Distributed Bingo

17-654: Analysis of Software Artifacts
18-841: Dependability Analysis of Middleware

Team 5:
Jack Yao
Bubba Beasley
Kai Liao
Alex Berendeyev

http://www.ece.cmu.edu/~ece846/team5
Team Members

Kai Liao  Bubba Beasley  Alex Berendeyev
Real-World Demonstration
Real-World Characteristics

- Each player gets a Bingo card to start
- A player joining mid-game can catch up with knowledge of previous draws
- The host announces each draw
- The winning player announces Bingo
- The host verifies the win
- The host announces the win
Baseline Application

- A distributed, online version of Bingo
- The clients *pull* data from the servers and the servers *push* data to the clients
- DB: SQL Server 2000, Windows XP, MSE Cave machine
  Servers: JBoss (JNDI, JMS) on Linux, ECE Game cluster
  Clients: Windows, Linux (theoretically anywhere)
  Automated command-line client
  Interactive GUI-based client
Publish/Subscribe with JMS

![Diagram of Publish/Subscribe with JMS]

Key:
- **-** JMS Connection
- Server
- Client
- JMS
Baseline Architecture

Join()  GetSnapshot()  DeclareBingo()
From the Real to the App

- Each player gets a Bingo card to start
- A player joining mid-game can catch up with knowledge of previous draws

*The Client asks the AnsweringServer to join and receives a bingo card and all previous draws.*

- The host announces each draw

*At regular intervals (5 seconds), the HostServer broadcasts the draws via the JMS.*
From the Real to the App

- The winning player announces Bingo
- The host verifies the win
- The host announces the win

The Client asks the AnsweringServer to verify Bingo

The AnsweringServer verifies and stores the winner's ID in the database

The HostServer checks for a winner in the DB and broadcasts that there is a win
GUI Interface

Java GUI on top of a Java command-line interface
Miscellany

- Each game: 100 draws, rather than 75
- Approximately $1.4 \times 10^{30}$ card combinations
  (1.4 billion trillion trillion cards)
- Duplicate cards are not a problem, so theoretically no limit on the number of players in a single game
- No guarantee of fairness in declaring a winner
Fault Tolerance Goals (1)

Server Faults

- JBoss Process crashes
- Machine crashes

Network Faults

"Sacred Machine" Assumptions

- Replication Manager
- Global Fault Detector
- DB Server

Replicas
N replicas
Tested with 3 replicas
Round-robin recovery of JBoss servers
FT Baseline Architecture

One Connection to the primary JMS Server

BingoClient

BingoServer

RepMan
FD (HealthChecker class)

One JBoss server up

JBoss Servers on replicas are not running

Key
- Remote Method Call
- JMS Connection
- DB Call
- SSH Connection

Server  Client  DBServer
Baseline Fail-Over Measurements

FT Baseline Fail-Over

Time (seconds)

Number of faults injected

Slow Server

Fast Server

FD

SN
Baseline Fail-Over Measurements

FT Baseline Client

Number of faults injected

Time (seconds)

Time B/w Messages

Slow Server

Fast Server
RT-FT Optimized 1 Architecture

Pre-established Connections to all JMS Servers

RepMan
FD (HealthChecker class)

All JBoss servers on replicas are up

Key
- Remote Method Call
- JMS Connection
- DB Call
- SSH Connection

BingoServer

BingoClient

AS
JNDI
JMS

DBServer

Server
Client
DBServer
RT Optimized 1 Fail-Over Measurements

![Graph showing RT Optimization 1 Fail-Over measurements with time on the y-axis and number of faults injected on the x-axis. The graph compares slow and fast servers with different time intervals.]
RT-FT Optimized 2 Architecture

FD runs as a daemon

Key
- Remote Method Call
- JMS Connection
- DB Call
- SSH Connection

Server  Client  DBServer
RT Optimized 2 Fail-Over Measurements

![Graph showing RT Optimization 2 Fail-Over (Repman=1s, HealthCheckerDaemon=0.5s) with number of faults injected on the x-axis and time in seconds on the y-axis.]
RT-FT Optimized 3 Architecture

Key:
- Remote Method Call
- JMS Connection
- DB Call
- SSH Connection

FD runs as a daemon + Local FD
RT Optimized 3 Fail-Over Measurements

![Graph showing RT Optimization 3 Fail-Over (Repman=0.1s, Local Checker=0.4s) with time in seconds on the x-axis and number of faults injected on the y-axis. The graph contains two sets of data points, one for FDI and one for SN.]
Measurements Insights (1)
Measurements Insights (2)

Average Fault Detection Time Comparison

Local FD = 1.0s  
Repman = 0.1s

Local FD = 0.7s  
Repman = 1.0s

FD = 3.1s

Real Time Tuning:
- Repman period
- Local FD period
- Broadcast period
Lessons Learned

- Potentially Publish/Subscribe (JMS) can hide server errors from clients

![Typical Pull Architecture](image)

![Our Push Architecture](image)

- JBoss Server should be run in the minimal configuration. (default configurations are not suited for RT)
Questions?