Improving Data Access of J2EE Applications by Exploiting Asynchronous Messaging and Caching Services

Samuel Kounev and Alex Buchmann

{skounev,buchmann}@informatik.tu-darmstadt.de

28th VLDB Conference, Hong Kong, China, 2002
Motivation

• Modern **J2EE Applications** are extremely data intensive and therefore the integration with existing backend databases is of crucial importance for system scalability and performance

• As a result, the **Data Access Layer** in J2EE is very susceptible to turning into a system bottleneck

• In this paper we use **ECperf** – a heavy-duty B2B benchmark as an example of a realistic application in order to show how **Asynchronous Messaging** and **Caching Services** could be exploited to reduce the load on the database and improve system performance and reliability
Agenda

• The J2EE Platform and Persistence
• Improving Entity Bean Performance
• The ECperf Benchmark
• Comparing BMP with CMP
• Elimination of Persistence Bottlenecks
• Summary and Conclusions
The J2EE Platform for E-Business

• A major technology for today’s e-business systems
• Industry standard defined by Sun Microsystems, Inc.
• Over 30 implementations on the market – Application Servers
• The latter provide a range of middleware services crucial for today’s e-commerce systems

• J2EE Applications are made up of EJB Components:
  • Session Beans
  • Entity Beans
  • Message-Driven Beans
Persistence Mechanisms in J2EE

Entity Beans:
- Bean-Managed Persistence (BMP)
- Container-Managed Persistence (CMP)

Benefits:
- decouple business logic from data access logic
- enable the container to cache data in the middle tier
- enforce control on the way data is accessed and modified

Session Bean Managed Persistence (SMP)
When reading large amounts of read-only data for listing purposes entity beans might be too slow → use SMP
Entity Bean Lifecycle

Does Not Exist

unsetEntityContext()

Pooled State

ejbRemove()
OR

ejbStore()

ejbPassivate()

Ready State

newInstance()

setEntityContext()

ejbCreate()

ejbPostCreate()
OR

ejbActivate()

ejbLoad()

ëjbHome()

ejbFind()

ëjbLoad()

business-method()

ejbStore()
Caching and Concurrency Control (CC)

- Commit Options A, B, C used in the EJB Specification [4].

- Deployment Options in *WebLogic Server 7.0* [1]:
  - EJB Concurrency Strategy:
    - Exclusive – CC handled by the container
    - Database – CC deferred to the underlying data store
    - Optimistic – implements optimistic CC
    - ReadOnly – eliminates calls to ejbStore()
  - Caching between Transactions Option – minimize ejbLoad() calls
  - Read-Mostly Pattern
The ECperf Benchmark

- Heavy-duty B2B E-Commerce Benchmark
- Measures performance and scalability of J2EE App. Servers
- Built in conjunction with App. Server vendors under the JCP
- ECperf was recently taken over by the SPEC-OSG Java Subcommittee and repackaged as SPECjAppServer
ECperf Business Model

**CUSTOMER DOMAIN**
Order Entry Application

**MANUFACTURING DOMAIN**
Parts ➔ Planned Lines ➔ Widgets

**Corporation Domain**

**SUPPLIER DOMAIN**

---

**CUSTOMER DOMAIN**
Order Entry Application

**MANUFACTURING DOMAIN**
Parts ➔ Planned Lines ➔ Widgets

**Corporation Domain**

**SUPPLIER DOMAIN**

---

**CUSTOMER DOMAIN**
Order Entry Application

**MANUFACTURING DOMAIN**
Parts ➔ Planned Lines ➔ Widgets

**Corporation Domain**

**SUPPLIER DOMAIN**

---

**CUSTOMER DOMAIN**
Order Entry Application

**MANUFACTURING DOMAIN**
Parts ➔ Planned Lines ➔ Widgets

**Corporation Domain**

**SUPPLIER DOMAIN**

---

**CUSTOMER DOMAIN**
Order Entry Application

**MANUFACTURING DOMAIN**
Parts ➔ Planned Lines ➔ Widgets

**Corporation Domain**

**SUPPLIER DOMAIN**

---
ECperf Application Design

- **Benchmark Components:**
  - **ECperf EJBs** – J2EE appl. deployed on the System Under Test (SUT)
  - **Supplier Emulator** – servlet simulating interactions with suppliers
  - **ECperf Driver** – Java appl. running on a client machine

- **Benchmark’s Throughput** function of chosen TX Injection Rate ($Ir$)

- **Performance Metric** provided is BBops/min = total number of business TXs completed in the Customer Domain + total number of workorders completed in the Manufacturing Domain, normalized per minute.
Our Deployment Environment

LAN

Client PC

- Client PC running the ECperf Driver
  - RedHat Linux 7.1, 192 MB Main Memory

WebLogic Server

SUT - System Under Test

- ECperf EJBs deployed on Web Logic Server 6.1
- Solaris 7, SUN Ultra Sparc II (Ultra 60)
- 2 x 360 MHz CPUs, 2 GB RAM

Oracle Server

RDBMS: Oracle 9i (9.0.1) Database Server

- Hosting the ECperf Database
  - Red Hat Linux 7.2
  - 1 x 1.7 GHz AMD XP CPU, 1 GB RAM

Supplier Emulator

ECperf Supplier Emulator

- Emulator deployed on Web Logic Server 6.1
  - Solaris 7, SUN Ultra Sparc II (Ultra 60)
  - 1 x 360 MHz CPU, 1 GB RAM
Performance of BMP vs. CMP
Performance of BMP vs. CMP

![Bar chart showing the comparison between BMP and CMP average commit times for different transaction injection rates.](graph.png)
Deployment Environment with Informix

LAN

Client PC

Client PC running the ECperf Driver
RedHat Linux 7.1, 192 MB Main Memory

SUT - System Under Test
ECperf EJBs deployed on Web Logic Server 6.1
Solaris 7, SUN Ultra Sparc II (Ultra 60)
2 x 360 MHz CPUs, 2 GB RAM

RDBMS: Informix Universal Server 9.20
Hosting the ECperf Database
Solaris 7, SUN Ultra Sparc II (Ultra 60)
2 x 360 MHz CPUs, 2 GB RAM

ECperf Supplier Emulator
Emulator deployed on Web Logic Server 6.1
Solaris 7, SUN Ultra Sparc II
1 x 360 MHz CPU, 1 GB RAM
• **Deploying ECperf out-of-the-box we ran into the following problem:**
  - There was extremely high data contention leading to data thrashing
  - Most transactions were being aborted because of deadlocks or timeouts

• **Monitoring the database we observed that:**
  - The `scheduleWorkOrder` transaction was taking relatively long
  - Most lock conflicts were occurring for the M_WORKORDER and S_PURCHASE_ORDER(LINE) tables and their indexes

• **To reduce data contention and eliminate the bottleneck we:**
  - *Decreased the Locking Granularity*
  - *Decreased the Transaction Isolation Level*
  - *Applied Transaction Chopping*
Getting to the Core of the Problem

ScheduleWorkOrder

Create Work Order

Get Bill of Materials

Get parts from Inventory

If parts depleted, contact Supplier

Create PO

Send PO

Confirm Receipt

ECperf DB

Supp. Domain

Supp. Emulator
Eliminating the Bottleneck

**Problem 1:** Sending the purchase order delays the transaction while holding locks on the inserted table and index entries.

**Problem 2:** The sending step is not undoable. If transaction is aborted after sending the order, we later get components delivered for which no purchase order exists.

**Solution:** the PO sending step replaced by sending a message to a JMS queue containing the ID of the new purchase order that needs to be sent. A message-driven bean created to process incoming queue messages by sending respective POs.
Asynchronous Redesign

**ScheduleWorkOrder**

Create Work Order

Get Bill of Materials

Get parts from Inventory

If parts depleted, contact Supplier

Create PO

Send Message

**Message-driven Bean**

Get next Message

Read PO

Send PO over HTTP

Confirm Receipt

**ECperf DB**

**Supp. Domain**

**PO Queue**

**Supp. Emulator**
Synchronous vs. Asynchronous (Informix)

![Graph showing the relative throughput for synchronous and asynchronous transaction injection rates.](Image)
Synchronous vs. Asynchronous (Oracle)

![Bar chart comparing synchronous and asynchronous transaction injection rates and relative throughput.](image-url)
Effect of Emulator Delays

![Graph showing the effect of emulator delays on relative manufacturing throughput. The graph compares synchronous and asynchronous emulations. As the average emulator delay increases, the relative throughput decreases for both synchronous and asynchronous cases.](image-url)
Evaluation

- This work was submitted to the ECperf Expert Group at Sun
- The identified problems are being addressed at the SPEC-OSG Java Subcommittee and the solution proposed will be implemented in **SPECjAppServer2003**
- **Both with Informix and Oracle our optimizations:**
  - Bring some significant performance and scalability gains
  - Solve the scheduleWorkOrder TX’s atomicity problem
  - Provide some further reliability benefits
- For further details see: “http://www.dvs1.informatik.tu-darmstadt.de/~skounev”
Summary and Conclusions

- **Entity beans** are the natural method for modeling persistent data in J2EE

- However, if not **configured and optimized** properly they could lead to serious performance degradation

- CMP in general performs better than BMP, because it gives the container complete control over the data access code and thus provides more opportunities for **caching of data** in the middle tier

- **Asynchronous Processing** could lead to astounding performance gains and reliability benefits if utilized properly
Thank You for your Attention!

We acknowledge the many fruitful discussions with:

- **Shanti Subramanyam**, ECperf Specification Lead at Sun
- **Steve Realmuto**, BEA’s Representative to the ECperf Expert Group
- **Chris Beer**, vice-chair of the OSG’s Java Subcommittee at SPEC and HP’s Representative to the ECperf Expert Group

For more information visit:

- [http://ecperf.theserverside.com](http://ecperf.theserverside.com)
- [http://www.dvs1.informatik.tu-darmstadt.de/](http://www.dvs1.informatik.tu-darmstadt.de/)