Experiences of using LQN and QPN tools for performance modeling of a J2EE Application

Nidhi Tiwari
nidhi_tiwari@infosys.com
Prabhakar Mynampati
Prabhakar_Mynampati@infosys.com
Infosys Technologies Limited

Performance of a J2EE application is influenced by the underlying infrastructure, operating system and middleware parameters. Usually a reactive approach of testing is used to configure these, which is costly and lengthy. Consequently a proactive approach of performance modeling is required. Layered Queuing Networks and Queuing Petri Nets are two such effective techniques for tuning environment. This paper articulates our experiences with these techniques for a J2EE application. The relative attributes of the two techniques are listed to provide an insight on their suitability in a context.

1. Overview

For any J2EE application to deliver the expected quality of service, firstly it needs to be designed and built properly so that it does not have any bottlenecks. Secondly it needs to be hosted in a conducive environment for meeting its non-functional requirements of response time, throughput etc. The second step here leaves architects perplexed as most of these applications have intricate layers of software (application servers, web servers and database servers) which need to be configured appropriately for threads, pools, processes. These then in turn be hosted on adequate hardware having the complexity of resource sharing and multi-processor systems to get the optimum performance in deployment environment.

Now assuming that a system is free of any application bottlenecks how should one approach the problem of setting so many variables suitably with minimum cost and time to get optimum performance? One easy way is to test and tune, however this is costly and time consuming. Instead software modeling can be used to proactively determine the environment for desired quality of service without incurring large costs.

To substantiate this, here we demonstrate the usage of two popular modeling techniques, Layered Queuing Network (LQN) [CARLRADS] [WOOD2003] and Queuing Petri Nets (QPN) [KOUN2003]. For these can be used to model scheduling strategies, simultaneous resource possession, synchronization, blocking and contentions for hardware as well as software resources. The relative strengths and weaknesses of these techniques are also discussed to assist in determining the appropriateness of one of these in a given context.

The remainder of this paper is organized as follows. Section 2 gives a sketch of various forms of performance models based on literature survey. Section 3 gives a brief on LQN modeling. QPN and HQPN formalism are introduced in section 4. Section 5 describes the SPECjAppServer2001 application and its configuration details used for performance modeling study. The LQN and QPN models of the given application are constructed in section 6 and 7 respectively. The analysis of outputs of applications' LQN and QPN models are presented, along with measured results for comparison purpose, in section 8. Comparison of LQN and QPN models is given in section 9. Section 10 concludes this paper with summary and conclusions for these techniques.

2. Literature Survey

Several approaches have been proposed for early software performance analysis. These performance modeling methods are used for:

- comparing two or more system architectures
- determining optimal value of various parameters (system tuning)
- finding performance bottlenecks
- characterizing the workload on the system
- determining the number and size of components (capacity planning) and
- predicting the performance at future loads

To facilitate this kind of early analysis and to make these models easily usable for developers group, number of methods and tools for automating various activities involved in performance modeling has been developed. We briefly describe various approaches, corresponding methods and tools available to realize the performance modeling based on literature survey [BALS2004], [INFEDAC], [VERN1987] [DONA1995]:

Queuing Network Models (QNM) – One of the popular approaches for performance modeling is QNM. In a queuing network model computer system is represented as a network of queues, where the network of queues is a collection of service centers which symbolize the system resources and customers which depict the users/transactions. Initial QNMs were designed to model resource contentions among independent jobs; they lacked the parallel system and synchronization representation [VERN1987]. Later they were extended, to represent the synchronization, simultaneous resource possession, software resources and dynamic software characteristics resulting in Extended Queuing Networks (EQN), Stochastic Rendezvous Network (SRVN) and Layered Queuing Networks (LQN).

Developing the QNM from a system architecture specification (in UML or Use Case Map) and then solving them are two crucial steps in the overall performance analysis process. In this direction a lot of good work has been done by Murray Woodside, Dorin C. Petriu [PETR2002], Roy Gregory Franks [GREG1999], Dr. Connie U. Smith [PERFENG], Daniel A. Menasce [MENA1997] and others, by providing the methods and tools to automate these two steps for developers.

Mary Vernon, John Zahorjan and Edward D. Lazowksa noted that QNM have the balanced accuracy and efficiency [INFEDAC], as they can be efficiently solved using analytical computational methods to obtain a certain degree of accuracy in results.

Stochastic Petri-Nets (SPN) – Another popular approach for performance modeling is based on Petri-nets. These started with the classical Petri-nets, which consisted of places to represent servers, tokens that reside at the places and transitions that represent events causing tokens to move from place to place [VERN1987]. These models allowed the analysis of both functional and non-functional properties of the systems. The classical Petri-nets evolved over time as SPN (Stochastic Petri-Nets), CPN (Colored Petri-Nets), GSPN (Generalized Stochastic Petri-Nets), QPN (Queuing Petri-Nets) and HiQPN (Hierarchical Queuing Petri-Nets).

Jane Hillstone [HILL2001] finds that the synchronization structures used in SPN makes them complex to construct and analyze. However the use of such low level notations gives them unrestricted expressiveness and equips them to model large classes of systems. As the SPN models are based on state chart diagrams, they depict the dynamic behavior of the system but provide very little insight into the system structure.

SPN models are based on the concept of markov chains; solutions of which can be used directly to compute the steady state performance measures. These results are very close to the actual results. The underlying markov process for low level PN models can be manually obtained and numerically solved. But as the SPN advances to GSPN and above, generation and solution of the markov process becomes complicated and needs to be done using computers. The major problem is that this markov solution method suffers from the state space explosion problem.

Stochastic Process Algebra (SPA) – The third approach to performance modeling commonly used by practitioners is based on Process Algebra. Process algebras are the mathematical theories to model concurrent systems by means of their algebra and provide solutions for examining the system structure and their behavior [DONA1995]. Thus process algebra models help in describing and analyzing both functional and performance properties of software specification.

Based on this approach following modeling tools and techniques have been developed: TIPP (Time Processes and Performability evaluation), EMPA (Extended Markovian Process Algebra), and PEPA (Performance Evaluation Process Algebra) [BALS2004]. These tools use either the software system design specified using process algebra and the associated performance parameters or UML diagrams like collaboration and state chart diagrams for SPA model creation.

Like SPN, these models are also capable of representing large class of systems and capturing the dynamic system behavior as they use low level notations. Jane Hillstone also mentions that equivalence relation in process algebra makes it possible to compare two or more SPA models [HILL2001].

All these variety of performance models can be evaluated using analytical methods [JAIN1991] or simulation techniques [PETR2004]. The significant outputs obtained from their evaluation are throughput, utilization and response time, which can be used for the software system's performance study. Amongst these, the QNM and SPN are widely practiced and have evolved over the time to include most of the system components. For this reason we chose the LQN and QPN methods of these, to perform software modeling for our case study. In this paper we show how use of these for the given J2EE application, helped us to find the exact number of software resources to be configured in respective layers for a defined workload.

The next 2 sections give brief overview of LQN and QPN modeling techniques.

3. Layered Queuing Network

Layered Queuing Network (LQN) models [CARLRADS] are used to model the concurrent and distributed systems. LQN use the concept of layers to naturally model the software servers in the multi-tier system architectures with their nested services and calls. Software servers are modeled as tasks, and hardware resources as devices in an acyclic graph.

Any software object which has its own thread of execution can be modeled as task [WOOD2003]. These tasks can have properties like queue, scheduling disciplines and multiplicity. The various classes of services provided by a task are defined using entries and their hardware demands. The service requests from one entry to another form the arcs of the LQN model.

Figure 1 below shows the LQN model for the create login-id transaction of simple client-server application using the graphical notations of tasks, entries and devices. Multiple clients accessing the server from their machines have been modeled using multiple tasks and CPUs in LQN below.

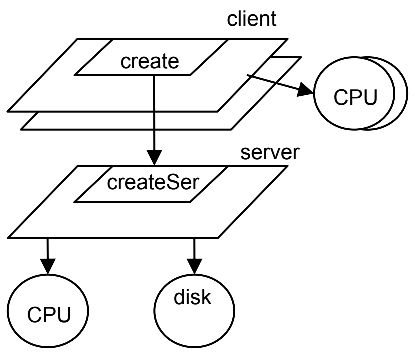

Figure 1: Simple Client-Server LQN model

The utilization, throughput, response time and queuing delays obtained by solving LQN model are used to diagnose the system for bottlenecks and scalability. What-if analysis can be done by varying the LQN input parameters like user load, multiplicity of hardware devices or software tasks.

4. Queuing Petri Nets and Hierarchical Queuing Petri Nets

Queuing Petri Nets formalized by Dr Falko Bause, is a combination of Queuing networks and Petri Nets to model the hardware and software aspects of system behavior [KOUN2003]. Using QPN models, in addition to the synchronization and blocking of software resources; hardware contentions and scheduling strategies can also be modeled.

In this formalism, a processing device is modeled as queuing place consisting of two components: queue and depository. The queuing place and its notation are shown below figure 2.

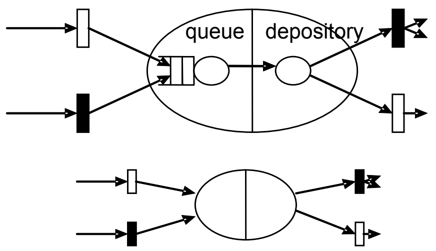

Figure 2: A Queuing Place and its Notation

The software processes are modeled using ordinary places. And the number of software processes (threads, connection pools etc) is specified as the number of process tokens available in these places. Transitions represent the conditions/events that can move a request token to next queuing place.

For efficient numerical analysis of complex QPN models, Dr Falko Bause developed the HQPN formalism exploiting the hierarchical structures. In HQPN a queuing place known as subnet contains whole QPN instead of single queue. A subnet place might contain an ordinary QPN or again another HQPN allowing for multiple hierarchical level nesting.

Detailed description of LQN is available in [WOOD2003] [CARLRADS]. QPNs and HiQPN details are given in the article [KOUN2003] by Samuel Kounev and Alejandro Buchmann.

Next section briefly introduces the SPECjAppServer2001 benchmark which will be modeled to study the LQN and QPN techniques for software and hardware bottlenecks identification.

5. The SPECjAppServer2001 Benchmark Application

SPECjAppServer2001 [SPECORG] is an Enterprise JavaBeans (EJB)TM benchmark to measure the scalability and performance of J2EE servers and containers. We decided to take this application for performance modeling study as its workload is based on a large distributed application, claimed to be big and complex enough to represent a real-world e-business system. The benchmark emulates a manufacturing, supply chain management (SCM) and order/inventory system, representative of one in use at a Fortune 500 company. Manufacturing, Supplier & Service Provider, Customer, and Corporate are the four business domains, implemented using EJB components, in the benchmark.

Before modeling this application we need to know the application architecture and deployment environment, the request classes to be modeled, hardware and software resources used by those request classes and their corresponding service demands on the hardware resources.

For this reason, initial deployment environment was taken from [KOUN2003] as shown in figure 3. It has a WebLogic server (WLS) as application server (AppServer) and Oracle 9i database server (DBS) for data persistence.

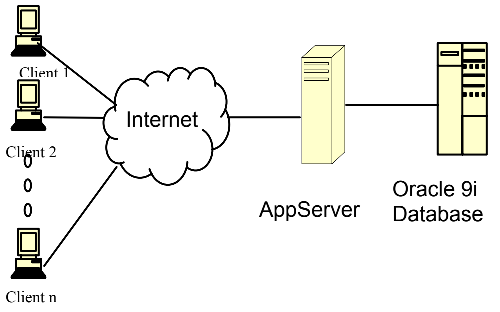

Figure 3: SPECjAppServer2001 deployment diagram

The NewOrder and ChangeOrder transactions in order entry application of customer domain were identified for modeling. The deployment diagram in figure 3 suggests that following software and hardware resources were used during these transactions processing:

- A WLS thread
- The CPU of a WebLogic server
- The network between a WebLogic server and the database server
- A database connection
- The CPU of the database server
- A database server process
- The disk subsystem of the database server (I/O)

Table 1 from reference document [KOUN2003] lists the service demands (CPU time) for the two request classes of system for our further study. These service demands values in reference document were determined by conducting some experiments with the order entry application and measuring the time spent by each transaction at each resource [KOUN2003]. The network service demands were ignored as all communications were taking place over a 100MBit LAN and communication times were negligible.

Transaction Type	**WLS-CPU (in sec)**	**DBS-CPU (in sec)**	**DBS-I/O (in sec)**
NewOrder	0.07	0.053	0.012
ChangeOrder	0.026	0.016	0.006

Table 1: Workload Service Demands

The creation of LQN and QPN models for this SPECjAppServer2001 benchmark are described below.

6. LQN Model for SPECjAppServer2001

The formulation of LQN model for the given J2EE application is explained here. LQN modeling starts with identification of nodes for the acyclic graph. So first the high level component instances in application architecture (figure 3) were modeled as shown figure 4 as following LQN tasks:

- *Users* - The multiple users accessing the system with some think time were modeled as multiple 'Users' reference tasks.
- *Application server (AppServer)* – Multiple instances of 'AppServer', which was an active task as it received request from users and sent it to Database server, were represented in model.
- *Database server (DBServer)* **–** Database connection pool having multiple instances of database connection was modeled in the LQN as multiple 'DBServer' tasks.
- *Database Disk (DBDisk)* – 'DBDisk' was modeled as multiple pure server tasks to model the database disk I/O operations.

Next the hardware devices (processor and disk) shown in the application deployment diagram (figure 3) were mapped to LQN devices/processors in figure 4:

- *User Processor (UserP)* **–** Multiple users accessed the system from separate machines, therefore multiple 'User' processors were shown in LQN model.
- *Application Server Processor (AppP)* **–** All the application server instances were running on single machine, so application server processor was modeled as a process sharing device.
- *Database Server Processor (DBP)* **–** Single DB server executed multiple database instances in sharing mode, therefore it was modeled as process sharing device.

- *Database Disk (DBDiskP)* – Only one database disk was considered in deployment diagram and as DB disk has First-In First-Out (FIFO) scheduling, it was modeled as single device having FIFO scheduling.

For LQN construction, once the nodes are identified, they are connected as per deployment environment. Thus above mentioned tasks and processors were connected using the arcs referring the links in deployment diagram as shown below in figure 4. The transaction flow paths are used to model the entries corresponding to the services provided by each task.

The NewOrder transaction was modeled by adding the corresponding entries in all the tasks with their service demands as shown in the figure 4. All the synchronous requests from entries to entries were modeled with the probability 1 for the number of calls for each entry.

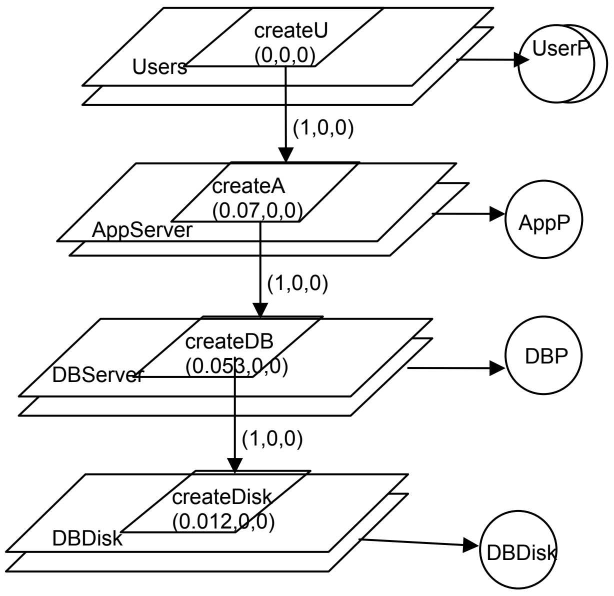

Figure 4: SPECjAppServer2001 LQN model

The flow of a NewOrder request in the given LQN model starts at the client task, waiting for the given think time. Next the request is queued at the AppServer task queue, waiting for the WLS thread. Once WLS thread is available the request is processed by the CreateW entry for the given service time on the WLS CPU in process sharing mode. Subsequently, the request is queued and processed in similar manner on the DBServer and DBDisk tasks, and their processors.

7. HQPN Model for SPECjAppServer2001

Assuming the SPECjAppServer 2001 deployment environment as shown in figure 3, the HQPN model for its NewOrder transaction was built. The HQPN model is shown in figure 5.

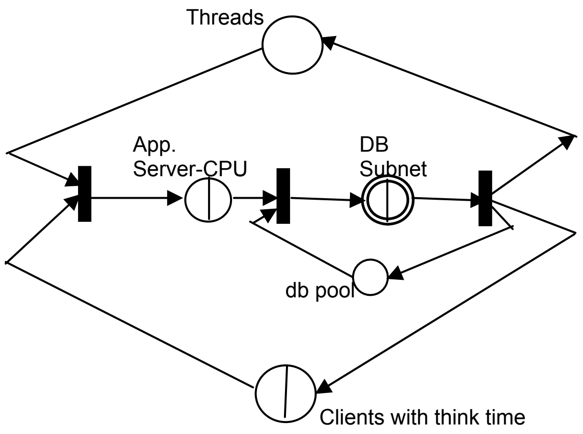

Figure 5: HQPN model of SPECjAppServer2001

The clients, AppServer CPU, DB server CPU, DB server Disk were represented with queuing places in the HQPN model. The database server was represented with DB subnet place to reduce state space explosion problem. The software resources like WLS threads, db connection pools, db processes were represented as ordinary places with equivalent number of tokens.

The traversal path of a client request in the model is as follows [KOUN2003]: Every client request spends the user specified think time at the client queue and then moves in the client depository. The request waits in the client depository for the availability of application server thread.

Once the WLS thread is available, the request is transitioned to the queue of the AppServer CPU place, where it receives the service from the CPU of application server for the specified service demand time. The request then moves to the depository of the AppServer place and waits for the JDBC connection. After obtaining the connection, the request enters into DB subnet place, which is the input place of the DB subnet.

In the DB subnet, the request is processed in similar fashion at the DBServer and DBDisk queuing place. After completing all the processing request moves back to the Client queuing place, releasing the DB Process, DB connection pool and AppServer thread.

The total response time for the client request in the HQPN model is calculated by summing up the residence times at the queue and depository of all queuing places and ordinary places in the client processing chain. Thus in this case, total response time for the NewOrder transaction would be sum of residence time at client depository, application server CPU queue and depository, DB process queue, DB server CPU queue and DB I/O queue.

8. Performance Models Analysis

The LQN and QPN performance models built above were used for what-if analysis. The various test conditions used and their results are described in this section. Case 1, 2 and 3 used the single class models with different settings for multiplicity of webserver threads and CPU's to identify the system bottlenecks and configure these parameters for required performance. While case 4 used the multi-class models to demonstrate the accuracy of the modeling results for multiple request classes too.

Generally, the performance models are acceptable if they predict the system resource utilizations and throughputs within 10% and response time prediction within 30% [HLIU2004]. Considering this, we experimented on the LQN and HQPN performance models built above for different configurations using the LQNS [GREG1999] and HiQPN [KOUN2003] tools.

For LQNS tool the LQN format files were used to provide the input models. The creation of LQN input files and execution of LQNS tool were quick and less resource intensive. The required performance measures were directly pulled from its output text file. Thus throughput and utilizations of the devices were picked from the output files. While the response times taken were the service times for the user entries in the output file as it includes - queuing for all processors, service time at all processors, queuing for all serving tasks and phase one service times at all serving tasks in the request's path [GREG2005].

For HiQPN tool, its user interface was used to input the HQPN model. The HiQPN input model creation and execution were time consuming and resource intensive as compared to LQN model. Operational laws were applied on its output to get the values of required performance metrics.

Next the results of these two models are compared against the measured results taken from reference document [KOUN2003] by calculating the percentage error for each case. The analysis of these models for different configurations is also given concisely.

Case 1:
Initially, the NewOrder transaction was considered for study. The system was configured to have 80 system users having average think time of 200 ms, 40 WLS threads, 40 JDBC connections and 30 DBS processes.

LQN model input file [GREG2005] for NewOrder transaction with same configuration settings was processed with LQNS tool. Appendix A shows the input LQN file for this case. The output file of LQNS for this case is given in Appendix B. Same information was used for HQPN input and using the operational laws, required performance measures were computed from its output. The performance metrics obtained from [KOUN2003], LQNS and HQPN tools with their percentage errors are shown in table 2.

Metric	Test Result	LQN Model	LQN Error	QPN Model	QPN Error
WLS-CPU Utilization	100%	99.9%	0.1%	100%	0%
DBS-CPU Utilization	65%	75%	15%	75%	15%
NewOrder Throughput	13.41	14.28	6.3%	14.28	6.3%
NewOrder Response Time	5.7 sec	5.4 sec	5.2%	5.4 sec	5.2%

Table 2: Case1 performance metrics

The output metrics of the LQN and QPN models were used for analyzing the hardware bottlenecks, software contentions and overheads as follows.

The LQNS output file points that 39.9 WLS threads were used out of the available 40 WLS threads, indicating that there may be contention for WLS threads. Besides the 99.9% WLS CPU utilization in file establishes it as a saturated device. Thus, on study of the CPU utilizations at various servers, the AppServer was identified to be the bottleneck device.

The LQNS tool reported the number of DBServer and DBDisk tasks (i.e. the processes or threads) utilized in this case as 2.75 and 0.20 respectively. Thus the rest 37 DBServer and 29 DBDisk threads were lying idle in memory and consuming memory. Consequently, reducing the number of threads to the required number would help reducing the memory costs and improve the system performance. The HQPN tool also reported the level of concurrency as the mean token population at each processing center which was used for similar analysis of the software contentions/overheads.

Case 2:
As indicated in previous case, there may be contention for WLS threads, so in this case the number of WLS threads was increased to 60, keeping other configuration same as in case 1. Thus the test and modeling were done with following configuration: 80 system users with average think time of 200 ms, 60 WLS threads, 40 JDBC connections, and 30 DBS processes. Following table summarizes the results of test and the results from modeling tools.

Metric	Test Result	LQN Model	LQN Error	QPN Model	QPN Error
WLS-CPU Utilization	100%	99.99 %	0.1%	100%	0%
DBS-CPU Utilization	65%	75.71 %	15%	75%	15%

NewOrder Throughput	13.43	14.28	6.3%	14.28	6.3%
NewOrder Response Time	5.73 sec	5.39 sec	5.2%	5.39 sec	5.2%

Table 3: Case2 performance metrics

The performance results in this case were found to be similar to Case 1. Nevertheless the response time remains 5.4 sec which is same as in case 1 against the expectation that it would decrease. The lqn model output files reveal that though the wait time reduced from 2.63 to 1.25, the service time at AppServer increased from 2.77 to 4.14 because of increased concurrency level. This signals that for a saturated device; increasing the number of threads or processes may not help in improving the performance.

Case 3:
Of the above two case, the bottleneck for the SPECjAppServer2001 for the deployment given in figure 3 was detected to be at the AppServer. Therefore in this case one more CPU was added at the AppServer. For testing and modeling, system was configured to have 30 clients with 1 sec think time, infinite WLS threads, DB connection pools and DB processes. The table 4 gives the results of testing, LQN modeling and QPN modeling with corresponding percentage deviations.

Metric	**Test Result**	**LQN Model**	**LQN Error**	**QPN Model**	**QPN Error**
WLS-CPU Utilization	68%	65%	4.4%	64%	5.8%
DBS-CPU Utilization	91%	98%	7.6%	96%	5.4%
NewOrder Throughput	17.56	18.59	5.8%	18.28	4.1%
NewOrder Response Time	0.673 sec	0.613 sec	8.9%	0.623 sec	8%

Table 4: Case3 performance metrics

Study of these performance metrics evinces that by adding one WLS CPU and changing the system configuration as mentioned above, the throughput has increased to 17.57 and response time has reduced drastically to 0.67 sec. As the user load is shared by two processors, the utilization has also come down to 68% per processor.

Case 4:
After verifying the two modeling techniques for single transaction in above cases, in this case these techniques were examined for transaction mix. NewOrder and ChangeOrder transactions of SPECjAppServer2001 were modeled and tested for 20 clients (10 for each request class) with average think time of 1 sec, 10 WLS threads (5 for each request class) and infinite number of DB connection pools and DB disk processes. Service demands for each entry on respective software servers input to models were picked up from table 1. The total number of users was uniformly distributed with ratio 1:1, for these two transactions.
The results of testing, LQN modeling and QPN modeling for this case are summed up in table 5 given below.

Metric	**Test Result**	**LQN Model**	**LQN Error**	**QPN Model**	**QPN Error**
WLS-CPU Utilization	77%	78%	1.2%	76%	1.2%
DBS-CPU Utilization	64%	55%	14%	54%	15.6%
NewOrder Throughput	7.47	7.8	4.4%	7.4	0.2%
ChangeOrder Throughput	9.15	9.03	1.3%	9.22	0.7%
NewOrder Response Time	0.318 sec	0.276 sec	13.2%	0.34 sec	6.9%
ChangeOrder Response Time	0.104sec	0.106 sec	1.9%	0.084 sec	19.2%

Table 5: Case4 performance metrics

Looking at the results in all the above cases it can be inferred that the model prediction errors of both LQN and QPN for single and multi-class requests are within the acceptable range [HLIU2004]. Except for the DBS-CPU utilization where the error is found to be 15%. This could be because of the data collection methodology used for performance testing and/or for measuring the service demand values in the reference paper [KOUN2003]. For instance if there were some other processes running on the database server, they would have also accounted for in DBS-CPU utilization.

9. LQN and QPN Comparison

The above section described how both LQN and QPN models of a J2EE application can be efficiently used to analyze the software and hardware configuration in deployment environment. Also the performance results obtained from these models were observed to be close. Now in this section we examine the key differences found in using the given software modeling methodologies.

As the efficacies of two techniques for modeling system performance are comparable, application architects often face the dilemma of which one to choose for a given problem context. To assist in this regard, we explore some of the substantial differences between LQN and QPN models on grounds of their usability, features, solution techniques, limitations, tools availability etc:

- QPN can be used to analyze both functional and performance aspects of system. Whereas LQN gives only the performance measures of a system.
- LQN can be analytically solved using the approximate MVA techniques with minimal

resources; while QPN is analytically solved using Markov process thus requires resources that are exponential in the size of the model to produce exact results.

- The accuracy in system's software contention results is less in analytical LQN model as compared to corresponding QPN model results.
- The LQN models can be used for modeling any number of concurrent user requests. However the QPN model cannot be used for large number of concurrent requests due to state space explosion problem.
- LQN does not have any computational limitations, so can be modeled for any number of layers (tiers)/resources. Nevertheless the QPN computation model becomes exponentially complex with addition of each ordinary place and queuing place.
- LQN supports both the open (geometric distribution) and closed requests. While the QPN is restricted only to modeling the closed requests.
- LQN can be used to model synchronous, asynchronous and forward calls. So the messaging systems can also be modeled with LQN. QPN supports only synchronous calls.
- LQN model consists of convenient primitives notations which makes LQN construction simple, convey more semantic information and guarantee that these models are well-formed (i.e. stable and deadlock free) [DONA1995]. On the other hand, the low level notations used in QPN give it added expressive power with some readability complexity.
- In QPN memory size constraints for performance can be modeled more accurately than in the LQN.

Here we observe that LQN and QPN have their own benefits and constraints. Thus, one of these should be appropriately chosen for modeling based on the trade-off between resource, time and accuracy requirements in the given context.

10. Summary and Conclusions

This paper presents how to use the two popular analytical modeling techniques, LQN and QPN for evaluating the performance of J2EE systems. It demonstrates how these techniques can be used for finding the approximate number of software resources to be configured in respective layers at a defined workload. Thus reduce the memory cost and increase the availability of memory for respective servers for a given J2EE application. The performance results obtained from corresponding analytical tools for SPECjAppserver2001 models are compared with its measurement results.

Modeling results in this paper reveal that the performance parameters like resource utilizations, throughputs and response time values obtained from both these modeling techniques are within the acceptable limits except for DBS-CPU subjecting to the measurement methods used. Based on these observations we conclude that it is time and cost effective to proactively do the performance modeling, rather than deploying the system on different configurations and carrying out many experiments

Other conclusions from the experience of using the two analytical modeling tools are as following. LQN solver adopts the approximate solutions in solving MVA, and so its performance results sometimes deviate from the actual measurement results. Its counterpart, the HQPN tool analyses the model with the help of Markov chains and solves the balance equations using numerical methods to give accurate results. However it was found that it greatly suffers from state space explosion problem and takes huge amount of resources and time for large number of user requests and/or components. Hence performance analysis using LQN models was found to be better approach as it has got a balance of efficiency and accuracy.

Acknowledgements

We acknowledge the use of LQNS tool by Murray Woodside and Greg Franks from Carleton University. We thank Dr. Falko Bause from the University of Dortmund for the HiQPN tool. We also acknowledge reference of the paper by Samuel Kounev and Alejandro Buchmann for SPECjAppServer2001 benchmark's readings and HQPN model.

References

[BALS2004] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, "Model-Based Performance Prediction in Software Development: A Survey", IEEE Transactions on Software Engineering, 30(5), May 2004.
[CARLRADS] http://www.sce.carleton.ca/rads/
[DONA1995] S. Donatelli, J. Hillston, and M. Ribaudo, "A comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri Nets", In Proc of 6th International Workshop on Petri Nets and Performance Models, Durham, North Carolina, 1995.
[GREG1999] Roy Greg Franks, "Performance Analysis of Distributed Server Systems", PhD thesis, Carleton University, Canada, Dec 20, 1999.
[GREG2005] Greg Franks, Peter Maly, Murray Woodside, Dorin C. Petriu, Alex Hubbard. "Layered Queuing Network Solver and Simulator User Manual", Carleton University, Canada, Dec 15, 2005.

[HILL2001] J. Hillston, L. Recalde, M. Ribaudo, and M. Silva, “A comparison of the expressiveness of SPA and bounded SPN models, In Proceedings of the 9th International Workshop on Petri Nets and Performance Models, Germany, September 2001.

[HLIU2004] Henry H. Liu, Pat V. Crain, “An Analytic Model For Predicting The Performance Of SOA-Based Enterprise Software Applications”, 30th International Computer Measurement Group Conference, USA, December 5-10, 2004.

[INFEDAC] www.inf.ed.ac.uk

[JAIN1991] Raj Jain, “The Art of Computer Systems Performance Analysis”, Published by John Wiley & Sons, Inc. 1991.

[KOUN2003] Samuel Kounev and Alejandro Buchmann, “Performance Modelling of Distributed E-Business Applications using Queuing Petri Nets”, IEEE International Synopsium on Performance Analysis of Systems and Software 2003.

[MENA1997] D. A Menasce, "A Framework for Software Performance Engineering of Client/Server Systems," Proc. of the 1997 Computer Measurement Group Conference, FL, December 9-12, 1997.

[PERFENG] http://www.perfeng.com/

[PETR2002] Dorin Petriu, Murray Woodside, "Software Performance Models from System Scenarios in Use Case Maps", Proc. 12 Int. Conf. on Modeling Tools and Techniques for Computer and Communication System Performance Evaluation (Performance TOOLS), 2002.

[PETR2004] Dorin B. Petriu and Gabriel Wainer, “A DEVS Library for Layered Queuing Networks”, Int workshop on Modeling and Applied Simulation, Italy, Oct 28-30, 2004.

[SPECORG] http://www.spec.org/osg/jAppServer2001/

[VERN1987] M. Vernon, J. Zahorjan, and E. Lazowska, “A comparison of performance Petri nets and queueing network models”, In Proc. of Int. Workshop on Modelling Techniques and Performance Evaluation, Paris, 1987.

[WOOD2003] Murray Woodside, “Layered Resources, Layered Queues and Software Bottlenecks: A tutorial” Performance Tools 2003 conference, Sept 2, 2003.

Appendix

Appendix A: LQN model

```
G
"Simple 3 tier application"
0.00001
100
1
0.5
#End of general information
-1

#Processor information
P 4
p UserP f
p WebP s
p DBP s
p DBDiskP f
#End of processor info
-1

#Task Information
T 0
t Users r user -1 UserP z 0.2 m 80
t WebSer n createW -1 WebP m 40
t DB n createDB -1 DBP m 40
t DBDisk n createDisk -1 DBDiskP m 30
#End of Task information
-1

#Entry Information
E 0
s user 0 0 0 -1
y user createW 1 0 0 -1
s createW 0.07 0 0 -1
y createW createDB 1 0 0 -1
s createDB 0.053 0 0 -1
y createDB createDisk 1 0 0 -1
s createDisk 0.012 0 0 -1
#End of Entry Information
-1
```

Appendix B: LQN solution

```
Service times:

Task Name    Entry Name    Phase 1
Users        user          5.40004
WebSer       createW        2.79972
DB           createDB       0.192588
DBDisk       createDisk     0.0143659

Throughputs and utilizations per phase:

Task Name    Entry Name    Throughput      Phase 1    Total
Users        user           14.2856        77.1427    77.1427
WebSer       createW        14.2856        39.9956    39.9956
DB           createDB       14.2856        2.75123    2.75123
```

```
DBDisk          createDisk      14.2856         0.205225    0.205225

Utilization and waiting per phase for processor:  UserP

Task Name       Pri     n       Entry Name      Utilization         Ph1 wait
Users            0      80      user             0                  0

Utilization and waiting per phase for processor:  WebP

Task Name       Pri     n       Entry Name      Utilization         Ph1 wait
WebSer           0      40      createW          0.999989           1.26857

Utilization and waiting per phase for processor:  DBP

Task Name       Pri     n       Entry Name      Utilization         Ph1 wait
DB               0      40      createDB         0.757135           0.0626121

Utilization and waiting per phase for processor:  DBDiskP

Task Name       Pri     n       Entry Name      Utilization         Ph1 wait
DBDisk           0      30      createDisk       0.171427           0.00236592
```