Performance Modeling and Prediction of Enterprise JavaBeans with Layered Queuing Network Templates

Jing Xu
Murray Woodside
Carleton University

Alexandre Oufimtsev
Liam Murphy
University College Dublin
Motivation and Approach

- **Motivation:**
  - Performance analysis should be done in early stage of design (otherwise you going to pay for it…)
  - Modeling J2EE is non-trivial

- **Approach:**
  - Define LQN templates for different types of EJBs
  - Construct a model of real-life application by instantiating the templates and composing them
  - Profile and calibrate the model from app. traces
  - Perform measurements on the real running application
  - Compare model prediction with measurement results
LQN: what and why?

- **LQN (Layered Queuing Network)**
  - Is a performance modeling language
  - Models system resources and behaviour in an intuitive way
  - Allows nested software structure and composition with component concepts
  - Captures resource contentions effectively
  - Does not suffer from state explosion problem
  - Provides Analytical & Simulation solver

Ref on LQN

http://www.sce.carleton.ca/rads/lqn/lqn-documentation/
A LQN model of a 3-tier System

Client

Application Server

Database

Client

ClientCPU

CustomerControllerBean

CustomerBean

AppCPU

Database

DBCPU

Client

[$ThinkTime]

Client

(1-20)

updateEmail

store

find

invokeSetEmail

update

read

[$update]

[$read]
Introduction of EJB

- Core technology of J2EE
- Server-side component architecture

- Entity Bean
- Session Bean (Stateful/Stateless)
- Message Driven Bean
LQN Component Model Template for Session Bean (Stateless)
LQN Component Model Template for Session Bean (Stateless)

- **methodInvoke**
  - **invokeMethod** [$s\_checkAccess$] Container (inf)
  - **getThread** [$s\_getThread$] Bean Thread Pool ($M$)
  - **prepareBean** [$s\_prepareBn$] ContServ (1)
  - **busiMethod** [$s\_method$] activebean (inf)
  - transactionService
  - serviceRequest

- **invokeMethod** [$s\_checkAccess$]
- **getThread** [$s\_getThread$]
- **prepareBean** [$s\_prepareBn$]
- **busiMethod** [$s\_method$]
- **activebean** (inf)

**Model Elements:**
- Container
- Bean Thread Pool
- ContServ
- activebean

**Variables:**
- $M$
- $s\_checkAccess$
- $s\_getThread$
- $s\_prepareBn$
- $s\_method$
- $s\_checkAccess$
- $s\_getThread$
- $s\_prepareBn$
- $s\_method$
- $s\_checkAccess$
- $s\_getThread$
- $s\_prepareBn$
- $s\_method$
LQN Component Model Template for Session Bean (Stateless)

Diagram:
- MethodInvoke
  - invokeMethod
    - [s_checkAccess]
    - Container
      - (inf)
  - getThread
    - [s_getThread]
    - Bean Thread Pool
      - (M)
  - prepareBean
    - [s_prepareBn]
    - ContServ
      - (1)
  - busiMethod
    - [s_method]
    - activebean
      - (inf)
- transactionService
  - ($ptranx)
- serviceRequest
  - ($pextServ)
LQN Component Model Template for Session Bean (Stateless)
LQN Component Model Template for Session Bean (Stateless)
LQN Component Model Template for Session Bean (Stateful)

```
<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>homeCreate</td>
<td>Initialize bean</td>
</tr>
<tr>
<td>homeRemove</td>
<td>Remove bean from pool</td>
</tr>
<tr>
<td>invokeMethod</td>
<td>Call business method</td>
</tr>
<tr>
<td>Container</td>
<td>Manage bean instances</td>
</tr>
<tr>
<td>create</td>
<td>Create bean instance</td>
</tr>
<tr>
<td>remove</td>
<td>Remove bean instance</td>
</tr>
<tr>
<td>busiMethod</td>
<td>Call business method</td>
</tr>
<tr>
<td>Bean Thread Pool</td>
<td>Manage thread pool</td>
</tr>
<tr>
<td>prepareBean</td>
<td>Prepare bean for execution</td>
</tr>
<tr>
<td>ContServ</td>
<td>Manage service</td>
</tr>
<tr>
<td>Passivate/Activate</td>
<td>Activate or passivate bean</td>
</tr>
<tr>
<td>CallBack</td>
<td>Call back thread pool</td>
</tr>
</tbody>
</table>
```

Symbols:
- $s_callback$ (1-$p$)
- $s_prepareBn$
- $s_cremove$
- $s_method$
- $s_create$
- $s_remove$
- $s_prepareBn$
- $s_checkAccess$
- $M$
- $p$
- $i$
- $sptranx$
- $spextServ$
LQN Component Model Template for Entity Bean
Example Application - Duke’s Bank

- Standard J2EE sample application by Sun’s Microsystems, shipped with every J2EE tutorial

- Modifications
  - CMP instead of BMP;
  - multiple users support;
  - stateful SB converted to stateless SB;
  - artificial congestion at the pool/cache

- Running on
  - MySQL, JBoss, Sun’s JVM
Testing Scenario: Update Customer Information
Testing Scenario

- Data access pattern:
  - Sequential access vs. Random access

- Scenario:
  Gradually increasing workload from 1 to 20 users with step size 1. Each user:
  - Updates each customer record in ascending order (300 records in total);
  - Waits for other clients to finish;
  - Updates a random customer record 300 times;
  - Waits for other clients to finish.
High Level LQN Model for the Scenario
LQN Component Model for Customer Controller

- Stateless Session Bean

```
 KundenControllerBean

 invokeSetEmail
 [s_checkAccess] Container (inf)

 getThread
 [s_getThread] Bean Thread Pool ($CntrM)

 prepareBean
 [s_prepareBn] ContServ (1)

 setEmail
 [s_setEmail] activebean (inf)

 transactionService

 findCustomer

 setEmail
```
LQN Component Model for Customer

- Entity Bean

```
storeEntity  [\$s\_cstore]  
findByPK     [\$s\_cfind]   
invokeSetEmail [\$s\_checkAccess]  
Container     (inf)

1/$I

instanceSetEmail  [0]  
Instance          (1)

getThread        [\$s\_getThread]  
Bean Thread Pool ($M)

prepareBean     [\$s\_prepareBn]  
ContServ         (1)

store          [\$s\_store]  
load           [\$s\_load]   
passivate      [\$s\_passivate]  
activate       [\$s\_activate]  
SetEmail       [\$s\_setEmail]   
activebean     (inf)
```

updateDB

readDB
Profiling and Model Calibration

- Tool: JProbe profiler
- Calibration done using single-user profiling data
- 2 factors used for adjusting execution demand parameters
  - Profiling Ratio Factor (PFC): to remove overhead introduced by the profiling tool
  - Warm System Factor (WSF): to adjust the parameter values obtained in cold system status to reflect the warm system behavior
System Measurement and Comparing with Model Prediction

- Measured and predicted response times

![Graph showing measured and predicted response times for sequential and random access.](image)
Conclusion and Future Research

- A procedure for constructing, calibrating, solving and analyzing models of real-life J2EE applications;

- Investigate if the similar approach can be applied to other technologies such as .NET;

- Extend our work to incorporate crucial parts of underlying software layers into models (JVM, OS).
Appendix A: System Measurement of Throughput

![Graph showing system measurement of throughput]

- X-axis: Number of Clients
- Y-axis: Throughput, cps