Darmstadt University of Technology
Department of Computer Science
Databases and Distributed Systems Group

Implementing and Optimizing Sun's ECperf Benchmark with BEA WebLogic Server

Samuel Kounev
skounev@informatik.tu-darmstadt.de

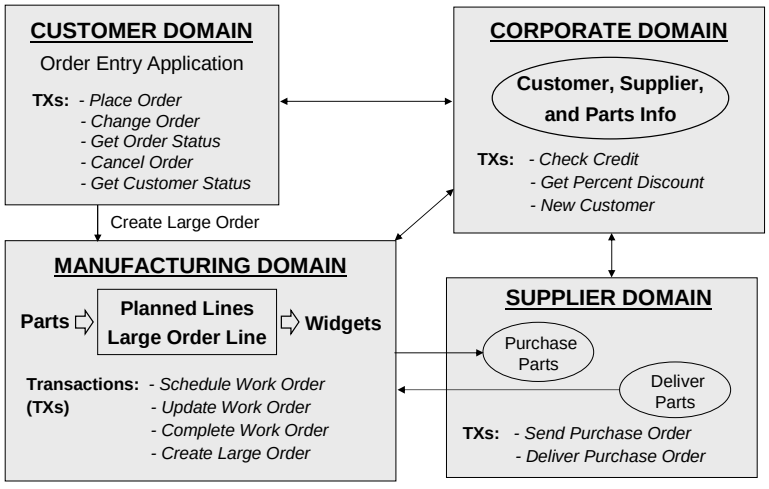
Agenda

- The ECperf Benchmark
- The ECperf Business Model
- The ECperf Application Design
- Our Deployment Environment
- The ECperf Persistence Bottleneck
- Eliminating the Persistence Bottleneck
- Evaluating the Performance Speedup
- Entity Bean Optimization Tips
- Other Optimization Tips
- Summary of Lessons Learned and Take-Away Points

The ECperf Benchmark

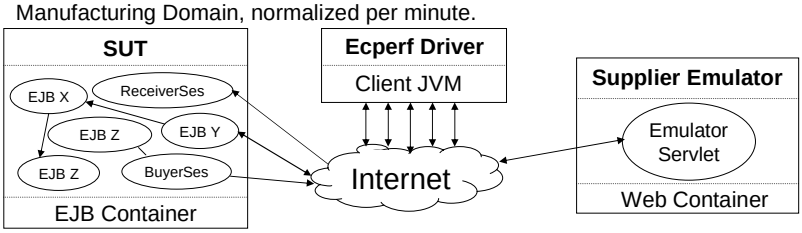
- Composed of a Specification and a Toolkit
- Measures Performance and Scalability of J2EE Appl. Servers
- Hosted on http://ecperf.theserverside.com
- Built in conjunction with Appl. Server Vendors under the JCP
- Non-exhaustive list of ECperf Expert Group members:

The ECperf Business Model

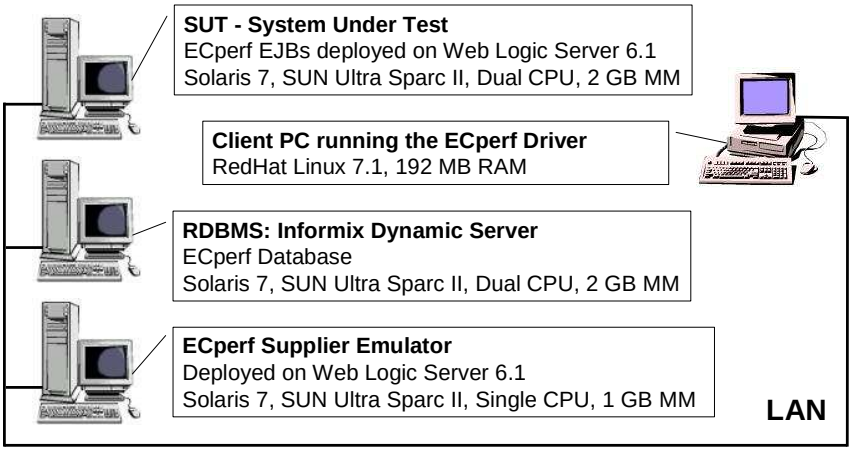


The ECperf Application Design

- **Benchmark Components:**
  1. *ECperf EJBs* - J2EE Application deployed on the *System Under Test (SUT*)
  2. *Supplier Emulator* - a servlet simulating interactions with external Suppliers
  3. *ECperf Driver* - a multithreaded Java Appl. running on a Client Machine
- RDBMS for Persistence - both Container Managed Persistence (CMP) and Bean Managed Persistence (BMP) supported
- Benchmark's Throughput function of chosen *Transaction 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



The ECperf Persistence Bottleneck

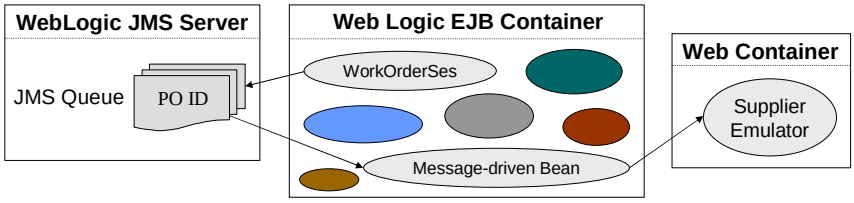
- **Deploying ECperf out-of-the-box we ran into the following problem:**
  - There was extremely high data contention leading to data thrashing
  - Most transactions were being aborted either because of deadlocks or timeouts
  - As a result we couldn't scale the benchmark beyond Ir = 2
- **Monitoring the database we observed that:**
  - The **scheduleWorkOrder** transaction of the **WorkOrderSes** EJB was taking relatively long to complete
  - 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* - set record-level locking
  - *Decreased the Transaction Isolation Level* - configured all beans to use SQL COMMITTED_READ isolation
  - *Applied Transaction Chopping* - broke down the scheduleWorkOrder transaction into smaller transactions

Eliminating the Persistence Bottleneck

- **Breaking down the scheduleWorkOrder transaction proved crucial**
- **scheduleWorkOrder proceeds as follows (simplified):**
  - Create a new **work order**
    - insert a new row in the M_WORKORDER table
  - Process the work order (stage 1 processing)
    - get the Bill of Materials needed
    - assign required parts from inventory
  - If new parts need to be purchased create a **purchase order - PO**
    - insert new rows in the S_PURCHASE_ORDER(LINE) tables
    - send the new purchase order to the Supplier Emulator (through HTTP)
- **Problem 1:** Sending the purchase order (the last step) 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

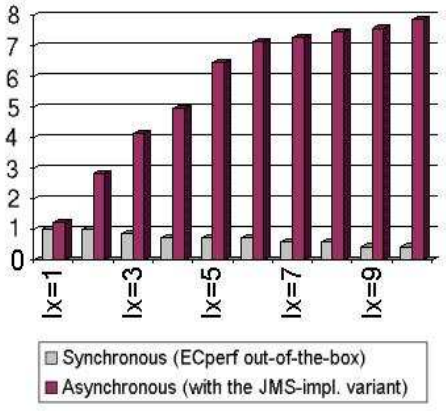
Eliminating the Persistence Bottleneck (continued)

- **Solution:** Move the PO sending step into a separate transaction and make it execute asynchronously
- Two variants for sending purchase orders implemented:
  - Quick Fix: A new agent implemented which periodically checks for new purchase orders and sends them to the Supplier Emulator
  - Long-term Messaging-based Solution - the PO sending step in scheduleWorkOrder replaced by sending a message to a WebLogic 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 (depicted below):



Eliminating the Persistence Bottleneck - Evaluation

- Proposed changes eliminated the bottleneck and lead to astounding performance gains in terms of throughput and concurrency as shown below:



Graph shows measured **BBops/min** relative to the BBops/min obtained when running ECperf out-of-the-box with **Ir = 1.**

Note: Here app. server becomes the bottleneck and not the database! All tests were run with a **single WLS instance**. We expect even better results if a multi-instance WLS cluster is run on the same hardware.

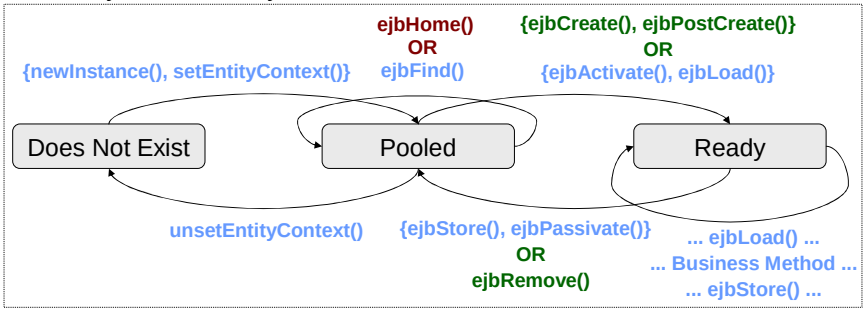
Eliminating the Persistence Bottleneck - Evaluation (continued)

- Our work was submitted as an official proposal to the ECperf Expert Group (see link below)
- Why were these problems not noticed before?
  - Tests were conducted with Oracle, which uses an optimistic multi-version concurrency control (CC) protocol. By contrast, Informix utilizes pessimistic locking-based CC, which performs worse under this workload.
- Both with Informix and Oracle our optimizations lead to some significant performance and scalability gains
- Expert Group vowed to address the problems we raised in the next version of ECperf
- For further details see:
  - "http://www.dvs1.informatik.tu-darmstadt.de/~skounev" ⇒ Publications Link

Entity Bean Optimization Tips

Introduction

- Lifecycle of an Entity Bean instance:



- Web Logic Server reads and writes persistent fields of entity beans from/to underlying storage using calls to ejbLoad() and ejbStore()
- By default ejbLoad() is called at transaction begin when the entity bean is first accessed and ejbStore is called at transaction commit

Entity Bean Optimization Tips

Utilizing Container's Caching Services

- Minimize database access calls - i.e. ejbLoad() and ejbStore()
- WLS offers 3 different concurrency strategies for entity beans that can be set in *"weblogic-ejb-jar.xml"* - Exclusive, Database and ReadOnly:
  - Use **ReadOnly** for beans that are never modified by an EJB client - this eliminates calls to ejbStore().
  - Use **Database** for Read-Write beans - this defers concurrency control to the database and usually performs better than **Exclusive**.
  - For beans that are only occasionally updated use the so-called **Read-Mostly Pattern** - implement a read-only bean for reading and a separate read-write bean for updating, mapping both of them to the same underlying data.

Entity Bean Optimization Tips

Bean Managed Persistence (BMP)

- Running ECperf with Bean-Managed (as opposed to Container-Managed) Persistence) we observed much worse performance
- Reason: Entity bean's data was being written to the database at every transaction commit, even if no changes had been made
- After adding code to check if data has been modified before accessing the database, throughput soared by at least a factor of 3 !
- **Further BMP Optimization Tips**:
  - Always prepare SQL statements - the database caches prepared statements in compiled form and this leads to a significant performance speedup
  - Make sure all statements are closed properly - so that respective database cursors are closed and resources freed
  - When reading large amounts of data mainly for listing purposes, consider bypassing entity beans and reading directly through JDBC in session beans

Entity Bean Optimization Tips

Container Managed Persistence (CMP)

- Even after optimizing the BMP code, we were still getting lower performance than with CMP - difference reached up to 50%.
- Reason: WLS provides some automatic optimizations such as:
  - Individual beans are loaded with one database call as opposed to two (ejbFind and ejbLoad) with BMP
  - Loading a collection of N beans requires N+1 database calls with BMP, while with CMP the container can combine those calls into a single database call
  - The container minimizes database access at transaction commit - i.e. only modified entity bean fields are written to the database
- Therefore, as a general rule of thumb, use CMP instead of BMP whenever possible

Entity Bean Optimization Tips

Watch Out for Deadlocks

- When multiple beans are involved in a transaction it is usually assumed that the container will invoke the database access calls for the beans in the same sequence as the respective beans are first accessed within the transaction. Accessing entity beans in different orders within different transactions could lead to deadlocks !
- For example with ECperf deadlocks were detected when reading order items in different orders in the Change Order and Order Status transactions
- This problem was fixed in Update 1 of ECperf by making sure that order items are always stored and accessed in sorted order
- **Take Away Point:** To avoid deadlocks make sure that entity beans are always accessed in the same order throughout the whole application !

Other Optimization Tips

- To improve initial response times:
  - Tune JDBC Connection Pool Size - set **initial-capacity** to the average number of concurrent client sessions that require JDBC connections
  - For every EJB tune the <initial-beans-in-free-pool> setting in the respective weblogic-ejb-jar.xml deployment descriptor
- Use the WLS Administration Console to monitor connection pools, bean pools, threads, transactions and other resources used during operation. At the same time use available tools and facilities to monitor all database servers in use. This might help you to discover subtle processing inefficiencies and scalability bottlenecks.
- Take a look at the Web Logic Performance and Tuning Guide for further tuning and optimization tips that are outside the scope of this presentation.

Summary of Lessons Learned & Take Away Points

- Use CMP instead of BMP whenever possible
- Use the lowest isolation level that does not compromise data integrity
- Configure the database locking granularity properly
- Exploit container's caching services to their full extent
- Always access entity beans in the same order in all transactions throughout the application

Summary of Lessons Learned & Take Away Points

- Make transactions as short as possible
  - Don't allow a transaction to span user interactions, network communications or any other activities that might potentially take a long time. Execute long operations asynchronously.
  - With CMP, use WLS monitoring facilities to make sure that transactions are demarcated correctly - check the transaction attribute settings.
- Use Asynchronous Processing and Messaging instead of traditional Request-Reply Processing whenever possible - Messaging brings significant performance, scalability and reliability benefits.
- DON'T GIVE UP UNTIL YOU HAVE EXPLOITED YOUR PLATFORM TO ITS FULL POTENTIAL !

Acknowledgements

Samuel Kounev and Alejandro Buchmann acknowledge the many fruitful discussions with:

- Shanti Subramanyam, ECperf Specification Lead at Sun Microsystems, Inc.
- Akara Sucharitakul, ECperf group at Sun Microsystems, Inc.
- Dan Fishman, Head of BEA's Research Lab
- Steve Realmuto, BEA's Representative to the ECperf Expert Group
- Chris Beer, vice-chair of the Java subcommittee at SPEC and Compaq's Representative to the ECperf Expert Group