Team 5: Fault-Terminators

18-749: Fault-Tolerant Distributed Systems

Patty Pun - tpun@andrew.cmu.edu
Kevin Smith - kevinsmith@cmu.edu
Felix Tze-Shun Yip - fty@andrew.cmu.edu
Yi Zhang - zhangyi@cmu.edu

Electrical & Computer ENGINEERING

Team Members

Patty Pun	Kevin Smith	Felix Tze-Shun Yip	Yi Zhang
tpun@andrew.cmu.edu	kevinsmith@cmu.edu	fty@andrew.cmu.edu	zhangyi@cmu.edu

http://www.ece.cmu.edu/~ece749/teams-06/team5/website/index.html

Baseline Application

- MAFIA-ONLINE
  - An online game that supports multiple game rooms
  - Users are assigned as villagers or mafia during log in
  - A system clock will keep track of the day time and night time
  - Mafia can vote to kill villagers during night time
  - Villagers vote in the day time to figure out who is mafia
  - Users interact with each other by broadcasting and receiving messages

Baseline Application

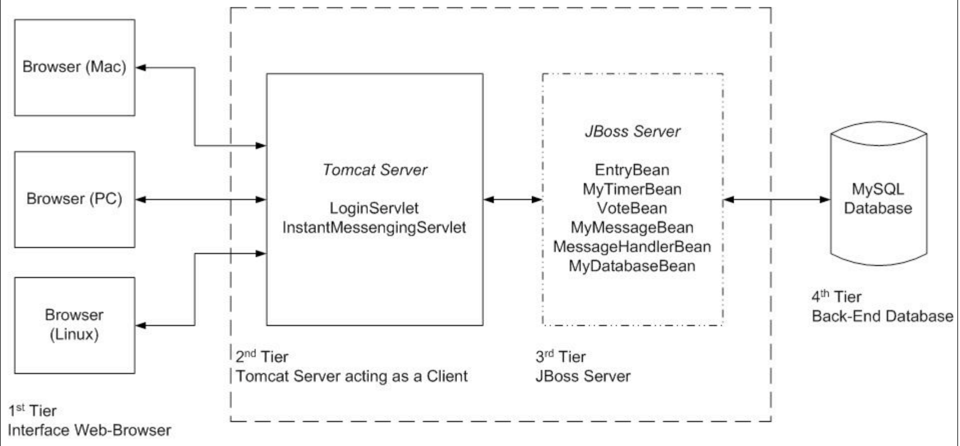
- Fun of MAFIA-ONLINE
  - Platform free: you only need a web browser to play the game
  - It has a nice web interface instead of boring command line display
  - It implements complicated game logic, highlighting timerBean and voteBean
  - It implements session beans, entity beans and message driven beans
  - It implements message broadcasting and blocking
  - You can vote to kill people (kick people out of the game room) when chatting

Baseline Application

- Middleware
  - EJB
- Platforms
  - Windows/Linux/Mac
- Language
  - JAVA
- Third-party software
  - Ant (Code Compilation & Development)
  - CVS (Version Control)
  - Eclipse (Development Environment)
  - Javadoc (Source Code Documentation)
  - JBoss (EJB Implementation)
  - MySQL (Database Management)
  - Tomcat (Web Server)
  - XDoclet (Interface File Generation)

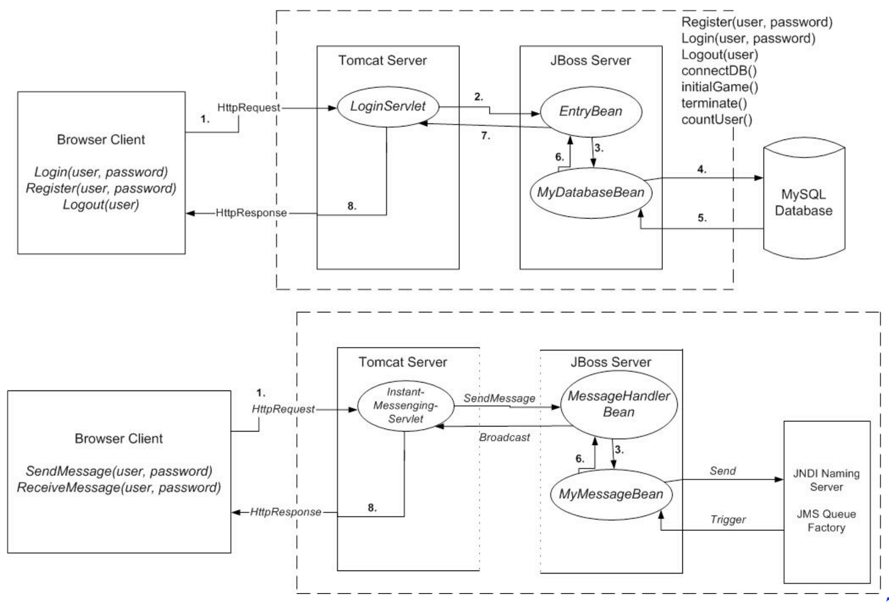
5

Baseline Architecture



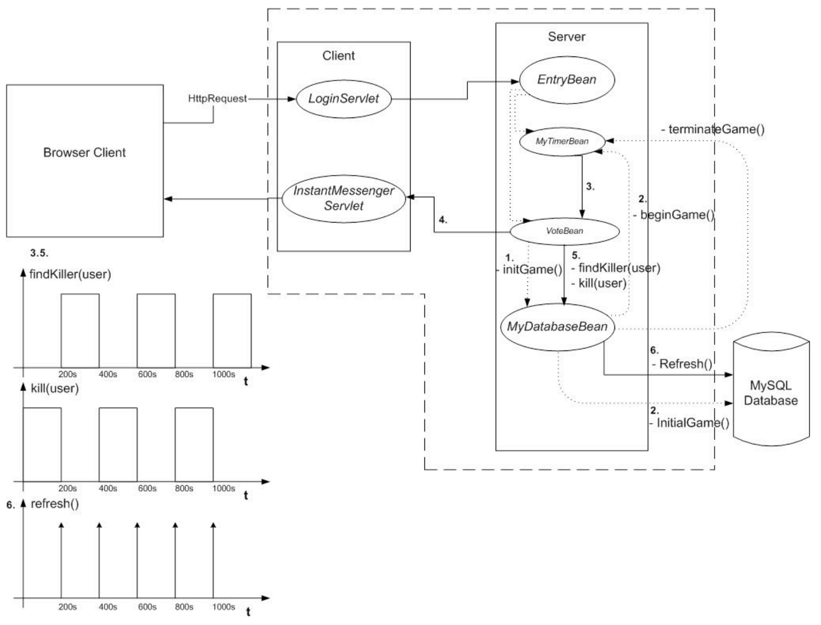
6

Baseline Architecture – Program Flow



7

Baseline Architecture – Game Flow



8

Fault-Tolerance Goals

- 100% Availability
  - Every resource should be available at all times
- Scalable
  - Performance should scale with number of increasing clients, servers, etc
- Complete Replication
  - There should not be a single point of failure anywhere
- Dynamic
  - Hostnames should not be hard coded anywhere
- Stateless servers
  - Stateful servers added complexity and chances for data loss
- Wishful thinking
  - If there is only one thing we learned, that will be that attaining everything is quite improbable

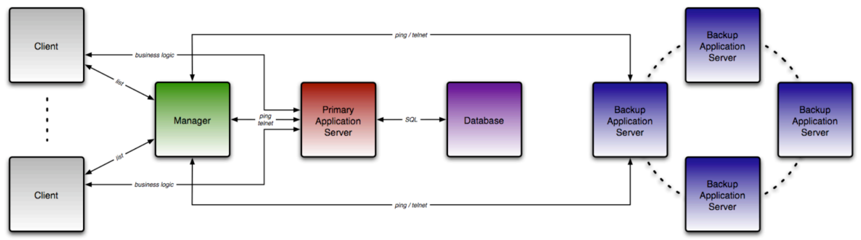
9

Fault-Tolerance Components

- Replication Manager
  - Replication manager should act as interface to list of servers
  - Should provide command line interface for modifying and viewing list
  - Options should be available for adding servers, removing servers, viewing the list, checking if a server is in the list, etc
- Fault Detector
  - Fault detector process should run continuously as background process
  - Will periodically ask replication manager for list of servers and poll all servers in that list
  - If the fault detector sees that a server has gone down, it will notify replication manager through command line interface
- Distributed Naming Service
  - All application servers must register their objects with naming service
  - Having naming service in central location introduces another single point of failure
- Replicated Application Servers
  - Must have replicated servers if fault-tolerance is desired

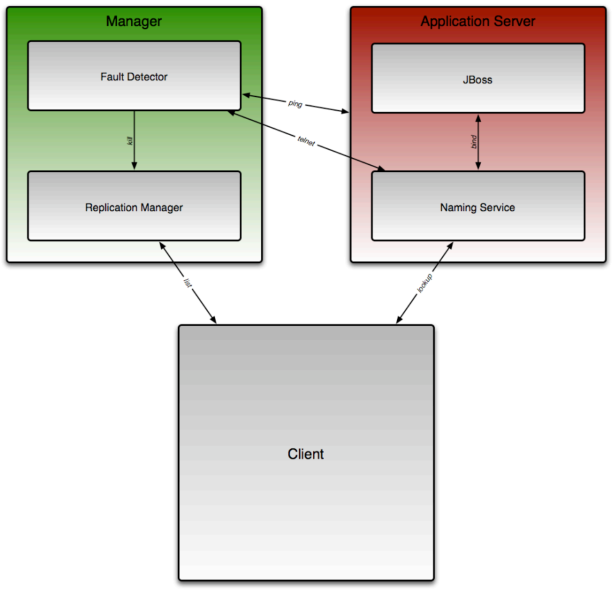
10

FT-Baseline Architecture


Client
business logic
list
Manager
ping / telnet
ping
telnet
Primary Application Server
SQL
Database
Backup Application Server
Backup Application Server
Backup Application Server
Backup Application Server
list
business logic
Client
ping / telnet

11

FT-Baseline Architecture



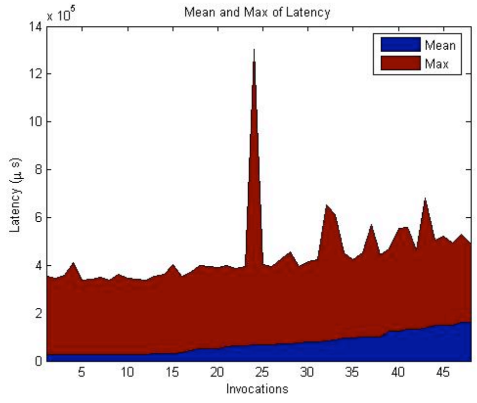
12

Mechanisms for Fail-Over

- In the event that the primary server goes down, clients should automatically be redirected to backup server
- If a client is unable to invoke a method on remote object, or fault detector cannot connect to server, it is assumed this server has gone down
- If client cannot invoke method on remote object, it could catch RemoteException, EJBException, or Exception
- When catching exception, client will tell replication manager to remove failed primary from list, and will then ask replication manager for name of new primary
- After receiving the name of the new primary, the client will create a new reference to the remote objects on this new primary
- If the replication manager reports an empty list, client will attempt to connect to a default server, and if that fails, client application must then notify user

13

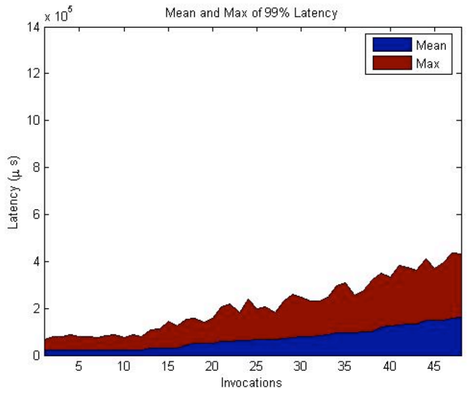
Fault-Tolerance Experimentation



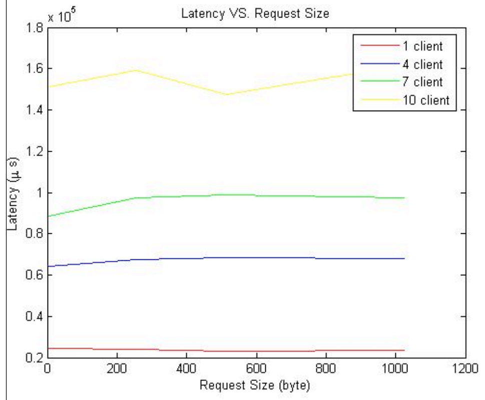
Area Plot for Maximal and Mean Value of Latency

Magical 1% Has Been Proved ☺

In fact, as the number of the runs increased, Magical 0.5% should be more proper.

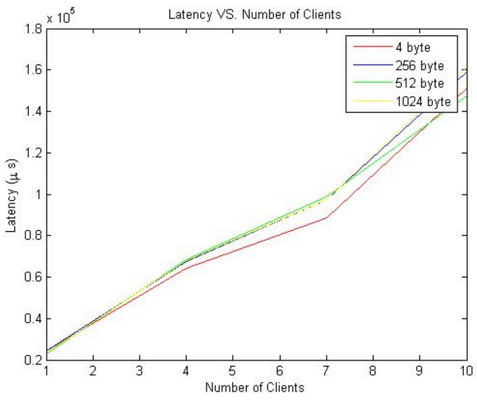


Fault-Tolerance Experimentation

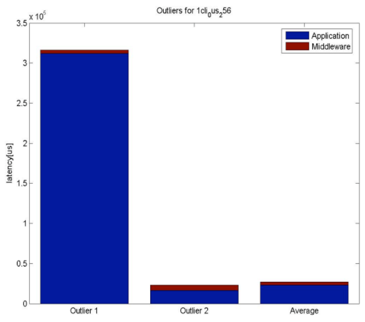


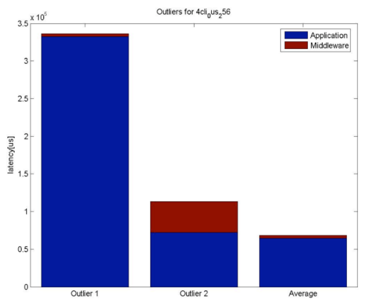
Request size is not the critical factor for latency

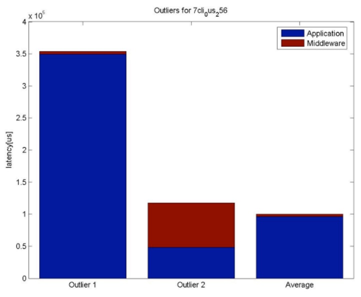
What's really important is the number of clients

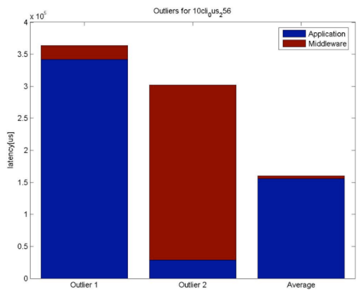


Fault-Tolerance Experimentation









16

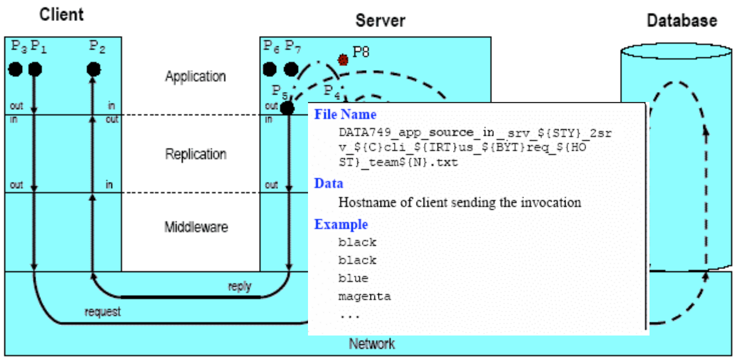
Fault-Tolerance Experimentation

- Data Discrepancy

$$Server(i) = P_5(i) - P_4(i)$$
$$Middleware(i) = Latency(i) - Server(i)$$

$\Rightarrow$ NEGATIVE Middleware Time

- Reason:
  - **First Come First Serve** $\neq$ **First Come First Leave**
- One more probe !!



File Name

```
DATA749_app_source_in_srv_${STY}_2sr
v_${C}cli_${IRT}us_${BYT}req_${HO
ST}_team${N}.txt
```

Data

Hostname of client sending the invocation

Example

```
black
black
blue
magenta
...
```

17

Fault-Tolerance Experimentation

……	
'iota'	'iota'
'eta'	'eta'
'gamma'	'gamma'
'theta'	'theta'
'lambda'	'lambda'
'epsilon'	'epsilon'
'zeta'	'delta'
'delta'	'zeta'
'iota'	'iota'
'eta'	'eta'
'gamma'	'gamma'
……	

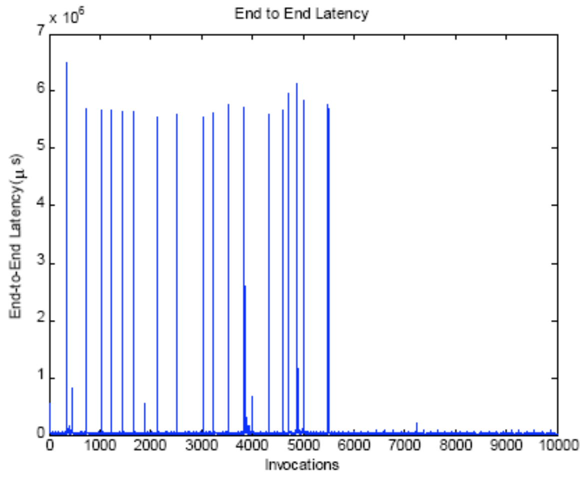
……	
'gamma'	'gamma'
'beta'	'beta'
'theta'	'theta'
'epsilon'	'epsilon‘
'lambda'	'delta'
'delta'	'lambda'
'kappa'	'kappa'
'zeta'	'zeta'
'iota'	'iota'
'eta'	'eta'
'gamma'	'gamma'
……	

……	
'beta'	'beta'
'zeta'	'zeta'
'kappa'	'delta'
'delta'	'kappa'
'zeta'	'zeta'
'delta'	'delta'
'kappa'	'kappa'
'zeta'	'zeta'
'beta'	'beta'
'delta'	'delta'
'kappa'	'kappa'
……	

18

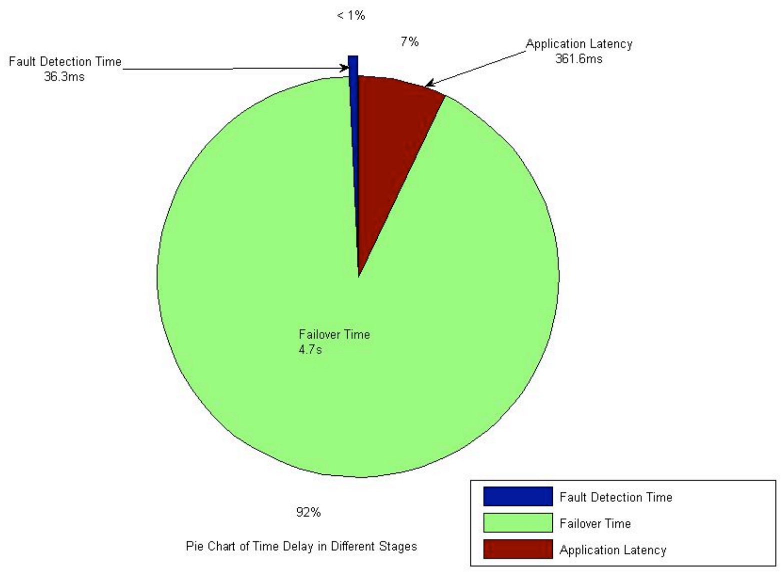
Bounded "Real-Time" Fail-Over Measurements

Average latency for fault free invocations: 20.55ms
Average latency for only fault invocations: **5751.9ms**
Minimum jitter in the presence of faults: 16.64ms
Maximum jitter in the presence of faults: 742.975ms



19

Bounded "Real-Time" Fail-Over Measurements



20

Bounded "Real-Time" Fail-Over Measurements

- Failover from the primary server to the backup server made up over 90% of recovery time for faulty invocations
- On average, this failover took 4.7 seconds
- This failover mechanism involves the client continuously communicating with the replication manager to tell it to update the list of servers
- Until the replication manager updates its list, and until the client is able to establish a connection with the new primary server, the client will be stuck in a loop

21

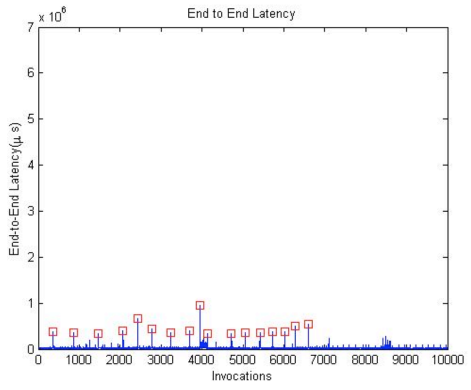
RT-FT-Performance Strategy

- Bottlenecks
  - Replication manager takes considerable amount of time to update its list
  - Client spends too much time waiting on replication manager to provide new server name
  - Client also spends considerable time creating new remote object reference after replication manager has provided valid server name.
- Solution
  - Both can be greatly reduced by always having second remote object reference available on client
  - When client starts, it can ask replication manager for the name of primary server as well as first backup server
  - Client can create two references pointing to the two different servers
  - In the event that the client cannot invoke a method on the primary server, it can immediately begin using the backup reference and continue processing
  - In separate thread, client can communicate with replication manager
    - Thread will ensure replication manager updates list
    - Thread will also update backup reference to point to the new backup server

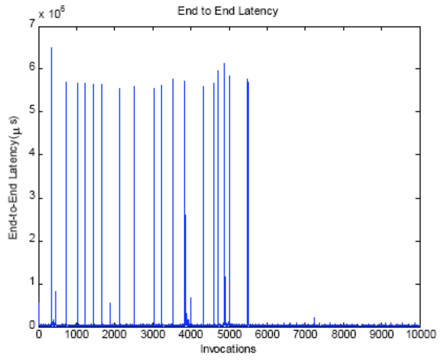
22

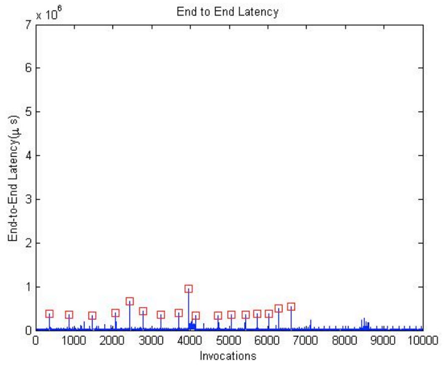
RT-FT-Performance Measurements

Average latency for fault free invocations: 21.30ms
Average latency for only fault invocations: **440.67ms**
Minimum jitter in the presence of faults: 3.4ms
Maximum jitter in the presence of faults: 520.3ms

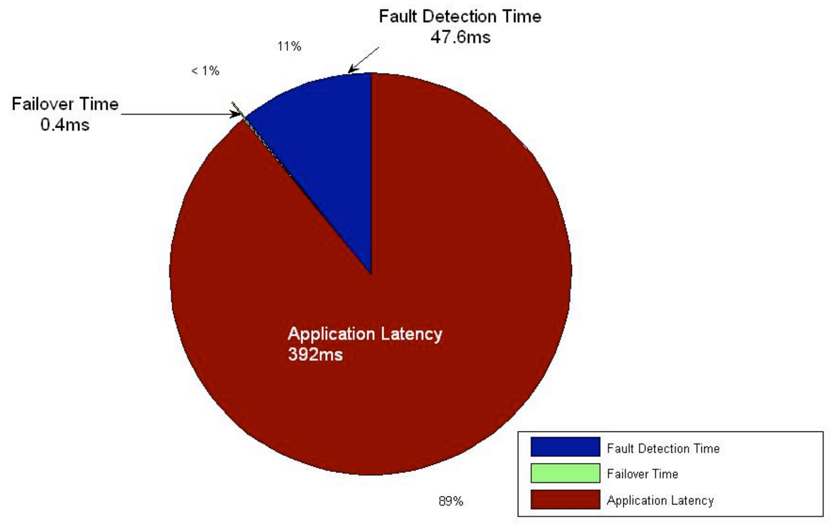


RT-FT-Performance Measurements



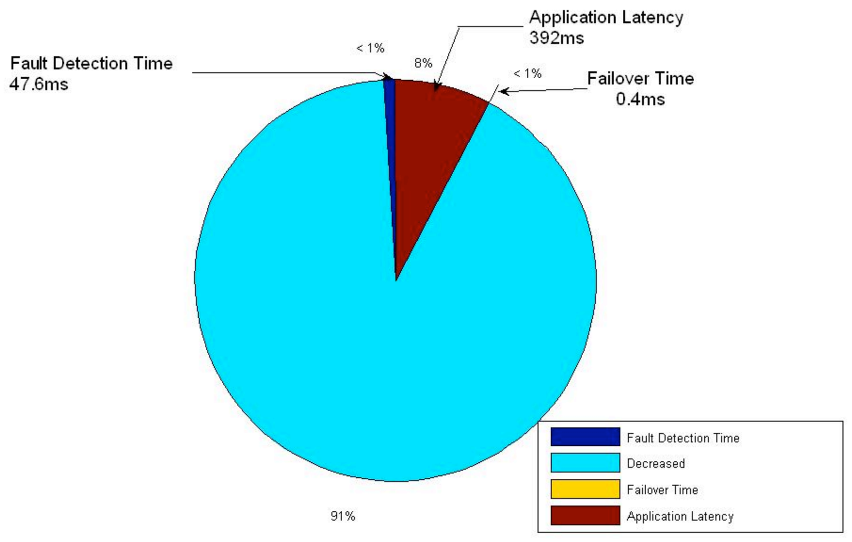


RT-FT-Performance Measurements



Pie Chart of Time Delay in Different Stages

RT-FT-Performance Measurements



Pie Chart of Time Delay in Different Stages

26

Other Features

- JMS
  - Queue Factory
    - send/onMessage
  - Topic Factory
    - publish/subscribe
- TimerTask
- Servlets
- Multi-threaded client

Other Features (cont'd)

- Lesson Learned from additional effort
  - JMS
    - Do not attempt to use before you understand JNDI
    - Lack of examples in books and internet
    - Unnecessary if there is only one client
  - TimerTask
    - Watch out timer function; they are very hard to debug !!
  - Servlet
    - EAR file deployable in Eclipse != deployable in cluster environment
    - XML files will get messy -> hard to integrate
    - Could not implement fault-tolerant architecture with only one client
  - Eclipse
    - It tends to give you an illusion that things work
    - Do not use Eclipse/Xdoclet unless you are really clear about what exactly you are implementing and have experience with EJB
    - Documentation quite difficult to find and not very useful

Insights from Measurements

- Our system DOES appear to adhere to the Magic 1% rule
- Trade off between latency and throughput
- Client requests are not handled in the same order they arrive
- For the fault free latency, what really matters is the number of the clients, and not the request size
- Middleware component of latency increases with increasing numbers of clients as well as with increasing reply size
- First client invocation on new bean exhibits considerably longer latency than subsequent invocations

29

Open Issues

- Requests are not handled in the same order they were received on server. This causes data discrepancies such as negative middleware time. Did other teams see this, and how did they resolve it?
- Add a timer or listener so users see new messages automatically
- Complete the original game we envisioned
- Why the first invocation takes longer than subsequent ones
- Graphical User Interface
- More efficient SQL queries
- Providing a web interface in a scalable way
- Why incomplete data from server probes

30

Conclusions

- What lessons did we learn?
  - Murphy’s Law: things will go wrong at the worst possible time
  - Be patient
  - Don’t hesitate to ask for help
  - Set realistic goals
- What did we accomplish?
  - In the high performance phase, latency was decreased by an order of magnitude
  - Gained experience in many tools such as Ant, JBoss, Eclipse, XDoclet, EJB, RMI, shell scripting, MATLAB, XML, Perl and more
- What would we do differently, if we could start the project from scratch now? (in essence we already did this … we tore apart our web-based project)
  - Dive straight into a Command-Line Interface
  - Seek help using middleware or switch to CORBA right away
  - Start with an instant messenger only, rather than a full-blown game
  - Get up to speed on new technologies as quickly as possible

31

Questions?

Patty Pun
tpun@andrew.cmu.edu

Kevin Smith
kevinsmith@cmu.edu

Felix Tze-Shun Yip
fty@andrew.cmu.edu

Yi Zhang
zhangyi@cmu.edu

http://www.ece.cmu.edu/~ece749/teams-06/team5/website/index.html

32