Performance Modeling of a JavaEE Component Application Using LQN: a Case Study

Alexander Ufimtsev
Liam Murphy

Performance Engineering Lab
School of Computer Science and Informatics
University College Dublin
Ireland

www.perfenglab.com
Motivation

- Software development projects fail (time, budget, QoS, altogether) for multiple causes
- Bad design contributes to approx. 20% of problems in enterprise systems [Ptak et al]
- Performance analysis should be done at the early stages of the design to avoid failures
- However, it is difficult to check outside proper test environment
Motivation

Component App
Application Server
Virtual Machine
Operating System
Hardware
Three types of Approaches

a) Application
   - Application Server
   - Virtual Machine
   - Operating System
   - Hardware

b) Application
   - Application Server
   - Virtual Machine
   - Operating System
   - Hardware

Profiling
   - Virtual Machine
   - Operating System
   - Hardware

Profiling
   - Virtual Machine
   - Operating System
   - Hardware

Profiling
   - Virtual Machine
   - Operating System
   - Hardware
Approach

- Construct a model of real-life application by instantiating the templates and composing them
- Perform measurements on the real running application
- Profile and calibrate the model from app. traces
- Compare model prediction with measurement results
LQN Templates Overview

(C)omponent interaction is augmented with instanciated (T)emplates of container services
Why LQN?

- 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 both Analytical & Simulation solver
A Better Case Study: ECPerf

- ECperf is a Enterprise JavaBeans (EJB) benchmark meant to measure the scalability and performance of J2EE servers and containers.
- ECperf stresses the ability of EJB containers to handle the complexities of memory management, connection pooling, passivation/activation, caching, etc.
ECPerf Overview

- ~30 beans, not including helper classes
- 50K LOC
ECPerf Overview Cont'd
ECPerf Startup Parameters

- rampUp = 480, stdyState = 600, rampDown = 180
- runOrderEntry = 1, runMfg = 1
- Transaction rate (txRate) was set from 1 to 50 in different tests. Orders=5*txRate, Manufacturing=3*txRate
  - txRate = 5 (40 threads: 25 order entry, 15 planned line)
Application Profiling and Measurement: Hardware

The testing environment includes three x86 machines:

- app server (PIII-866 Mhz / 512 Mb RAM),
- database (PIII-800Mhz / 512 Mb RAM)
- client (PIV-2.2 Ghz / 1024 Mb RAM)
Performance Test: Throughput

Performance
St. Dev.
Performance Test: Response Time

![Graph showing response time as a function of scale factor for different operations.]

- Manufacturers
- NewOrder
- CngOrder
- OrderStatus
- CustStatus

Response Time, sec

Scale Factor

1 3 5 7
Approach Refining

- Communication – local & remote
- Container Services
- Connection Pooling
- Transaction Management
- Security
- Garbage Collection
- Naming
- Database
Addressing Ambiguity

- Growing DB size
- Initial number of clients in DB depends on the load
- Transactions are retried 5-20 times when failed
Model Overview

Client

Application Server

ClientCPU

AppCPU

Network

DBCPU

update [update]

read [read]

Database
Workload

Diagram:
- runScript
- Driver
  - configure
  - mfgAgent
    - loop
    - plannedLine
  - configure
  - ordersAgent
    - loop
    - orderLine
  - configure
  - loAgent
    - loop
    - lorderLine
Performance Modeling of JavaEE Application with LQN: Case Study

System Under Test (SUT): emulation of activity stations

remote call

createWidget plannedLine

scheduleWO WorkOrderSes
updateWO WorkOrderSes
completeWO WorkOrderSes

create getid process WorkOrderEnt

findByPK finish WorkOrderEnt

findByPK update WorkOrderEnt

findByPK addInvent assemblyEnt

sleep System
Modeling vs Testing Results

![Graph showing Modeling vs Testing Results]

- **BBops/min**: Benchmarking Modelling Mod. Var
- **Scale Factor**: 1 3 5 7
- **Values**: 101.666666666667, 297.333333333333, 479, 555, 700, 555, 47
Conclusion & Future Work

- More automation required when modeling real-life systems :-) 
- Model works until systems starts approaching peak performance state 
- More work is needed to understand why results go wrong at the 'border area' – system changes dynamically with load or hidden bottleneck
Questions?
Appendix A: Entity Bean Template
Appendix B: Stateless Session Bean
Appendix C: Stateful Session Bean

Diagram showing the state transitions of a stateful session bean in a JavaEE application with LQN: Case Study.