Outline

- Architectures
- Peer-to-Peer Computing
  - Introduction
  - Chord
- Q&A
Architectures

- Software Architecture
  - How Software Components Are Organized
  - How Software Components Should Interact

- System Architecture
  - Final Instantiation of a Software Architecture

- Important Styles of Architecture for (Autonomic) Distributed Systems
  - Layered Architectures
  - Object-Based Architectures
  - Data-Centered Architectures
  - Event-Based Architectures
Architectural Styles

(a) layered architectural style
(b) The object-based architectural style
(a) The event-based architectural style
Architectural Styles (Cont’d)

(b) The shared data-space architectural style
System Architectures

- **Centralized Architecture**
  - Clients That Request Services from Servers
  - Support for Vertical Distribution
    - Placing different components on different machines

- **Decentralized Architecture**
  - Process Being a Client and a Server
  - Support for Horizontal Distribution
    - Spitting up a client or server physically into logically equivalent parts with each part operating on its own share of data set
Peer-to-Peer Architectures

- **Overlay Network**
  - Network in which the nodes are formed by the processes and the links represent the possible communication channels

- **Structured P2P Architecture**
  - Overlay network is constructed using a deterministic procedure
    - Distributed Hash Table (DHT)

- **Unstructured P2P Architecture**
  - Overlay network is constructed using a random algorithm
Centralized Architectures

General interaction between a client and a server
Application Layering

- Following Layered Architectural Style
  - User-Interface Level
  - Processing Level
  - Data level
The simplified organization of an Internet search engine into three different layers

User interface

Keyword expression

Query generator

HTML page containing list

HTML generator

Ranked list of page titles

Ranking algorithm

Web page titles with meta-information

Database with Web pages
Multitiered Architectures

- The simplest organization is to have only two types of machines:
  - A client machine containing only the programs implementing (part of) the user-interface level
  - A server machine containing the rest,
    - the programs implementing the processing and data level
Multitiered Architectures (Cont’d)

Alternative client-server organizations (a)–(e)
Multitiered Architectures (Cont’d)

User interface (presentation)

Application server

Database server

Wait for result

Request operation

Wait for data

Request data

Return data

Return result

Time

An example of a server acting as client
Structured Peer-to-Peer Architectures

The mapping of data items onto nodes in Chord

Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc. All rights reserved
Structured Peer-to-Peer Architectures (Cont’d)

The mapping of data items onto nodes in CAN
Peer-to-Peer (P2P) Computing

Definition

- Computing by Sharing Data & Resources on a Very Large Scale w/o Server Requirements

Important Characteristics

- Each Node’s Resource Contribution
- Same Functional Capabilities & Responsibilities of Nodes
- No Central Administration
- Limited Degree of Anonymity
- Unpredictable Availability
- Fault Tolerance

Key Issue: Efficient Data Placement & Access
1st-Generation P2P Systems

- File Sharing and Storage Applications
  - Napster Music Exchange Service
    - Use of central servers to locate files
  - Gnutella
    - Distributed service using scoped broadcast queries

Main Problem: Limited Scalability or No Guarantee That Files Can Be Located
2nd-Generation P2P Systems

Middleware

- Application-Independent Management of Distributed Resources on a Global Scale
  - Routing Overlay for locating nodes and objects
    - Scalable
    - Load balanced
    - Adaptive to network dynamics
    - Fault tolerant
    - Efficiently discovering
    - Secure

Implementing Key-Based Routing (KBR) Interface:
Routing of Messages to a Live Node Responsible for the Destination Key

Using Randomly Distributed Keys to Determine the Placement of Objects and to Retrieve Them
## IP vs Overlay Routing

<table>
<thead>
<tr>
<th></th>
<th>IP</th>
<th>Application-level routing overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scale</strong></td>
<td>IPv4 is limited to (2^{32}) addressable nodes. The IPv6 name space is much more generous ((2^{128})), but addresses in both versions are hierarchically structured and much of the space is pre-allocated according to administrative requirements.</td>
<td>Peer-to-peer systems can address more objects. The GUID name space is very large and flat ((&gt;2^{128})), allowing it to be much more fully occupied.</td>
</tr>
<tr>
<td><strong>Load balancing</strong></td>
<td>Loads on routers are determined by network topology and associated traffic patterns.</td>
<td>Object locations can be randomized and hence traffic patterns are divorced from the network topology. Routing tables can be updated synchronously or asynchronously with fractions of a second delays. Routes and object references can be replicated (n)-fold, ensuring tolerance of (n) failures of nodes or connections.</td>
</tr>
<tr>
<td><strong>Network dynamics</strong></td>
<td>IP routing tables are updated asynchronously on a best-efforts basis with time constants on the order of 1 hour. Redundancy is designed into the IP network by its managers, ensuring tolerance of a single router or network connectivity failure. (n)-fold replication is costly.</td>
<td></td>
</tr>
<tr>
<td><strong>Fault tolerance</strong></td>
<td>Each IP address maps to exactly one target node.</td>
<td>Messages can be routed to the nearest replica of a target object. Security can be achieved even in environments with limited trust. A limited degree of anonymity can be provided.</td>
</tr>
<tr>
<td><strong>Target identification</strong></td>
<td>Addressing is only secure when all nodes are trusted. Anonymity for the owners of addresses is not achievable.</td>
<td></td>
</tr>
<tr>
<td><strong>Security and anonymity</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Structured P2P Overlay Networks

Supporting Higher-Level Interfaces

- Distributed Hash Table (DHT)
  - Basic Interface: put(), get(), remove()
  - E.g., Pastry
- Distributed Object Location & Routing (DOLR)
  - Basic Interface: publish(), unpublish(), routeToObject()
  - E.g., Tapestry

Ignoring/Considering Network Distances

- Shortest Overlay-Hop Routing
  - E.g., Chord
- Locally Optimal Routing
  - E.g., Tapestry
Chord Protocol [Keifer03]

Simple Key Location

```c
// ask node n to find the successor of id
n.find_successor(id)
    if (id ∈ (n; successor])
        return successor;
    else
        // forward the query around the circle
        return successor.find_successor(id);
```
Cord Protocol (Cont’d)

- Scalable Key Location

\[
finger[i] = \text{successor} \ (n + 2^{i-1})
\]
Cord Protocol (Cont’d)

Scalable Key Location

```cpp
// ask node n to find the successor of id
n.find_successor(id)
if (id ∈ (n; successor])
    return successor;
else n0 = closest_preceding_node(id);
return n0.find_successor(id);
```

```cpp
// search the local table for the highest
// predecessor of id
n.closest_preceding_node(id)
for i = m downto 1
    if (finger[i] ∈ (n; id))
        return finger[i];
return n;
```
Cord Protocol (Cont’d)

- Node Joining/Leaving
Cord Protocol (Cont’d)

Properties of Chord

- Load Balance
  - Acting as a Distributed Hash Function

- Decentralization
  - Fully distributed

- Scalability
  - Lookup cost growing as the log of # of nodes

- Availability
  - Enabling the node responsible for a key to be found via automatic internal-table adjustment

- Flexible naming
  - Using flat key-space