CS 194: Distributed Systems

Distributed Coordination-based Systems

Scott Shenker and Ion Stoica
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

Coordination Systems

- Handle all communication and cooperation between processes/objects in a distributed system
  - Emphasize not on transparency
  - Object distribution is explicit

- Can be classified along two dimensions:
  - Temporal: do sender and receiver need to be active simultaneously?
  - Referential: do sender need to know the identifier of the receiver?

Taxonomy of Coordination Models

<table>
<thead>
<tr>
<th>Temporal</th>
<th>Referential</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coupled</td>
<td>Direct</td>
</tr>
<tr>
<td></td>
<td>Meeting</td>
</tr>
<tr>
<td></td>
<td>oriented</td>
</tr>
<tr>
<td></td>
<td>Generative</td>
</tr>
<tr>
<td></td>
<td>communication</td>
</tr>
</tbody>
</table>

TIB/Rendezvous System

- Meeting oriented model (a.k.a. publish/subscriber)
- Build around concept of information bus
- Messages are subject-based addressed
  - A message doesn’t specify destination, but a subject name
  - A message is delivered to all objects interested in message’s subject

Message Format

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>The name of the field, possibly NULL</td>
</tr>
<tr>
<td>ID</td>
<td>Integer</td>
<td>A message-unique field identifier</td>
</tr>
<tr>
<td>Size</td>
<td>Integer</td>
<td>The total size of the field (in bytes)</td>
</tr>
<tr>
<td>Count</td>
<td>Integer</td>
<td>The number of elements in the case of an array</td>
</tr>
<tr>
<td>Type</td>
<td>Constant</td>
<td>A constant indicating the type of data</td>
</tr>
<tr>
<td>Data</td>
<td>Any type</td>
<td>The actual data stored in a field</td>
</tr>
</tbody>
</table>

TIB/Rendezvous Architecture

- Publish on A
- Pub on B
- Pub on C
- Pub on D
- Pub on E
- Pub on F
- Pub on G
- Pub on H
- Pub on I
- Pub on J
- Pub on K
- Pub on L
- Pub on M
- Pub on N
- Pub on O
- Pub on P
- Pub on Q
- Pub on R
- Pub on S
- Pub on T
- Pub on U
- Pub on V
- Pub on W
- Pub on X
- Pub on Y
- Pub on Z

Multicast message on A to subscribers
Multicast message on B to subscribers

Network
Wide-area Architecture

- Use IP multicast on LANs
- Implement overlay multicast in wide-area

Communication Primitives

- `send()`: send message; non-blocking operation
- `sendreply()`: send a reply upon receiving a message; non-blocking operation
- `sendrequest()`: send message; blocks until a reply is received
- No receive operation; received messages are handled via events

Events

- To subscribe to a subject, create a listener event object
- Listener event contains reference to a callback function
- When a message arrives, create an event object and enque it in an event queue
- Each event queue is associated a dispatcher thread
- Dispatcher thread removes object at the head of the queue and invokes callback function

Processing Listener Events

Processing Incoming Messages

Queue Groups

- Assign priorities to event queues

(a) Priority scheduling of events through a queue group
(b) Semantically equivalent queue for the queue group
Naming

- A (subject) name matches a set of sender to a group of receivers
  - Does not identify a resource/object in the system
  - Consists of labels separated by "."

<table>
<thead>
<tr>
<th>Example</th>
<th>Valid?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Books.Computer_systems.Distributed_Systems</td>
<td>Yes</td>
</tr>
<tr>
<td>ftp.cuss.uu.nl</td>
<td>No (starts with a &quot;.&quot;)</td>
</tr>
<tr>
<td>NEWS.res.com.so</td>
<td>Yes</td>
</tr>
<tr>
<td>Marten_van_Steen</td>
<td>No (empty label)</td>
</tr>
<tr>
<td>Marten.R_van_Steen</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Synchronization

- Core of TIB/Rendezvous: FIFO-ordered messages per source
- In addition, transaction messaging: sending/receiving messages can be part of a transaction
  - A separate layer on top of core layer
- A transaction limited to operations that are part of only one process
- Transaction manager: stores a message until it has been delivered to all subscribers

Example

- Process P groups two publish and a receive operations in a transaction
- “Published” messages are sent to corresponding transaction managers
- Messages are published only after transaction is committed

Reliability

- Sending RV daemon
  - assigns a unique sequence number
  - stores it for 60 seconds
- Receiver RV daemon
  - detects whether a message is lost based on sequence numbers
  - request message retransmission
- Pragmatic General Multicast (PGM): scalable implementation of reliability
- Note: this is a “good-enough”, not guaranteed reliability

Security

- Goal: establish a secure channel between a publisher and a subscriber
  - Referential decoupling between publisher and subscriber is lost
- Sender publishes encrypted data including its identity
- Each subscriber sets up a secure channel with the sender
- All subscribers share the same key to decrypt messages

PGM Example

- A message is sent along a multicast tree
- A router will pass only a single NACK for each message
- A message is retransmitted only to receivers that have asked for it.
Establishing a Secure Channel

- Diffie-Helman key exchange + public-key cryptography
- Assume Alice and Bob already:
  - obtained certificates containing each other's public key
  - established a shared key $K_{AB}$ using Diffie-Helman

$$\begin{align*}
1 & \quad A, K_{AB} \rightarrow R_B \\
2 & \quad K_{AB}(R_A \oplus R_B \oplus K_{AB}) \rightarrow B \\
3 & \quad K_{AB}(R_A \oplus R_B \oplus K_{AB}) \rightarrow B
\end{align*}$$

| 19 |

Jini

- Generative communication model
- Built around the concept of tuple space
  - First proposed by Linda
- Tuple space
  - Distributed associative memory
  - Instantiated as a JavaSpace in Jini
- In addition, Jini
  - Provide distributed event and notification system
  - Allow clients discover services when become available

| 20 |

JavaSpace

- write(): create an object copy and store it in JavaSpace
- read(): return tuples from JavaSpace that match a template
- take(): like read, but removes tuple from JavaSpace

| 21 |

Example: JavaSpace “Hello World”

- Use pattern matching to get desired objects from the space
- “null” value represent wildcard
- A message object with the “content” field set to “null” will return any message object
- A message object with the content field set to “Berkeley” will only return a message object with the content set to that value

| 22 |

Example: JavaSpace “Hello World”

```java
public class Message implements Entry {
    public String content;
    public Message() {
    }
}
```

Message msg = new Message();
String content = "Hello World";
JavaSpace space = SpaceAccessor.getSpace();
space.write(msg, null, Lease.FOREVER);
System.out.println(result.content);
"Hello World"

| 23 |

Example: JavaSpace “Hello World”

```java
Message template = new Message(); //Content is null
Message result = (Message)space.read();
System.out.println(result.content);
"Hello World"
```

Long.MAX_VALUE - timeout parameter

| 24 |
**Layered Architecture of Jini**

- Transaction manager
- JavaSpace server
- Security server
- Transaction interfaces
- Events & notification
- Leasing interfaces
- Jini user-defined services
- Extra facilities
- Lookup service
- Java RMI
- Other core facilities
- Jini infrastructure

**Events**

- A client can register with an object that has events of interest
- A client can tell object to pass event to another process
- Notification implemented by remote call

**Using Events with JavaSpaces**

1. Request notification for T
2. Insert a copy of C
3. Notify when C is inserted
4. Look for tuple that matches T
5. Return C (and optionally remove it)

**JavaSpace Implementation**

- Replicate JavaSpace at all machines
- Store tuples locally, search everywhere
- Partial replication and searching
  - Use DHTs?

- Discussion: what are advantages & disadvantages of these approaches?

**Replicate Everywhere**

Process doing a write
Process doing a read
Tuple broadcast
Subspaces
Process doing a read tuple into local JavaSpace

**Search Everywhere**

Process doing a write
Inserts tuple into local JavaSpace

Discussion: what are advantages & disadvantages of these approaches?
Partial Replication and Searching

A broadcast tuple to those machines

B broadcasts template to those machines

Lookup Service

- Can be implemented using JavaSpaces
  - Each service inserts a tuple describing itself
  - JavaSpace notifies interested clients when service becomes available

- Instead, Jini provides a specialized lookup service
  - A service registers itself using (attribute, value)-pairs
  - E.g., service parameters, location

Service Item

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceID</td>
<td>The identifier of the service associated with this item</td>
</tr>
<tr>
<td>Service</td>
<td>A (possibly remote) reference to the object implementing the service</td>
</tr>
<tr>
<td>AttributeSets</td>
<td>A set of tuples describing the service</td>
</tr>
</tbody>
</table>

Predefined tuples:

<table>
<thead>
<tr>
<th>Tuple Type</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceInfo</td>
<td>Name, manufacturer, vendor, version, model, serial number</td>
</tr>
<tr>
<td>Location</td>
<td>Floor, room, building</td>
</tr>
<tr>
<td>Address</td>
<td>Street, organization, organizational unit, locality, state or province, postal code, country</td>
</tr>
</tbody>
</table>

Transactions

- Aim to provide ACID properties
  - Atomicity: all operations of a transaction take place, or none of them do
  - Consistency: completion of a transaction must leave the participants in a “consistent” state
  - Isolation: activities of one transaction must not affect any other transactions
  - Durability: results of a transaction must be persistent

- Jini
  - Supply the mechanism of two-phase commit protocol
  - Leave the policy to the participants in a transaction

Transactions

- A transaction is represented by a long identifier, obtained from a transaction manager
- Each transaction is associated a lease; if lease expires, transaction is aborted