Midterm exam

- Tuesday, Oct 28th
- 1:15pm – 2:40pm
- Location: Science II 140
- One page of letter-size, one-sided summary sheet allowed.
- No cellphone, smartphone, tablet, laptop, etc.
- Covers up to lecture on Oct 21st.
- Be on time!
Disclaimer

• This review helps you better prepare for the midterm exam.
• It is by no means comprehensive.
• Study all materials for the exam.
Midterm topics

- Introduction to Distributed Systems
- Distributed Architectures
- Communication in Distributed Systems
- RPC & RMI
- Naming
- Synchronization & Coordination
- Client and Server Design Issues
Introduction to distributed systems

- Heterogeneity: (need for openness)
  - hardware, platforms, languages
  - interoperability and portability

  *key interfaces* in software and communication protocols need to be standardized, e.g., Interface Definition Language (IDL)

- Scalability
  - size, geographical, administrative
  - scalability techniques

- Transparency
  - access, location, relocation, migration, replication, concurrency, failure

- ...
Scalability

- **Size scalability**
  - Number of users and/or processes
  - Concerning centralized services/data/algorithm

- **Geographical scalability**
  - Maximum distance between nodes
  - Synchronous communication in LAN vs. asynchronous communication in WAN

- **Administrative scalability**
  - Number of administrative domains
  - Policy conflicts from different orgs (e.g., for security, access control)
Scalability techniques

• Hiding communication latency
  • Asynchronous communication
  • Code migration (to client)

• Distribution
  • Splitting a large components to parts (e.g., DNS)

• Replication
  • Caching (decision of clients vs. of the server)
  • On demand (pull) vs. planned (push)
Transparency

<table>
<thead>
<tr>
<th>Transparency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Hide differences in data representation and how an object is accessed</td>
</tr>
<tr>
<td>Location</td>
<td>Hide where an object is located</td>
</tr>
<tr>
<td>Relocation</td>
<td>Hide that an object may be moved to another location while in use</td>
</tr>
<tr>
<td>Migration</td>
<td>Hide that an object may move to another location</td>
</tr>
<tr>
<td>Replication</td>
<td>Hide that an object is replicated</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Hide that an object may be shared by several independent users</td>
</tr>
<tr>
<td>Failure</td>
<td>Hide the failure and recovery of an object</td>
</tr>
</tbody>
</table>

Achieving transparency is a very difficult problem.
Distributed architectures

• Software architecture
  • Logical organization of the collection of software components that make up a distributed application
  • Layered architectures, Object-based architectures, Data-centered architectures, Event-based architectures
  • Application layering for client-server systems

• System architecture
  • Instantiation of a software architecture
  • i.e., physical placement of software components on computers
  • Centralized architectures, Decentralized architectures, Hybrid architectures
Application layering

- User interface
  - Keyword expression
  - Database queries
  - Database with Web pages

- Query generator
  - HTML generator
  - Ranking algorithm
    - Ranked list of page titles
    - Web page titles with meta-information
  - HTML page containing list

User-interface level
Processing level
Data level
Two-tiered architecture

(a) Thin Client

(b) (c) (d) (e) Fat Client
Communication in distributed systems

- Types of communication
  - Synchronous vs. Asynchronous
  - Transient vs. Persistent

- Communication models
  - Remote procedure call
    - steps of a remote procedure call
    - pass value parameters, remote/local object references in RPC and RMI
  - Message-oriented communication
    - message queuing system
Steps of a Remote Procedure Call

1) Client procedure calls client stub in normal way
2) Client stub builds message, calls local OS
3) Client's OS sends message to remote OS
4) Remote OS gives message to server stub
5) Server stub unpacks parameters, calls server
6) Server does work, returns result to the stub
7) Server stub packs it in message, calls local OS
8) Server's OS sends message to client's OS
9) Client's OS gives message to client stub
10) Stub unpacks result, returns to client
RPC & RMI

• Five different classes of failures can occur in RPC systems
  • The client is unable to locate the server
  • The request message from the client to the server is lost
  • The reply message from the server to the client is lost
  • The server crashes after receiving a request
  • The client crashes after sending a request

• Idempotent request vs. non-idempotent request

• Invocation semantics
  • maybe, at-least-once, at-most-once, exactly-once
## Invocation semantics

<table>
<thead>
<tr>
<th>Retransmit request message</th>
<th>Duplicate filtering</th>
<th>Re-execute procedure or retransmit reply</th>
<th>Invocation semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Maybe</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Re-execute procedure</td>
<td>At-least-once</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Retransmit reply</td>
<td>At-most-once</td>
</tr>
</tbody>
</table>
Naming

• Flat naming
  • Broadcasting and multicasting, forwarding pointers, home-based approach, distributed hash table, hierarchical approach

• Structured naming
  • Closure mechanism, linking and mounting
  • Naming service, DNS
Lookup in Chord: finger table

- The $i^{th}$ entry in the table at node $n$ contains the identity of the first node, $s$, that succeeds $n$ by at least $2^{i-1}$ on the ID circle.
- $O(\log N)$ steps

### Table: Start, Interval, Succ.

<table>
<thead>
<tr>
<th>Start</th>
<th>Interval</th>
<th>Succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[2, 3)</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>[3, 5)</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>[5, 9)</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>[9, 1)</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>[8, 9)</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>[9, 10)</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>[11, 15)</td>
<td>12</td>
</tr>
<tr>
<td>15</td>
<td>[15, 7)</td>
<td>15</td>
</tr>
</tbody>
</table>
Lookup in Chord: finger table

Resolving key 26 from node 1 and key 12 from node 28 in a Chord system.
Implementation of name resolution

- The principle of iterative name resolution.
Implementation of name resolution

• The principle of recursive name resolution.
Synchronization & coordination

• Physical clock synchronization algorithms:
  • Cristian, NTP, Berkeley algorithm

• Logical time and clock
  • Lamport timestamps, vector timestamps, totally ordered logical clocks

• Distributed mutual exclusion
  • Centralized algorithm, Distributed (Ricart-Agrawala’s) algorithm, Ring-based algorithm, Maekawa’s algorithm

• Election algorithm
  • Bully algorithm, Ring algorithm
Cristian’s algorithm

- \( p \) should set its time to \( t + T_{\text{round}}/2 \)
- If the value of the minimum transmission time \( \textit{min} \) is known or can be conservatively estimated
  - Earliest time at which \( S \) could have placed its time in \( m_t \) was \( \textit{min} \) after \( p \) dispatched \( m_r \)
  - Latest point at which it could do so was \( \textit{min} \) before \( m_t \) arrived at \( p \)
  - Time by \( S \)’s clock when message arrives at \( p \) is in range \( [t + \textit{min}, t + T_{\text{round}} - \textit{min}] \)
  - Accuracy \( \pm (T_{\text{round}}/2 - \textit{min}) \)
The Berkeley algorithm

a) The time daemon asks all the other machines for their clock values
b) The machines answer
c) The time daemon tells everyone how to adjust their clock
Lamport’s algorithm

- Each process has its own logical clock
- LC1: $C_p$ is incremented before each event at process $p$
- LC2:
  - When process $p$ sends a message it piggybacks on it the value $C_p$
  - On receiving a message $(m, t)$, a process $q$ computes $C_q = \max(C_q, t)$ and then applies LC1 before timestamping the receive event
Vector timestamps

- Shortcoming of Lamport’s clocks: if $L(e) < L(f)$, we cannot conclude that $e \rightarrow f$
- Vector clocks
  - A process keeps an array of clocks, one for each process
  - Like Lamport timestamps, processes piggyback vector timestamps on messages they send each other
    \begin{align*}
    V &= W \text{ iff } V[j] = W[j] \text{ for } j = 1, 2, \ldots, N \\
    V &\leq W \text{ iff } V[j] \leq W[j] \text{ for } j = 1, 2, \ldots, N \\
    V &< W \text{ if } V \leq W \text{ and } V \neq W
    \end{align*}
Vector timestamps for the events

Physical time

(1,0,0) (2,0,0)

(0,0,1) (2,2,2)

p_1

a b m_1

p_2

(2,1,0) (2,2,0)
c d m_2

p_3

e f

p_1, p_2, p_3
Server managing a mutual exclusion token for a set of processes

1. Request token
2. Release token
3. Grant token

Queue of requests
Ricart and Agrawala’s algorithm

On initialization

\[ \text{state} := \text{RELEASED}; \]

To enter the section

\[ \text{state} := \text{WANTED}; \]

\{ request processing deferred here \}

Multicast request to all processes;

\[ T := \text{request’s timestamp}; \]

Wait until (number of replies received = \((N - 1)\));

\[ \text{state} := \text{HELD}; \]

On receipt of a request \(<T_i, p_i>\) at \(p_j\) \((i \neq j)\)

\[ \text{if } (\text{state} = \text{HELD} \text{ or } (\text{state} = \text{WANTED} \text{ and } (T_j, p_j) < (T_i, p_i))) \text{ then} \]

queue request from \(p_i\) without replying;

\[ \text{else} \]

reply immediately to \(p_i\);

end if

To exit the critical section

\[ \text{state} := \text{RELEASED}; \]

reply to any queued requests;
Multicast synchronization

\[ \text{Multicast synchronization} \]
A ring of processes transferring a mutual exclusion token
Comparison

• A comparison of three mutual exclusion algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Messages per entry/exit</th>
<th>Delay before entry (in message times)</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>3</td>
<td>2</td>
<td>Coordinator crash</td>
</tr>
<tr>
<td>Distributed</td>
<td>2 (n – 1)</td>
<td>2 (n – 1)</td>
<td>Crash of any process</td>
</tr>
<tr>
<td>Token ring</td>
<td>1 to ∞</td>
<td>0 to n – 1</td>
<td>Lost token, process crash</td>
</tr>
</tbody>
</table>
Maekawa’s algorithm

• Every node needs permission from other nodes in its quorum before it enters critical section
• Quorums are constructed in such a way that no two nodes can be in their critical section at the same time
• The size of each node’s quorum is $O(\sqrt{N})$, which can be shown to be optimal
Construction of quorum sets

Consider a system with 9 nodes

The quorum for any node includes the nodes in its row and column

Quorum for Node 1 = \{1,2,3,4,7\}
Quorum for Node 9 = \{3,6,7,8,9\}

There is a non-null intersection for the quorums of any two nodes
Election algorithms

• An election is a procedure carried out to choose a process from a group, for example to take over the role of a process that has failed

• Main requirement: elected process should be unique even if several processes start an election simultaneously

• Algorithms:
  • Bully algorithm: assumes all processes know the identities and addresses of all the other processes
  • Ring-based election: processes need to know only addresses of their immediate neighbors
The bully algorithm

The election of coordinator $p_2$, after the failure of $p_4$ and then $p_3$
A ring-based election in progress

Note: The election was started by process 17. The highest process identifier encountered so far is 24. Participant processes are shown darkened.
Client and server design issues

- Client design issues
  - multithreading, transparency, thin clients
- Server design issues
  - Organization of the server
    - multithreaded, single threaded, multi-process, iterative, concurrent, finite state machine
  - How to contact the server
    - Service end points, super server
  - Stateful or Stateless
  - How to interrupt the server
  - Access Transparency
    - TCP handoff, remote-access model, upload-download model