Chapter 2: System Model

- Introduction
- Architecture Models
- Fundamental Models
- Summary

What is a model?

- Each model is intended to provide an abstract, simplified but consistent description of a relevant aspect of distributed system design
Architecture model

- **Architecture model**
  - define the way in which they are mapped onto the underlying network of computers
  - determine the distribution of data and computational tasks amongst the physical nodes of the system
  - evaluating the performance, reliability, scalability and other properties of distributed systems.

- **Including**
  - Client-server model
  - Peer process model
  - Variations of the client-server model

Fundamental model

- **Are concerned with a more formal description of the properties that are common in all of the architectural models**

- **Including**
  - **The interaction model**: deal with performance and with the difficulty of setting time limits in a distributed system
  - **The failure model**: give a precise specification of the faults that can be exhibited by processes and communication channels
  - **The security model**: discuss the possible threats to processes and communication channels
Difficulties for and threats to DTS

- Widely varying modes of use
- Wide range of system environments
- Internal problems
- External threats

Chapter 3: System Model

- Introduction
- Architecture Models
- Fundamental Models
- Summary
Build architectural models

- **Simplifies and abstracts the functions of the individual components**
  - Achieved by classify processes as server, client and peer processes
- **Then considers:**
  - The placement of the components
  - The interrelationships between the components

Software and hardware *service layers* in distributed systems

![Diagram of software and hardware service layers]

- Applications, services
- Middleware
- Operating system
- Computer and network hardware
Platform

- Are the lowest-level hardware and software layers
- e.g.
  - Intel x86/Windows
  - Intel x86/Linux
  - Intel x86/Solaris
  - SPARC/SunOS
  - PowerPC/MacOS

Middleware

- Its purpose is to mask heterogeneity and provide a convenient programming model
  e.g. **OMG’s CORBA, Java RMI, DCOM**
- Support of abstractions
  - Remote method invocation: **Sun RPC**
  - Group communication: **Isis**
  - Notification of events: COBAR
  - The replication of shared data
  - Transmission of multimedia data
Limitation of middleware

• some systems require support at the application level.
  E.g. transfer of large electronic mail
• ‘the end-to-end argument’ [1984]
  – some communication-related functions can be completely and reliably implemented only with the knowledge and help of the application standing at the end points of the communication system
  E.g. TCP, DNS and the Web

System architectures

• The division of responsibilities between system components (applications, server and other processes) and the placement of the components on computers in the network
Arch. 1: Client/Server

- Be historically the most important and remain the most widely employed
- Servers may in turn be clients of other servers

Arch. 2: Services provided by multiple servers

- Partition service objects on different servers
  - e.g. Web, CDAL
- Maintain replicated service objects on several hosts
  - e.g. Sun NIS, realcourse
Arch. 3: Proxy servers and caches

- **Cache**
  - a store of recently used data objects that is closer than the objects themselves
  - E.g., web page cache at web browser or web proxy proxy server

Arch. 4: Peer to Peer

All processes play similar roles
Interacting cooperatively to perform a distributed activity
Maintain consistency or synchronize at application level
Example: a peer to peer whiteboard
Napster

I have X!

1.2.3.4

Publish

insert(X, 1.2.3.4)

...

Napster

Where is file A?

Query

Reply

search(A) --> 4.3.2.1

Fetch

4.3.2.1

Where is file A?
Gnutella

I have file A.
Reply

Query
Where is file A?

KaZaA

insert(X, 123.21.23)
Publish
I have X! 123.2.21.23
KaZaA

Where is file A?

search(A) --> 123.2.22.50

123.2.22.50

Query

Replies

search(A) --> 123.2.0.18

123.2.0.18

BitTorrent

Tracker

A

B

C

D
BitTorrent

Consistent hashing

- Each node's identifier is generated by a hash function
  - Node identifier hash = hash(IP address)
- Each resource is identified by a key value
  - Key identifier hash = hash(key)

A key is stored at its successor: node with next higher ID

In Chord hash function is Secure Hash SHA-1
Chord：保存键值的索引

Node 105

Key=Hash(“book1”) = 5
Key=Hash(“book2”) = 20
Key=Hash(“book3”) = 80

Circular ID space

Key 5
K5
K20

N105
N90
K80

Chord：查找

“Where is key 80?”

“N90 has K80”

N120
N10
N105
N90
K80
N32
N60
**Key Location**

- Each node preserves a finger table, each table contains $\log(N)$ fingers, pointing to $\log(N)$ neighbors.
- Information includes neighbor node identifiers, coverage intervals, stored keys, and successors.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{finger}[k].\text{start}$</td>
<td>$(n+2^{k-1}) \mod 2^m$, $1 \leq k \leq m$</td>
</tr>
<tr>
<td>$\text{interval}$</td>
<td>$[\text{finger}[k].\text{start}, \text{finger}[k+1].\text{start})$</td>
</tr>
<tr>
<td>$\text{node}$</td>
<td>first node $\leq n.\text{finger}[k].\text{start}$</td>
</tr>
<tr>
<td>successor</td>
<td>the next node on the identifier circle; $\text{finger}[1].\text{node}$</td>
</tr>
<tr>
<td>predecessor</td>
<td>the previous node on the identifier circle</td>
</tr>
</tbody>
</table>

$k$ is the finger table index

**Lookup(id)**

![Finger table for Node 1](image)

Finger table for Node 1

Finger tables and key locations with nodes 0,1,3 and keys 1,2 and 6
To find the successor of an id:
Chord returns the successor of the closest preceding finger to this id.

**Lookup PseudoCode**

```
// ask node n to find id's successor
n.find_successor(id)
    n' = find_predecessor(id);
    return n'.successor;
```

```
// ask node n to find id's predecessor
n.find_predecessor(id)
    n' = n;
    while (id ∉ (n', n'.successor))
        n' = n'.closest_preceding_finger(id);
    return n';
```

```
// return closest finger preceding id
n.closest_preceding_finger(id)
    for i = m downto 1
        if (finger[i].node ∈ (n, id))
            return finger[i].node;
    return n;
```

**Finding successor of identifier 1**

---

**Lookup cost**

- The finger pointers at repeatedly doubling distances around the circle cause each iteration of the loop in `find_predecessor` to halve the distance to the target identifier.

In an N node Network the number of messages is of

\[ O(\log N) \]
路由表

- **路由表内容**
  - **id**—文件标识符
  - **next_hop**—存储文件id的另一个节点
  - **file**—保存在本地的id标识文件

- **搜索过程**
  - 如果文件id存储在本地，停止搜索，上传文件
  - 如果不在本地，搜索路由表中最接近的id，将请求转到next_hop
  - 如果所有节点都没有找到，返回失败，返回路由表中下一个最接近的id

文件路由原理

- 网络趋向于一个小世界—small world, 类似六度分
  隔（Six Degrees of Separation）理论
- 因此，大部分查询只需经过少量跳数
Distributed Hash Table

- Distributed data structure systems can be in a ring, tree, hypercube, hop table, butterfly network ...
- CFS, OceanStore, PAST, ChordDNS

<table>
<thead>
<tr>
<th>put(key, data)</th>
<th>get(key)</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed Hash Table (DHash)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lookup(key)</th>
<th>node IP address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chord (Chord)</td>
<td></td>
</tr>
</tbody>
</table>

Variations on the client-server model

- Reasons of variation
  - The use of mobile code and mobile agents
  - Users need for low-cost computers with limited hardware resources
  - The requirement to add and remove mobile devices in a convenient manner
Arch. 1.1: Mobile Code

- For good interactive response, e.g. applet

a) client request results in the downloading of applet code

b) client interacts with the applet

Arch. 1.2: Mobile Agent

- A running program that travels from one computer to another in a network
- Carrying out a task on someone’s behalf, e.g. worm [Xerox PARC]
Arch. 1.3: Network Computer

- Download operating system and any application software from a remote file server
- All the application data and code is stored by a file server
- Users may migrate

Arch. 1.4: Thin Client

- A GUI on a computer that is local to the user
- Execute application programs on a remote computer
- Drawback: high latencies
- Implementation: X-11, VNC[AT&T 1998]
Arch. 1.5: Spontaneous network

- **Integrate mobile devices and other devices into a given network**
- **Key features**
  - Easy connection to a local network
  - Easy integration with local services
- **Key design issues**
  - Convenient connection and integration
  - Limited connectivity
    - mobile device move around continuously, disconnection
  - Security and privacy
  - Discovery Services
    - registration service, lookup service

Spontaneous networking in a hotel
Interfaces and objects

- Interface definitions
  -- A set of functions available or invocation
- In object-oriented languages
  -- Many objects can be encapsulated in processes
- **Distribution of responsibilities**
  -- a static client-server architecture or the more dynamic object-oriented model

Design requirements for distributed architectures

- Resource sharing is taken for granted, but effective data sharing on a large scale remains a substantial challenge.
- Performance issues
- Quality of services
- Use of caching and replication
- Dependability issues
**Performance issues**

- **Responsiveness.** E.g. the performance of web-browsing clients
  - the load and performance of the server and network
  - delay in the client and server operating system’s communication and middleware services as well as code of the service

- **Throughput**
  - the rate at which computational work is done
  - It is affected by processing speeds and by data transfer rates

- **Balancing computational loads**
  - E.g. applets, several computers for a service

**Quality of service**

- Reliability, security, performance and adaptability
- The failure model, the security model and the interaction model
- QoS is commandeered to refer to the ability to meet time-critical data (RSVP)
- Qos applies to OSs as well as networks.
Use of caching and replication

- Performance issues are barriers to successful deployment of distributed systems
- Replication and caching, with a variety of different cache consistency protocols
- E.g. Web-caching

Design requirements for distributed architectures

- Dependability issues
  - Correctness
  - Fault tolerance
    - redundancy, e.g. data and processes be replicated, messages be retransmitted
  - Security
    - e.g. locate sensitive data in computers that can be secured effectively against attack
Chapter 3: System Model

- Introduction
- Architecture Models
- Fundamental Models
- Summary

A system model should address

- What are the main entities in the system?
- How do they interact?
- What are the characteristics that affect their individual and collective behavior?
Purpose of a fundamental model

- Make explicit all the relevant assumptions about the system we are modeling
- Make generalizations concerning what is possible or impossible by logical analysis and mathematical proof

Fundamental models intend to discuss

- **Interaction model**
  - The processes interact by passing messages, resulting in communication and coordination.
  - delay are often of considerable duration
  - It’s difficult to maintain the same time notion in distributed system
  - Accuracy with which of the independent processes can be coordinated is limited by the two facts mentioned above
Fundamental models intend to discuss

- **Failure model**
  - a fault occurs in computers or network
  - Failure model can provide the a basis for the analysis of the potential effect
  - and for the design of system to be able to tolerate faults of each type while continuing to run correctly

Fundamental models intend to discuss

- **Security model**
  - Modular nature of distributed and their openness exposes them to attack by both external and internal agent.
  - Security model provide the basis for analysis of threats to system and for the design of system that are able to resist them
Interaction model

- Examples of interaction in distributed system
  - DNS, NIS
    - multiple server processes cooperate with one another
  - P2P voice conference system
    - with strict real-time constraints

Implementation of a interaction model

- Distributed algorithm
  - a definition of the steps to be taken by each of the processes of which the system is composed, including the transmission of messages between them
  - The proceeding rate and transmission timing can not be predicted.
  - difficult to describe all the states, because of failures of processes and message transmissions
Two factors affecting interacting processes

- Communication performance is always a limited characteristic
- It is impossible to maintain a single global notion of time

Performance of communication channels

- **Latency**
  - the time taken for the first string of bits transmitted through a network to reach its destination
  - accessing network
  - OS communication services
- **Bandwidth**
  - total amount of information that can be transmitted over computer network in a given time
- **Jitter**
  - variation in the time taken to deliver a series of messages
  - E.g. consecutive samples of audio data are played with differing time intervals
Computer clocks and timing events

- Clock drift rate
  - the relative amount that a computer clock differs from a perfect reference clock
- Timing event
  - e.g., GPS, Logical time

Two variants of the interaction model

- Synchronous distributed system
  - The time to execute each step of a process has known lower and upper bounds
  - Each message transmitted over a channel is received within a known bounded time
  - Each process has a local clock whose drift rate from real time has a known bound
Two variants of the interaction model

• Asynchronous distributed system – no bounds on
  – Process execution speed
    ▪ e.g. each step may take an arbitrarily long time
  – Message transmission delay
    ▪ e.g. a message may be received after an arbitrarily long time
  – Clock drift rate
    ▪ the drift rate of a clock is arbitrary

Examples of Syn. DS and Asyn. DS

• Asynchronous DS
  – Email
  – FTP
• Synchronous DS
  – VOD
  – Voice Conference System
Event ordering

- Example of disorder of messages
  - A group including X, Y, Z and A
  - X send “Meeting” to all; Y and Z reply “Re: Meeting” to all
  - At A, the messages received are Z.“Re: Meeting”, X.“Meeting”, Y. “Re: Meeting”

Failure model

- Define the ways in which failure may occur in order to provide an understanding of the effects of failures
- Taxonomy[Hadzilacos and Toueg, 1994]
  - Omission failures
  - Arbitrary failures
  - Time failures
1. Omission failures

- A process or communication channel fails to perform actions that it is supposed to do
- **Process omission failure: Crash**
  - *Fail-stop*: Crash that can be detected by other processes certainly, e.g., by timeouts in synchronous DS
- **Communication omission failures: dropping messages**
  - Send omission, receive omission, channel omission
- **Benign failures**
2. Arbitrary (Byzantine) failures

- The worst possible failure semantics
- Arbitrarily omit intended processing steps steps or take unintended processing steps.
  - E.g. return a wrong value in response to an an invocation
- Arbitrary failures in process is hard to be detected
- Arbitrary failures in communication channel exist but rare.
  - E.g. checksum, sequence number

<table>
<thead>
<tr>
<th>Class of failure</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail-stop Process</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may detect this state.</td>
</tr>
<tr>
<td>Crash Process</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may not be able to detect this state.</td>
</tr>
<tr>
<td>Omission Channel</td>
<td>Channel</td>
<td>A message inserted in an outgoing message buffer never arrives at the other end’s incoming message buffer.</td>
</tr>
<tr>
<td>Send-omission Process</td>
<td>Process</td>
<td>A process completes a send, but the message is not put in its outgoing message buffer.</td>
</tr>
<tr>
<td>Receive-omission Process</td>
<td>Process</td>
<td>A message is put in a process’s incoming message buffer, but that process does not receive it.</td>
</tr>
<tr>
<td>Arbitrary (Byzantine) Process or channel</td>
<td>Process or channel</td>
<td>Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.</td>
</tr>
</tbody>
</table>
3. Timing failures

- Applicable in syn. distributed system, but not in asyn. distributed system

<table>
<thead>
<tr>
<th>Class of Failure</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>Process</td>
<td>Process’s local clock exceeds the bounds on its rate of drift from real time.</td>
</tr>
<tr>
<td>Performance</td>
<td>Process</td>
<td>Process exceeds the bounds on the interval between two steps.</td>
</tr>
<tr>
<td>Performance</td>
<td>Channel</td>
<td>A message’s transmission takes longer than the stated bound.</td>
</tr>
</tbody>
</table>

Masking failures

- Hide
  - e.g. replicated servers

- Convert
  - e.g. Checksum: arbitrary failure -> omission failure

- Reliability of one-to-one communication
  - Validity
    - any message in the outgoing message buffer is eventually delivered to the incoming message buffer
  - Integrity
    - the message received is identical to one sent, and no messages are delivered twice, against retransmit protocols and spurious messages
Security model

- The security of a distributed system
  - The processes
  - The communication channels
  - The objects
- Protecting the objects
  - Access rights: who is allowed to perform the operations of an object
  - Principal: the authority who has some rights on the object
The enemies

- **Threats to processes**
  - To servers: invoke with a false identity, e.g. cheating a mail server
  - To clients: receive false result, e.g. stealing account password

- **Threats to communication channels**
  - Copy, alter or inject messages
  - Save and replay, e.g. retransfer money from one account to another

The enemies (2)

- **Denial of service**
  - excessive and pointless invocation on services or message transmissions in a network
  - result in overloading of physical resources (network bandwidth, server processing capacity)

- **Mobile code**
  - malicious mobile program, e.g. Trojan horse attachment
Defeat security threats

- Cryptography and shared secret
  - Identify each other by the shared secrets that are only known by themselves
  - Cryptography is the base
- Authentication
  - proving the identities supplied by their senders

Secure channels

- Each process knows reliably the identities of the principal on whose behalf the other process is executing
- Ensure the privacy and integrity of the data transmitted across it
- Each message includes physical or logical time stamp
Chapter 3: System Model

- Introduction
- Architecture Models
- Fundamental Models
- Summary

Architecture Models

- **Client / Server**
  - e.g. Web, FTP, NEWS
- **Multiple Servers**
  - e.g. DNS
- **Proxy and Cache**
  - e.g. Web Cache
- **Peer processes**
- **Variations of C/S**
  - Mobile code, mobile agent, network computer, thin client, spontaneous networks
Fundamental Models

- **Interaction models**
  - synchronous DS and asynchronous DS

- **Failure models**
  - omission failures
  - arbitrary failures
  - timing failures

- **Security model**
  - the enemies
  - the approaches of defeating them