DISTRIBUTED SYSTEMS

Introduction

Dr. Yingwu Zhu
Definition of a Distributed System (1)

A collection of independent computers that appears to its users as a single coherent system.
Definition of a Distributed System (2)

A distributed system organized as middleware. The middleware layer extends over multiple machines, and offers each application the same interface.

A distributed system organized as middleware. The middleware layer extends over multiple machines, and offers each application the same interface.
## Transparency in a Distributed System

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

Different forms of transparency in a distributed system (ISO, 1995).
Can centralized systems work?
Centralized Model

- No networking
- Traditional time-sharing system
- Direct connection of user terminals to system
- One or several CPUs
- Not easily scalable
- Limiting factor: number of CPUs in system
  - Contention for same resources
Client-Server Model

• Environment consists of **clients and servers**
• **Service**: task machine can perform
• **Server**: machine that performs the task
• **Client**: machine that is requesting the service
# Scalability Problems

Examples of scalability limitations for centralized models.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized services</td>
<td>A single server for all users</td>
</tr>
<tr>
<td>Centralized data</td>
<td>A single on-line telephone book</td>
</tr>
<tr>
<td>Centralized algorithms</td>
<td>Doing routing based on complete information</td>
</tr>
</tbody>
</table>
The difference between letting (a) a server or (b) a client check forms as they are being filled.

“Empower the clients!”
Scaling Techniques (2)

An example of dividing the DNS name space into zones.
“(Data & Computing) Distribution/partition”
Scaling Techniques (3)

- Replication (data & services)
  - Availability
  - Load balancing
  - Improve latency by accessing the nearby copy
  - Caching
- Downsides
  - Consistency issues: modification on one copy → other copies?
Distributed Systems
Characteristics of decentralized algorithms

- No machine has complete information about the system state. (partial info)
- Machines make decisions based only on local information. (local decisions)
- Failure of one machine does not ruin the algorithm. (failure mask)
- There is no implicit assumption that a global clock exists. (No global clock / asynchrony)
  (No exact clock synchronization), clock drift!
Pitfalls when Developing Distributed Systems

False assumptions made by first time developer:

- The network is reliable.
- The network is secure.
- The network is homogeneous.
- The topology does not change.
- Latency is zero.
- Bandwidth is infinite.
- Transport cost is zero.
- There is one administrator.
Types of Distributed Computing Systems

For High-Performance Computing

• Cluster Computing
  • similar hardwares (workstations, or PCs)
  • high-speed LANs

• Grid Computing
  • Constructed by a federation of computing systems under different admin. domains, different hardware, software and networking technology!
  • Higher heterogeneity, geographically distributed
  • Seamless access to storage, processing & bandwidth
  • Build a “supercomputer” on the fly via networked, loosely coupled computers
## Transaction Processing Systems (1)

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN_TRANSACTION</td>
<td>Mark the start of a transaction</td>
</tr>
<tr>
<td>END_TRANSACTION</td>
<td>Terminate the transaction and try to commit</td>
</tr>
<tr>
<td>ABORT_TRANSACTION</td>
<td>Kill the transaction and restore the old values</td>
</tr>
<tr>
<td>READ</td>
<td>Read data from a file, a table, or otherwise</td>
</tr>
<tr>
<td>WRITE</td>
<td>Write data to a file, a table, or otherwise</td>
</tr>
</tbody>
</table>

Example primitives for transactions.
Transaction Processing Systems (2)

Characteristic properties of transactions:

• **Atomic**: To the outside world, the transaction happens indivisibly.

• **Consistent**: The transaction does not violate system invariants.

• **Isolated**: Concurrent transactions do not interfere with each other.

• **Durable**: Once a transaction commits, the changes are permanent.
A nested transaction.
The role of a TP monitor in distributed systems.
ARCHITECTURES
Components (modular unit)
- Provide interfaces
- Can be replaced if we respect its interfaces

Connector:
- Meditates communication, coordination, or cooperation among components
- Eg. RPCs
Architectural Styles (2)

Important styles of architecture for distributed systems

• Layered architectures
• Object-based architectures
• Data-centered architectures
• Event-based architectures
Architectural Styles (3)

The layered architectural style and ...
E.g., network layers
(b) The object-based architectural style. Object $\rightarrow$ component, e.g., client/server architecture.
Architectural Styles (5)

(b) The shared data-space architectural style. E.g., networked apps rely on a distributed file system where all communication takes place via files.
Figure 2-2. (a) The event-based architectural style and … e.g., publish/subscribe systems
System Architecture

- Centralized architecture
- Decentralized architecture
Centralized Architectures

General interaction between a client and a server.
Client/Server architecture

- The user-interface level
- The processing level
- The data level
The simplified organization of an Internet search engine into three different layers.
2-Tiered Architectures

- Common from mid 1980’s-early 1990’s
- 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 (UI)
  - A server machine containing the rest,
  - the programs implementing the processing and data level
Multitiered Architectures

Alternative client-server organizations (a)–(e).

Clients: thin $\rightarrow$ fat
3-tiered Architecture (3)

3-tiered architecture.
The role of clients & servers blurs! Server-side solutions become distributed across multiple machines!
3-tiered architecture

- User interface
- some data validation/formatting

- queueing/scheduling of user requests
  - transaction processor (TP)
  - Connection mgmt
  - Format conversion

- Database
- Legacy application processing
Beyond 3-tiered

Most architectures are multi-tiered
Decentralized Architecture

- Peer-to-peer systems
  - Chord
  - BitTorrent
  - .......

Structured Peer-to-Peer Architectures (1)

The mapping of data items onto nodes in Chord.

The diagram shows a circle with nodes labeled 0 to 15, each associated with a range of data keys. The nodes are connected in a ring, with arrows indicating the mapping of data keys to their corresponding nodes. For example, node 15 is associated with the data keys {13, 14, 15}, and node 0 is associated with {0, 1}. The diagram illustrates how data items are distributed across the nodes in the Chord system.
Typical Dist. Systems Design Goals

• Heterogeneity – can the system handle a large variety of types of PCs and devices?
• Robustness – is the system resilient to host crashes and failures, and to the network dropping messages?
• Availability – are data, services always there for clients?
• Transparency – can the system hide its internal workings from the users?
• Concurrency – can the server handle multiple clients simultaneously?
• Efficiency – is it fast enough?
• Scalability – can it handle 100 million nodes? (nodes=clients and/or servers)
• Security – can the system withstand hacker attacks?
• Openness – is the system extensible?