FTRMI: Fault-Tolerant Transparent RMI

Diogo Reis and Hugo Miranda

University of Lisbon
LaSIGE

SAC, March 28th, 2012
Remote Procedure Calls
What?

- A simple approach for distributed computing
- Hides the network from the application (client and server) programmer

Client side

// Do something
x=f(y);
// Do more

Server side

int f(int y) {
// Do something
    return z;
}
Remote Procedure Calls

Who?

- Implementation examples
  - Procedural: ONC (SUN) RPCs, Web Services
  - OO: CORBA, JRMI
Remote Procedure Calls

How?

JRMI as an example
The Availability Limitation of RPC’s

- What if server fails?
  - Server name is well-known
  - Stubs cannot reroute remote calls to alternative servers
    - Server state would not be available at the replica
Approaches for State-full Consistent Replication of Servers

Middleware Aware

- Client and server stacks cooperate to support replication
- Disadvantages
  - Clients and servers use non-standard protocol
    - Must run special version of the middleware
- Examples
  - Jgroup/ARM
  - Filterfresh
  - FT-CORBA
  - ...
Approaches for State-full Consistent Replication of Servers

Middleware Unaware

- Replication is hidden from the application and the middleware
- “Proxies” capture and (possibly) interpret the client/server traffic
- Disadvantages
  - Respecting the protocols raises limitations on the operations that can be provided
- Examples
  - Aroma
    - Snoops traffic at client and server side
  - FTRMI
Fault-Tolerant RMI (FTRMI)

Overview

- Proxy placed on the server side
  - Between the standard library class and server skeleton
  - Class with the same name and API of the original JRMI
    - sun.rmi.server.UnicastServerRef
    - -Xbootclasspath/p
- No code change at the client or server
FTRMI Process

Client

Proxy Layer (Stub)

RemoteReference Layer

Transport Layer

Server

Proxy Layer (Skeleton)

FTRMI Layer

RemoteReference Layer

Appia Layer

Transport Layer

Server

Proxy Layer (Skeleton)

FTRMI Layer

RemoteReference Layer

Appia Layer

Transport Layer

Network Layer

---
request total order multicast reply
Incoming calls

- Intercept remote calls before they are delivered to the server
- Use Linux `libcap` to retrieve call’s TCP/IP connection information
  - Sequence and Ack number
  - IP origin and destination addresses
- Deliver the call and TCP data to the Appia Group Communication Service
  - Appia enforces the atomic broadcast properties
FTRMI Implementation
Calls received from clients
FTRMI Implementation

Calls received from Appia

- Forward the call to skeleton
- Intercept the reply
- Prepare a TCP segment that matches the TCP expected reply at the client
- Send the TCP segment
  - Using raw sockets
FTRMI Implementation
Handling calls received from Appia
Client Transparency

- Client’s TCP will receive multiple copies of the reply
- Consider all but the first as duplicates
  - Discard
FTRMI Implementation
Multiple reply handling at the clients
Fault Tolerance

- Appia protocols
  - Provide atomic broadcast
  - Support for a distributed state machine
  - Support for state synchronisation when servers recover

- TCP
  - Duplicate detection
    - Case where server that received the request fails
FTRMI experimented with 3 distinct total order protocols provided by Appia

- FTRMI-1 Regular, Coordinator-based Total Order
- FTRMI-2 Regular, Causal Order-based Total Order
- FTRMI-U Uniform Total Order

And compared with 2 approaches

- JRMI Off-the-shelf, not replicated JRMI
- Jgroup/ARM middleware-aware framework
  - Clients and servers share a GCS

Remote procedure

- int procedure(String s)
Evaluation
Latency

arguments size: 2000 bytes
• JRMI always presents the best performance results
• FTRMI scales well
• Server-Server $4 \times -10 \times$ more than Client-Server traffic
• Some protocols don’t have a linear relation between latency and traffic
Conclusions & Future Work

- **FTRMI**
  - Transparent replication of JRMI servers
    - without code changes at the client or the server
    - No need to use specialised libraries at the client side
  - Encouraging performance results

- **Future Work**
  - Extend fault tolerance to the JRMI Registry
  - Experiment this approach in other RPC frameworks