Motivation

• Modern E-Business Systems are gaining in size and complexity, which makes it difficult for deployers to estimate the size and capacity of the deployment environment needed to meet SLAs.

• Deployers are faced with questions such as the following:
  - Does the system scale? Are there potential system bottlenecks?
  - What is the maximum load level that the system is able to handle?
  - What would the avg. response time, throughput and utilization be under the expected workload?
Motivation (contd.)

- The main problem is **how to predict system performance under a particular workload**. Performance models are increasingly used for this purpose.

- **Queueing Networks** and **Generalized Stochastic Petri Nets** are among the most popular models exploited, but they both have some serious disadvantages.

- In this paper we look at a new modelling formalism – **Queueing Petri Nets (QPNs)**, which eliminates these disadvantages.

- We study a real-world e-business system and show how QPN models can be exploited for performance analysis.

---

Agenda

- Traditional Queueing Networks and Petri Nets
- (Hierarchical) Queueing Petri Nets
- Case Study: SPECjAppServer2001
- QPN Performance Models
- Model Analysis and Validation
- Summary and Conclusions
**Queueing Networks (QNs)**

- **QN**: Set of interconnected queues
- **Queue**: waiting area and servers
- Scheduling strategies (FCFS, PS, ...)
- Single-class vs. multi-class
- Open, closed or mixed

**PROS**: Very powerful for modelling **hardware contention** and scheduling strategies. Many efficient analysis techniques available.

**CONS**: Not suitable for modelling blocking, synchronization, simultaneous resource possession and **software contention** in general. Although Extended QNs provide some limited support for the above, they are very restrictive and inaccurate.

---

**Petri Nets (PNs)**

- **PN**: places, tokens and transitions. marking, transition enabling/firing
- **CPNs**: allow tokens of different colors and transition modes
- **GSPNs**: allow timed transitions
- **CGSPNs**: CPNs + GSPNs

**PROS**: Suitable both for qualitative and quantitative analysis.

Lend themselves very well to modelling blocking, synchronization, simultaneous resource possession and software contention.

**CONS**: No direct means for modelling scheduling strategies. Not as many algorithms/tools for efficient quantitative analysis are available as for Queueing Networks.
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:** Extremely difficult to analyze! 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 and enable larger models to be analyzed.

**Analysis Tools for HQPNs**

Currently only one tool available:

The **HQPN-Tool** from the University of Dortmund.
Supports a number of structured analysis methods.
Available free of charge for non-commercial use.
The SPECjAppServer2001 Benchmark

- Heavy-duty B2B E-Commerce Benchmark
- Successor of Sun's ECperf™ 1.1 Benchmark
- Measures performance and scalability of J2EE App. Servers
- Developed by SPEC OSG Java Subcommittee.
- For more info visit: http://www.spec.org/osg/jAppServer/

SPECjAppServer2001 Business Model

CUSTOMER DOMAIN
Order Entry Application
TXs:
- Place Order
- Change Order
- Get Order Status
- Get Customer Status
Create Large Order

MANUFACTURING DOMAIN
Parts → Planned Lines
Large Order Line → Widgets
Transactions: (TXs)
- Schedule Work Order
- Update Work Order
- Complete Work Order
- Create Large Order

CORPORATE DOMAIN
Customer, Supplier, and Parts Info
TXs:
- Check Credit
- Get Percent Discount
- New Customer

SUPPLIER DOMAIN
TXs:
- Send Purchase Order
- Deliver Purchase Order

Dagstuhl-2003 © S. Kounev and A. Buchmann
Case Study: Order Entry Application

CUSTOMER DOMAIN
Order Entry Application
TXs:
- Place Order
- Change Order
- Get Order Status
- Get Customer Status
Create Large Order

MANUFACTURING DOMAIN
Planned Lines
Large Order Line
Widgets
Transactions:
- Schedule Work Order
- Update Work Order
- Complete Work Order
- Create Large Order

CORPORATE DOMAIN
Customer, Supplier, and Parts Info
TXs:
- Check Credit
- Get Percent Discount
- New Customer

SUPPLIER DOMAIN
TXs:
- Send Purchase Order
- Deliver Parts

Deployment Environment

WebLogic Server 7.0 Cluster
Each node equipped with:
AMD XP 2000+ CPU, 1 GB RAM
Running on SuSE Linux 8.0

Oracle 9i (9.0.1) Database Server
Hosting the SPECjAppServer DB
1.7 GHz AMD XP CPU, 1 GB RAM
Running on Red Hat Linux 7.2
Capacity Planning Issues

We are interested in finding answers to the following questions:

• What level of performance does the system provide under load?
• Average response time, throughput and utilization = ?
• Are there potential system bottlenecks?
• How many application servers would be needed to guarantee adequate performance?

Need also optimal values for the following configuration parameters:

• Number of threads in WebLogic (WLS) thread pools
• Number of connections in WLS database connection pools
• Number of processes of the Oracle server instance

Workload Characterization

1. Describe the types of requests (request classes) that arrive at the system: NewOrder, ChangeOrder, OrderStatus, CustStatus.

2. Identify the hardware and software resources used by each request class: HW: WLS-CPU, Network, DBS-CPU, DBS-Disk, SW: WLS Thread, DB Connection, DBS Process.

3. Measure the total service time (service demand) of each request class at each processing resource:

<table>
<thead>
<tr>
<th>TX-Type</th>
<th>WLS-CPU</th>
<th>DBS-CPU</th>
<th>DBS-I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewOrder</td>
<td>7ms</td>
<td>2.3ms</td>
<td>1.2ms</td>
</tr>
<tr>
<td>ChangeOrder</td>
<td>0.6ms</td>
<td>1.8ms</td>
<td>0.5ms</td>
</tr>
<tr>
<td>OrderStatus</td>
<td>7ms</td>
<td>4ms</td>
<td>0.4ms</td>
</tr>
<tr>
<td>CustomerStatus</td>
<td>1.1ms</td>
<td>6ms</td>
<td>0.1ms</td>
</tr>
</tbody>
</table>
First Cut System Model

Hierarchical System Model: High-Level QPN

- We isolate the database server and model it using a separate QPN, represented by subnet "DBS" above.
- The above QPN is called High-Level QPN (HLQPN) of our hierarchical model.
Hierarchical System Model: Low-Level QPN

- The nested DBS subnet of our HQPN - called Low-Level QPN (LLQPN).
- Places Input, Output and Actual Population are standard for each subnet.

Scenario 1: Single Request Class

- Single request class – the NewOrder TX
- 80 concurrent clients with avg. client think time of 200ms
- 60 WLS Threads, 40 JDBC Connections, 30 Oracle processes

Analysis Results

Modelling Error

<table>
<thead>
<tr>
<th>Metric</th>
<th>Model</th>
<th>Measured</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLS CPU Utilisation</td>
<td>100%</td>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>DBS-PQ Utilisation</td>
<td>78%</td>
<td>80%</td>
<td>2%</td>
</tr>
<tr>
<td>NewOrder Throughput</td>
<td>66.25</td>
<td>69.35</td>
<td>3%</td>
</tr>
<tr>
<td>NewOrder Resp. Time</td>
<td>17.54</td>
<td>18.68</td>
<td>1.14</td>
</tr>
<tr>
<td>Service Queue Length</td>
<td>37.56</td>
<td>34.72</td>
<td>2.84</td>
</tr>
</tbody>
</table>
Scenario 1a: Same, but only with 40 Threads

Modelling Error

- More contention for threads, but less contention for CPU time.
- In both cases, we can reduce the number of DB connections and DBS processes, since they are not effectively utilized.

Scenario 2: Multiple Request Classes

- Two request classes – NewOrder and ChangeOrder
- Some simplifications needed to avoid explosion of the Markov Chain
- Assume that there are plenty of JDBC connections and DBS processes
- Drop places DB-Conn-Pool and DBS-Process-Pool
- 20 clients: 10 NewOrder and 10 ChangeOrder, Avg. think time = 1 sec
- Only 10 WLS Threads
Scenario 3: Multiple Application Servers

- We modify the HLOPN to include multiple WLS places
- 30 NewOrder clients with avg. think time of 1 sec
- No contention for JDBC connections, DBS processes and WLS threads

Scenario 3: Modelling Error

<table>
<thead>
<tr>
<th>Metric</th>
<th>Model</th>
<th>Measured</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>For 2 Application Servers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WLS-CPU Utilization</td>
<td>64%</td>
<td>58%</td>
<td>6%</td>
</tr>
<tr>
<td>DBS-CPU Utilization</td>
<td>88%</td>
<td>81%</td>
<td>6%</td>
</tr>
<tr>
<td>NewOrder Throughput</td>
<td>10.3s</td>
<td>17.8s</td>
<td>4%</td>
</tr>
<tr>
<td>NewOrder Resp. Time</td>
<td>620ms</td>
<td>632ms</td>
<td>2%</td>
</tr>
<tr>
<td>For 3 Application Servers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WLS-CPU Utilization</td>
<td>43%</td>
<td>64%</td>
<td>2%</td>
</tr>
<tr>
<td>DBS-CPU Utilization</td>
<td>86%</td>
<td>97%</td>
<td>1%</td>
</tr>
<tr>
<td>NewOrder Throughput</td>
<td>10.4s</td>
<td>17.8s</td>
<td>6%</td>
</tr>
<tr>
<td>NewOrder Resp. Time</td>
<td>623ms</td>
<td>678ms</td>
<td>7%</td>
</tr>
</tbody>
</table>
Summary and Conclusions

- QPN models enable us to integrate both hardware and software aspects of system behavior in the same model.

- Combining the expressiveness of Queueing Networks and Petri Nets, QPNs are not just powerful as a specification mechanism, but are also very powerful as a performance analysis and prediction tool.

- However, if this power is to be exploited to its full potential, improved solution methods and software tools for QPNs are needed, which enable larger models to be analyzed.

Thank You for Your Attention!

Questions?