IMPLEMENTATION OF FAULT TOLERANT ATOMIC BROADCAST

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
Fault Tolerance

Ability to recover from unexpected situations
Abstract Techniques

Information redundancy
- e.g. Hamming code

Time redundancy
- e.g. timeout and retransmission

Physical redundancy
- e.g. redundant array of independent disks
**Faults in Distributed Systems**

- **Crash Failure**: Process works correctly until it halts
- **Omission Failure**: Receive or send omission
- **Timing Failure**: Does not respond in expected period
- **Arbitrary Failure**: Unexpected response at arbitrary time
INFORMATION REDUNDANCY IN DISTRIBUTED SYSTEMS

Active replication

Active replication: A, B, C, replaced by A, AA, AAA ...; B, BB, BBB .., C, CC, CCC .., and if A sends message M to B in original, A* send message to B* in new system, and B* chooses majority result

Passive Replication

Passive Replication: A, B, C, augmented with by AA, AAA, ...; BB, BBB, ...; CC, CCC, ... and if A’s state changes, the corresponding change is made on AA and AAA. If A fails, AA takes over. If AA fails, AAA takes over ....
**Fault Tolerance Problems: Consensus Problem**

<table>
<thead>
<tr>
<th>Set of processes decide on some value</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. Who Relays, whether a transaction should be committed, which value to choose</td>
</tr>
</tbody>
</table>
Asynchronous vs. Synchronous Systems

Asynchronous Systems
No bound on the time required to respond to a message

Synchronous Systems
Bound on the time required to respond to a message
**Impossibility Results in in Distributed System**

- **Asynchronous Systems**
  - Cannot achieve consensus as long as one faulty process
  - Do not know if a process is faulty or taking too long

- **Synchronous Systems**
  - Can achieve consensus as long as ratio of total/faulty processes is above a certain threshold ($M$ faulty in $3M + 1$ total processes)
  - Rounds of communication with timeouts
CONSISTENCY PROBLEM IN BROADCAST

FIFO

Messages Mi1, M2 sent by P are received in order by every receiver Q

Causal Broadcast

If P sends a message M2 after seeing M1 then M2 is received after M1 in every receiver Q
ATOMIC BROADCAST

- Communication History
- Privilege-Based
- Moving Sequencer
- Destination Agreement
**Fixed Sequencer**

- **Broadcast-Broadcast**
  - P broadcasts M to sequencer and all destinations. Sequencer sends sequence number and hashcode of M to all destinations. Destinations deliver messages based on sequence number.

- **Unicast-Unicast-Broadcast**
  - P unicasts message to sequencer, which unicasts sequence number to it. P broadcasts message with sequence number.
Techniques in Distributed Systems

Active replication

Active replication: A, B, C, replaced by A, AA, AAA ...; B, BB, BBB .., C, CC, CCC .., and if A sends message M to B in original, A* send message to B* in new system, and B* chooses majority result

Atomic broadcast without fault means that all processes will have the same state at quiescence
Fault Tolerant Atomic Broadcast

Asynchronous Systems

If we can do fault tolerant atomic broadcast, then we could have consensus, which is impossible

Synchronous Systems

Application-level vs network based broadcast

Reliable vs unreliable communication
Fixed Sequencer, Unicast Broadcast: Basic Idea and Assumptions

Assume each message has been sent to each of the current session members—no latecomer.

Assume synchronous system, when a process fails, within a specified time (chosen by TCP/IP) period all other processes know because of probe messages, and any in-transit messages are discarded.

Next relayer chosen based on purely local information, no expensive synchronization done but it is possible to solve the consensus problem.

Assume no erroneous or malicious code or hardware.

Peer to peer: any process can act as a relayer, no special sequencer.
NON FAULT TOLERANT ARCHITECTURE WITH SEPARATION OF CONCERNS

- Session Port
- Client Object
- Relayer Elector
- Connection Manager

Client object can be composed of model, in and out couplers, reference relayer elector

Awareness?
Connections

- Client Object
- Relayer Object
- Connection Manager
- Session Port
- Relayer Elector

Relayer elector is session port unaware

Connection manager responds to join and leave commands and calls relayer elector to get current relayer

Client reference (possibly subclassed) relayer connector if relayer elector is simply a function call
# Events

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sent-broadcast</td>
<td>A message sent by a relayer-client to a relayer.</td>
</tr>
<tr>
<td>Received-broadcast</td>
<td>A message received by a relayer from a relayer-client.</td>
</tr>
<tr>
<td>Sent-relay</td>
<td>A message sent by a relayer to a relayer-client.</td>
</tr>
<tr>
<td>Received-relay</td>
<td>A message received by a relayer-client from a relayer.</td>
</tr>
<tr>
<td>Process left</td>
<td>A process has left the session.</td>
</tr>
<tr>
<td>Process joined</td>
<td>A process has left the session.</td>
</tr>
</tbody>
</table>
Basic Fault-Tolerance Algorithm

Client algorithm

When the leaving of a relayer is detected, the next broadcast is sent to the new relayer

Server algorithm

When the leaving of a relayer is detected, the client object is removed from the list of clients

Assumption: A relayer does not die in the middle of sending messages

Passive voice?
Fault Tolerant Architecture with Separation of Concerns

- FT Manager responds to leave command by changing destination of broadcasts and updating client map
- Can be subclass of Relayer Elector and same connections
- Assumption: A relayer does not die in the middle of sending messages
# Reacting to Partial Broadcast

<table>
<thead>
<tr>
<th>Passive Replication Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Any message sent by a remote client to the dead relayer should be received by all clients</td>
</tr>
<tr>
<td>2. Any message sent by the dead relayer to a remote client should be received by all clients</td>
</tr>
</tbody>
</table>

Key idea: one or more clients has the message(s) that old relayer partially broadcast, which can be rebroadcast using the old relayer

<table>
<thead>
<tr>
<th>1. Sent messages must be buffered</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Received messages must be buffered</td>
</tr>
</tbody>
</table>
# Changes to Basic Algorithm

## Client Algorithm

<table>
<thead>
<tr>
<th>When the leaving of a relayer is detected, synchronization phase is entered and one or more messages are rebroadcast</th>
</tr>
</thead>
<tbody>
<tr>
<td>When messages are sent/received, they are buffered and the sequence numbers of received messages used to determine what is rebroadcast</td>
</tr>
<tr>
<td>During synchronization phase new messages are buffered and at end of phase they are sent</td>
</tr>
</tbody>
</table>

## Server Algorithm

<table>
<thead>
<tr>
<th>Each relayed message wrapped with current sequence number before being sent to client object</th>
</tr>
</thead>
<tbody>
<tr>
<td>When the leaving of a relayer is detected, the new relayer goes into a synchronization phase before doing new relay</td>
</tr>
</tbody>
</table>

## Architecture

- Session Port
- Client Object
- Relayer Elector
- Connection Manager
- Architecture changes?
FILTER OBJECTS

- Session Port
- Client Object
- Relayer Elector
- Connection Manager
- Send and Receive Filter
As Filters are GIPC aware, best to no FT logic in them

Separate receive and send filter

FT manager can be separated into master and slave FT managers
<table>
<thead>
<tr>
<th>Message Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sent-broadcast: A message sent by a relayer-client to a relayer.</td>
</tr>
<tr>
<td>Received-broadcast: A message received by a relayer from a relayer-client.</td>
</tr>
<tr>
<td>Sent-relay: A message sent by a relayer to a relayer-client.</td>
</tr>
<tr>
<td>Received-relay: A message received by a relayer-client from a relayer.</td>
</tr>
<tr>
<td>Send finish synchronization message: A message received by the master FT manager to the slave FT manager to indicate synchronization is over</td>
</tr>
<tr>
<td>Receive finish synchronization message: A message received by the master FT manager to the slave FT manager to indicate synchronization is over</td>
</tr>
</tbody>
</table>

Must somehow generate (application-specific) sent-broadcasts and sent-relay messages and distinguish them from synchronization messages
FILTER OBJECT

Needs a way to distinguish between messages are internally communicated by GIPC and those generated by client and relay object.

We can specify the messages generated by client and relay objects.
As Filters are GIPC aware, best to no FT logic in them

Separate receive and send filter

FT manager can be separated into master and slave FT managers
### GIPC and RPC Calls

<table>
<thead>
<tr>
<th>Assume that client and relay objects make remote procedure calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>The messages passed to filter objects are instances of SerializableCall</td>
</tr>
<tr>
<td>The toHeader() method of such a call tells its signature which can be specified in a list</td>
</tr>
</tbody>
</table>
## GIPC and RPC Calls

**Server algorithm**

<table>
<thead>
<tr>
<th>Each relayed message wrapped with current sequence number before being sent to client object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each sent-relay in response to a received-broadcast must be assigned the same sequence number</td>
</tr>
<tr>
<td>Server FT manager Keeps track of N, the number of clients</td>
</tr>
<tr>
<td>Assigns sequence number on received-broadcast and wraps the next N sent-relay messages with N</td>
</tr>
</tbody>
</table>
Server algorithm

During synchronization phase new messages are buffered and at end of phase they are sent

Synchronized method can broadcast,
## Threads

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sent-broadcast: A message sent by a relayer-client to a relayer.</td>
<td>App thread</td>
</tr>
<tr>
<td>Received-broadcast: A message received by a relayer from a relayer-client.</td>
<td>RPC thread</td>
</tr>
<tr>
<td>Sent-relay: A message sent by a relayer to a relayer-client.</td>
<td>RPC thread</td>
</tr>
<tr>
<td>Received-relay: A message received by a relayer-client from a relayer.</td>
<td>RPC thread</td>
</tr>
<tr>
<td>Send finish synchronization message: A message received by the master FT manager to the slave FT manager to indicate synchronization is over.</td>
<td>Select Thread</td>
</tr>
<tr>
<td>Receive finish synchronization message: A message received by the master FT manager to the slave FT manager to indicate synchronization is over.</td>
<td>Select Thread</td>
</tr>
</tbody>
</table>

Original transmission and retransmission can occur in different threads

Retransmission in select thread
Buffering

Client algorithm

Sent/received messages are buffered and the sequence numbers of received messages used to determine what is rebroadcast.
**Replicated vs. Distributed Buffering**

**Replicated Buffering**

- A client buffers all received messages and unbuffers messages it knows have been received.
- Unbuffers messages it knows have been received by all sites.
- Last sequence number sent with every broadcast and periodically a special synchronization message sent with sequence number.

**Distributed Buffering**

- A client buffers messages it has sent and at most one message from the current relayer.
- When a message is echoed back or a relayer message is received, previous message is unbuffered.

**Efficient but less flexible:** assumes synchronized broadcast (GIPC group function call)

**No fault occurs during synchronization phase – one fault at a time assumption**
MORE WRAPPING

Distributed Buffering

Need a way to know its message has been bounced back or that a relayer message has been received

Wrap message with message with unique (host and local sequence number) which must be unwrapped
DISTRIBUTED BUFFERING

Client algorithm

When the leaving of relayer is detected, the slave FT Manager sends the buffered message with the highest sequence number (relayer or local)

Server algorithm

When the leaving of a relayer is detected, the master FT manager chooses the maximum received message and sends it to all clients that have not received the message or all sites and sends a finish synchronization message

A relayed message is discarded if its sequence number is not expected
# Key Ideas

<table>
<thead>
<tr>
<th>Synchronous Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passive Replication</td>
</tr>
<tr>
<td>Client, Relayer, FT Manager, Filter, Connection Manager Architecture</td>
</tr>
<tr>
<td>Partial vs. Complete Broadcast</td>
</tr>
<tr>
<td>Distributed vs. Replicated Buffering</td>
</tr>
</tbody>
</table>