References: CACM, October 2003, Vol. 46, No. 10 (Mark Little)
Distributed System (Algorithms)

- Centralized Algorithms

  Coordinator Site → sites connect; Submit request to access resources at distributed databases

- Distributed Algorithms

  Resource Coordinators → Many resource Coordinators

  → Prevent the Central Coordinator failure problem.

- Peer-to-peer (P2P) Algorithms

  Fully Distributed Algorithms

- Current Protocols on LAN based Systems → are often Distributed Algorithms
CASE 1 → Distributed Deadlock Detection

+ Blocked Transaction →
Generate TWFG for its own detection (Distributed approach)

+ Blocked Transaction →
Send out a Probe for its own detection (Distributed approach)

• Existing Algorithms do not consider P2P solutions

→ Cost of P2P algorithms is high
  ( example - Distributed Mutual Exclusion ).
[I] Distributed System

- **CASE 1** → Distributed Commit Algorithm

- 2 Phase Commit

Transaction Aromicity → All sites must commit a Transaction or reject it

- Coordinator

- Informs about a request + in Phase 2 obtains an agreement

Distributed Algorithm does not use a P2P solution
[I] Distributed System

- **LECTURE 1** - Air Traffic Controls Application over Cloud Computing
  - P2P Solution (else Network Partitioning? 2 Coordinators?)

- **LECTURE 5** - Web Services for E-Commerce Applications
  - P2P Solution (except the directory services)
## Distributed System

<table>
<thead>
<tr>
<th>client</th>
<th>• Dynamic Discovery → finds a service</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>• Dynamic Discovery → finds a client</td>
</tr>
</tbody>
</table>

### Outline
- [A] Dynamically
- [B] Over the network
- [C] No prior knowledge of each other
- [D] Based on IP Multicast
- [E] Need! →
- Automatic Self-healing network
- [F] Show status of all available and running services
- Jini —→ also called Apache River
  —→ a network architecture for the construction of distributed systems in the form of modular co-operating services.
  —→ Originally developed by Sun,
  —→ released under an open source license (Apache license...
Services try to contact a lookup service (LUS), by dynamic multicast discovery.

The strategy is more convenient than Java remote method invocation, which requires the client to know the location of the remote service in advance.
Jini uses a lookup service to broker communication between the client and service.

This uses a centralized model

It does not scale well to very large systems.
### Summary - Jini

- **Jini** - Distributed System based on LAN

- Shows status of all available and running services

- No human administration

- If a service fails or stops → automatic update of system information

- Jini - Maintains a location service (directory)

- Based on JAVA (minimum support + JAVA)
Vendor Independence (use J2EE compliant server)
company (use J2EE compatible clients)

[A] server supports:
- Security
- Transactions: Concurrency, logging, 2 phase commit
- persistence
- networking code implementation

[B] Based on JAVA

[C] Based on LAN

[C] Focus on learn one API - work on Business logic
Web Services
• Standards ← being worked out
• Tools ← to be developed
• Aim ← Interoperability

Business Applications →
Old legacy application [Web Services] →
XML interface with SOAP messages
• Security : (!) HTTPS for mutual authentication; XML ? text
• Transactions : No support
• Dynamic Discovery [OK]
• Not JAVA based →

J2EE may support Web Services interface for Web (outside)
Service

Steps

• 1. Send a message from one software system to another may not have human involvement

result

• Starts with software but may move a Robot

example

• [A] Send text to high speed printer
• [B] Huge calculations for Genetic Match service
• [C] Play a game of GO
• [D] Buy concert tickets; Reserve travel
• [E] move a camera
• [F] Dail a phone, send a FAX

definition

• [Aim] Same as Object-Oriented programs
  - Have least amount of coupling
  - client ←→ Server
  → know least about one another
Service - How to use

- How do clients find a service?
- How to know what can be done?
- Which methods to call?

→ Service needs to declare itself

- **Service Interface** - Jini / J2EE / Web Services
  - use different ways
- **Service** - Methods!
  - Arguments to Methods!
  - return values
**Service Interface**

<table>
<thead>
<tr>
<th>Interface: Advice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method: one</td>
</tr>
<tr>
<td>: getAdvice()</td>
</tr>
<tr>
<td>Return Parameter: string</td>
</tr>
<tr>
<td>+ useful random advice</td>
</tr>
<tr>
<td>Input Parameter: null</td>
</tr>
</tbody>
</table>

- Jini and J2EE(EJB): use JAVA Remote Interface (Based on RMI)

getAdvice() - declare a Remote Exception → java.rmi.Remote

- [Jini → Jini]; [J2EE → J2EE]
Inter-Enterprise Level

Company A

System – System (workflow)

Process

Applications

Internet (Dedicated links (?) LAN ?)

Company B

- Car Maker (A) ←→ Part Supplier (B)

- Traditional Transaction Semantics [ X ]

- Traditional Transaction Protocols [ X ]
Web Services Transactions

- Long Duration of activity
- Commitment to be negotiated at run time
- Isolation Levels?
- ACID properties?
- Inter-Enterprise Level differences
<table>
<thead>
<tr>
<th>BT</th>
<th>Web Services Transactions (WST)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Transactions (BT)</td>
<td>Web Services Transactions (WST)</td>
</tr>
<tr>
<td>[A] Need → extended Transaction Model</td>
<td></td>
</tr>
<tr>
<td>[B] need → defined Interoperable Transaction Protocol</td>
<td></td>
</tr>
<tr>
<td>[C] Message Flows → help to negotiate transaction guarantees at inter-enterprise level</td>
<td></td>
</tr>
<tr>
<td>Proposal</td>
<td>Business Transaction Protocol (BTP)</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for Advanced Structured Information Systems</td>
</tr>
</tbody>
</table>
| Meets requirements | [A] long running  
[B] collaborative  
[C] business applications  
[D] allows relaxation of Isolation levels  
[E] ACID property implementation in a controlled manner  
[F] Suitable for Web Services Environment |
[I] Concept of Atomic Transactions

Multi-party (multiple) business operations $\rightarrow$ Guaranteed Consistent Outcome

- Multiple participants with separate concerns
- final state $\rightarrow$ Acceptable to all participants
- final state $\rightarrow$ Consistent with business rules
Concept of Atomic Transactions

- Two Phase Commit Protocol
# Concept of Atomic Transactions

<table>
<thead>
<tr>
<th>State</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Consistent: Supported by rollback</td>
<td>• Final outcome - consistent with all</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>• [A] Concensus among parts</td>
</tr>
<tr>
<td>• [B] 1. State Changes → temporary</td>
</tr>
<tr>
<td>→ LOG (used for redo/undo)</td>
</tr>
<tr>
<td>• [C] 2. Understand → outcome at each level</td>
</tr>
<tr>
<td>• [D] 3. support → completion (finish/cancel)</td>
</tr>
<tr>
<td>• [E] Steps (1.2.3) → decided by workflow</td>
</tr>
<tr>
<td>Long Transaction</td>
</tr>
<tr>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>• Internet - slow and asynchronous</td>
</tr>
<tr>
<td>- no guarantee</td>
</tr>
<tr>
<td>• Inter-enterprise business</td>
</tr>
<tr>
<td>• Tasks - send an order / wait for despatch</td>
</tr>
<tr>
<td>• More steps as verification, validation, quality control</td>
</tr>
</tbody>
</table>
[I] Concept of Atomic WS Transactions

Application activity

failure

t1 ➔ t2 ➔ t4 ➔ tcl ➔ t5' ➔ t6'
t3 ➔
time
<table>
<thead>
<tr>
<th>I] ACID property</th>
<th>[26]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ACID</strong></td>
<td>• too strong for Long Duration Transactions</td>
</tr>
<tr>
<td><strong>Need</strong></td>
<td>• Can guarantee consistency in the presence of failures</td>
</tr>
</tbody>
</table>
| **Assume**                      | • [A] Short transaction  
|                                  | • [B] Executes in closely coupled environment  
|                                  | • [C] Performs stable state changes  |
| **BTP**                          | • Long lived Application Functions  
|                                  | • Copy ACID ideas  
|                                  | • [A] Phase I - make participant changes durable (LOG)  
|                                  | → rollback.commit at final steps  
|                                  | • [B] Protocol to get a consensus in Phase II  
|                                  | → may overwrite earlier values  
|                                  | • [C] Problem **Blocking** → Long  
|                                  | - if coordinator fails before Phase 2 |
Long-Running Activity

- Long Transaction $\rightarrow$ many independent short duration transactions ($t_{11} + t_{12} + t_{13} + ...$)
- Logical Long Running Transaction $\rightarrow$ sum($t_{11} + t_{12} + ...$)

- **purpose** - structure of short duration transaction
  ... acquire and use resources for only the short required duration
- **each ti** $\rightarrow$ a part $\rightarrow$ a coordinated split part
<table>
<thead>
<tr>
<th>Example - Long running transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>2PC</strong></td>
</tr>
<tr>
<td>• Not for all but only for sub-transaction</td>
</tr>
<tr>
<td><strong>Overall</strong></td>
</tr>
<tr>
<td>• A final check at completion time for activity</td>
</tr>
<tr>
<td><strong>example</strong></td>
</tr>
<tr>
<td>• [A] t1: Book a Taxi</td>
</tr>
<tr>
<td>• [B] t2: reserve a table at restaurant</td>
</tr>
<tr>
<td>• [C] t3: reserve a seat at a theater</td>
</tr>
<tr>
<td>• [D] t4: book room at a hotel</td>
</tr>
<tr>
<td>• [E] ...</td>
</tr>
<tr>
<td>• [Poit] if t2 - t6 do not need → why block t1 resources until the end</td>
</tr>
<tr>
<td>• [Failure] (and if concurrent access) - run a compensating transaction [t4 aborts, tcl is run and t6 and t5 are changed]</td>
</tr>
</tbody>
</table>
BTP - Ensure →
atomicity between multiple participants

2 Phase Completion →
- prepare
- confirm
- cancel / finish

Does not imply ACID

Back-end Implementation →
+ details of prepare / confirm / cancel!
+ details of Consistency and Isolation
Exercise

- Use Traditional Models:
  Ensure pair-wise communication (use atomicity between multiple participants)

1. Consider an interaction based Deadlock detection procedure for web services environment using (a) Transaction wait for graph (TWFG) method, (b) Probe Method

2. Consider a pair-wise interaction based transaction commit procedure for Web services environment