB platform is shared by the three scenarios in this load application, and their usage patterns of the ESB are different from each other as discussed in section 3. Overall four classes of workload are identified based on the message flow branches, namely loan validation (VAL), manager loan review (MLR), large loan (LL) and normal loan (NL).

In the queuing network model, the mode that workload arrives at the system determines if the system is modeled as a closed or an open model. For a closed model, new job/request arrivals are only triggered by job completions (followed by think time), as in Figure 6 (a). By contrast in an open model, new jobs/requests arrive independently of job completions, as in Figure 6 (b). Schroeder et al demonstrated that closed and open system models yield significantly different results, even when both models are run with the same workload and service demands [1]. This means determining the type of the model as closed or open is critical to the accuracy of the performance modelling results.

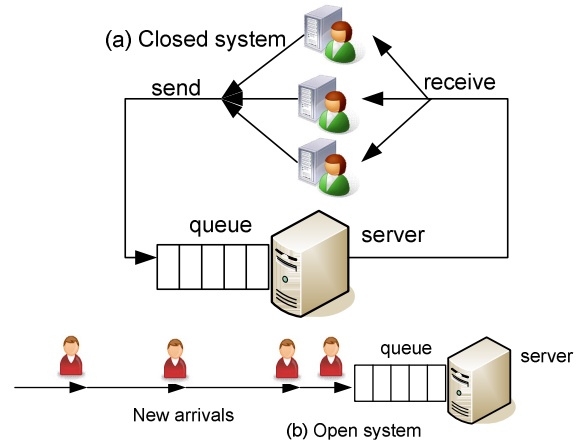


Figure 6 Closed vs open system

By the nature of the loan application, the requests arrive at the ASB independently of job completions, and we can consider modeling the system as an open model. In order to model and validate the system in an open model, we need to generate workload in the open model accordingly. However, most available and popular web-based workload generators assume a closed system model when generating workload, including TPC-W, RUBis, Microsoft Web Application Stress Tool, WESBtone, SPECJ2EE and many others [1]. For these tools, a fixed number of concurrent clients are specified and requests are generated from each client. A new request starts after the previous invocation completes with a configurable time interval between two consecutive requests.

This issue hence raises the engineering problem of how to model an open system with only workload generators available for closed systems. Of course, one can implement a workload generator that generates open workload with a certain distribution of the job arrival rate. However, this requires extra software development effort and it is non-trivial to have a fully fledged workload generator with functions for collecting metrics.

Our solution to this problem applies closed and open models in two stages. At the stage of performance modeling and validating, we use the available closed system workload generator and model the system as a closed model under a workload represented as the number of requests. Using this closed model, we can validate the assumption made on the type of resources and the accuracy of input parameter values estimated. Then at the stage of predicting performance, we replace the model as an open one with the workload presented as arrival rates. In this open model, the type of resources and the parameter values remain the same as they are in the closed model.

4.3 Model calibration

The performance model in Figure 5 is solved using the mean value analysis algorithm for multi-class closed systems [10](see chapter 9). One of the key inputs of this algorithm is the service demand of each resource in the queuing network model. In this case a resource or a queue in the queuing network model represents either a proxy service or a business service. Therefore the service demand is the time spent on processing a request by the service.

The service demand of a business service can be measured individually by the end-to-end delay of a request. However, proxy services can not be measured in this way, as a proxy service is embedded in the ASB and it is only executed when routing is triggered. It is also not practical to instrument the ASB, as this would effect actual performance.

The solution for estimating the service demand of proxy services is described as follows. We can measure the end-to-end delay of a message flow between the Time of The First Bit (TTFB) of the request sent and the Time of The Last Bit of (TTLB) of the response received, which is $t = TTFB - TTLB$. One message flow can consist of several proxy and business services. The message flow can be manipulated by changing the request input parameters to trigger different branches of the routing condition or by changing the routing table itself.

We estimate the service demand of a proxy service by solving linear equations combining a set of delay measurements of different message flow branches. An example is shown in Figure 7 for estimating the service demand of proxy service *LoanGateway1* and *LoanGateway3*.

$$\begin{cases} T_{G3} + T_{G1} + t_{MLR} = t_1 & (1) \\ T_{G3} + t_{MLR} = t_2 & (2) \end{cases}$$

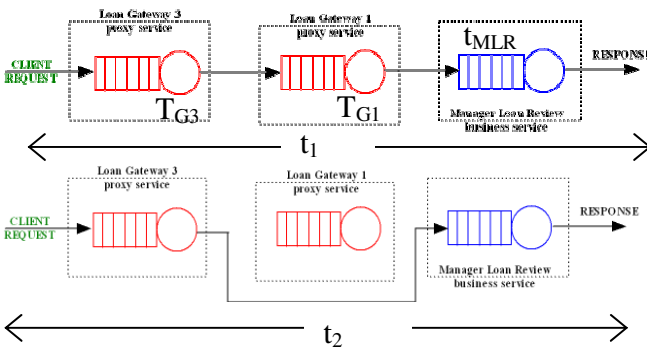


Figure 7 Estimating the service demand of proxy services

The time t_1 , t_2 and t_{MLR} are measured using the Microsoft Web Application Stress Tool. Metrics including the overall throughput, the total number of hits, TTFB and TTLB are collected by this tool. Similarly we can obtain

the service demands of the rest of the proxy services as shown in Table 1.

Service name	Service type	Service demand (ms)
LoanGateway1	Proxy	2.76
LoanGateway2	Proxy	3.39
LoanGateway3	Proxy	6.66
managerLoanReview (MLR)	Business	118.31
loanValidation (VAL)	Business	79.14
normalLoan (NL)	Business	121.22
loanSaleProcessor (including credit rating service) (LS)	Business	176.52

Table 1 The service demand of the loan application

4.4 Validation and prediction

The performance model is populated and solved based on the service demands shown in Table 1. The modeled performance data is then compared to the measured results from the Microsoft Web Stress Tool. The error of performance modeling is shown in Table 2. We validate the model with different workloads by setting the number of clients (we only show 100 and 50 clients due to the space limitations) and transaction mix as indicated in the table. The errors of performance modeling are within 7% with the majority of the errors within 5%. The results validate the mapping between services and queuing network model elements, and the estimation of the parameter values.

	Throughput (requests/s)	VAL (18%)	MLR (28%)	LL (18%)	NL (36%)
100 clients	Modeled	12.64	8.45	5.66	8.25
	Measured	12.44	8.23	5.58	8.00
	Error (%)	1.60	2.70	1.55	3.12
50 clients	Modeled	12.63	8.45	5.66	8.24
	Measured	11.91	8.54	5.75	8.66
	Error (%)	6.08	1.04	1.56	4.75
100 clients	Throughput (requests/s)	VAL (18%)	MLR (18%)	LL (54%)	NL (9%)
	Modeled	13.13	8.67	5.76	9
	Measured	12.44	8.23	5.58	8.45
Error (%)	5.56	5.33	3.30	5.63	

Table 2 Error of performance modeling

Predicting the performance involves solving the performance model as an open system with the workload characterized as the request arrival rate. For a resource in an open system, which represents the proxy or business service in this loan application, if the arrival rate exceeds the service rate of the resource, then the resource is saturated and response time and requests queued increase significantly.

Using the model, we can analyze and predict the maximum arrival rate that a service can handle before it degrades performance. For example, we can change the transaction mix by the ratio of service invoked. To illustrate this, we configured the workload with high usage of the *normalLoan* and *managerLoanReview* services respectively. We scale the arrival rate from 1 request per second to 25 requests per second and the predicted response time of each class of request is shown in Figure 8. Given the software and hardware capacity, Figure 8 shows 20 requests per second reaches the maximum arrival rates that *normalLoan* and *managerLoanReview* can handle. This result can provide administrators with a quantitative guide for throttling requests when the arrival rate of requests exceeds this level.

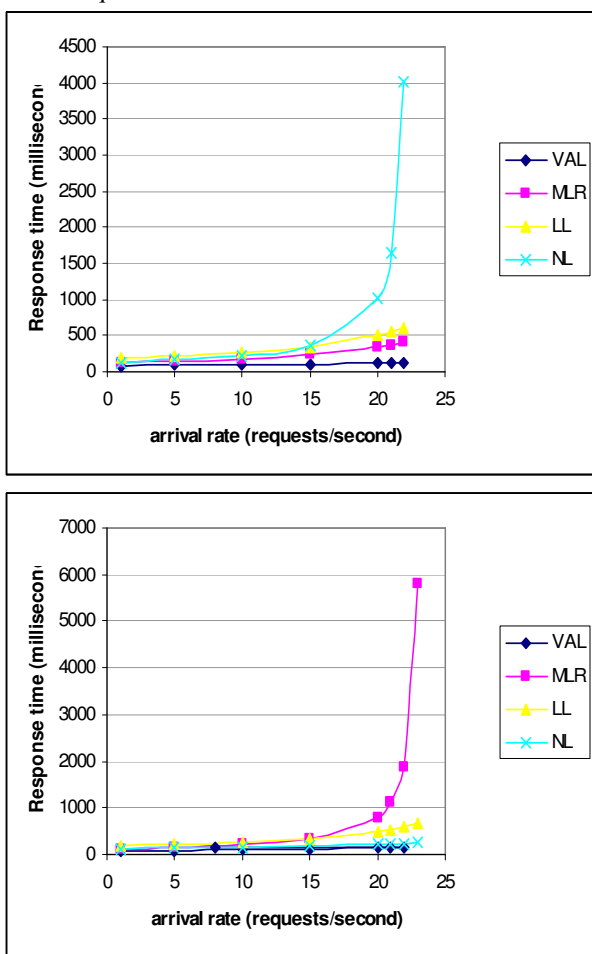


Figure 8 Performance predicted for the open system

5. Discussions on potential extensions

So far we have described our performance modeling approach for ESBs, which focuses on performance characteristics of routing messages to different services. A baseline performance model is derived from this work.

ESBs have more advanced features to facilitate the integration of services, such as message security management. We envision our approach can be extended to model the performance of ESB-based systems with security management enabled.

Figure 9 shows an abstract security model in an ESB. Similar to Web Service security management, security of ESB messages can be controlled at different levels: transport level message encryption, and message level authentication and authorization. The key issue in performance modeling is to identify the critical components involved in enabling this feature. We can see from the abstract security model in Figure 9 that proxy services are responsible for invoking authentication or authorization providers and CAs to obtain the necessary security tokens. One solution is to calibrate the service demand of the proxy services in the baseline model. The extra delay occurred by security management are augmented to the service demand of the proxy services. This value can be obtained by benchmark measurement. Alternatively, the security management can be abstracted and modeled as another resource and combined with the baseline model. Accordingly, the delay of the security management serves as the service demand of this newly introduced resource. It remains our future work to evaluate and compare the accuracy of these two modeling solutions.

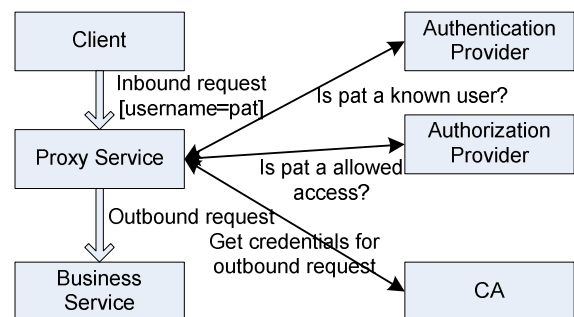


Figure 9 Abstract Security Model in ESB

6. Related Work

Performance of SOAs is mainly studied as Web Service performance, as a web service is one of the key enabling technologies of SOA. Menasce discussed the QoS issues in Web Services [6] and presented how a systematic capacity planning process can be applied to Web Services [10]. [3] applied layered queuing networks to a web service-based health software architecture. Other work focuses on the performance evaluation and analysis of the SOAP protocol in Web Services [4][6][7][8]. To the best of our knowledge, there hasn't been work done in modeling the performance of ESB-based applications.

7. Conclusion

The loan application demonstrates the core features of an ESB, including content-based routing and message transformation. The performance model developed forms a baseline model and it remains our future work to extend this model to analyze other features of an ESB such as the security management.

This case study focuses on the performance modeling method instead of doing performance evaluation of the ESB with stress testing. Therefore, the workload imposed is synthetic. For a real world application, the workload characterization approach described in section 4.2 can be applied. Obtaining the parameter values to solve the model requires the measurement of the delay incurred by individual ESB infrastructure gateway/proxy service. This can be achieved using performance testing tools and linear regression.

One contribution of this paper is that it combines both closed system and open system analysis method at the model validation and prediction stage. Simplified assumptions have been made to reduce the engineering effort for performance measurements. These assumptions are validated by empirical results, otherwise inappropriate assumptions need to be revised and revalidated. The case study using a loan application validates our approach and shows that it is practical for COTS ESBs. The experience gained from this work provides us with insightful understanding of performance analysis issues involved in ESB enable applications.

In the future, this approach will be further validated to different ESB implementations and applications integrated through different communication protocols by ESBs .

8. Acknowledgements

National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

9. Reference

- [1] Almeida, V. A. and Menascé, D. A. 2002. Capacity Planning: An Essential Tool for Managing Web Services. *IT Professional* 4, 4 (Jul. 2002), 33-38. DOI= <http://dx.doi.org/10.1109/MITP.2002.1046642>
- [2] Bianca Schroeder, Adam Wierman and Mor Harchol-Balter. Open vs closed: a cautionary tale, *Networked System Design and Implementation NSDI*, 2006.
- [3] Carolyn McGregor, Josef Schiefer, "A Framework for Analyzing and Measuring Business Performance with Web Services," *cec*, p. 405, 2003 IEEE International Conference on E-Commerce Technology (CEC'03), 2003.
- [4] Chen, S., Yan, B., Zic, J., Liu, R., and Ng, A. 2006. Evaluation and Modeling of Web Services Performance. In *Proceedings of the IEEE international Conference on Web Services (Icws'06) - Volume 00* (September 18 - 22, 2006). ICWS. IEEE Computer Society, Washington, DC, 437-444. DOI= <http://dx.doi.org/10.1109/ICWS.2006.59>
- [5] Christina Catley, Dorina C. Petriu, Monique Frize, Software Performance Engineering of a Web service-based Clinical Decision Support infrastructure. In *Proceedings of the 4th international Workshop on Software and Performance* (Redwood Shores, California, January 14 - 16, 2004). WOSP '04. ACM Press, New York, NY, 130-138. DOI= <http://doi.acm.org/10.1145/974044.974066>
- [6] Davis, D. and Parashar, M. Latency Performance of SOAP Implementations. In *Proceedings of the IEEE Cluster Computing and the GRID 2002 (CCGRID'02)*, Berlin, Germany, IEEE, 2002
- [7] Gregor Hohpe, Bobby Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional (October 10, 2003), ISBN-13: 978-0321200686
- [8] Kohlhoff, C. and Steele, R. Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems. In *Proceedings of the WWW2003*, Budapest, Hungary, 2003
- [9] Menascé, D. A. 2002. QoS Issues in Web Services. *IEEE Internet Computing* 6, 6 (Nov. 2002), 72-75. DOI= <http://dx.doi.org/10.1109/MIC.2002.1067740>
- [10] Menascé, D. A and Almeida, V. A., *Capacity Planning for Web Services: metrics, models and methods*, Prentice Hall, 2001, ISBN 0-13-065903-7.
- [11] Martin Keen, et al. *Patterns: Implementing an SOA Using an Enterprise Service Bus*, IBM redbook, 2004, ISBN 0738490008. <http://www.redbooks.ibm.com/abstracts/SG246346.html>
- [12] [http://mule.mulesource.org/wiki/display/MULE/Loan Broker](http://mule.mulesource.org/wiki/display/MULE/Loan+Broker)
- [13] <http://edocs.bea.com/alsb/docs25/interm/examples.html#tutorial>