[I] Distributed System (Algorithms) [2]
- 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

[I] Distributed System [4]
- CASE 1 → Distributed Commit Algorithm
- 2 Phase Commit
  Transaction Aromaticity → 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 (Algorithms) [3]
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)

University of Aizu
CSV888: Building Reliable Networked Applications
2014 - Transactions and Web Services
References: CACM, October 2003, Vol. 46, No. 10 (Mark Little)
**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)

---

**Example - 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)

---

**Example - J2EE**

- Vendor Independence (use J2EE compliant server)
  - company (use J2EE compatible clients)
  - J2EE - Automatic Self healing network
    - [A] server supports:
      - Security
      - Transactions: Concurrency, logging, 2 phase commit
      - persistence networking code implementation
    - [B] Based on JAVA
    - [C] Based on LAN
    - [D] Focus on learning one API - work on Business logic

---

**Outline**

- Dynamic Discovery → finds a service
  - [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
[I] 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

---

[I] 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]

[J2EE] may support Web Services interface for Web (outside)

---

[I] Service Interface

Interface : Advice
Method: one
  : getAdvice()
Return Parameter: string
  + useful random advice
Input Parameter: null

• Jini and J2EE(EJB) : use JAVA Remote Interface (Based on RMI)
getAdvice() - declare a Remote Exception → java.rmi.Remote
[Jini → Jini]; [J2EE → J2EE]

---

[I] 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
## Inter-Enterprise Level

- System - System (workflow)
- Process
- Applications

Internet (Dedicated links (?), LAN ?)

- Car Maker (A) \(\leftrightarrow\) Part Supplier (B)
- Traditional Transaction Semantics
- Traditional Transaction Protocols

## Web Service Model

- Web Services
- WSDL
- XML Interface
- WS

J2EE

- EJB
- Java Interface
- XML WSDL
- development tool

## Web Services Transactions

- Long Duration of activity
- Commitment to be negotiated at run time
- Isolation Levels
- ACID properties
- Inter-Enterprise Level differences

## Business Transactions (BT)

- Business Transactions (BT)
- Web Services Transactions (WST)
- [A] Need → extended Transaction Model
- [B] need → defined Interoperable Transaction Protocol
- [C] Message Flows → help to negotiate transaction guarantees at inter-enterprise level
**[I] Concept of Atomic Transactions** [18]

- Multiple participants with separate concerns
- Final state → Acceptable to all participants
- Final state → Consistent with business rules


- Business Transaction Protocol (BTP)
- Organization for Advanced Structured Information Systems

**[I] Concept of Atomic Transactions** [19]

- [A] Concensus among parts
- [B] 1. State Changes → temporary
  → LOG (used for redo/undo)
- [C] 2. Understand → outcome at each level
- [D] 3. support → completion (finish/cancel)
- [E] Steps (1.2.3) → decided by workflow

**State**
- Consistent: Supported by rollback

**Overall**
- Final outcome - consistent with all

**Outline**
- Concensus among parts
- 1. State Changes → temporary
  → LOG (used for redo/undo)
- Understand → outcome at each level
- support → completion (finish/cancel)
- Steps (1.2.3) → decided by workflow

**Two Phase Commit Protocol**
**Concept of Atomic WS Transactions**

- Internet - slow and asynchronous
- Inter-enterprise business
- Tasks - send an order / wait for despatch
- More steps as verification, validation, quality control

**Long-Running Activity**

- Long Transaction → many independent short duration transactions (t1 + t2 + t3 + ...)
- Logical Long Running Transaction → sum(t1 + t2 + ...)
- **purpose** - structure of short duration transaction
  - acquire and use resources for only the short required duration
- Each ti → a part → a coordinated split part

**ACID property**

<table>
<thead>
<tr>
<th>ACID</th>
<th>Need</th>
<th>Assume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too strong for Long Duration Transactions</td>
<td>Can guarantee consistency in the presence of failures</td>
<td>[A] Short transaction</td>
</tr>
<tr>
<td>[B] Executes in closely coupled environment</td>
<td>[C] Performs stable state changes</td>
<td></td>
</tr>
<tr>
<td>Long lived Application Functions</td>
<td></td>
<td>[A] Phase I - make participant changes durable (LOG) rollback.commit at final steps</td>
</tr>
<tr>
<td>Copy ACID ideas</td>
<td></td>
<td>[B] Protocol to get a consensus in Phase II may overwrite earlier values</td>
</tr>
<tr>
<td>[C] Problem</td>
<td>[Blocking] → Long</td>
<td></td>
</tr>
<tr>
<td>- if coordinator fails before Phase 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
[I] Example - Long running transaction  [25]

2PC

- Not for all but only for sub-transaction
- A final check at completion time for activity

Example

- [A] t1: Book a Taxi
- [B] t2: reserve a table at restaurant
- [C] t3: reserve a seat at a theater
- [D] t4: book room at a hotel
- [E] ...
- [Point] if t2 - t6 do not need to block t1 resources until the end
- [Failure] (and if concurrent access) - run a compensating transaction [t4 aborts, tci is run and t6 and t5 are changed]

[I] Exercise  [27]

- 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