Performance Prediction using QPN Models: From Capacity Planning to Online Performance Management

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

January 16, 2008
Department of Software Engineering, Charles University, Czech Republic

Roadmap

- Introduction to Queueing Petri Nets
- Modeling Case Studies
  - Modeling Distributed Component Systems
  - Modeling Event-Based Systems
  - Online QoS Control in Grid Environments
- Concluding Remarks

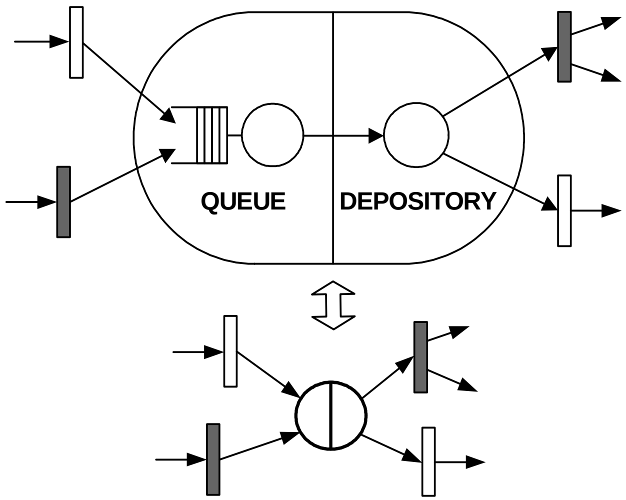
Queueing Networks vs. Petri Nets

- **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

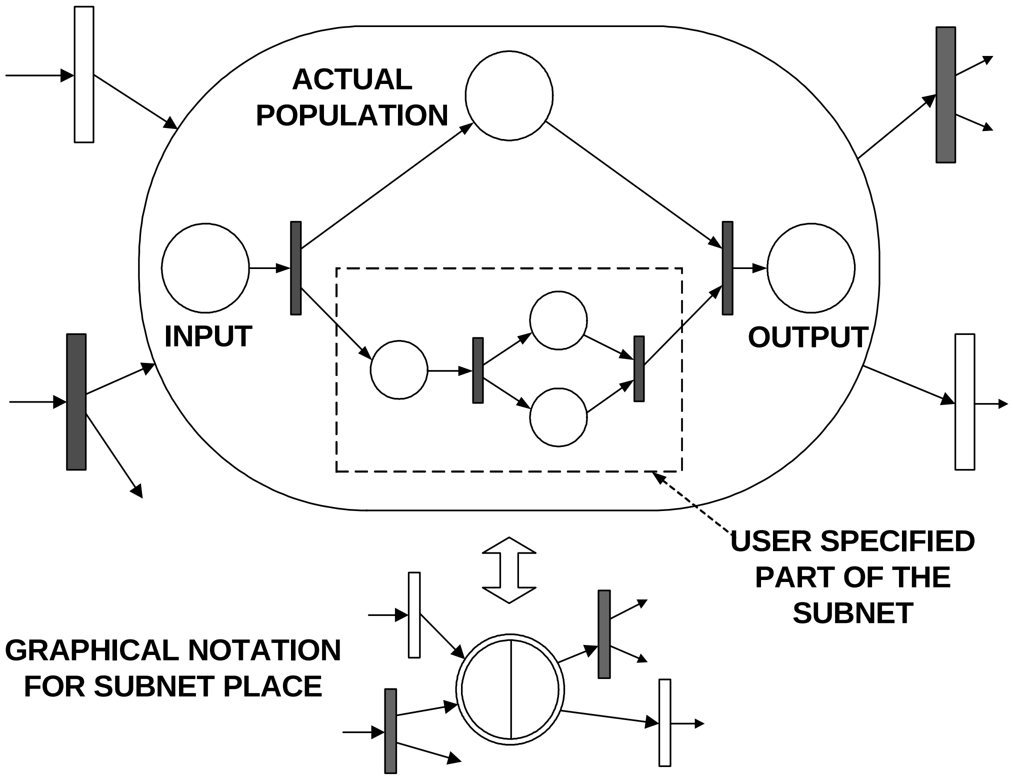


PROS: Combine the modelling power and expressiveness of QNs and PNs. Facilitate the modelling of both hardware and software aspects of system behavior in the same model.

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



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.

Performance Modeling Methodology

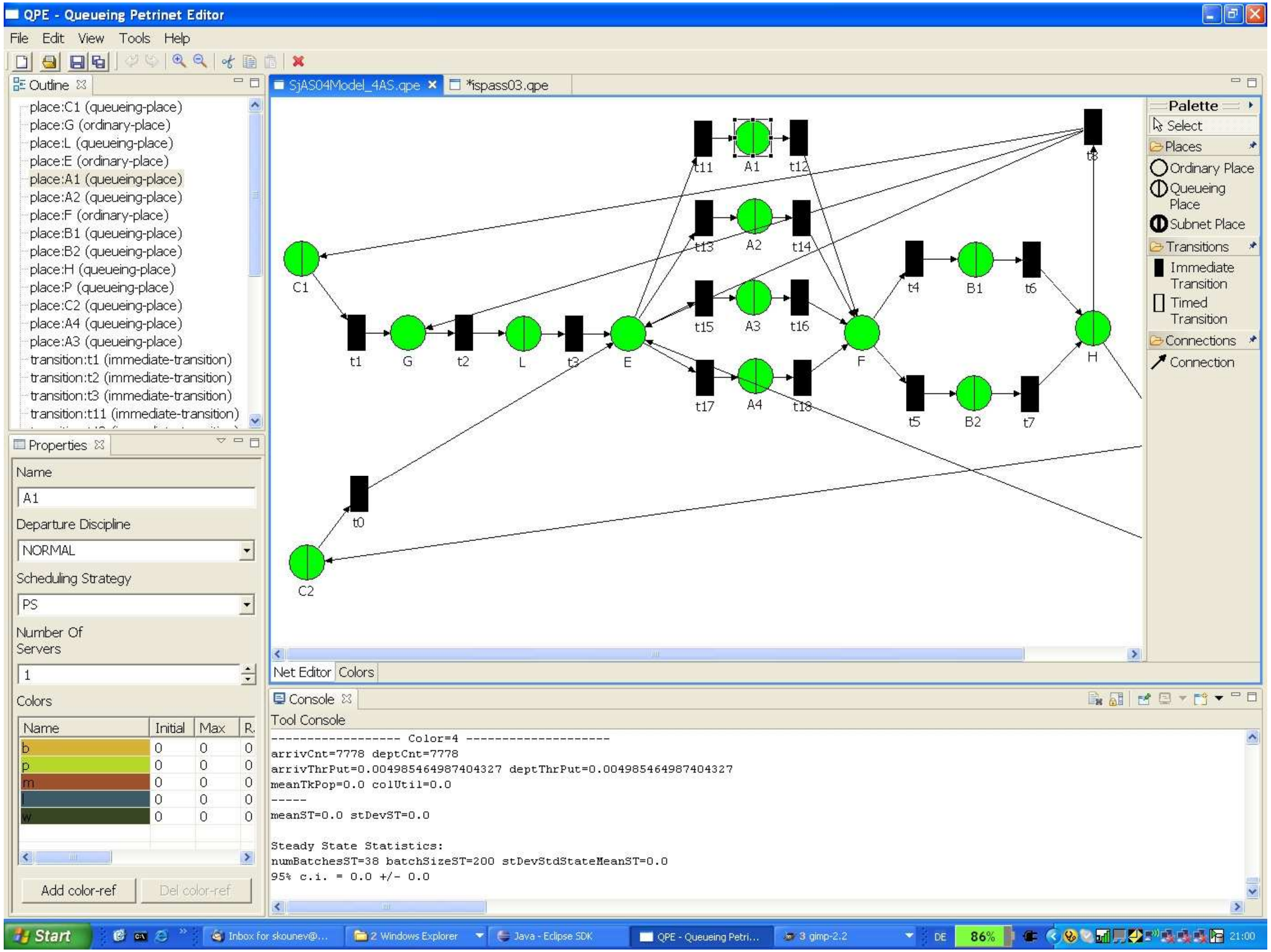
1. Establish performance modeling objectives.
2. Characterize the system in its current state.
3. Characterize the workload.
4. Develop a performance model.
5. Validate, refine and/or calibrate the model.
6. Use model to predict system performance.
7. Analyze results and address modeling objectives.

"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.

QPME - QPN Modeling Environment

- 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

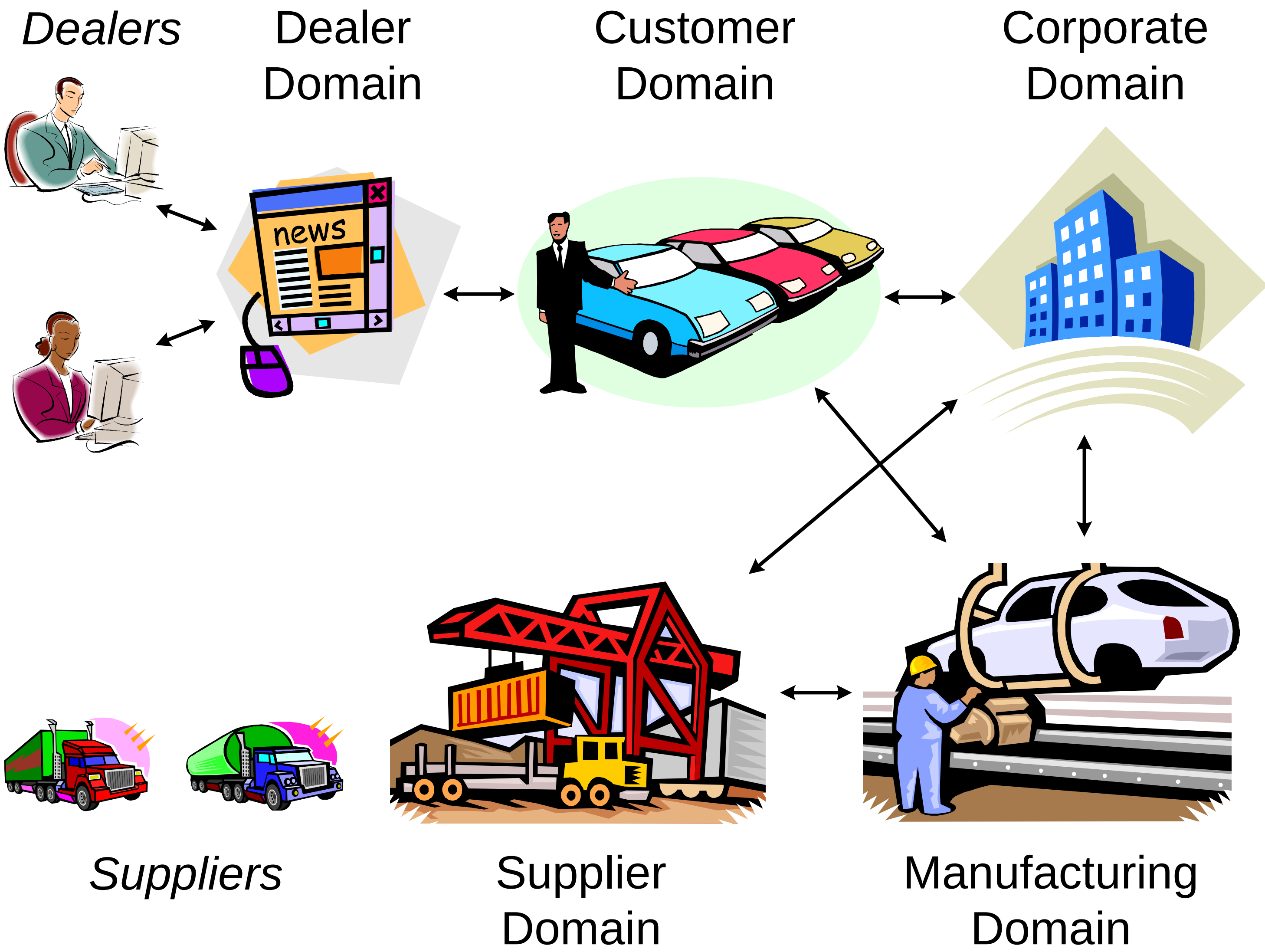
QPME – QPN Modeling Environment (2)



Roadmap

- Introduction to Queueing Petri Nets
- Modeling Case Studies
  - Modeling Distributed Component Systems
  - Modeling Event-Based Systems
  - Online QoS Control in Grid Environments
- Concluding Remarks

SPECjAppServer2004 Business Model



SPECjAppServer2004 Application Design

SPECjAppServer Driver made up of two components:

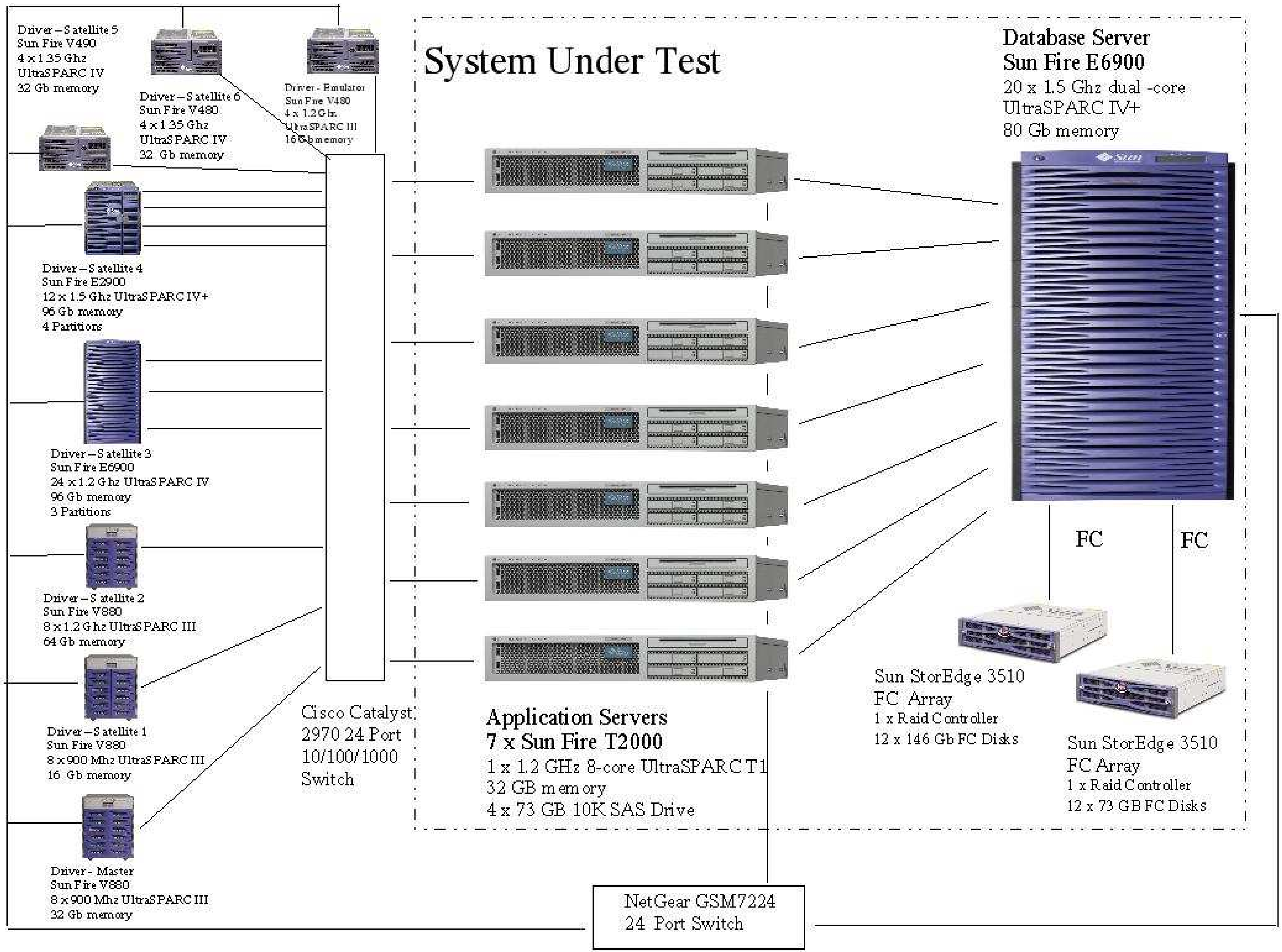
1. DealerEntry Driver:

- 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.

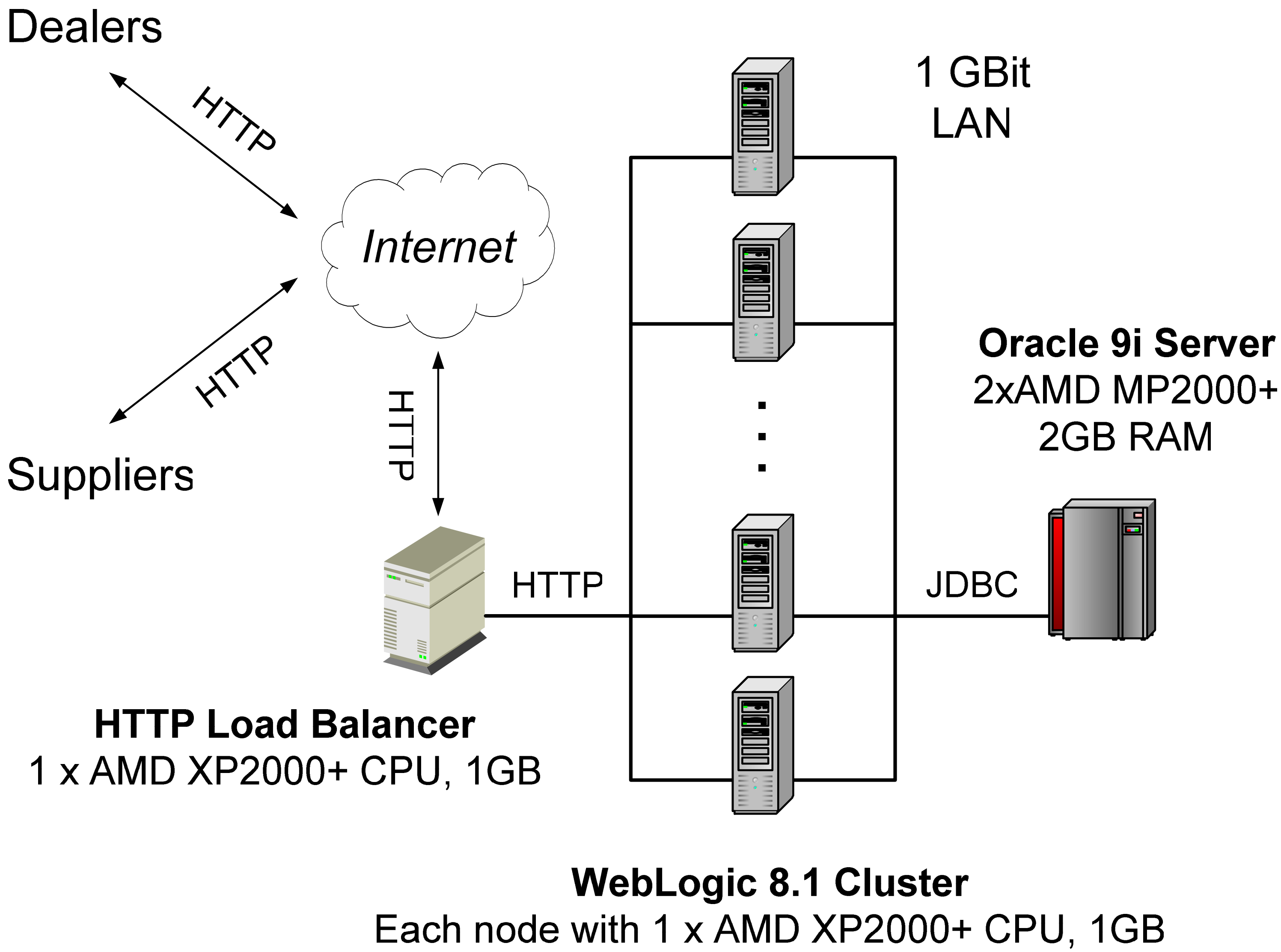
2. Manufacturing Driver:

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

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.
- 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.

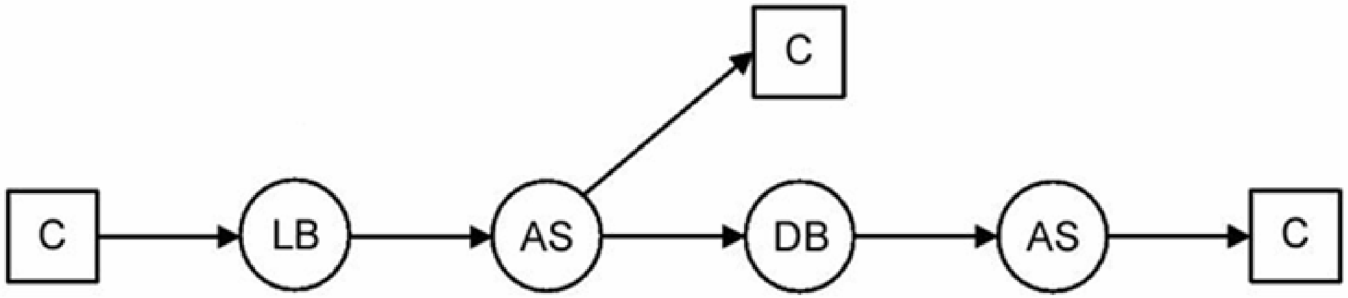
2. Characterize the System

SYSTEM COMPONENT DETAILS

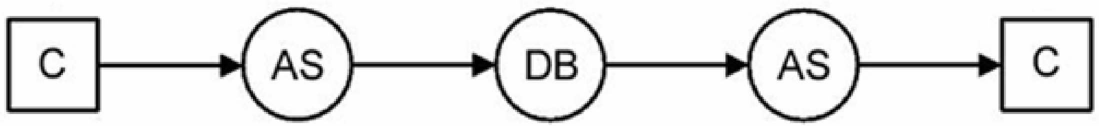
Component	Description
Load Balancer	WebLogic 8.1 Server (HttpClusterServlet) 1 x AMD Athlon XP2000+ CPU 1 GB RAM, SuSE Linux 8
App. Server Cluster Nodes	WebLogic 8.1 Server 1 x AMD Athlon XP2000+ CPU 1 GB RAM, SuSE Linux 8
Database Server	Oracle 9i Server 2 x AMD Athlon MP2000+ CPU 2 GB RAM, SuSE Linux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



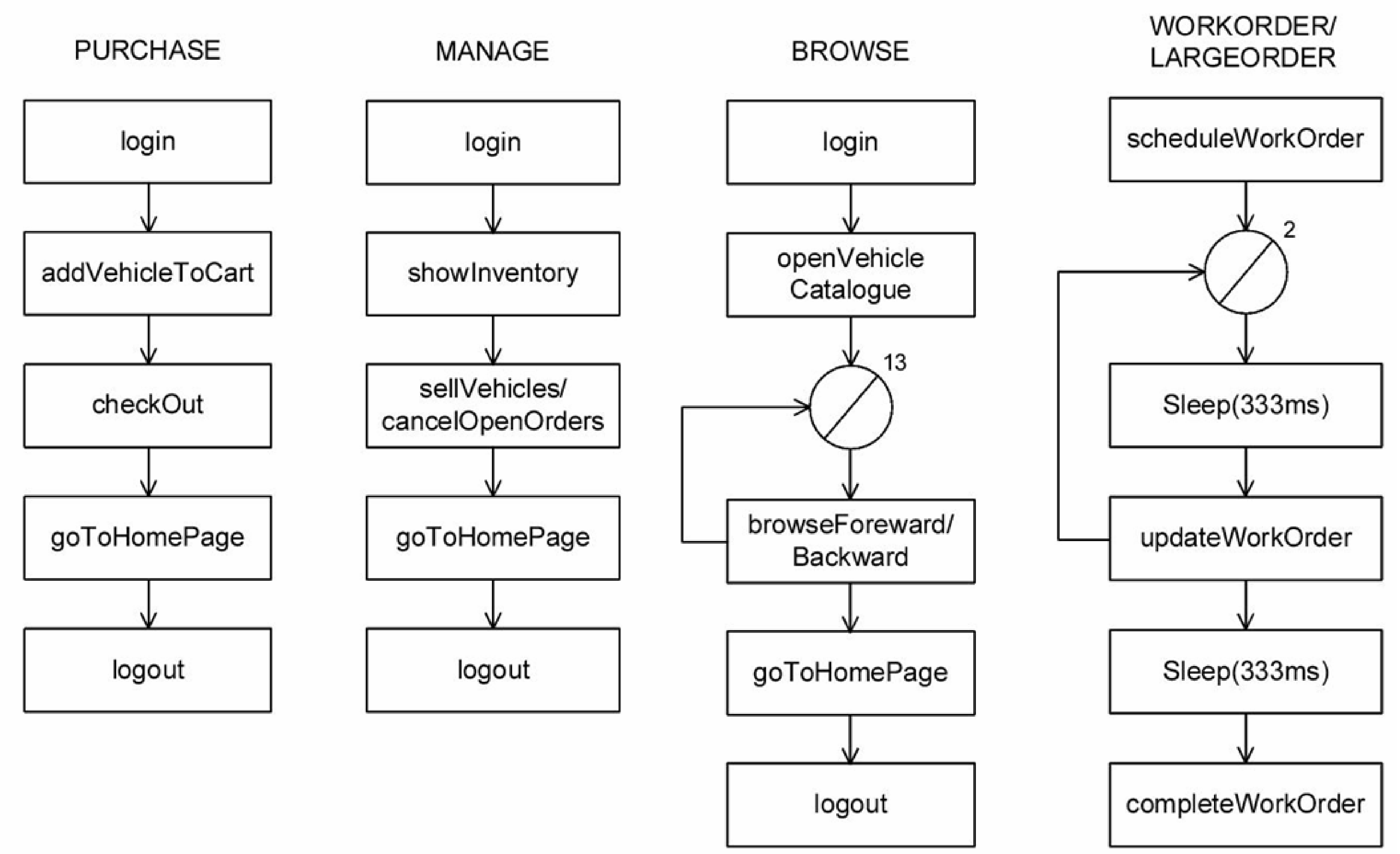
(A). Subtransactions of Browse, Purchase and Manage



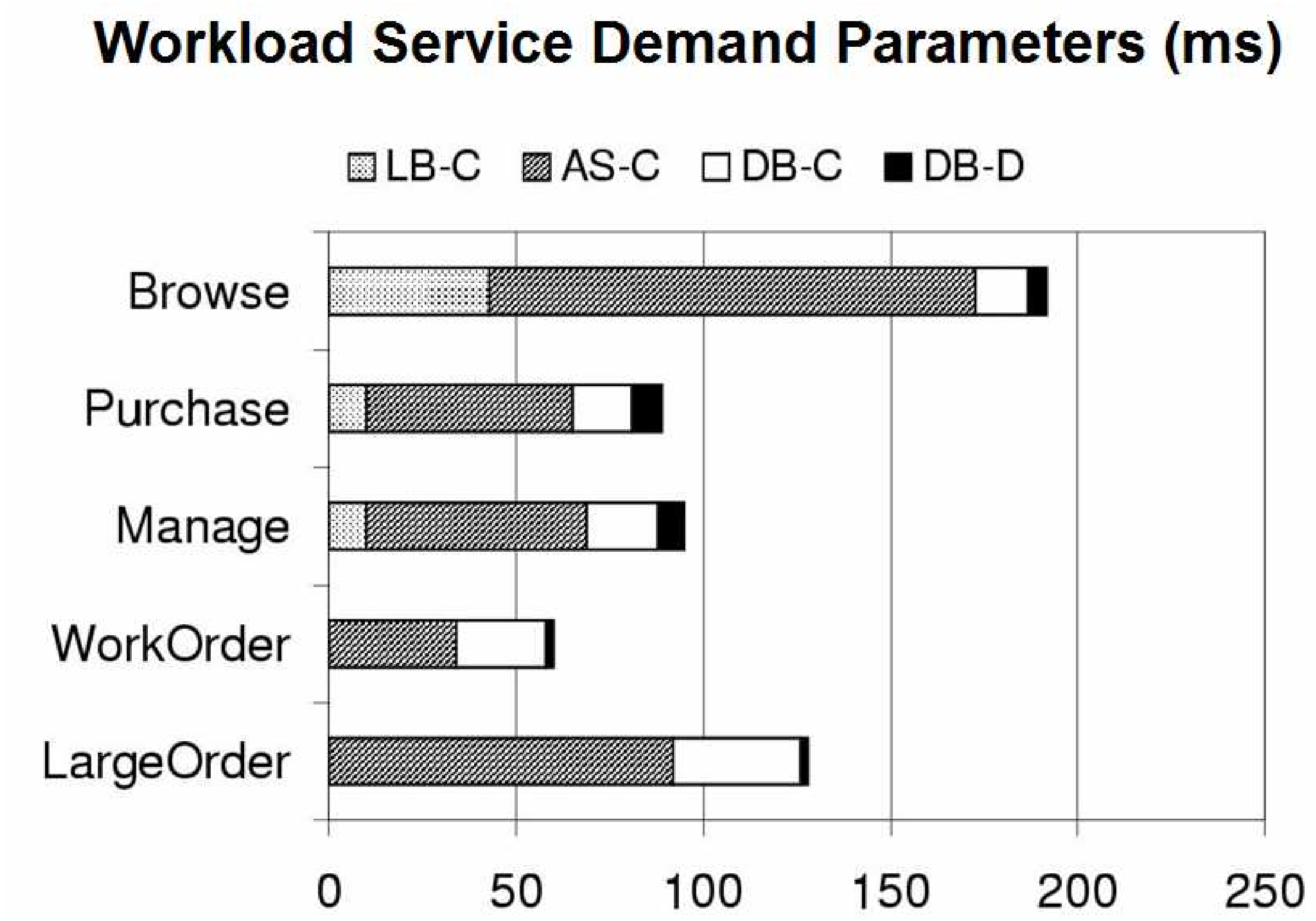
(B). Subtransactions of WorkOrder and LargeOrder

3. Characterize the Workload (2)

Describe the processing steps (subtransactions).



3. Characterize the Workload (3)

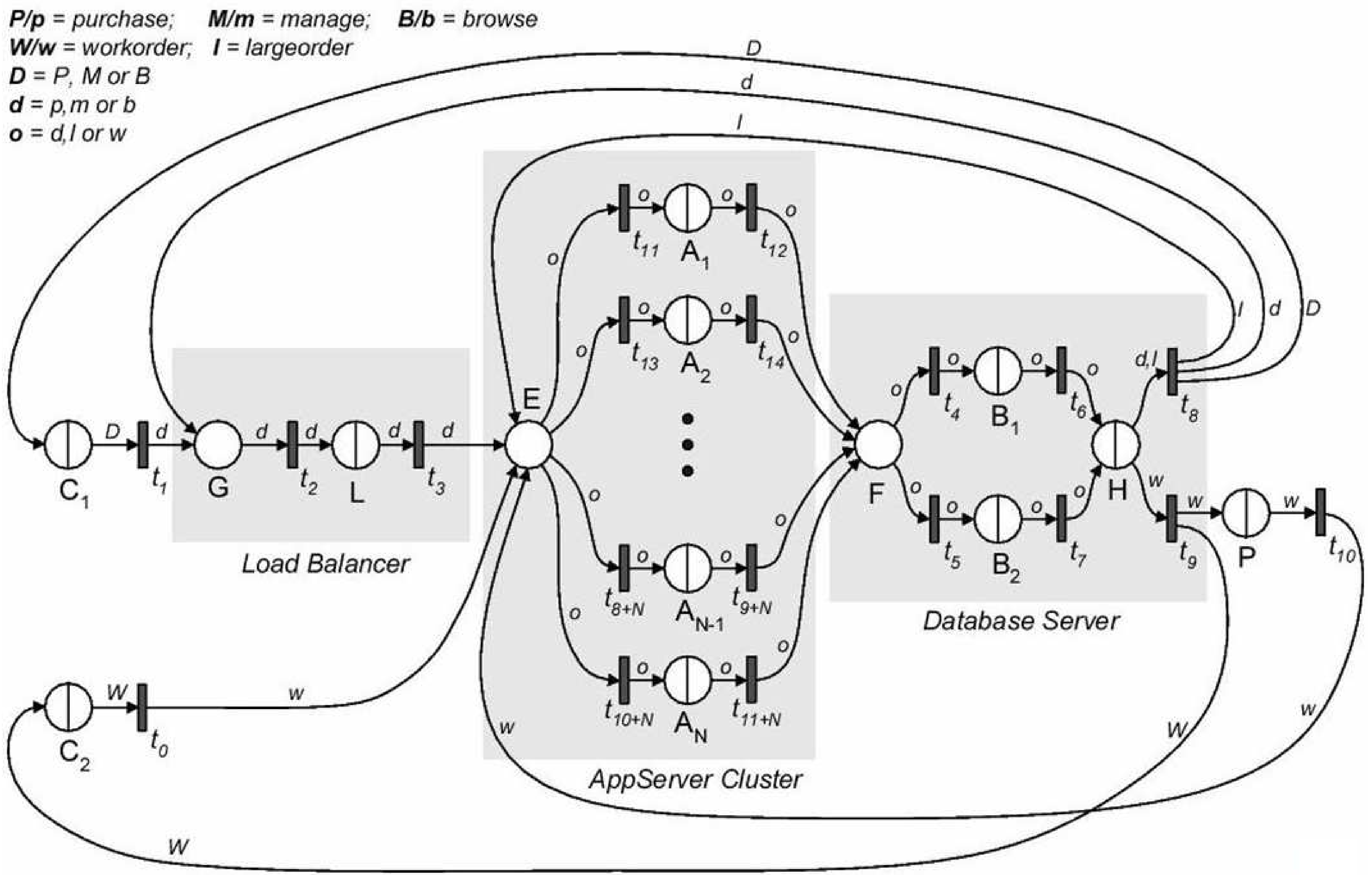


3. Characterize the Workload (4)

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



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%

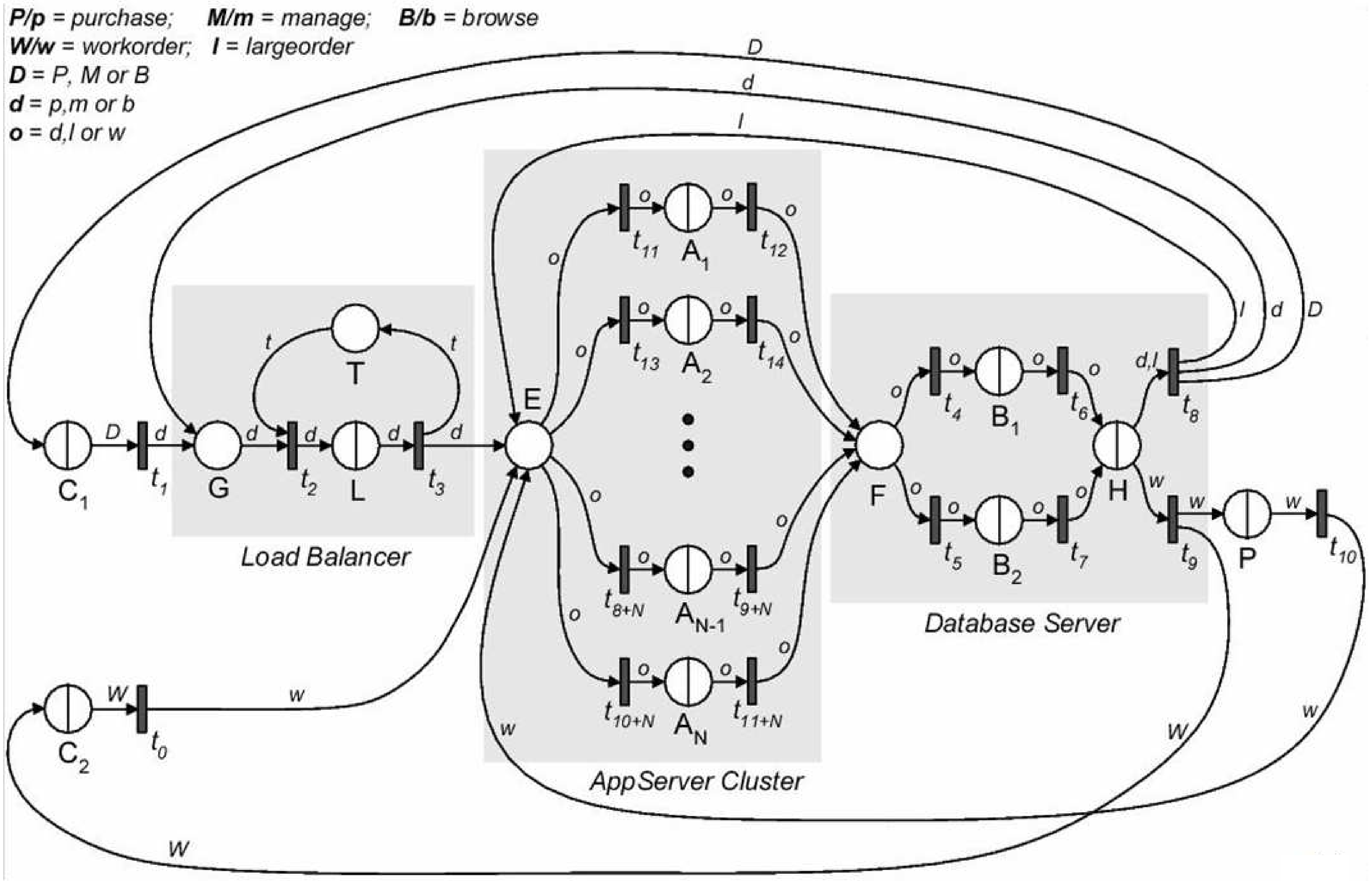
6. Predict System Performance (2)

ANALYSIS RESULTS FOR SCENARIOS UNDER 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 (3)



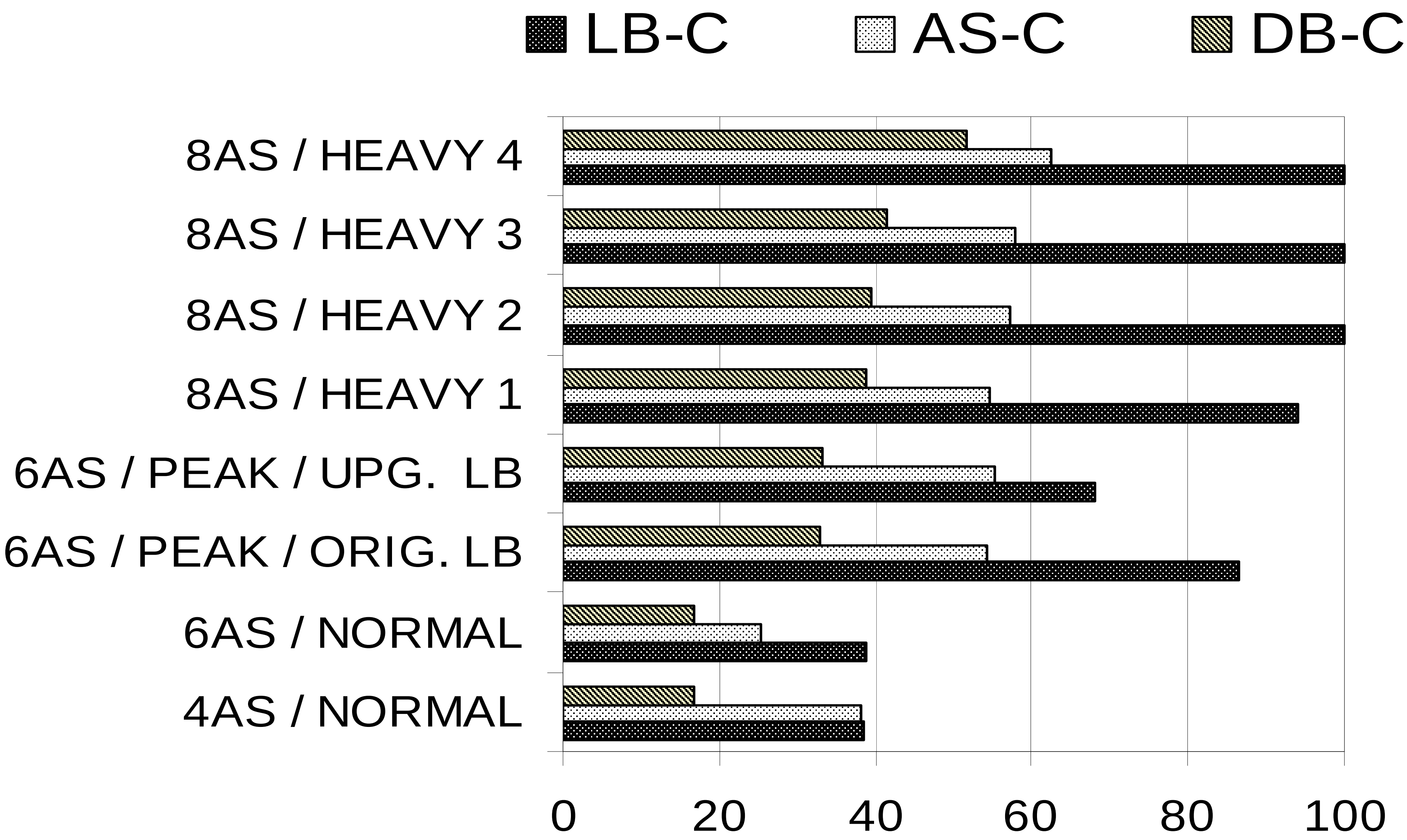
6. Predict System Performance (4)

	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



Roadmap

- Introduction to Queueing Petri Nets
- Modeling Case Studies
  - Modeling Distributed Component Systems
  - Modeling Event-Based Systems
  - Online QoS Control in Grid Environments
- Concluding Remarks

Event-Based (EB) Systems

- Originally motivated by the need for *decoupled* and *asynchronous* information dissemination in large-scale information-driven applications:
  - Stock trading
  - Internet-wide news distribution
  - Air traffic control
  - Electronic auctions
- More recently, gaining attention in other domains:
  - Manufacturing, supply chain management
  - Transportation, health-care and others
- Publish/subscribe now a building block in major new software architectures including ESB, EAI, SOA and EDA.

Challenges Faced

1. What performance would a deployment of the system exhibit?
   - ✓ Event throughput?
   - ✓ Event notification latency and hop count?
   - ✓ Utilization of system components?

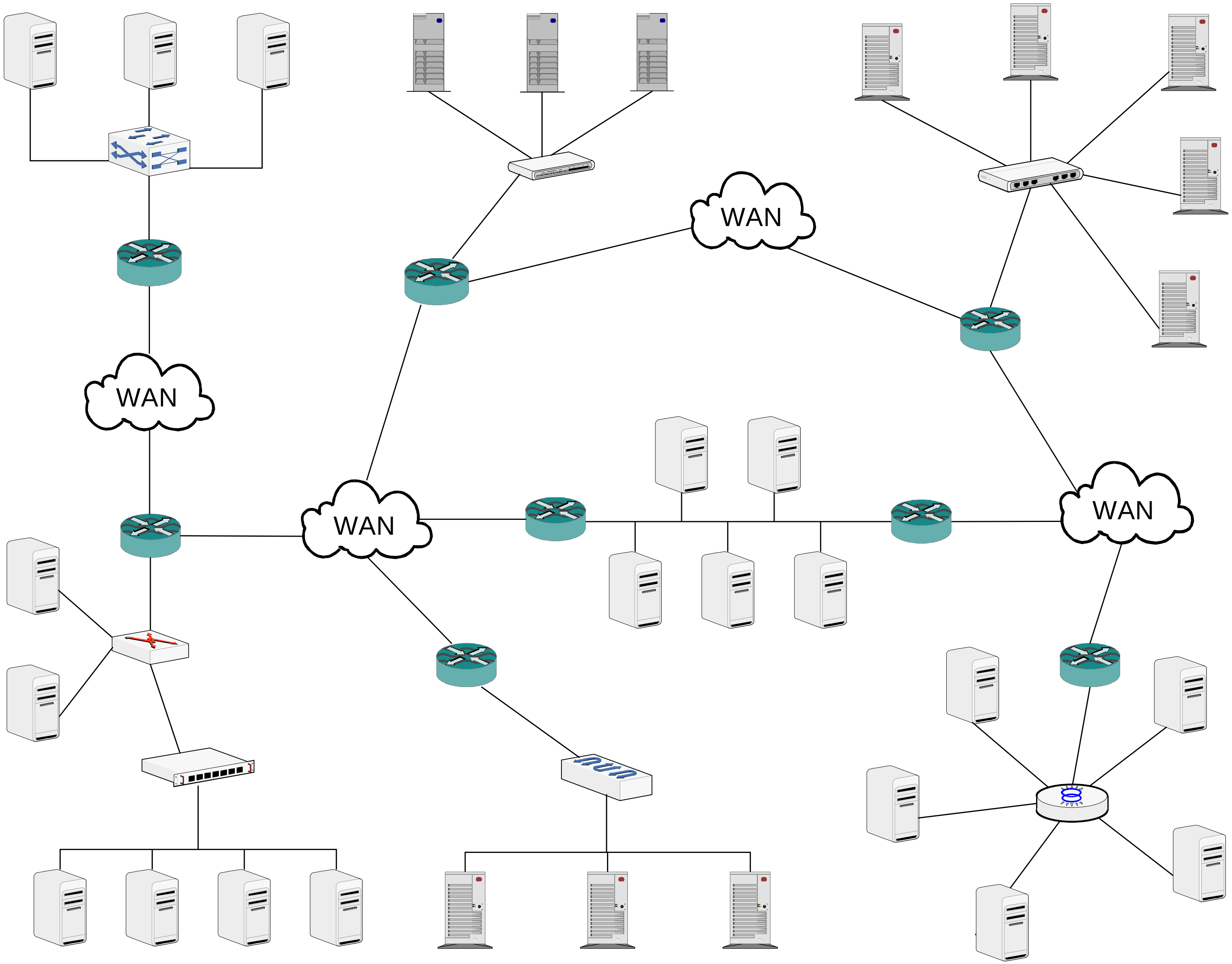
2. What maximum load would the system be able to handle?
   - ✓ Max # publishers, # subscribers, event publication rates

3. How much hardware would be needed to meet SLAs?

4. What would be the optimal broker topology?

5. How to validate the scalability of the application design?

Example EB System Infrastructure



Architecture Model of DEBS

Event Matching Layer

Predicate indexing algorithms

Testing network algorithms

Event Routing Layer

Event flooding
Subscription flooding

Filtering-based
Rendezvous

Basic gossiping
Informed gossiping

Overlay Layer

Broker network

P2P structured overlay

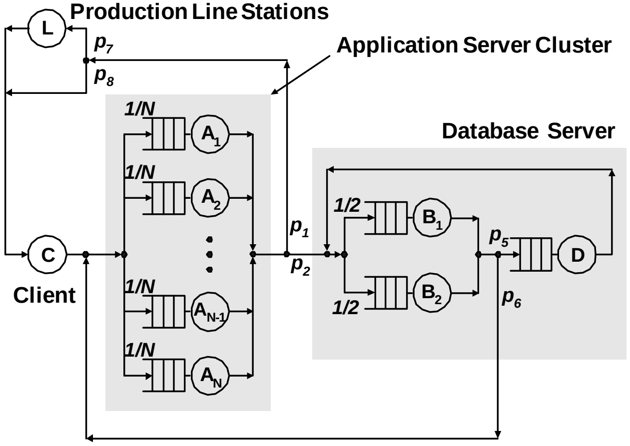
P2P unstructured overlay

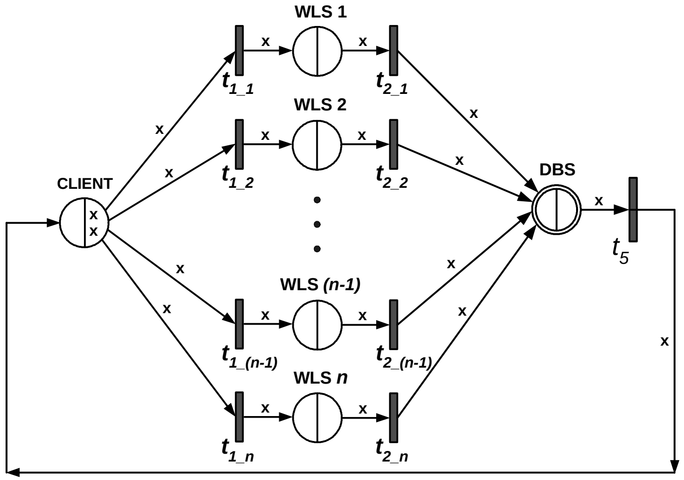
Network Layer

TCP UDP IP multicast RMI IIOP SOAP 802.11g 802.15.4

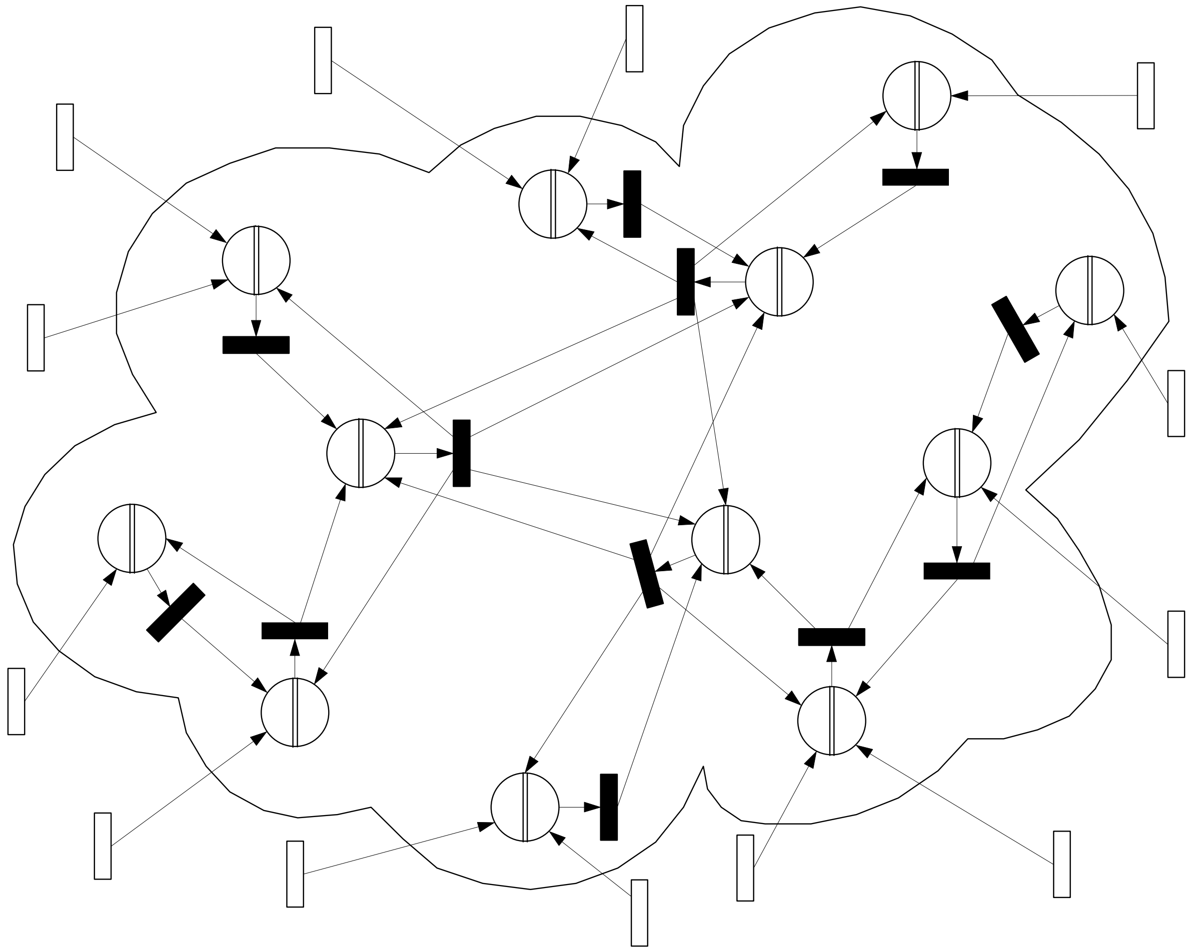
Analytical Analysis

- Used basic operational laws to derive performance metrics as functions of measured routing probabilities and service times
- Obtained approximations for the event delivery times
- For accurate performance prediction, a more detailed performance model must be built
- For example, queueing network or queueing Petri net





Hierarchical Queueing Petri Net

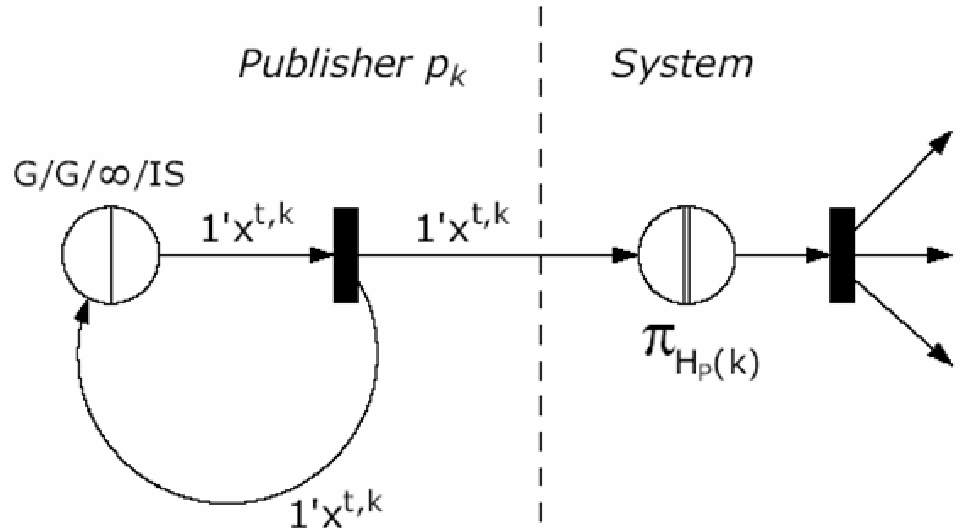


- System nodes modeled as nested QPNs.
- Each with single output transition → can set routing probabilities locally!

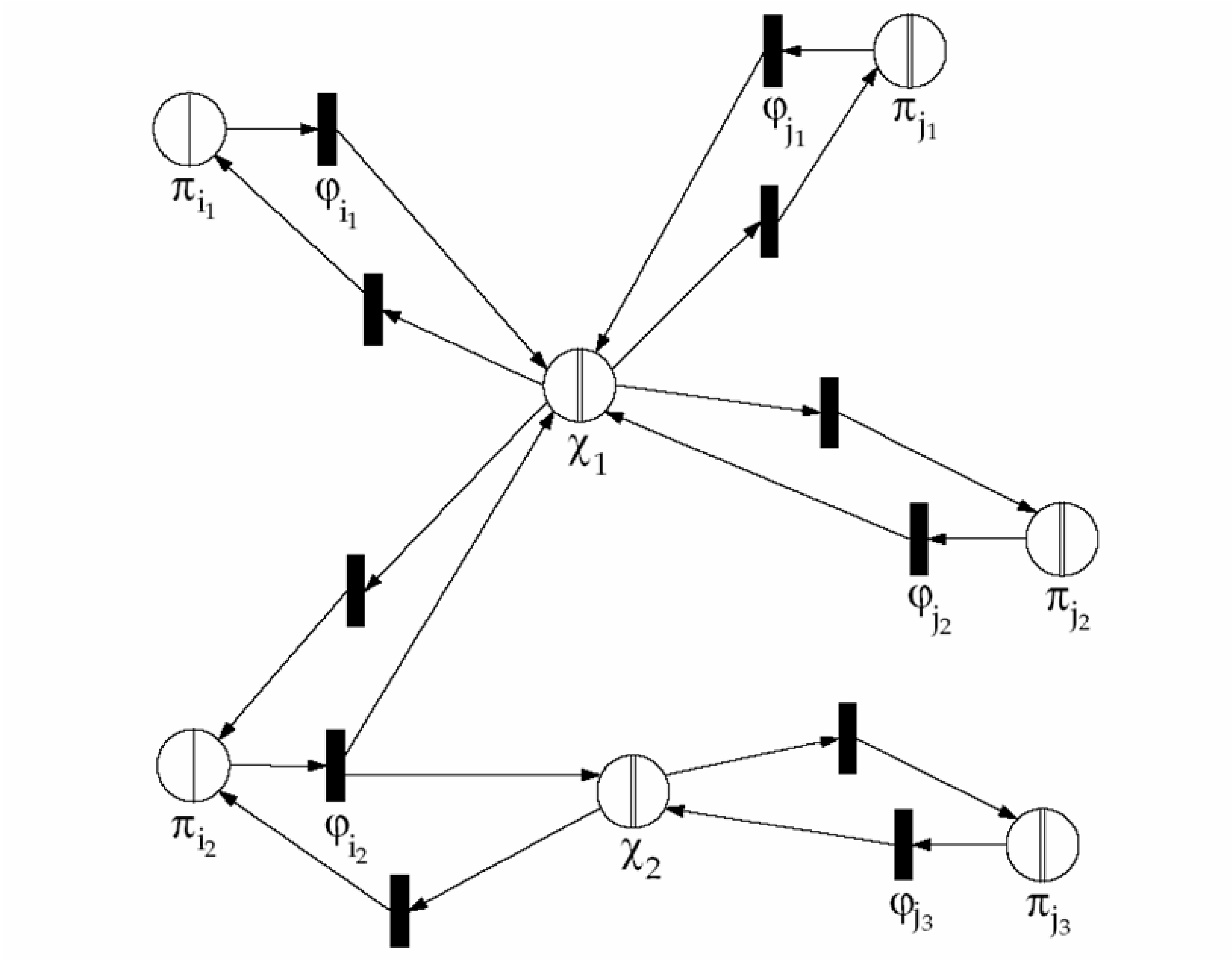
Modeling Non-Possion Event Publications

- Assume *non-exponential* distribution of the time between successive event publications

- Use queueing place with the respective service time distribution



Modeling Network Connections



Automated Workload Characterization

- DEBS subject to the following types of dynamic changes:

WORKLOAD

- Publishers/subscribers joining or leaving the system.
- Publishers changing their event publication rates.
- Subscribers altering their subscriptions.
- Addition of new event types.

CONFIGURATION

- Nodes joining the system.
- Nodes leaving the system (e.g. due to failures).
- Addition of new network links.
- Removal or failure of network links.

Automated Workload Characterization (2)

- Need to monitor the following workload parameters:

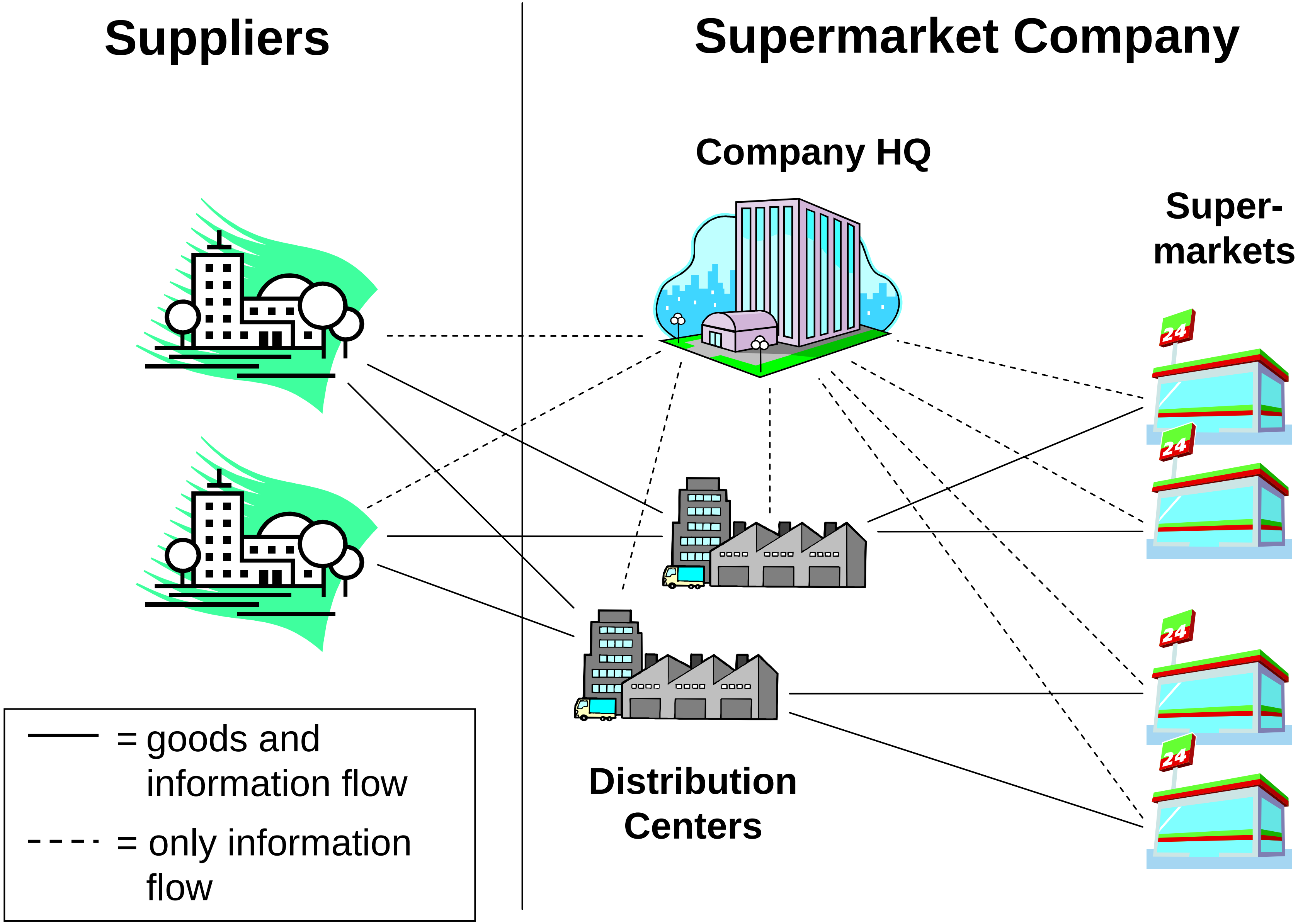
LOCAL

- Event publication rates, arrival rates, throughput
- Routing probabilities
- Node utilization (CPU, I/O)

GLOBAL

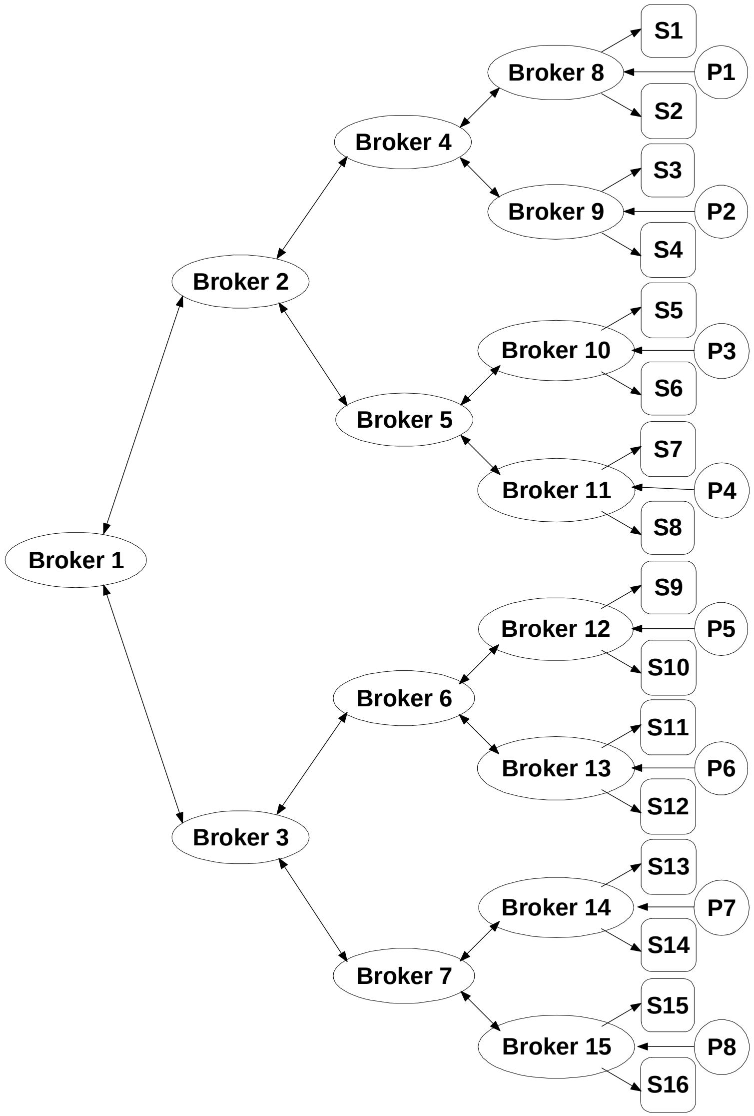
- Sets of system nodes, connections, network links
- Sets of publishers, subscribers, event types
- Utilization of network links
- Dissemination trees of events of interest
- Event delivery latencies over delivery paths of interest

Motivating Application



Modeling Network Connections

- Modeled dissemination of requests for quotes (as in the SPECjms2007 scenario)
- Hierarchical broker topology
  - 15 brokers
  - 8 publishers
  - 16 subscribes
- Used SIENA pub/sub system
- Enhanced with self-monitoring functionality



Performance Prediction

	Scenario 1		Scenario 2	
Broker	Model	Measured	Model	Measured
1	94.66	93.46	61.88	62.11
2	94.65	96.15	61.88	62.11
3	89.93	89.29	59.28	59.17
4	90.40	89.29	58.27	57.80
5	83.42	84.03	56.42	56.18
6	85.24	84.75	56.35	56.18
7	71.90	71.94	48.63	48.54
8	78.91	79.37	51.12	51.28
9	67.15	68.03	43.49	43.48
10	67.14	67.11	47.01	46.95
11	59.54	59.88	41.72	41.67
12	58.26	58.82	40.01	40.16
13	73.09	72.46	48.23	48.08
14	56.35	57.47	38.49	38.46
15	63.11	63.29	42.97	42.92

Broker throughput
(messages / sec)

Delivery latency
(msec)

	Scenario 1		Scenario 2	
Subscriber	Model	Measured	Model	Measured
1	9.48	8.98	24.60	26.71
2	19.01	18.56	24.79	25.93
3	28.82	27.27	7.90	9.05
4	29.03	27.79	16.39	17.59
5	38.34	37.01	32.61	35.20
6	38.00	37.77	32.63	35.52
7	39.06	38.12	33.27	36.25
8	38.71	37.87	33.28	35.47

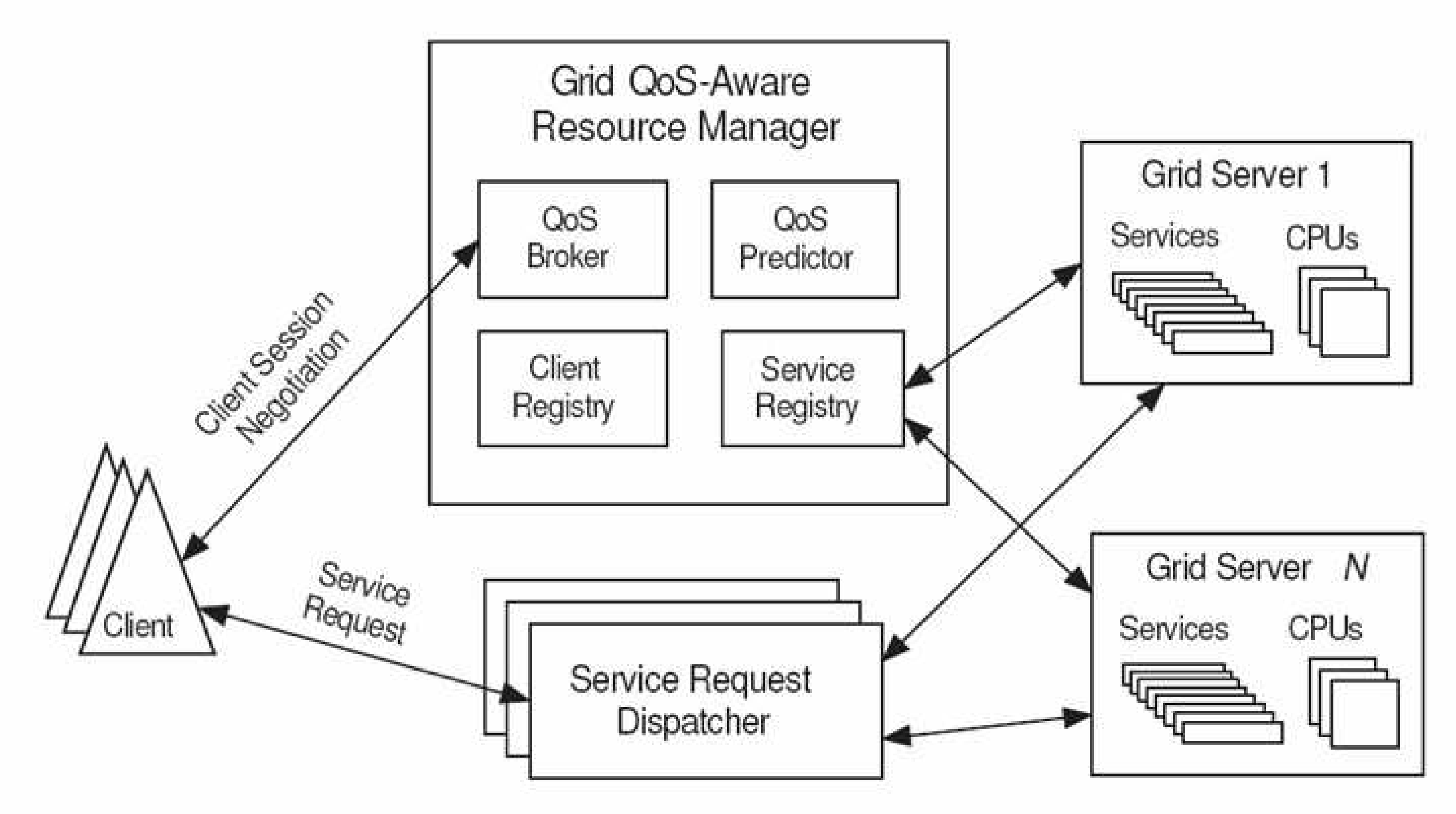
Roadmap

- Introduction to Queueing Petri Nets
- Modeling Case Studies
  - Modeling Distributed Component Systems
  - Modeling Event-Based Systems
  - Online QoS Control in Grid Environments
- Concluding Remarks

Motivation

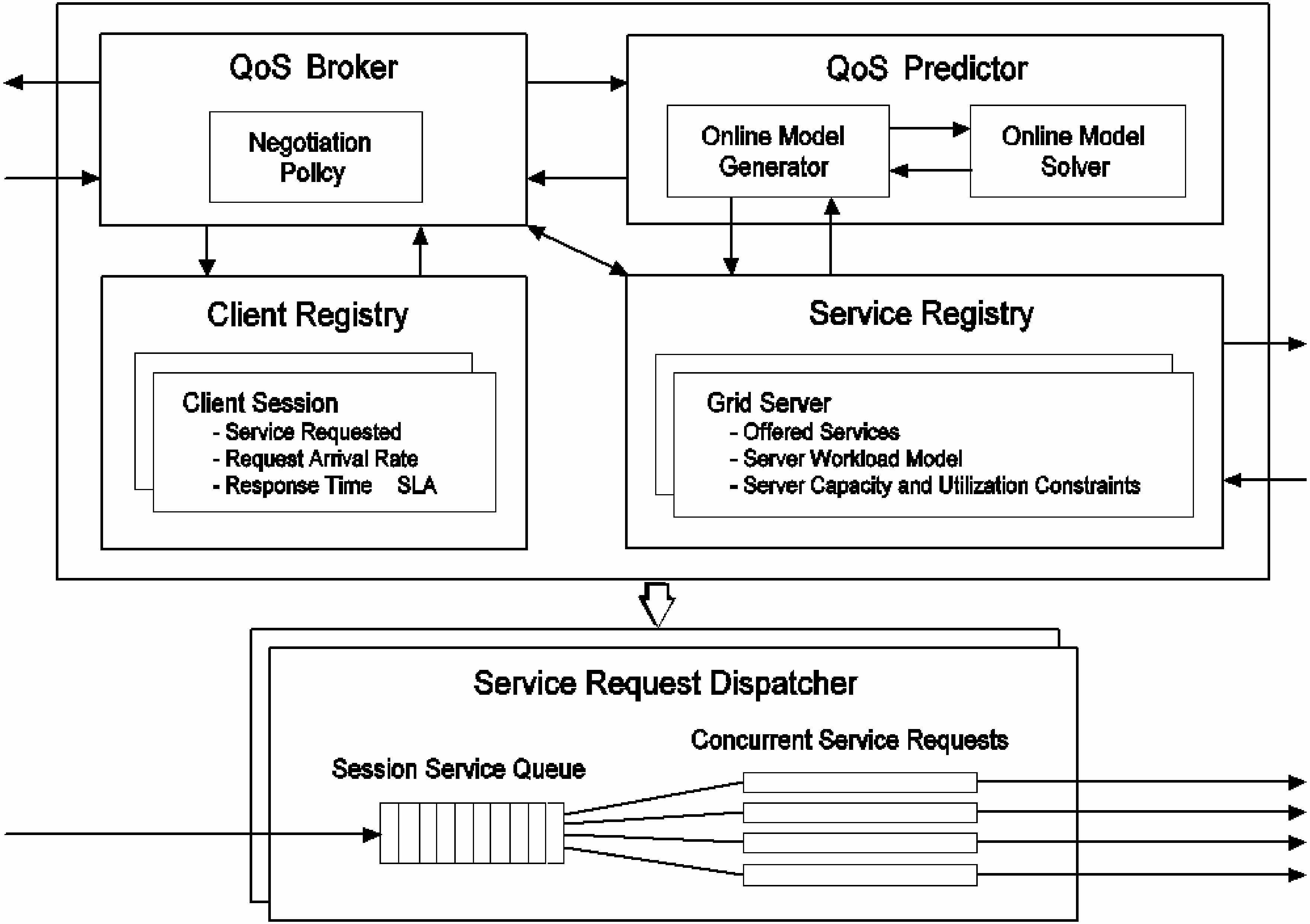
- Grid computing gaining grounds in the enterprise and commercial domains
- Grid and SOA technologies converging
- Enterprise Grid environments highly dynamic
  - Unpredictable workloads
  - Non-dedicated resources
- QoS management a major challenge
- Off-line capacity planning no longer feasible
- On-the-fly performance prediction needed

Resource Manager Architecture

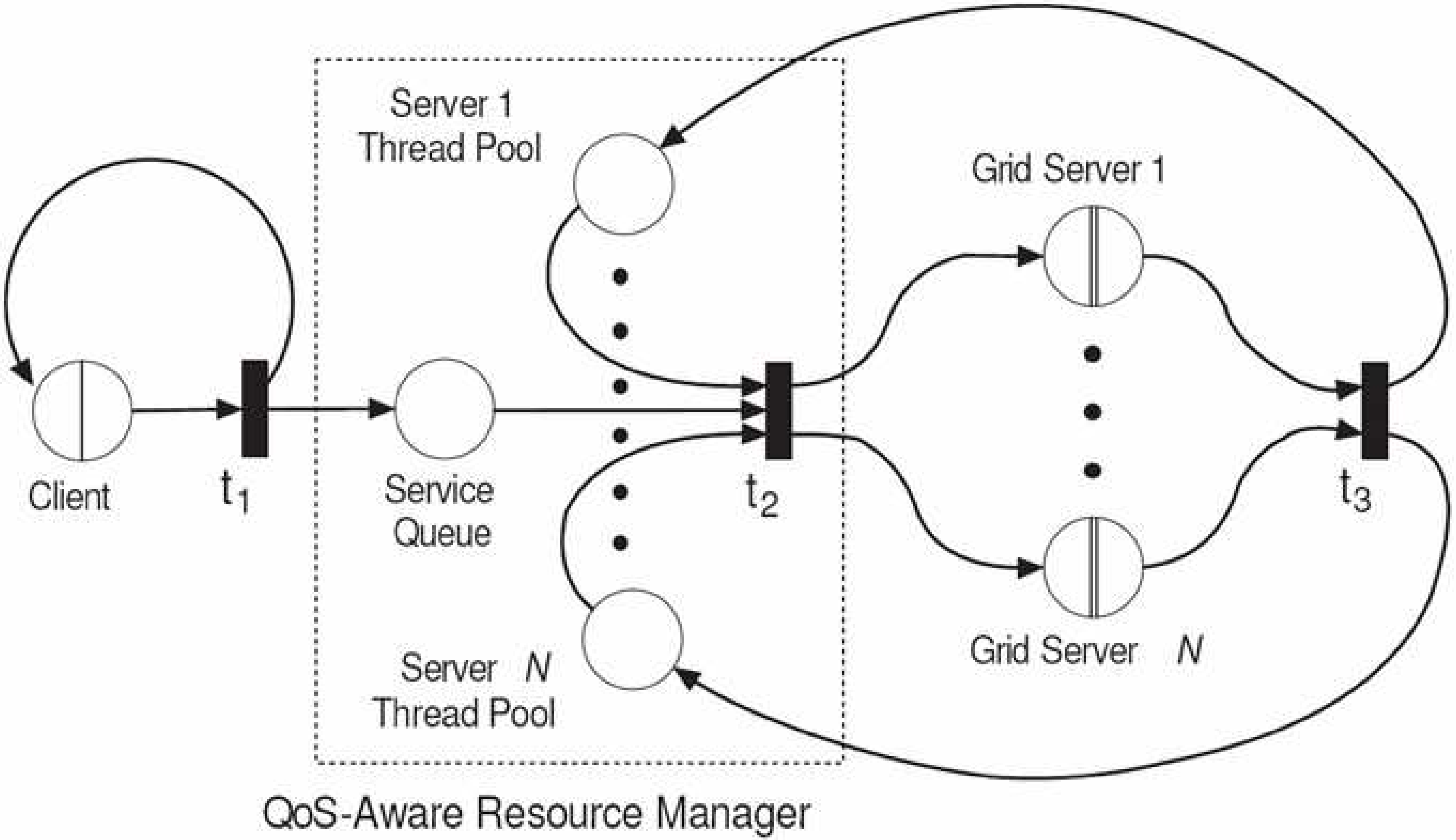


Joint work with Ramon Nou and Jordi Torres (UPC).

Resource Manager Architecture (2)



QoS Predictor

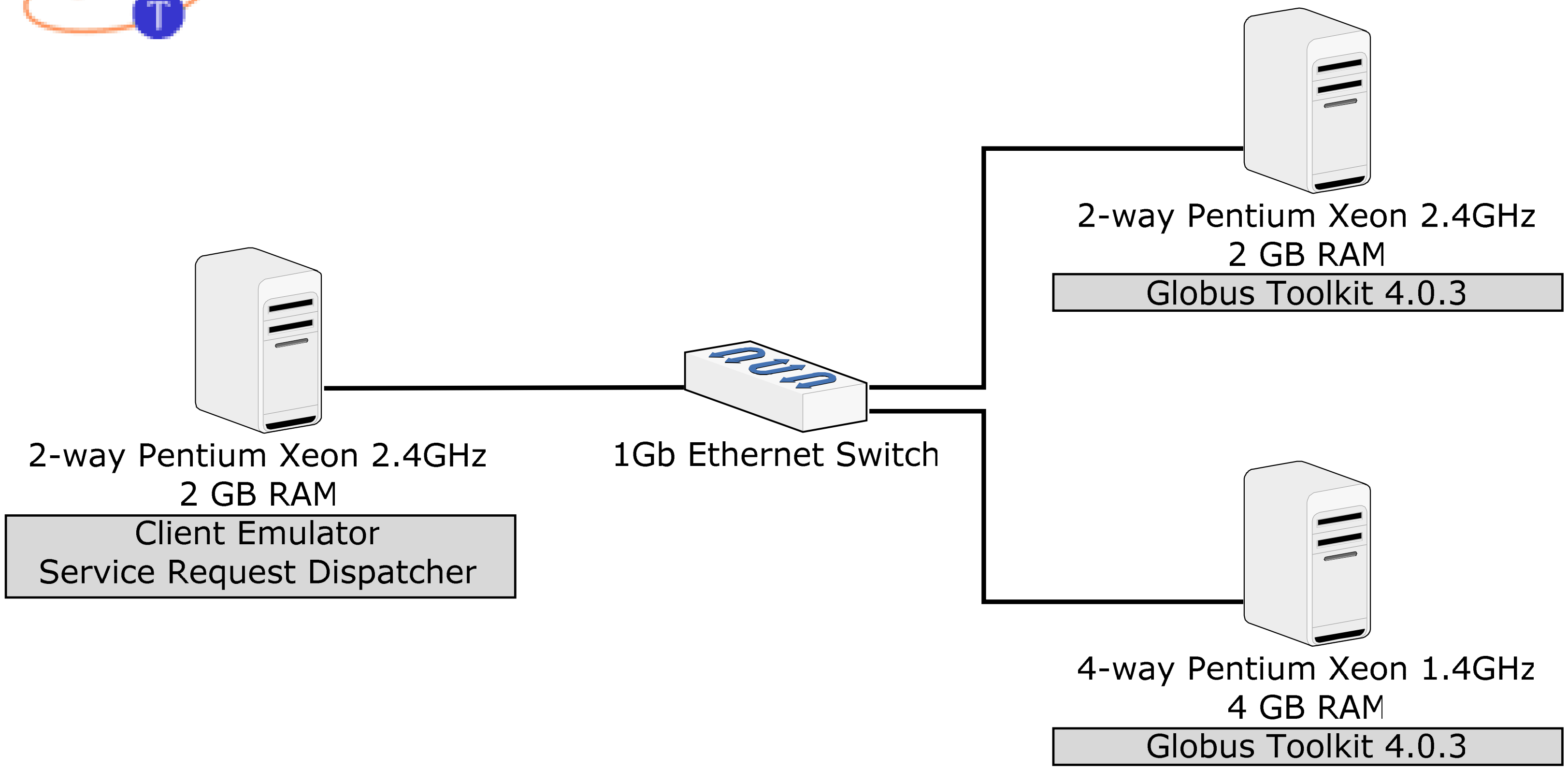


Resource Allocation Algorithm

- New session request (v, λ, ρ) arrives
- Assign new session unlimited # threads on each server
- If required throughput cannot be sustained, reject request
- For each over-utilized server limit the number of threads
- If an SLA of an active session is broken, reject request
- Else, SLA of new session broken, send *counter offer*
- Else accept request

Deployment Environment

the globus toolkit



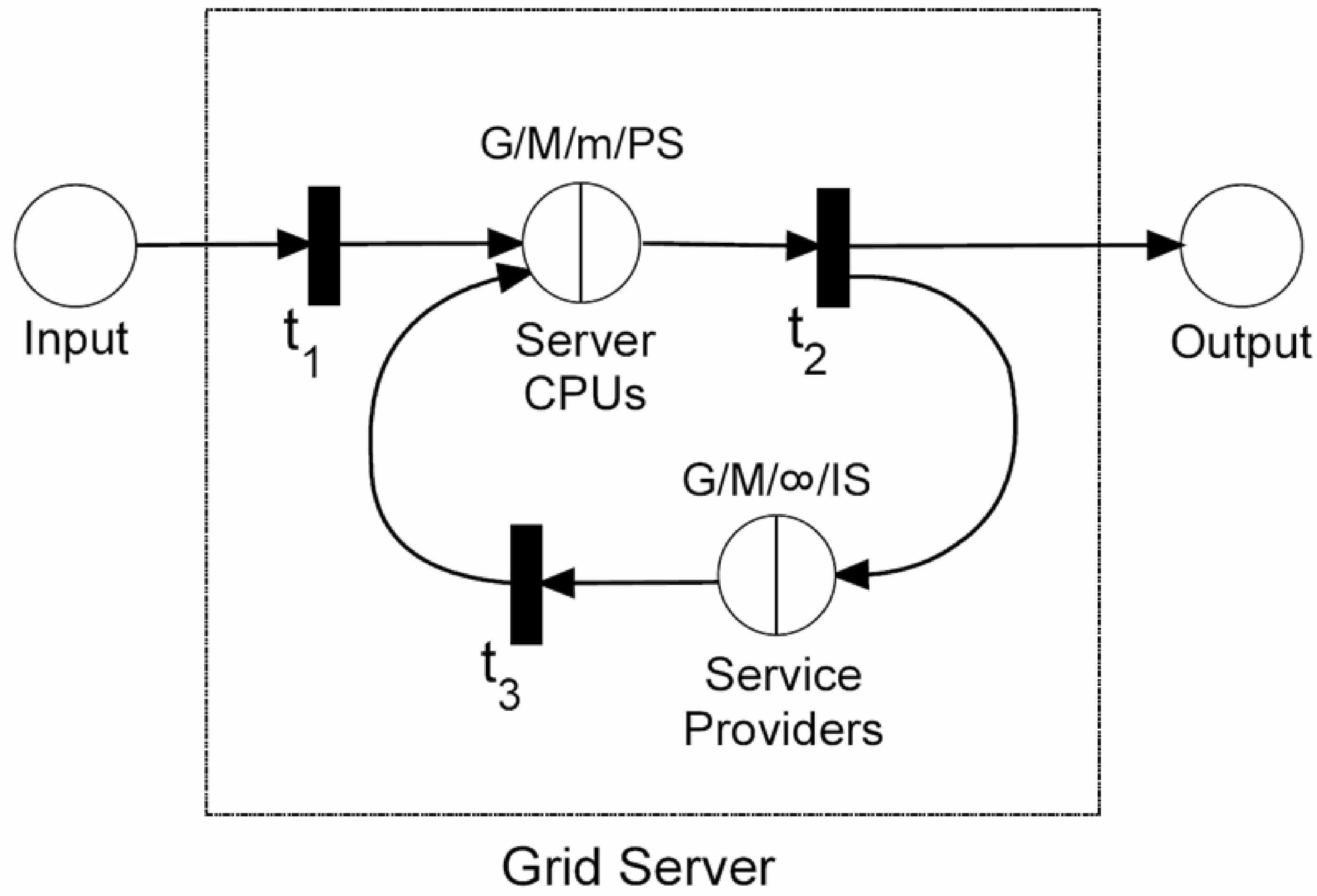
Sample Workload

- Assume three services available
- Each service
  - executes CPU-intensive business logic
  - might call external third-party services
- Service workload model

Service	Service 1	Service 2	Service 3
CPU service demand on 2-way server	6.89	4.79	5.84
CPU service demand on 4-way server	7.72	5.68	6.49
External Service Provider Time	2.00	3.00	0.00

- Workload models stored in service registry

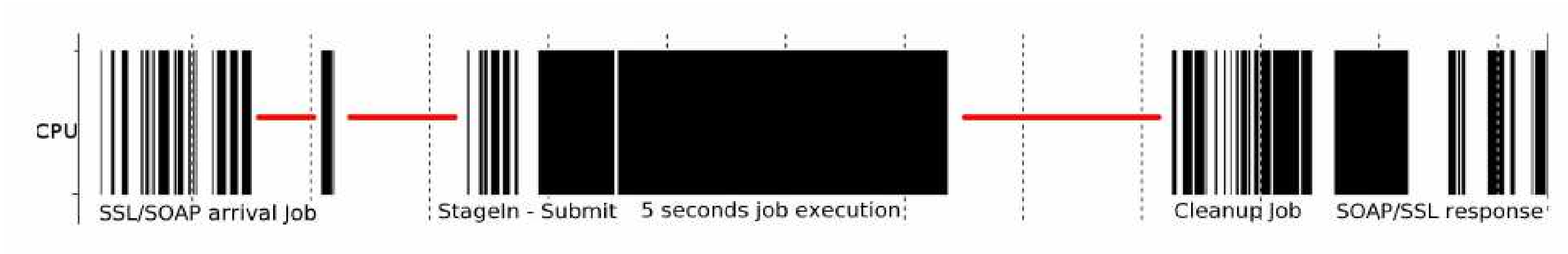
Grid Server Model



Model Validation and Calibration

Services	No of threads allocated	Request interarrival time (sec)	Request response time (sec)		Error (%)
			measured	predicted	
2	unlimited	4	11.43	10.47±0.033	8.3%
1—3	unlimited	8 / 8	13.66 / 12.91	12.21±0.019 / 11.17±0.031	11% / 13%
3	5	2.5	10.93	8.14±0.030	**25%**
1—3	2/2	8 / 8	18.15 / 9.79	15.58±0.23 / 7.8±0.05	14.1% / **20.3%**

- Model failed initial validation attempt
- Service execution trace (BSC-MF / Paraver)

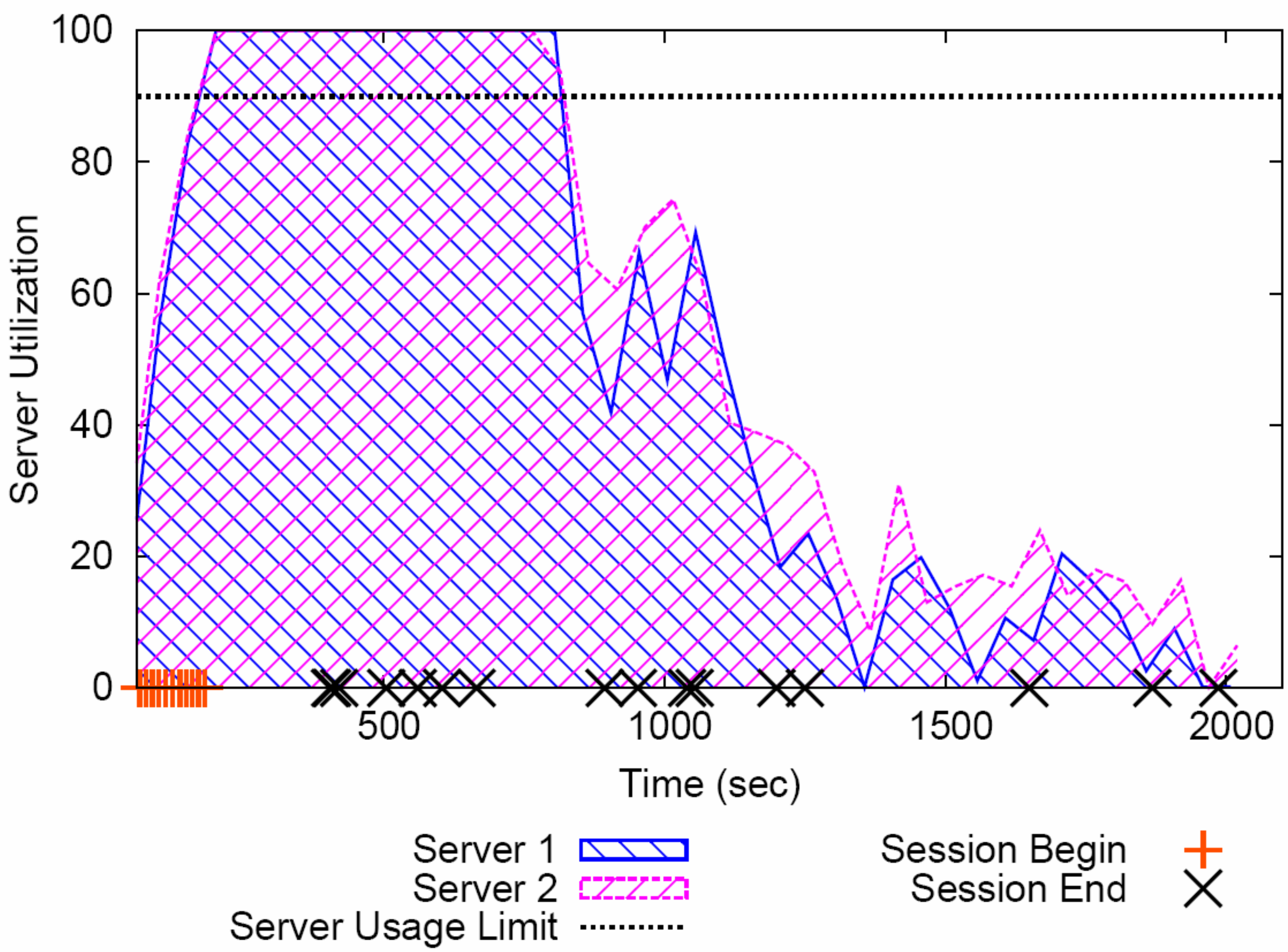


- Calibrated model by adding the 1 sec delay

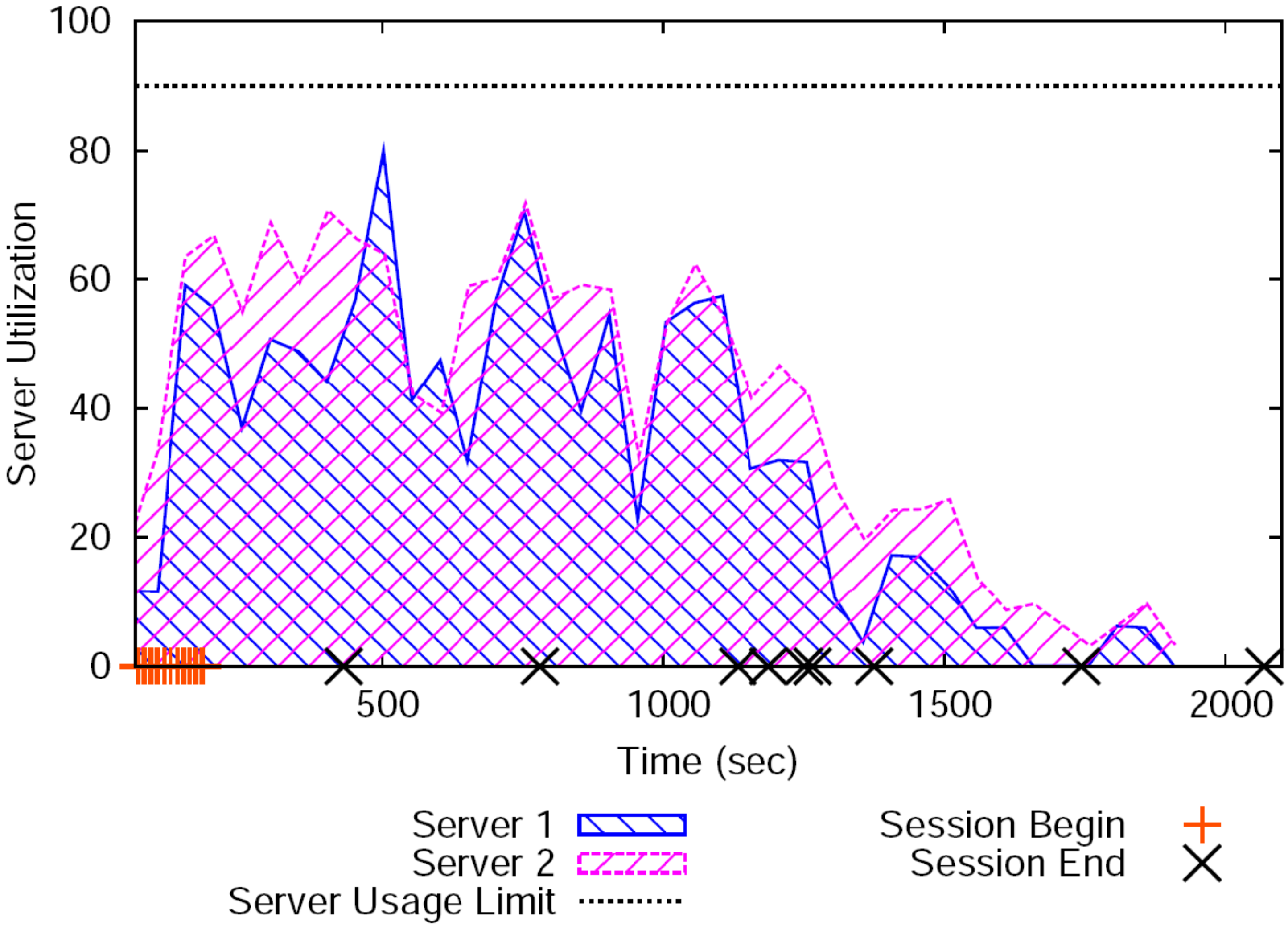
Scenario 1

- 16 session requests
- Run until all sessions complete
- Each session has 20-120 service requests (avg. 65)
- SLAs between 16 and 30 sec
- 90% maximum server utilization constraint
- Will compare two configurations
  - Without QoS Control
    - Incoming requests simply load-balanced
  - With QoS Control
    - QoS-aware admission control enforced

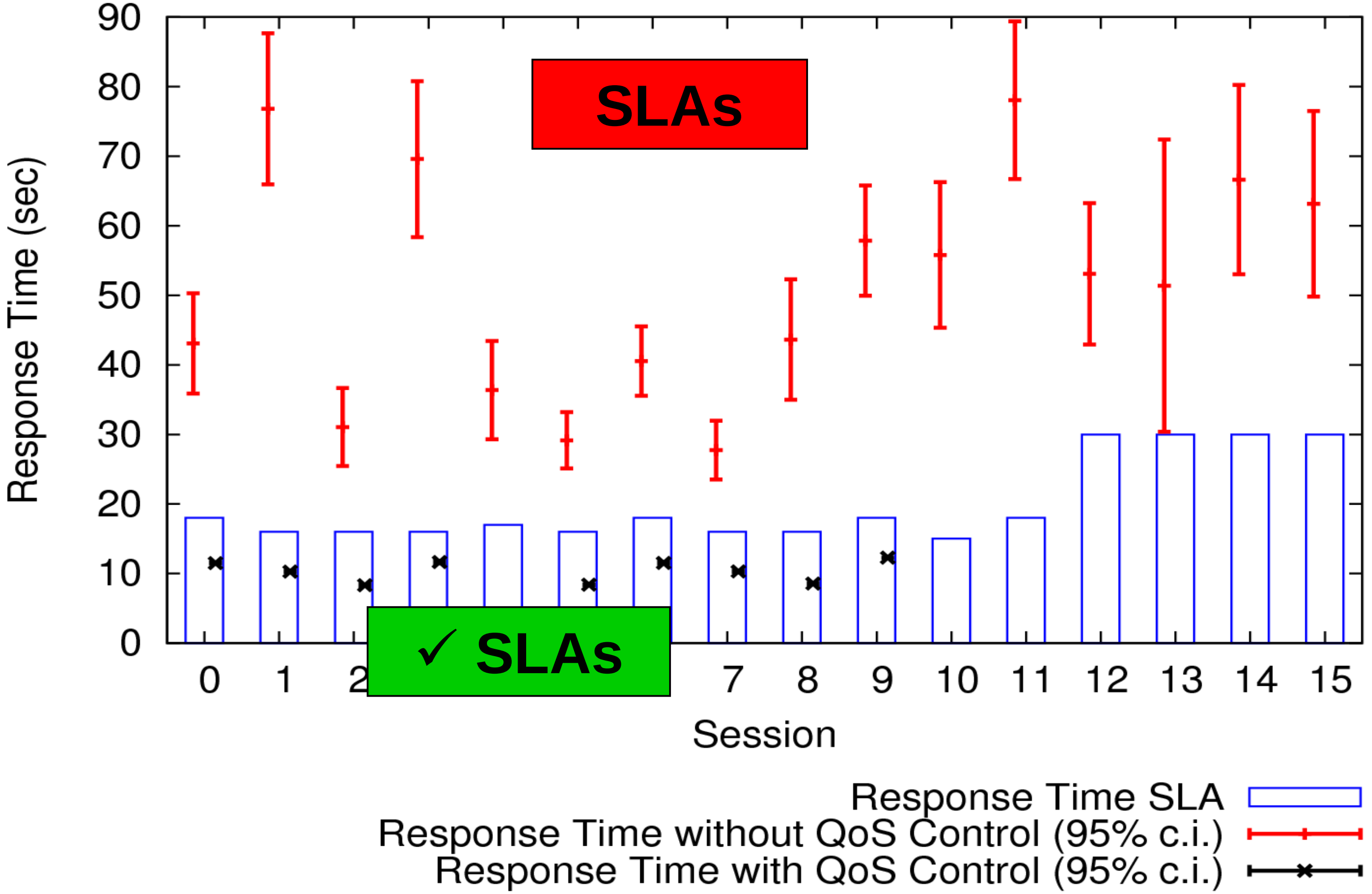
CPU Utilization – Without QoS Control



CPU Utilization – With QoS Control



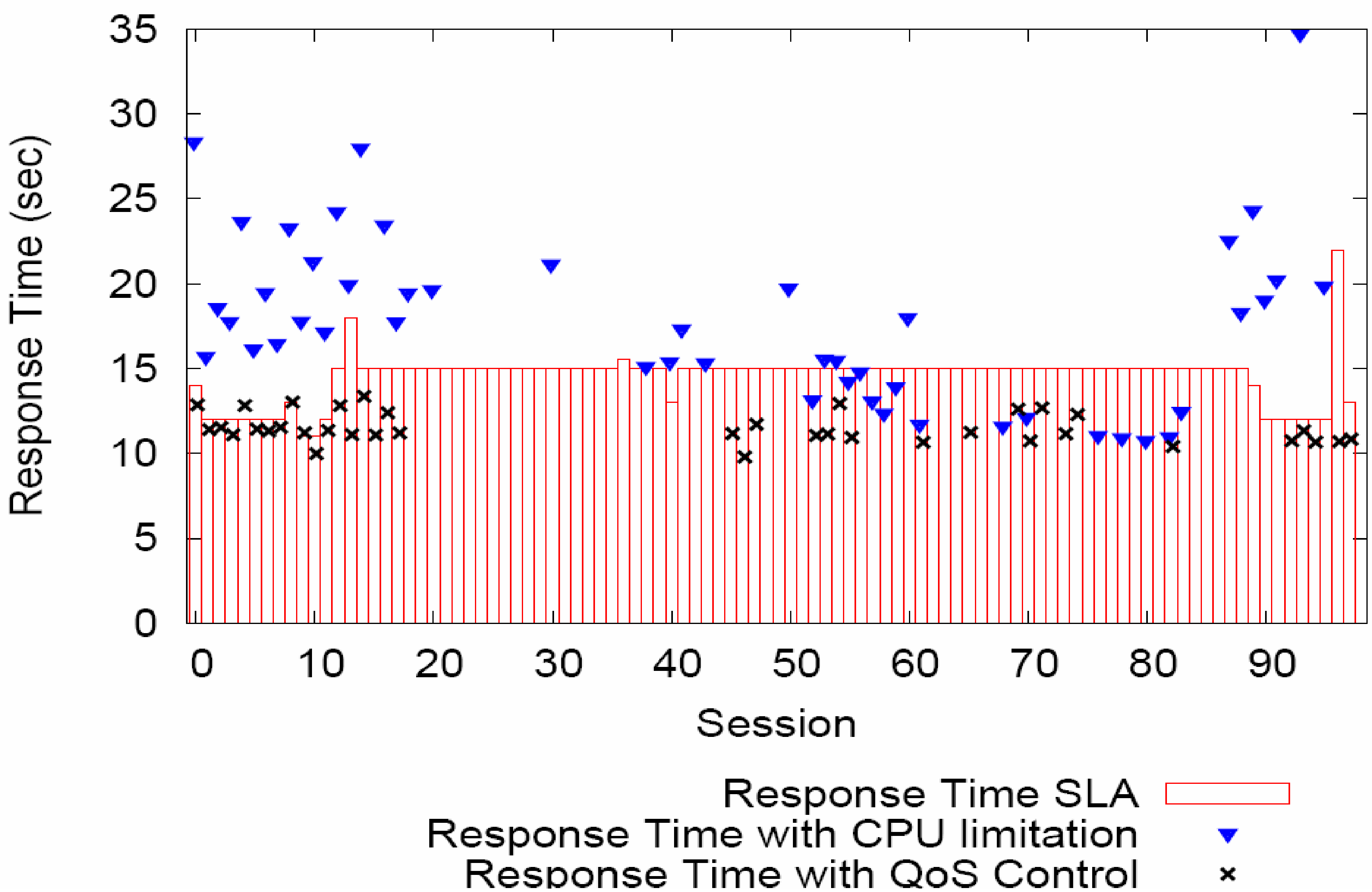
Average Session Response Times



Scenario 2

- 99 session requests executed over period of 2 hours
- Run until all sessions complete
- Average session duration 18 minutes (92 requests)
- 90% maximum server utilization constraint
- Will compare two configurations
  - Without QoS Control
    - Incoming requests simply load-balanced
    - **Reject session requests when servers saturated**
  - With QoS Control
    - QoS-aware admission control enforced

Scenario 1


35
30
25
20
15
10
5
0
Response Time (sec)
0
10
20
30
40
50
60
70
80
90
Session
Response Time SLA
Response Time with CPU limitation
Response Time with QoS Control

Overhead for QoS Control

- Time to reach a decision in Sc. 1 < 11 sec
- Several approaches to boost performance
  - Speed up model analysis
    - Distribute simulation to utilize multi-core CPUs
    - Use analytical product form solution techniques
  - Optimize resource allocation algorithm
    - Allocate resources bottom up instead of top down
    - Cache analyzed configurations
    - Aggregate sessions of the same type
    - Generate model of minimal size
- Subject of on-going and future work

Summary

- Presented three case studies using QPN models
  - Capacity planning for distributed component systems (SPECjAppServer2004)
  - Performance prediction of event-based systems
  - Online QoS Control in Grid environments
- Hierarchical Queueing Petri Nets
  - Well suited to modeling distributed component systems
  - Balance between model complexity and expressiveness
  - Flexibility in choosing the level of detail and accuracy
  - Integration of hardware and software aspects
  - Hierarchical structures facilitate model composition
  - Intuitive graphical representation
- Balancing accuracy and speed is a major challenge

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.

Further Reading (2)

S. Kounev, R. Nou and J. Torres, *Autonomic QoS-Aware Resource Management in Grid Computing using Online Performance Models*, In Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS-2007), Nantes, France, October 23-25, 2007.

R. Nou, S. Kounev and J. Torres, *Building Online Performance Models of Grid Middleware with Fine-Grained Load-Balancing: A Globus Toolkit Case Study*, In Formal Methods and Stochastic Models for Performance Evaluation, Springer LNCS 4748/2007, Proceedings of the 4th European Performance Engineering Workshop (EPEW-2007), Berlin, Germany, September 27-28, 2007.

S. Kounev and A. Buchmann, *On the Use of Queueing Petri Nets for Modeling and Performance Analysis of Distributed Systems*, Book chapter to appear in Vedran Kordic (ed.) Petri Net, Theory and Application, Advanced Robotic Systems International, Vienna, Austria, 2007.

Selected papers available for download at

http://www.cl.cam.ac.uk/~sk507

http://www.dvs.tu-darmstadt.de/staff/skounev/

Questions

Thank You for your Attention!