UNIVERSITY OF CAMBRIDGE
Computer Laboratory

opera

Performance Modeling and Evaluation of Distributed Component Systems using Queueing Petri Nets

Samuel Kounev
OPERA Group, Systems Research Group
University of Cambridge – Computer Laboratory

May 4, 2007
PEPA Club, LFCS, University of Edinburgh

Roadmap

- Research Interests
- Introduction to Queueing Petri Nets
- Performance Modeling Methodology
- Case Study: Modeling SPECjAppServer2004
- QPN Modeling Environment (QPME) Demo
- Concluding Remarks

Research Interests

1. Performance modeling and evaluation
2. System sizing and capacity planning
3. Software performance engineering
4. Benchmarking and experimental performance analysis
5. Performance tuning and optimization
6. Autonomic computing and self-managed systems

Target Domains

1. Distributed component-based systems
2. Enterprise middleware
3. Java EE and related technologies
4. Large-scale e-business applications
5. Event-based systems
6. Grid computing environments
7. XML-based Web services
8. Service Oriented Architectures
9. RFID and EPCglobal-related applications

Current Projects

1. Performance modeling and evaluation of event-based systems
2. SPECjms2007 - benchmark for message-oriented middleware
3. Autonomic QoS management in Grid computing and SOA using online performance models
4. QPME - Queueing Petri Net Modeling Environment
5. Performance and scalability analysis of SAP' Web application server (Netweaver)
6. A Transport Information Monitoring Environment: Event Architecture and Context Management (TIME-EACM)

Motivation

- Distributed component systems increasingly ubiquitous.
- Quality of service requirements of crucial importance!
- System architects and deployers faced with questions such as:

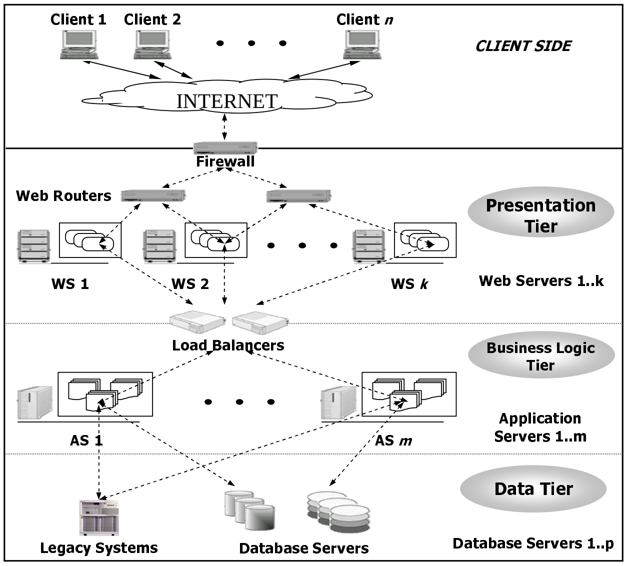
MEASURING

- Which **platform** would provide the best cost/performance ratio for a given application?
- How do we ensure that the selected platform does not have any inherent scalability bottlenecks?

PREDICTING

- For a given application design, what performance would the **application** exhibit under the expected workload?
- How do we ensure that the application does not have any inherent scalability bottlenecks?

Distributed Component System (DCS)



If (n = 1000)

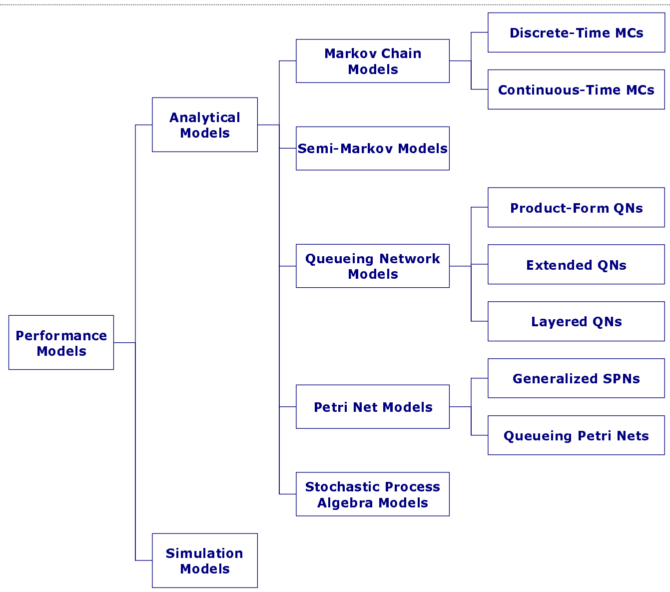
k=? m=? p=?

so that all

SLAs

are fulfilled.

Space of Performance Models



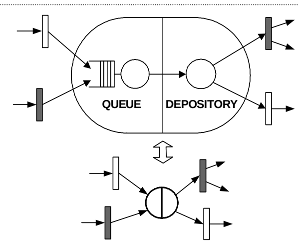
Queueing Networks vs. Petri Nets

- **Queueing Networks**
  - Very powerful for modelling **hardware contention** and scheduling strategies. Many efficient analysis techniques available.
  - Hard to model blocking, synchronization, simultaneous resource possession and **software contention** aspects.

- **Stochastic Petri Nets**
  - Suitable both for qualitative and quantitative analysis.
  - Easy to model blocking, synchronization, simultaneous resource possession and software contention aspects.
  - However, no direct means for modelling queues.

Queueing Petri Nets (QPNs = QNs + PNs)

- Introduced by **Falko Bause** in 1993.
- Combine queueing networks and Petri nets
- Allow integration of queues into places of PNs
- Ordinary vs. queueing places
- **Queueing place** = queue + depository

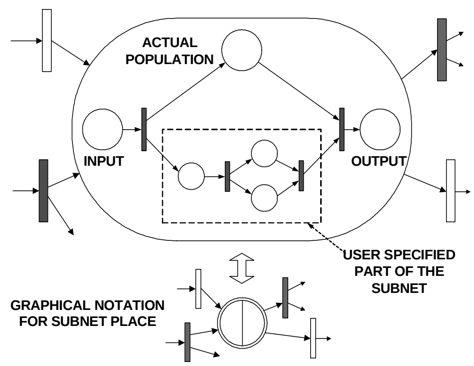


PROS: Combine the modelling power and expressiveness of QNs and PNs. Facilitate the modelling of both hardware and software aspects of system behavior in the same model.

CONS: Analysis suffers the **state space explosion** problem and this imposes a limit on the size of the models that are analyzable.

Hierarchical Queueing Petri Nets (HQPNs)

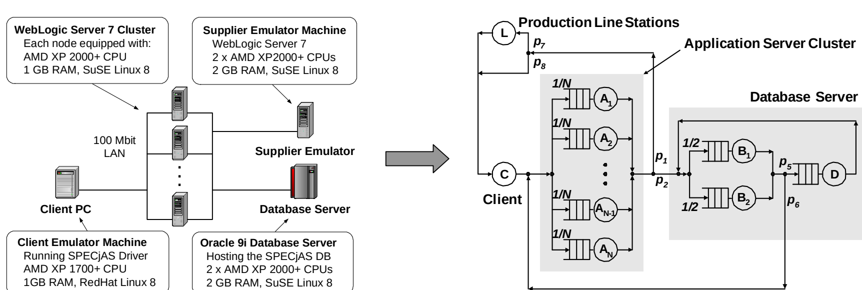
- Allow hierarchical model specification
- **Subnet place**: contains a nested QPN
- Structured analysis methods alleviate the state space explosion problem



Modeling using Queueing Networks

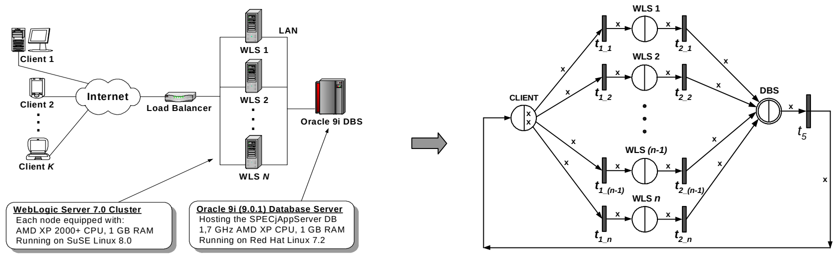
"Perf. modeling and evaluation of large-scale J2EE applications", CMG-2003

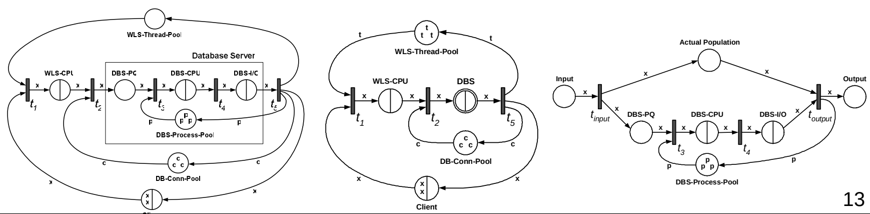
- Benchmark deployment modeled using Queueing Networks (QNs).
- Two problems encountered:
  - Poor model expressiveness: no way to accurately model asynchronous processing and software contention.
  - Large non-product form QNs not tractable.



Modeling using Queueing Petri Nets

"Performance Modeling of Distributed E-Business Applications using Queueing Petri Nets", IEEE ISPASS-2003.





13

Modeling using Queueing Petri Nets (2)

- **Excellent model expressiveness** for both hardware and software aspects of system behavior.

METRIC	Model	Measured	Error
WLS-CPU Utilization	100%	100%	0%
DBS-CPU Utilization	75%	65%	15%
NewOrder Throughput	14.28	13.43	6.3%
NewOrder Resp.Time	5399ms	5738ms	5.9%
Thread Queue Length	17.14	18	4.7%

- **However, state space explosion problem**:
  - Model had to be restricted to part of the application.
  - Max 20 concurrent customers.
  - Models of realistic systems not tractable!

SimQPN – Simulator for QPNs

- Tool and methodology for analyzing QPNs using simulation.
- Provides a scalable simulation engine optimized for QPNs.
- Can be used to analyze models of realistic size and complexity.
- Light-weight and fast.
- Portable across platforms.
- Validated in a number of realistic scenarios.

"SimQPN - a tool and methodology for analyzing queueing Petri net models by means of simulation",
Performance Evaluation, Vol. 63, No. 4-5, pp. 364-394, May 2006.

SimQPN

Performance Modeling Methodology

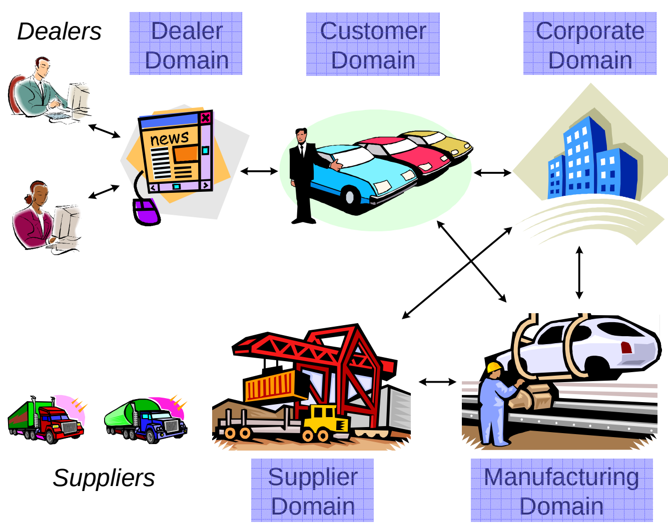
1. Establish performance modeling objectives.
2. Characterize the system in its current state.
3. Characterize the workload.
4. Develop a performance model.
5. Validate, refine and/or calibrate the model.
6. Use model to predict system performance.
7. Analyze results and address modeling objectives.

"Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets", IEEE Transactions on Software Engineering, Vol. 32, No. 7, pp. 486-502, July 2006.

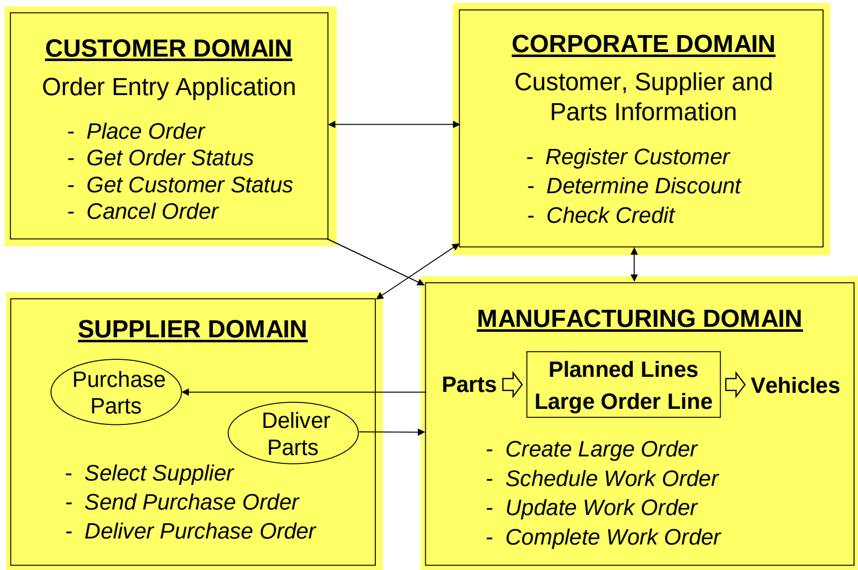
Roadmap

- Research Interests
- Introduction to Queueing Petri Nets
- Performance Modeling Methodology
- Case Study: Modeling SPECjAppServer2004
- QPN Modeling Environment (QPME) Demo
- Concluding Remarks

SPECjAppServer2004 Business Model



SPECjAppServer2004 Business Domains



OSG Java Subcommittee

OSG Java Subcommittee

SPECjAppServer2004 Application Design

Benchmark Components:

1. ***EJBs*** – J2EE application deployed on the *System Under Test (SUT)*
2. ***Supplier Emulator*** – web application emulating external suppliers
3. ***Driver*** – Java application emulating clients interacting with the system and driving production lines

- RDBMS used for persistence
- Asynchronous-messaging used for inter-domain communication
- Throughput is function of chosen *Transaction Injection Rate*
- Performance metric is **JOPS = JAppServerOpsPerSecond**

OPERA Group © S. Kounev 21

SPECjAppServer2004 Application Design (2)

SPECjAppServer Driver made up of two components:

1. DealerEntry Driver:

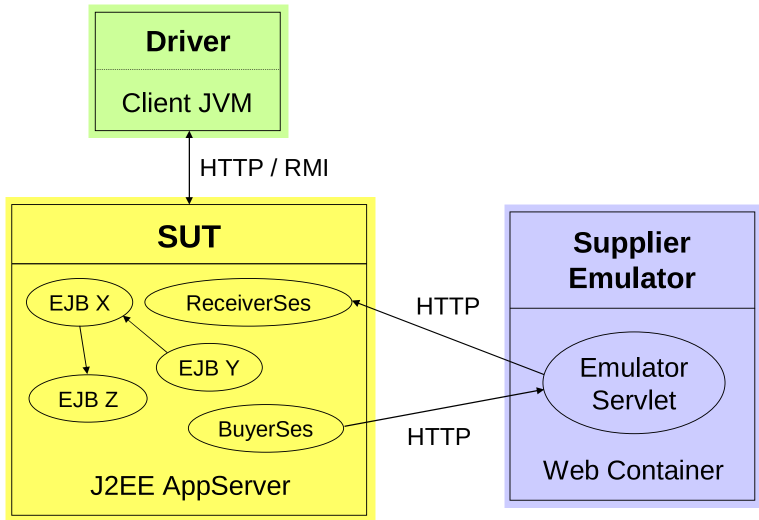
- Emulates automobile dealers interacting with the system.
- Exercises the dealer and order-entry applications using 3 business transaction types: Browse, Purchase and Manage.
- Each transaction emulates a client session.
- Communicates with the SUT through HTTP.

2. Manufacturing Driver:

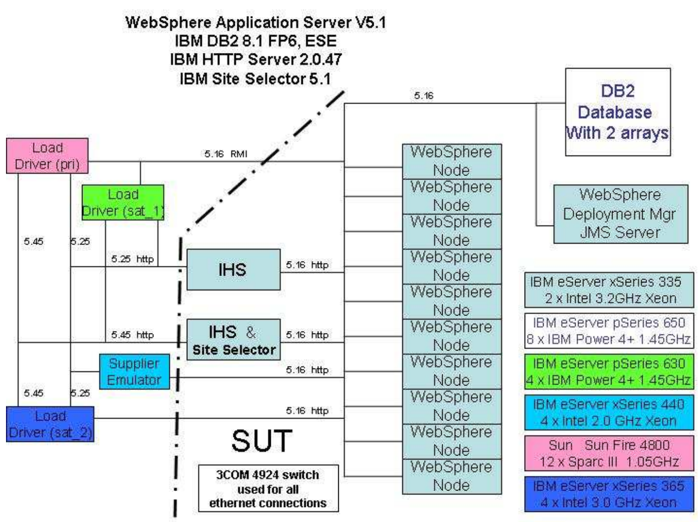
- Drives production lines in the manufacturing domain.
- Exercises the manufacturing application.
- Unit of work is WorkOrder.
- Communicates with the SUT through RMI.

OPERA Group © S. Kounev 22

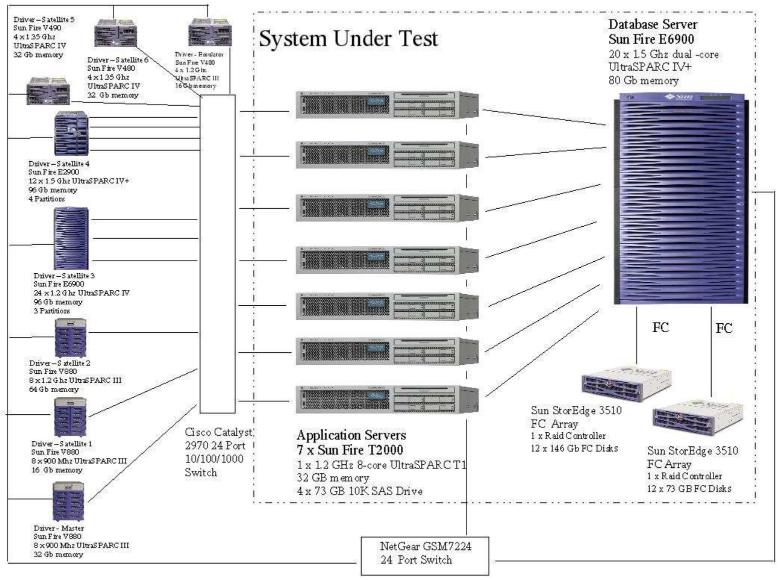
SPECjAppServer2004 Application Design (2)



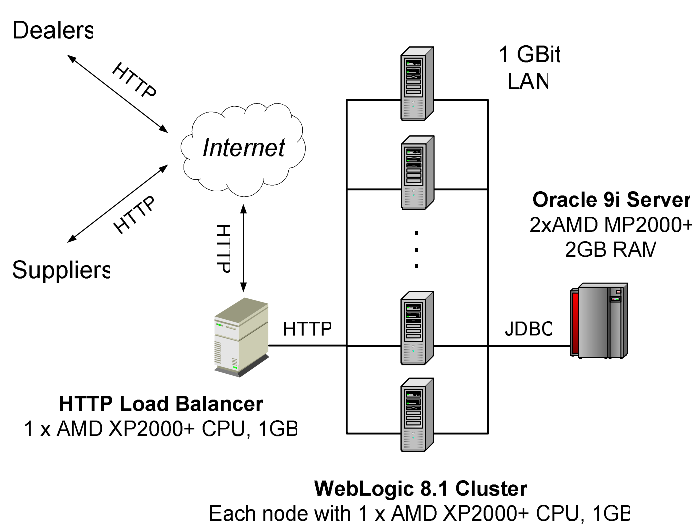
Sample Deployment Environment (IBM)



Sample Deployment Environment (Sun)



Case Study - Deployment Environment



1. Establish Modeling Objectives

Normal Conditions: 72 concurrent dealer clients (40 Browse, 16 Purchase, 16 Manage) and 50 planned production lines in the mfg domain.

Peak Conditions: 152 concurrent dealer clients (100 Browse, 26 Purchase, 26 Manage) and 100 planned production lines in the mfg domain.

Goals:

- Predict system performance under normal operating conditions with 4 and 6 application servers.
- Predict how much system performance would improve if the load balancer is upgraded with a slightly faster CPU.
- Study the scalability of the system as the workload increases and additional application server nodes are added.
- Determine which servers would be most utilized under heavy load and investigate if they are potential bottlenecks.

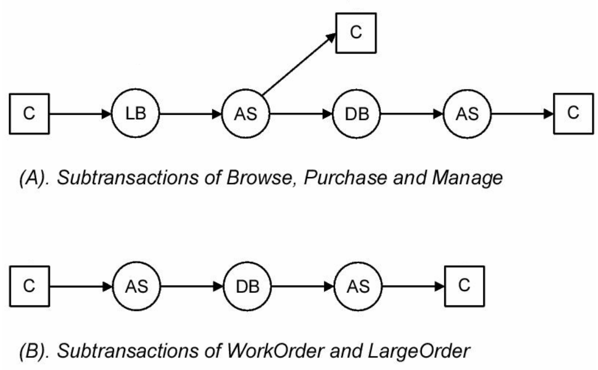
2. Characterize the System

SYSTEM COMPONENT DETAILS

Component	Description
Load Balancer	WebLogic 8.1 Server (HttpClusterServlet) 1 x AMD Athlon XP2000+ CPU 1 GB RAM, SuSE Linux 8
App. Server Cluster Nodes	WebLogic 8.1 Server 1 x AMD Athlon XP2000+ CPU 1 GB RAM, SuSE Linux 8
Database Server	Oracle 9i Server 2 x AMD Athlon MP2000+ CPU 2 GB RAM, SuSE Linux 8
Local Area Network	1 GBit Switched Ethernet

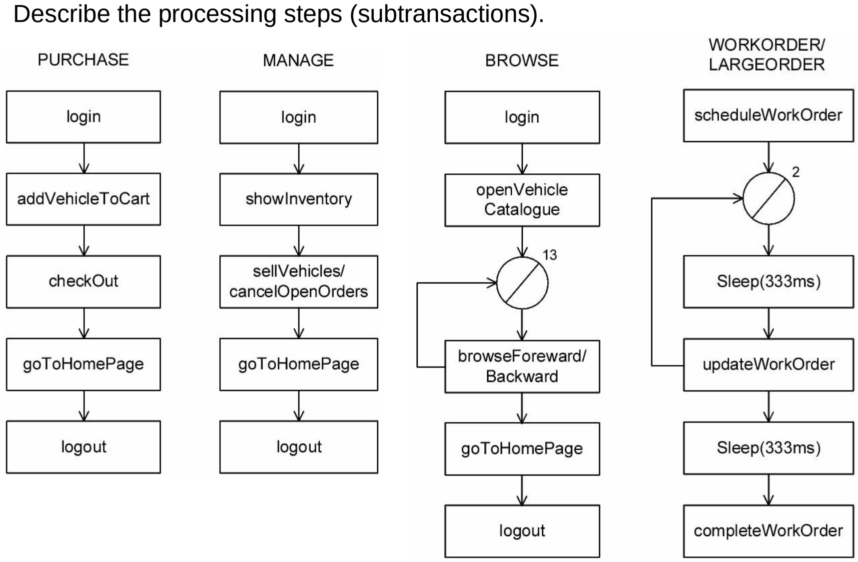
3. Characterize the Workload

1. **Basic Components:** Dealer Transactions and Work Orders.
2. **Workload Classes:** Browse, Purchase, Manage, WorkOrder and LgrOrder.
3. **Inter-Component Interactions:**

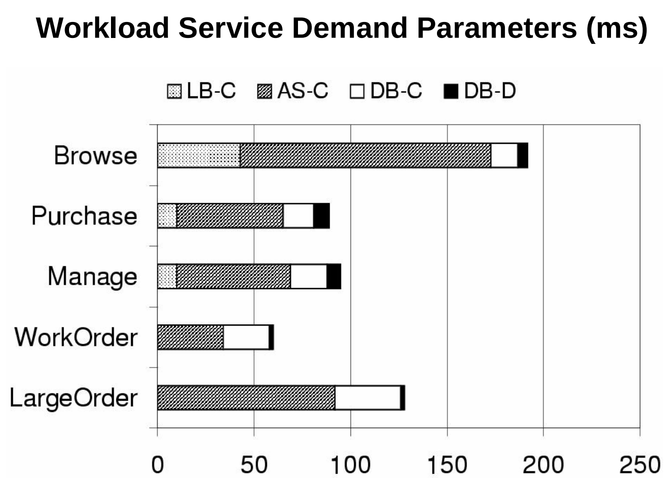


3. Characterize the Workload (2)

Describe the processing steps (subtransactions).



3. Characterize the Workload (3)

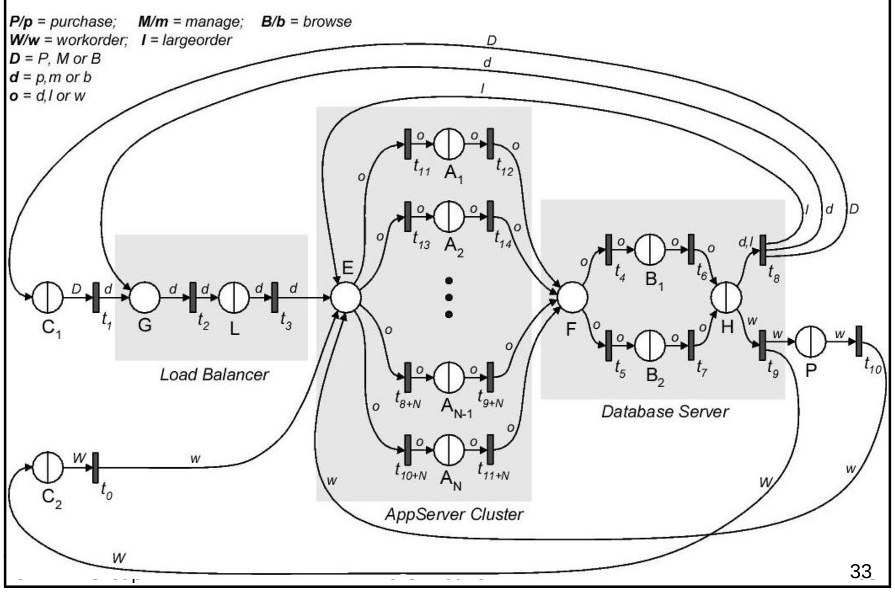


3. Characterize the Workload (4)

WORKLOAD INTENSITY PARAMETERS

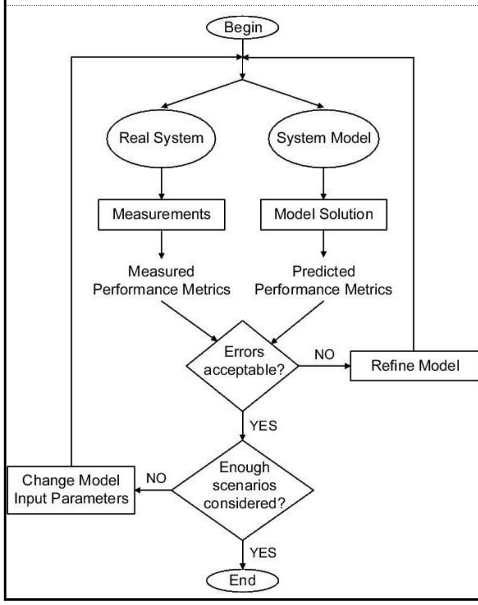
Parameter	Normal Conditions	Peak Conditions
Browse Clients	40	100
Purchase Clients	16	26
Manage Clients	16	26
Planned Lines	50	100
Dealer Think Time	5 sec	5 sec
Mfg Think Time	10 sec	10 sec

4. Develop a Performance Model



33

5. Validate [Refine, Calibrate]



Assume 2 AS nodes available.

Two Specific Validation Scenarios:

1: 20 B, 10 P, 10 M, 30 PL

2: 40 B, 20 P, 30 M, 50 PL

Max. Modeling Error:

- For Throughput: 8.1%
- For Utilization: 10.2%
- For Resp. Times: 12.9%

34

6. Predict System Performance

ANALYSIS RESULTS FOR SCENARIOS UNDER NORMAL CONDITIONS WITH 4 AND 6 AS NODES

	4 App. Server Nodes			6 App. Server Nodes		
METRIC	Model	Measured	Error	Model	Measured	Error
X_B	7.549	7.438	+1.5%	7.589	7.415	+2.3%
X_P	3.119	3.105	+0.5%	3.141	3.038	+3.4%
X_M	3.111	3.068	+1.4%	3.117	2.993	+4.1%
X_W	4.517	4.550	-0.7%	4.517	4.320	+4.6%
X_L	0.313	0.318	-1.6%	0.311	0.307	+1.3%
R_B	299ms	282ms	+6.0%	266ms	267ms	-0.4%
R_P	131ms	119ms	+10.1%	116ms	110ms	+5.5%
R_M	140ms	131ms	+6.9%	125ms	127ms	-1.6%
R_W	1086ms	1109ms	-2.1%	1077ms	1100ms	-2.1%
U_{LB}	38.5%	38.0%	+1.3%	38.7%	38.5%	+0.1%
U_{AS}	38.0%	35.8%	+6.1%	25.4%	23.7%	+0.7%
U_{DB}	16.7%	18.5%	-9.7%	16.7%	15.5%	+0.8%

6. Predict System Performance (2)

ANALYSIS RESULTS FOR SCENARIOS UNDER PEAK CONDITIONS WITH 6 APP. SERVER NODES

	Original Load Balancer			Upgraded Load Balancer		
METRIC	Model	Measured	Error	Model	Measured	Error
X_B	17.960	17.742	+1.2%	18.471	18.347	+0.7%
X_P	4.981	4.913	+1.4%	5.027	5.072	-0.8%
X_M	4.981	4.995	-0.3%	5.013	5.032	-0.4%
X_W	8.984	8.880	+1.2%	9.014	8.850	+1.8%
X_L	0.497	0.490	+1.4%	0.501	0.515	-2.7%
R_B	567ms	534ms	+6.2%	413ms	440ms	-6.5%
R_P	214ms	198ms	+8.1%	182ms	165ms	+10.3%
R_M	224ms	214ms	+4.7%	193ms	187ms	+3.2%
R_W	1113ms	1135ms	-1.9%	1115ms	1123ms	-0.7%
U_{LB}	86.6%	88.0%	-1.6%	68.2%	70.0%	-2.6%
U_{AS}	54.3%	53.8%	+0.9%	55.4%	55.3%	+0.2%
U_{DB}	32.9%	34.5%	-4.6%	33.3%	35.0%	-4.9%

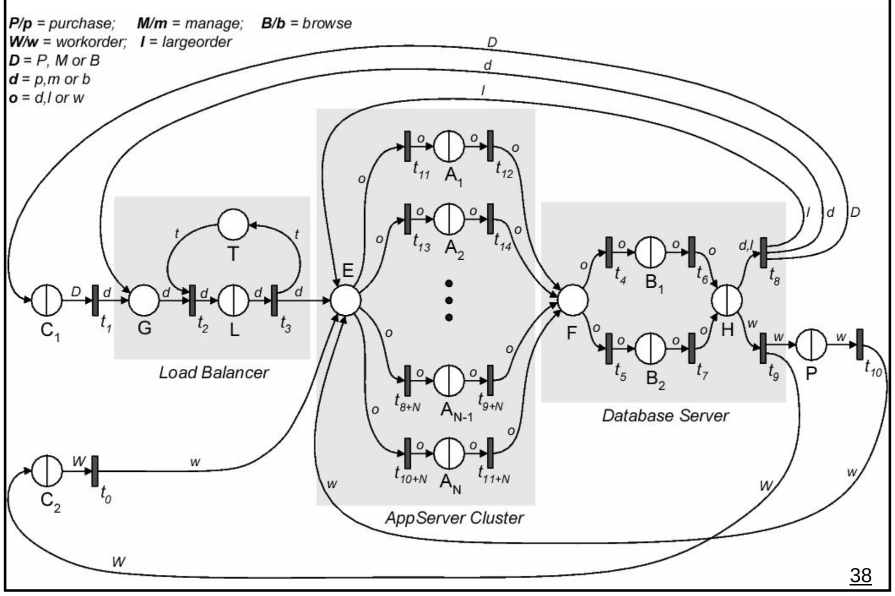
6. Predict System Performance (3)

ANALYSIS RESULTS FOR SCENARIOS UNDER HEAVY LOAD WITH 8 APP. SERVER NODES

	Heavy Load Scenario 1			Heavy Load Scenario 2		
METRIC	Model	Measured	Error	Model	Measured	Error
X_B	26.505	25.905	+2.3%	28.537	26.987	+5.7%
X_P	4.948	4.817	+2.7%	4.619	4.333	+6.6%
X_M	4.944	4.825	+2.5%	4.604	4.528	+1.6%
X_W	8.984	8.820	+1.8%	9.003	8.970	+0.4%
X_L	0.497	0.488	+1.8%	0.460	0.417	+10.4%
R_B	664ms	714ms	-7.0%	2012ms	2288ms	-12.1%
R_P	253ms	257ms	-1.6%	632ms	802ms	-21.2%
R_M	263ms	276ms	-4.7%	630ms	745ms	-15.4%
R_W	1116ms	1128ms	-1.1%	1123ms	1132ms	-0.8%
U_{LB}	94.1%	95.0%	-0.9%	99.9%	100.0%	-0.1%
U_{AS}	54.5%	54.1%	+0.7%	57.3%	55.7%	+2.9%
U_{DB}	38.8%	42.0%	-7.6%	39.6%	42.0%	-5.7%

150 Browse Clients *200 Browse Clients*

6. Predict System Performance (4)



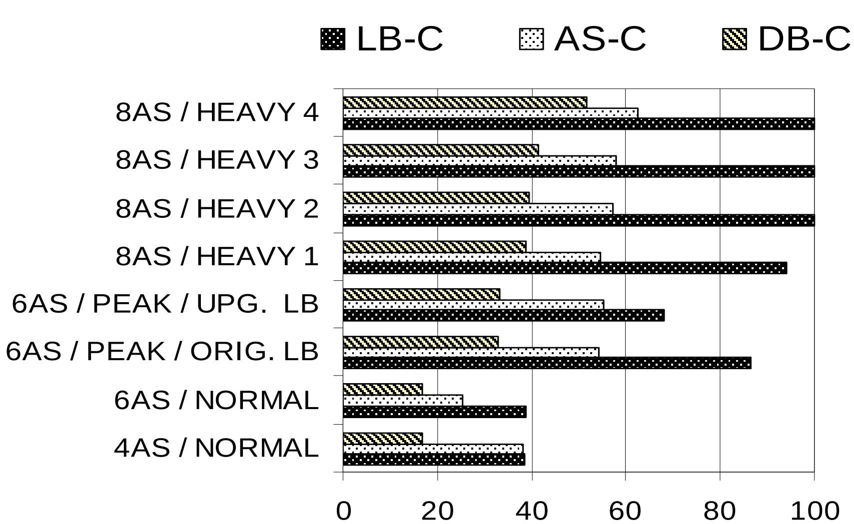
6. Predict System Performance (4)

	Heavy Load Sc. 3 with 15 Threads			Heavy Load Sc. 3 with 30 Threads		
METRIC	Model	Measured	Error	Model	Measured	Error
X_B	28.607	27.323	+4.7%	28.590	27.205	+5.1%
X_P	4.501	4.220	+6.7%	4.499	4.213	+6.8%
X_M	4.489	4.387	+2.3%	4.494	4.485	+0.2%
X_W	10.784	10.660	+1.2%	10.793	10.800	-0.1%
X_L	0.447	0.410	+9.0%	0.450	0.446	+0.1%
R_B	5495ms	5740ms	-4.2%	5495ms	5805ms	-5.3%
R_P	1674ms	1977ms	-15.3%	1665ms	2001ms	-16.8%
R_M	1685ms	1779ms	-5.3%	1670ms	1801ms	-7.3%
R_W	1125ms	1158ms	-2.8%	1125ms	1143ms	-1.6%
U_{LB}	100.0%	93.0%	+7.5%	99.9%	100.0%	-0.1%
U_{AS}	57.9%	57.8%	+0.2%	57.9%	58.0%	-0.2%
U_{DB}	41.6%	44.0%	-5.5%	41.6%	44.0%	-5.5%
N_{LBQ}	146	161	-9.3%	131	146	-10.3%

Sc.3: 300 B, 30 P, 30 M, 120 PL → Max Error **16.8%**

Sc.4: 270 B, 90 P, 60 M, 120 PL → Max Error **15.2%**

7. Analyze Results & Address Objectives



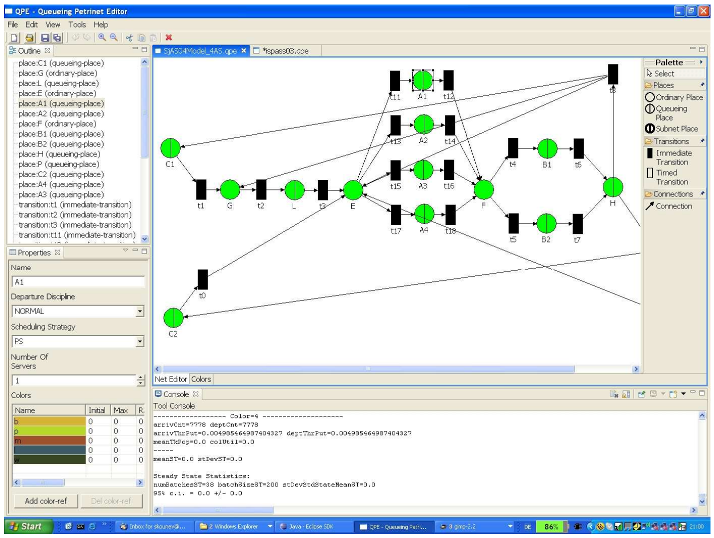
Benefits of using QPNs

1. QPN models allow the integration of hardware and software aspects of system behavior.
2. Using QPNs, DCS can be modeled *accurately*.
3. The knowledge of the structure and behavior of QPNs can be exploited for efficient simulation using SimQPN.
4. QPNs can be used to combine qualitative and quantitative system analysis.
5. QPN models have an intuitive graphical representation facilitating model development.

QPME

- A performance modeling tool based on QPNs
- QPN Editor (QPE) and Simulator (SimQPN)
- Based on Eclipse/GEF
- Provides a user-friendly graphical user interface
- Runs on all platforms supported by Eclipse

QPME – QPN Modeling Environment



Summary & Conclusions

- Presented a systematic approach for performance prediction.
- Studied a **representative** application and predicted its performance under **realistic** load conditions.
- Model predictions were validated against measurements on the real system. The modeling error did not exceed 21.2%!

- QPN models can be exploited for **accurate** performance prediction in realistic scenarios.
- Proposed methodology provides a powerful tool for sizing and capacity planning.

Further Reading

S. Kounev, “Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets”, IEEE Transactions on Software Engineering, Vol. 32, No. 7, pp. 486-502, July 2006.

S. Kounev, C. Dutz, A. Buchmann, „QPME - Queueing Petri Net Modeling Environment“, In Proceedings of the 3rd International Conference on Quantitative Evaluation of SysTems (QEST-2006), Riverside, CA, September 11-14, 2006.

S. Kounev and A. Buchmann, “SimQPN - a tool and methodology for analyzing queueing Petri net models by means of simulation”, Performance Evaluation, Vol. 63, No. 4-5, pp. 364–394, May 2006.

S. Kounev, “Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction”, Shaker Verlag, Dec. 2005, ISBN: 3832247130.

S. Kounev and A. Buchmann, “Performance Modeling of Distributed E-Business Applications using Queueing Petri Nets”, In Proc. of the 2003 IEEE Intl. Symposium on Performance Analysis of Systems and Software, Austin, Texas, March 6-8, 2003.

Papers available for download at http://www.cl.cam.ac.uk/~sk507

Questions

Thank You for your Attention!

QUESTIONS?