Team 2:
Sales Inventory Management System

Vamshi Ambati
Myung-Joo Ko
Ryan Frenz
Cindy Jen
Team Members

Rfrenz @andrew.cmu.edu  Vamshi @andrew.cmu.edu  Cdj @andrew.cmu.edu  Mko1 @andrew.cmu.edu

Ryan Frenz  Vamshi Ambati  Cindy Jen  Myung-Joo Ko

http://www.ece.cmu.edu/~ece846/team2
Introduction

- Baseline Application
- Fault Tolerance
- Real-time
- High Performance
- Conclusion
Baseline Application

Jan 21 – Feb 13
Baseline Application

- **Sales Inventory Management System**
  - **Menu-Based Client**
    - Login, create/view/remove Sales/Purchase Orders, Inventory, Users
  - **Sales/Purchasing Management**
    - Handle requests to create, view, or remove sales or purchase orders, respectively
  - **User Management**
    - Controls add/view/remove of users
    - Login/logout
  - **Inventory Management**
    - Add/view/create inventory
Baseline Application

- Java-based, 3-tier application
- Middleware: EJB / Jboss
- OS: Linux (MS Compatible too)
- DB: MySql
- Deployment tool: Ant
Baseline Application

Why Interesting?

- Strong data integrity requirements
  - Data seen at any client must be accurate at any given time, regardless of other clients accessing/modifying it
Baseline Architecture

Client Tier
- Client applications
  - AddUser()
  - ViewProfile()
  - GetItemList()
  - GetItemDetail()
  - PlacePurchaseOrder()
  - GetPurchaseOrderList()
  - GetPurchaseOrderDetail()
  - PlaceSalesOrder()
  - GetSalesOrderList()
  - GetSalesOrderDetail()
  - GetOperationId()
  - LogIn()
  - LogOut()

Middle Tier
- User Management
- Inventory Management
- Purchase Order Management
- Sales Order Management

Backend
- Database
  - MySql
  - User
  - Item
  - SalesOrder
  - PurchaseOrder

EJB Container
- Application Server (JBoss)
- Naming Service (JNDI)

Client applications
Server application
Database

JDBC
Remote Method Invocation
JNDI call
Baseline Architecture

4 EJB Patterns

<table>
<thead>
<tr>
<th></th>
<th>Bean Managed Entity Beans with Session Beans</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Beans Only</td>
<td>Session Beans Only</td>
</tr>
</tbody>
</table>
Fault Tolerance + Baseline
Feb 13 – March 16
Fault-Tolerance Goals

- Replicate Sales, Purchasing, Inventory, User, and Operations Servers
- Server modules are ‘stateless’
  - We simply store a record of the last transaction in the database to prevent duplication in the case of a fault
- ‘Sacred’ Machine
  - Database, Replication Manager, Fault-Injector
FT-Baseline Architecture
Mechanisms for Fail-Over

- Fail-Over through exception handling

- Fault-Detection through replication manager-
  Crashed replica is restarted upon detection

Exceptions Caught:
- NamingException
  - JNDI is down (and consequently replica)
- RemoteException
  - JNDI is still up but replica is down
- CreateException
  - Any DB Problem (unavailable, duplicate create, etc)
- Create Exception → notify user (don’t fail over)
- Naming and Remote → retry with backup replica
- Next request → start over, trying primary first
Mechanisms for Fail-Over

- Replica references are obtained at time of request

- This allows for a simple fault-tolerance model
  - If anything goes wrong while obtaining references, we assume the worst and fail-over
  - If fault was transient, we’ll be back to primary upon next request

- But herein lies the bottleneck
  - Performing JNDI lookup and creating remote object for the same replica(s) every transaction
  - Big spike in fail-over, due to two lookups (with the first timing out)
Fail-Over Measurements (1)

**RTT (1 Client)**

- Operation Number (one run through use case)
- Time in milliseconds

**RTT (20 Clients)**

- Operation Number (one run through use case)
- Time in milliseconds
Fail-Over Measurements (2)

Fault Free Case (1 Client)
- View User List: 79%
- Create Server Home: 12%
- JNDI Lookup Time: 9%

Faulty Case (1 Client)
- Create Server Home: 38%
- View User List: 19%
- JNDI Lookup Time: 43%

Notice JNDI increases from 9% to 43%
Fail-Over Measurements (3)

Fault Free Case (20 Client)
- View User List, 56%
- Create Server Home, 25%
- JNDI Lookup Time, 19%

Faulty Case (20 Client)
- View User List, 28%
- Create Server Home, 27%
- JNDI Lookup Time, 45%

JNDI 19% --> 45%
Real Time + FT + Baseline

March 16 – April 5
RT-FT-Baseline Architecture (1)

- Upper-Bound the fail-over
- Target JNDI bottleneck by simply checking reference status instead of doing lookup
  - Instead of ‘lookup1-exception-lookup2’, we want ‘check-failover’
- Separate JNDI lookups into a background thread
  - Runs at the beginning of execution, then sleeps until needed (i.e. when we catch an exception from the primary server).
RT-FT-Baseline Architecture (2)

- **Fault-Free Case** → thread runs once and never again
- **Faulty-Case** → thread runs in the background, caching live references
  - Main execution simply checks if the primary reference is valid
  - If it is not live, move on to secondary object and signal the thread to update the primary
Other Possibilities to bound fail-over

- Client-Side timeouts to reduce ‘failed’ lookup times
- Would bound fail-over to a constant factor of the timeout value + second lookup
- However, after implementing the background thread to cache server references, adding timeout functionality does not improve fail-over times in the cases we consider (at least one ‘live’ server)
- Did not implement based on these observations
‘Real-Time’ Fail-Over Measurements

**Before**

RTT(1 Client)

Upper Bound ~ 5900 ms

**After**

RTT(Caching / 1 Client)

Upper Bound ~ 1500 ms

75% decrease!
‘Real-Time’ Fail-Over Measurements

Before

Upper Bound ~ 78000 ms

After

92% decrease!

Upper Bound ~ 6000 ms
High Performance+
RT+FT+Baseline

April 5 – April 13
High Performance Strategy

- “Functionality-Based” Load Balancing
  - Motivation
    - Webservers
    - Our Design
  - Benefits
    - Administrative actions do not suffer
    - QOS can be assured by following some policy to split the functionality
    - Decent level of Load Balancing is achieved with minimal effort
High Performance Architecture

Client Tier
- Client applications

Middle Tier
- User Management
- Purchase Order Management
- Inventory Management
- Sales Order Management
- Application Server (JBOSS)
- Naming Service (JNDI)

Backend
- Database
- MySql

KEY
- Client applications
- Server application
- Database
- JDBC
- Remote Method Invocation
- JNDI call
Performance Measurements(1)
Performance Measurements(2)

Mean RTT vs. # Clients

- Normal
- Balanced
Conclusion


Insights From Measurements

- FT-Baseline
  - JNDI/Reference lookups take large majority of RTT, especially in faulty case
  - Even in fault-free, doing the same lookup every time hurts RTT

- RT-FT-Baseline
  - Caching of references nearly eliminates ‘spikes’ by reducing JNDI lookup time which is a bottleneck in Fault tolerant systems

- High-Performance
  - Still room for improvement of performance
Accomplishments

- Nearly full ‘spike’ reduction with client-side reference caching (reduced RTT upper bound by 75% for 1 client, and 92% for 20 clients)

- Fully-interactive client with automated test benches

- Functionality-Based Load Balancing
What we learned

- Working in distributed Teams
  - Thanks to Assignment-1. CVS made life easy
  - Try Subversion

- JBoss is great but ‘heavy’
  - Startup times, etc make testing difficult

- Design decisions make project smoother

- To conquer FT, RT, HP you need to fight 3 battles
  - Keeping the server stateless makes the battle less complicated
What more could we have done

- **Optimizations on Server**
  - Fine-tuning JBOSS may improve the performance

- **Bounded Failover**
  - Exception handling, TCP timeouts could be bounded

- **Hard-coded JNDI servers**
  - Separate the JNDI from JBOSS
  - Should probably be modularized in a config file or similar

- **Load Balancing**
  - Functionality-based + Standard Load balancing Algorithms

- **Use Cases:**
  - Authentication
  - Searching
  - Messaging
Had we started over!

- **Administrative**
  - Would register for the course

- **Technical**
  - Would have
    - thought twice about EJB and JBOSS
    - thought about Operation ID stuffing at the time of designing the system
  - (Stateless vs Stateful server)