

# Accuracy of Performance Prediction for EJB Applications: A statistical analysis

Yan Liu, Ian Gorton

National ICT Australia,  
1430, NSW, Australia  
{jenny.liu,ian.gorton}@nicta.com.au

**Abstract.** A challenging software engineering problem is the design and implementation of component-based (CB) applications that can meet specified performance requirements. Our PPCB approach has been developed to facilitate performance prediction of CB applications built using black-box component infrastructures such as J2EE. Such deployment scenarios are problematic for traditional performance modeling approaches, which typically focus on modeling application component performance and neglect the complex influence of the specific component technology that hosts the application. In this paper, an overview of the PPCB modeling approach is given. Example results from predicting the performance of a J2EE application are presented. These results are then statistically analyzed to quantify the uncertainty in the predicted results. The contribution of the paper is the presentation of concrete measures of the confidence an architect can have in the performance predictions produced by the PPCB.

## 1 Introduction

Distributed component-based technologies such as the Java 2 Enterprise Edition (J2EE) and .NET have become important infrastructure technologies for building multi-tier applications. The overall performance of such component-based applications depends on a number of factors. These include the implementation of the supporting component container, the architectural decisions taken in the design of components, application-specific deployment configurations, and the specific application client behavior [6]. It is consequently challenging for software architects to design a system with a priori confidence that it will perform well enough to meet its requirements.

Consequently, architects are forced to develop prototypes to evaluate the performance of an application design [2,6,11]. For complex applications, this can be time-consuming and expensive. We believe that the process of predicting the performance of a component-based system based on an architecture-level design could significantly reduce the engineering costs and risks of a deployed system failing to meet performance requirements.

In related research, performance modeling has proved a useful approach [1, 3,4,14,17,20,24]. A performance model can represent the underlying architecture as

well as application behavior in terms of its performance characteristics. A common practice is to build a prototype and use this to obtain measures for the values of parameters in the model [12]. However, for a complex application, this is expensive and time-consuming.

Therefore, we propose using *benchmarking* to overcome these difficulties. Benchmarking is the process of running a specific program or workload on a machine or system and measuring the resulting performance [21]. In PPCB (Performance Prediction of Component Based systems), benchmarking is used to provide values for certain parameters in the performance model that is used for prediction. However, the abstraction inherent in the performance model and approximations in the benchmarking measurements introduce uncertainty in to the resulting predictions.

The major contribution of this paper beyond that of [15] is the use of statistical methods to analyze the predicted performance for an application. This provides statistical evidence of the accuracy of the results of the PPCB. The results of the analysis reveal a high level of accuracy in the predictions. To the best of our knowledge, this is the first time such high levels of confidence in performance predictions of applications executing on black-box based component infrastructures have been published.

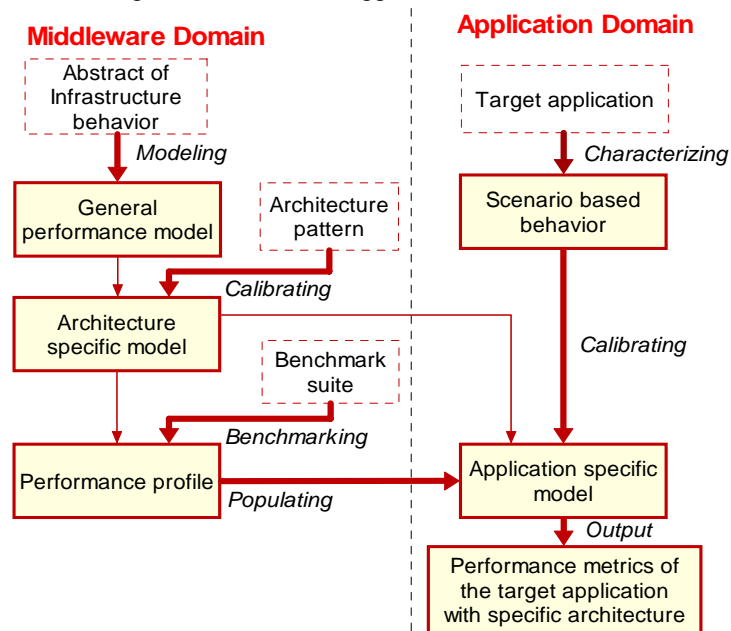
The structure of this paper is as follows. Section 2 gives an overview of the PPCB approach. Section 3 presents the example application on which we conduct performance prediction using PPCB, along with some sample performance prediction results. Section 4 details the statistical analysis, and Section 5 discusses related work.

## 2 PPCB Overview

The essence of our framework shown in Figure 1 is combining performance modeling and benchmarking techniques. This enables performance prediction at the design level of software applications that are based on specific component technology. A comprehensive description of the performance model is beyond the scope of this paper, and can be found in [15,16]. Given a component technology, such as an implementation of Enterprise JavaBeans (EJB), the approach has the following steps:

1. **Modeling.** We establish a general model  $P$  for the chosen technology, by identifying the main components of the system, and noting where queuing delays occur. This abstracts details of the infrastructure components and their communication.
2. **Calibrating.** The model has to be calibrated for a specific architecture before it can be used to predict performance, so we must develop the function  $f^A$ , which is the function used to calibrate the generic performance model  $P$  to specific architecture  $A$ . An architectural choice can be mapped to a set of infrastructure components and their communication pattern [4]. The operations of service components can be further aggregated into computing modules. Calibrating the performance model means deriving mathematical models with parameters characterizing those computing modules.

3. **Characterizing.** The purpose of characterizing an application is to determine the load that an application places on the underlying component infrastructure when the application takes the form of architecture *A*. For a given application, we can determine how often each component is executed. This depends on their business logic, which tells us how often methods are called, and what operations are performed by which computing modules.
4. **Benchmarking.** The above produces a performance prediction for the designed system in the form of an equation with parameters. Some of the parameters represent observable or tunable features of the configuration, but other parameters reflect internal details of the black-box middleware platform that hosts the application components. We therefore implement a simple application, with minimal business logic and a simple architecture, on the target middleware platform, and measure its performance. Solving the performance model corresponding to the simple application allows us to determine the required parameter values, which we describe as the performance profile of the platform.
5. **Populating.** The parameters of the middleware platform profile can be substituted into the performance model of the designed application, giving the required quantitative prediction of performance of that application.



**Fig. 1.** The performance prediction framework

The PPCB approach provides a solution to overcome the difficulties in populating the performance profile of component infrastructure, or middleware. Instead of prototyping the system and measuring it, the explicit parameter values of the performance

model are obtained by benchmarking a simple application. These benchmark results can then be applied to any applications that execute of the benchmarked middleware platform.

### 3 Example Results

The PPCB approach has been applied to predict the performance of a J2EE application, *Stock-Online*. In this section, we briefly describe the benchmark design and how benchmarking can be integrated with the performance model of an infrastructure to predict the overall performance of an EJB-based application. The predicted results are summarized and they are used in the next section for statistical validation.

#### 3.1 Predicting Stock-Online performance

Stock-Online [5] is a simulation of an on-line stock-broking system. It supports six business transactions and enables users to buy and sell stocks, inquire about the up-to-date prices of particular stocks, and get a holding statement detailing the stocks they currently own. There are four database tables to store details for accounts, stock items, holdings and transaction history.

We have implemented Stock-Online with EJB components. Three distinct implementations have been created that employ very different component architectures. These are:

1. One architecture solution uses Container Managed Persistence (CMP) entity beans, applying the standard EJB design pattern of a session bean as a façade to entity beans. A single session bean implements all transaction methods. Four entity beans, one each for the database tables, manage the database access. Transactions are container managed. We refer this architecture as CMP.
2. The second architecture optimizes the access mechanism to persistent data in the CMP architecture for business scenarios with intensive read-only operations. This architecture is implemented using the Read-Mostly (RM) EJB design pattern [19]. Read-only and read-write operations are separated into two entity beans, which are mapped to the same database data. Read-only operations have direct access to cached data inside the container, thus reducing the overhead of access to the database. The synchronization of cache data and persistent data is managed by the container.
3. The third architecture leverages an Optimistic Concurrency Control [13] (OCC) algorithm. A container that supports OCC does not hold a lock for any persistent data. The ACID transaction properties are managed by the database system. This increases the concurrency of the application when there is no conflict of two simultaneously running transactions.

The deployment environment for each of these solutions is identical. It consists of a commercial J2EE application server as the container for Stock-Online and a commercial relational database for persistence. The clients, J2EE container and database

each execute on separate machines. The client requests are from web server hosted components under a full, sustained request load. Given this scenario, it is desirable for an architect to determine the level of performance that the system can provide under load without building its solution.

Basically, we have developed a queuing network model of the J2EE application server infrastructure and calibrated it for the three different component architectures. The approach to characterize an application behavior from scenarios is developed and presented in [15,16]. This produces a performance prediction for the designed system in the form of an equation with parameters, which we describe as the performance profile of the platform. These parameters capture the performance characteristics of the internal behavior of an EJB container. The parameters and the descriptions are listed in Table 1 below. Importantly, the performance profile and its value are obtained by benchmarking without access to the implementation and deployment of Stock-Online.

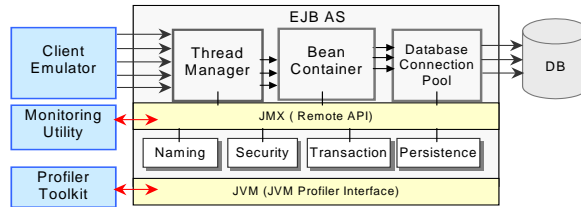
$T_{SINIT}$	The service time of the container's initialization process
$T_s$	The service time of a session bean's operation, which doesn't include the time waiting for replies from nested beans' operations
$T_o$	$T_o = T_{SINIT} + T_s$
$T_1$	The service time for the container to access the entity data in its cache
$T_2$	The service time of the container to active/passivate an entity bean instance to secondary storage
$T_{create}$	The service time of the container to create an entity bean object
$T_{remove}$	The service time of the container to remove an entity bean object
$T_{load}$	The service time to load an entity data into the container
$T_{store}$	The service time to store updates of an entity data
$T_{insert}$	The service time to insert a new record of an entity into the database
$T_{delete}$	The service time to remove a record of an entity from the database

**Table 1.** Performance profile of an EJB container

### 3.2 Benchmark Design

Component technologies leverage many standard services (e.g. security, transactions) to support application development. The benchmark scenario is thus designed to exercise the key elements of a component infrastructure involved in the application execution.

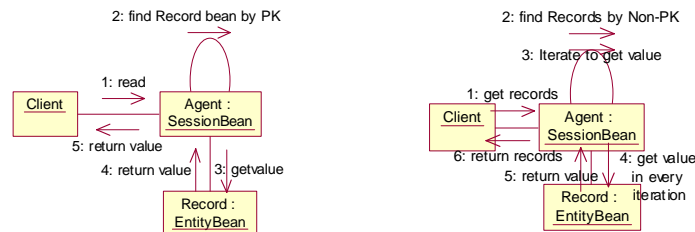
We have designed and implemented a benchmark suite for modeling the performance of EJB-based applications. The benchmark suite consists of four modules, namely a workload generator, benchmark application, monitoring utility and profiling toolkit in Figure 2.



**Fig. 2.** A benchmark suite for EJB technologies

The benchmark clients simulate active requests from proxy applications, such as servlets executing in a web server. Under heavy workloads, this kind of proxy client has an ignorable interval between two successive requests<sup>1</sup>. Its population in a steady state is consequently bounded<sup>1</sup>. Hence the benchmark client spawns a fixed number of threads for each test. Each thread submits a new service request immediately after the results are returned from the previous request to the application server. The ‘thinking time’ of the client is thus effectively zero. The benchmark also uses some utility programs to collect the measurement of black-box metrics, such as response time and throughput.

The implementation of the benchmark application involves a session bean object *Agent* and an entity bean object *Record*. Container managed persistence (CMP) is used for entity beans and transactions are container-managed. The example collaboration diagram in Fig3 shows the benchmark application scenario for *read /write* and *get Records*.



**Fig. 3.** Benchmark application events

A monitoring utility is implemented using the Java Management Extensions (JMX) API. It collects performance metrics for the application server and the EJB container at runtime, for example the number of active server threads, active database connections and the hit ratio of the entity bean cache.

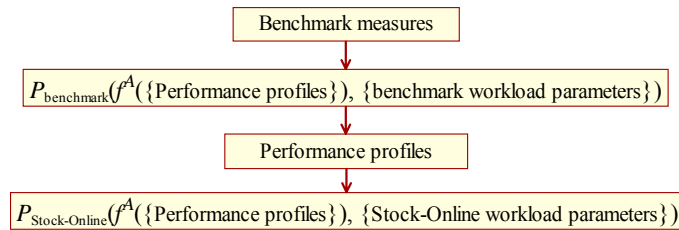
A profiling toolkit *OptimizeIt* [18] is also employed. *OptimizeIt* obtains profiling data from the Java virtual machine, and helps in tracing the execution path and collecting statistics such as the percentage of time spent on a method invocation, from

<sup>1</sup> A web server has configuration parameters to limit the active workload. For example, Apache uses *Max-Client* to control the maximum number of workers, thus the concurrent requests to the application server are bounded.

which we can estimate the percentage of time spent on a key subsystems of the J2EE server infrastructure. Profiling tools are necessary for COTS component-based systems, as instrumentation of the source code is not possible.

### 3.3 Predicted Performance

The explicit parameter values for the performance profile are obtained by solving the performance model using the inputs measured from benchmarking. Then the populated performance profile provides inputs for predicting the performance of Stock-Online. This relationship is shown in Figure 4. Detailed solutions are presented in [15, 16].



**Fig. 4.** Dataflow of the Stock-Online performance prediction

In order to assess the accuracy of the model's predictions, we have implemented and measured the performance of Stock-Online for each of the three different implementations (i.e. CMP, RM and OCC). Below, we present two sample sets of results that are used for subsequent analysis:

- Figures 5 and 6 compare actual versus predicted performance of Stock-Online for a single J2EE server configuration (in this case, utilizing 20 threads). The aim is to infer how accurate the future predictions are based on the measured samples. The predicted client response time for the three architecture models under different workload with the same server configuration is shown.
- Figures 7, 8 and 9 compare the predicted and measured optimal response times as the J2EE thread pool setting is varied, under a stable client workload.

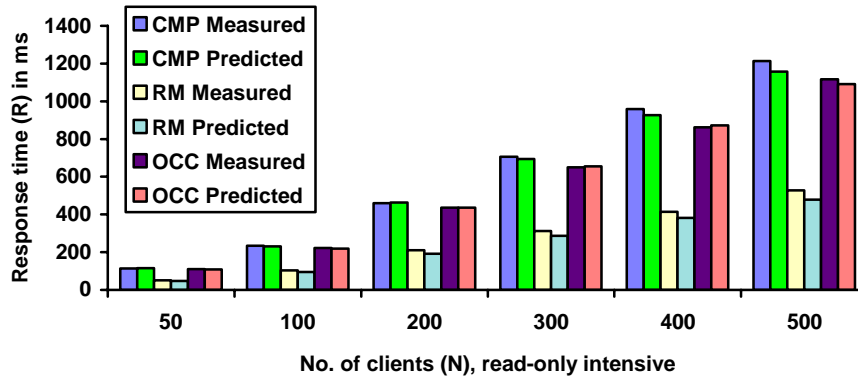


Fig. 5. Stock-Online Performance (*Read-only intensive* business model)

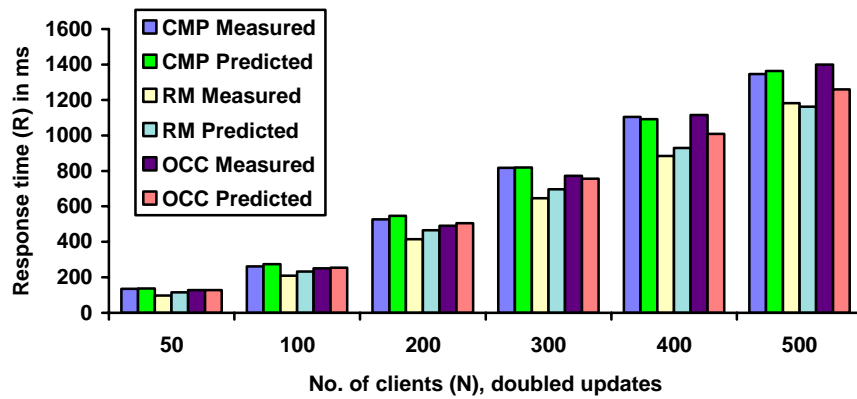


Fig. 6. Stock-Online Performance (*Doubled updates* business model)

## 4. Statistical Analysis

As can be seen in Figure 5 to Figure 9, the predicted performance, while close, is not 100% accurate. Hence we need to assess the effectiveness of the overall approach. Hence, similarly to [9, 10], two appropriate statistical methods are used:

- Statistical intervals for the first data set.
- Linear correlation analysis for the second data set

### 4.1 Statistical Intervals

Statistical intervals can be used to quantify the uncertainty in the sample data [8,23]. We use *tolerance intervals* to estimate the boundary of the prediction error. A toler-

ance interval covers a fixed proportion of the population with a stated confidence [8]. The prediction error is defined as:

$$Error = \left| \frac{Predicted - Measured}{Measured} \right|$$

We calculated the statistical intervals as follows:

Step 1: Use *Shapiro-Wilk* normality test to determine if the original distribution of *Error* is a normal distribution. We use the *shapiro.test* function in the S-Plus[22] library MASS. IF it is normal distribution, then DO Step 3.

Step 2: Transform the original data using the *BoxCox* function. The *BoxCox* function in S-Plus library MASS computes the profile likelihood function for the largest linear model to be considered as a guide in choosing a value for  $\lambda$ , which will then remain fixed [22].

$$y^{(\lambda)} = \begin{cases} (y^\lambda - 1)/\lambda & \lambda \neq 0 \\ \log y & \lambda = 0 \end{cases} \quad (1)$$

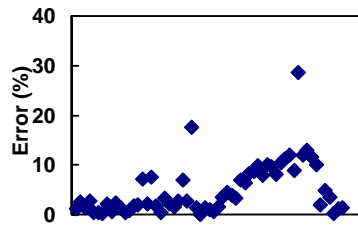
Step 3: The statistical software *SInt* [8] is used to calculate the tolerance interval for the (transformed) normal distribution of *Error*. As we are interested in the upper bound of the *Error* given a confidence level, only a one-side tolerance interval is considered.

The statistical intervals of prediction error can be also calculated as distribution-free intervals. [8] shows that a distribution-free interval (if one exists) will generally be longer than a corresponding interval based on a particular distribution.

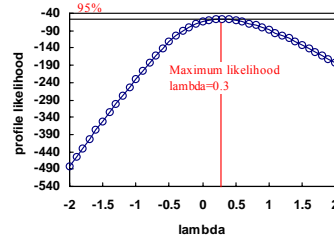
The original statistical error metrics are listed in Table 2, 3 4 for the three architecture models CMP, RM and OCC respectively. The statistical results, for example the statistical intervals of the CMP model in Table2, can be interpreted as:

*90% of CMP model prediction error will not exceed roughly 14.75% and we have 95% confidence level that this upper bound is correct.*

This gives a concrete measure of the confidence an architect can have in the predictions produced by the performance modeling and benchmarking approaches we have developed for black box component-based applications. The statistical intervals show that the accuracy of the RM and OCC models are a little lower than the CMP model. One reason is that both the RM and OCC models do not cover the overhead of invalidating an entity bean cache element involved in conflicting transactions. This is mainly due to a technical limitation. These parameters depends on the internal implementation of the EJB container, and currently the monitoring and profiling tool can not identify the operations involved in these functions and their associated performance metrics. According to [17] however, a prediction error under 30% is still acceptable for capacity planning of a system.



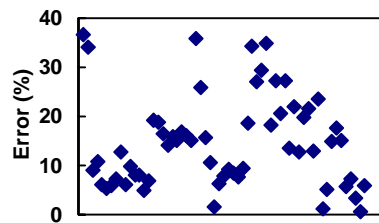
Original distribution of *Error*



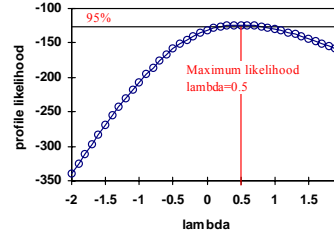
*boxcox* outputs

Statistical intervals	Value
Sample size	$N=61$
Sample mean	$\bar{x} = 4.97$
Sample standard deviation	$s = 5.19$
Percentile of the population	$p = 90\%$
Confidence level	$\gamma = 0.95$
Upper-bound of Error	UB = 14.75

**Table 2.** Statistical intervals for the CMP model



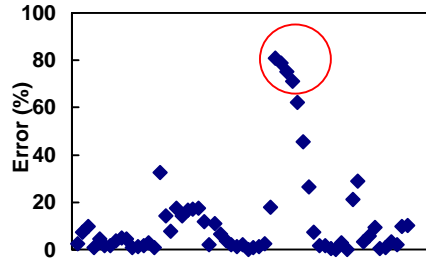
Original distribution of *Error*



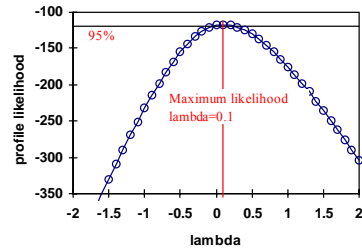
*boxcox* outputs

Statistical intervals	Value	
	Sample size	$N=61$
Sample mean	$\bar{x} = 14.72$	$\bar{x} = 14.72$
Sample standard deviation	$s = 9.31$	$s = 9.31$
Percentile of the population	$p = 80\%$	$p = 50\%$
Confidence level	$\gamma = 0.95$	$\gamma = 0.95$
Upper-bound of Error	UB = 25.15	UB = 15.21

**Table 3.** Statistical intervals for the RM model



Original distribution of  $Error^2$



boxcox outputs

Statistical intervals	Value	
Sample size	$N=61$	$N=61$
Sample mean	$\bar{x} = 13.05$	$\bar{x} = 13.05$
Sample standard deviation	$s = 20.40$	$s = 20.40$
Percentile of the population	$p = 80\%$	$p = 68\%$
Confidence level	$\gamma = 0.95$	$\gamma = 0.95$
Upper-bound of Error	UB = 26.16	UB = 14.66

Table 4. Statistical intervals for the OCC model

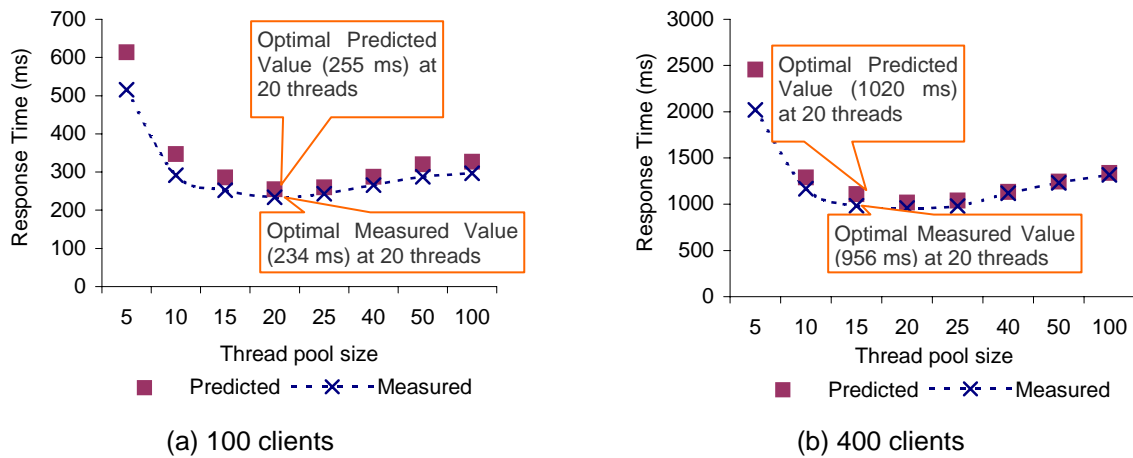
## 4.2 Linear Correlation

Linear correlation analysis can measure the strength of the linear relationship between two variables. The thread pool size for an EJB container is an important tuning option. Our model can be used to predict the performance under different settings of the thread pool size, and consequently be used to find the optimal value with the best performance under a given workload. The linear relationship between predicted and measured response time is assessed to indicate the accuracy of the predicted optimal value. The calculation is shown in Table 5. The correlation coefficients of the three models indicate a good linear relationship between predicted and measured response time under the various settings of thread pool size. The statistical results can be interpreted as, for example in the CMP model, approximately 98.37% of the variation in the values of predicted response time is accounted for by a linear relationship with measured response time and the confidence level is 95%.

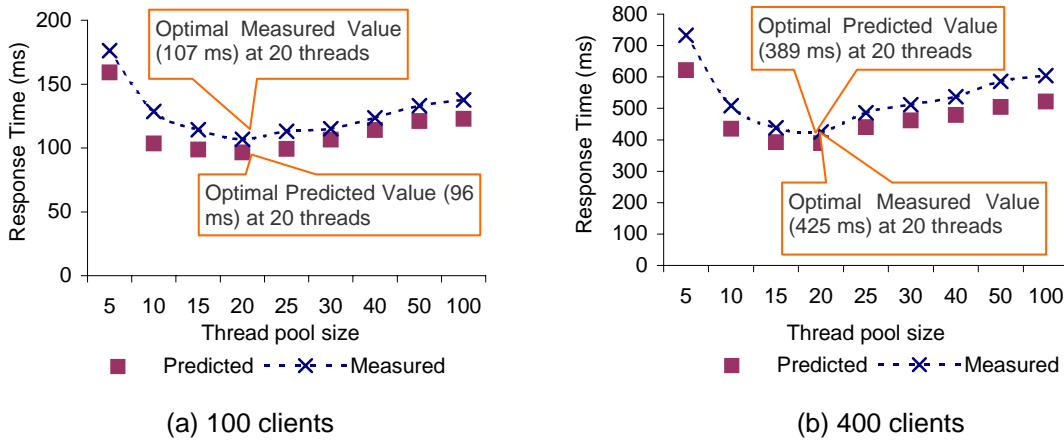
<sup>2</sup> The high *Error* in the circled samples is due to the ratio of transactions rolled back is high when the ratio of updating transaction increases. The overhead of rolling back transaction is not covered in the model

Model	Correlation Coefficient R	Coefficient of determination R <sup>2</sup>	p-value
CMP	0.9919	0.9837	0
RM	0.9981	0.9962	0
OCC	0.9984	0.9769	0

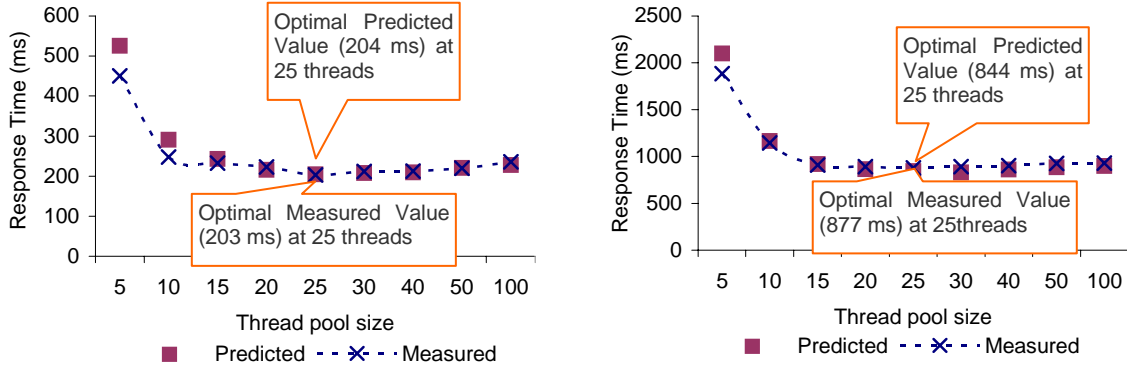
**Table 5.** Linear correlation between predicted and measured response time



**Fig. 7.** The response time vs. thread pool size (CMP)



**Fig. 8.** The response time vs. thread pool size (RM)



(a) 100 clients

(b) 400 clients

Fig. 9. The response time vs. thread pool size (OCC)

## 5 Future Work and Conclusions

In the Prediction-Enabled Component Technology (PECB) framework, the quality of the reasoning framework is evaluated by statistical intervals [9,10]. The accuracy of the prediction model is higher than ours, however, their performance model is developed for a white-box system and detailed measurements for each component can be easily discovered through source code instrumentation. This is not possible in our example of using a black-box COTS component technology. However, we hope our models may influence component technology vendors to expose APIs that allow the measurements of important parameters required for performance prediction.

In this paper, we statistically evaluate our PPCB approach for predicting the performance of black-box CB applications. The results demonstrate that 80% of the prediction error is within upper bound of 27% with a confidence level of 95%. This provides statistical evidence to the architects that our approach is accurate enough for predicting the performance for different architectures at the design level.

While these results are encouraging, and to our knowledge, the first that focus on black-box components that have been presented in the literature, they are of course based on a small sample. More evidence is required that the approach is broadly applicable and scalable. To this end we are working to:

- Enhance the PPCB approach to cover additional architectural features such as asynchronous messaging, widely varying message sizes and complex distributed transactions.
- Test the approach on more complex applications.
- Design software engineering tools that hide the complexity of the modeling and analysis steps in PPCB from an architect.

## References

1. Balsamo, S., Personè, V.D. N. Inverardi, P.: A Review on queueing network models with finite capacity queues for software architectures performance prediction, *Performance Evaluation*, Volume/Issue: vol 51/2-4, (2002) 269 – 288.
2. Cecchet, E., Marguerite, J., Zwaenepoel, W.: Performance and scalability of EJB applications, *Conference on Object Oriented Programming, Systems, Languages and Applications* (2002) 246-261.
3. Chen, S.; Liu, Y.; Gorton, I.; Liu, A.: Performance Prediction of Component Based System, *Journal of Systems and Software*, In Press (2004).
4. Gomaa, H., Menascé, D.A.: Design and performance modeling of component interconnection patterns for distributed software architectures, *Proc. Workshop on Software and Performance* (2000) 117-126.
5. Gorton, I.: *Enterprise Transaction Processing Systems*, Addison-Wesley (2000).
6. Gorton, I., Liu, A.: Performance evaluation of alternative component architectures for EJB applications, *IEEE Internet Computing*, vol.7, no. 3, (2003) 18-23.
7. Gorton, I., Liu, A., Brebner, P.: Rigorous evaluation of COTS middleware technology, *IEEE Computer*, vol. 36, no.3 (2003) 50-55.
8. Hahn, G. J.; Meeker, W.Q.: *Statistical Intervals: a guide for practitioners*, New York: John Wiley & Sons, 1991
9. Hissam, S. A., Moreno, G., Stafford, J., Wallman, K.: Packaging predictable assembly, *Component Deployment: IFIP/ACM Working Conference, LNCS 2370* (2002) 108-224.
10. Hissam, S. A., Hudak, J., Ivers, J., Klein, M., Larsson, M., Moreno, G., Northrop, L., Plakosh, D., Stafford, J., Wallnau, K., Wood, W.: *Predictable Assembly of Substation Automation Systems: An Experiment Report, Second Edition, CMU/SEI-2002-TR-031, ESC-TR-2002-031* (2002).
11. Juse, K.S., Kounev, S., Buchmann, A.: PetStore-WS: measuring the performance implications of web services, *Proceedings of the International Conference of the Computer Measurement Group* (2003).
12. Kounev, S., Buchmann, A.: Performance modeling of distributed E-Business applications using queuing petri nets, *Proc. of IEEE Int'l Symp on Performance Analysis of Systems and Software* (2003).
13. Kung, H. T. and Robinson, J. T.: On optimistic methods for concurrency control, *ACM Transactions on Database Systems*, vol. 6, No. 2, (1981) 213 - 226.
14. Lazowska, E., Zahorjan, J., Graham, S., Sevcik, K.: *Quantitative System Performance*, Prentice Hall (1984).
15. Liu, Y.; Fekete, A.; Gorton, I.: Predicting the performance of middleware-based applications at the design level, *4th International Workshop on Performance and Software Engineering* (2004) 166-170.
16. Liu, Y.: *A Framework to Predict the Performance of Component-based Applications*, PhD Thesis, University of Sydney, Australia (2004).
17. Menascé, D., Almeida, V.A.F.: *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Prentice-Hall, 2000.
18. OptimizeIt Suite, <http://www.borland.com/optimizeit/>
19. Rakatine, D.: *The Seppuku Pattern*, 2002.  
[http://www.theserverside.com/patterns/thread.tss?thread\\_id=11280](http://www.theserverside.com/patterns/thread.tss?thread_id=11280)

20. Rolia, J. A., Sevik, K.C.: The method of layers, IEEE Transaction on Software Engineering, vol. 21, no. 8. (1995) 689-700.
21. Saavedra, R. H., Smith, A. J.: Analysis of benchmark characteristics and benchmark performance prediction, ACM Transactions on Computer System, vol. 14, no. 4, (1996) 344-384.
22. Venables, W. N.; Ripley, B. D.; Modern Applied Statistics with S-Plus, Springer, (2002).
23. Walpole, R. E., Myers, R. H.: Probability and Statistics for Engineers and Scientists, Fifth Edition, Macmillan Publishing Company (1993).
24. Woodside, C.M., Neilson, J.E., Petriu, D.C., Majumdar, S.: The Stochastic Rendezvous Network Model for Performa of Synchronous Client-Server-Like Distributed Software, IEEE Transactions on Computers, vol. 44, no. 1, January (1995) 20-34.

## **Acknowledgements**

National ICT Australia is funded through the Australian Government's *BackingAustralia's Ability* initiative, in part through the Australian Research Council. We would also like to thank Professor Weber from the University of Sydney for his advise on statistical models.