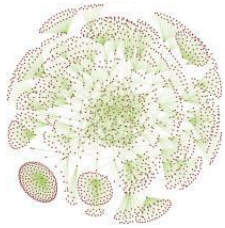
Distributed Computing

Review



Ben Abdallah Abderazk
E-mail: benab@u-aizu.ac.jp

Distributed System

- Collection of computers that are connected together and (sometimes) interact.
- Many independent problems at same time
  - Similar
  - Different
- Or …
  - One very big problem (or a small number)
- Computations that are physically separated
  - Client-server
  - Inherently dispersed computations

Types

- Types of Distributed Systems
  - Distributed Computing Systems
  - Distributed Information Systems
  - Distributed Pervasive Systems

DS Advantages & Disadvantage

- **Advantages**
  - Economics
  - Speed
  - Inherent distribution
  - Reliability
  - Incremental growth
- **Disadvantages**
  - Complex Software
  - Network
  - More components to fail
  - Security

Architectures

- **Software Architectures** – describe the organization and interaction of software components; focuses on logical organization of software
- **System Architectures** - describe the placement of software components on physical machines

Architecture

- Almost all new large systems are distributed systems
- Distributed systems support resource sharing, openness, concurrency, scalability, fault tolerance and transparency
- Client-server architectures involve services being delivered by servers to programs operating on clients
- User interface software always runs on the client and data management on the server

Architecture

- In a distributed object architecture, there is no distinction between clients and servers
- Distributed object systems require middleware to handle object communications
- The CORBA standards are a set of middleware standards that support distributed object architectures

Process Managements

- Key reasons: performance and flexibility
- Process migration (aka *strong mobility*)
  - Improved system-wide performance – better utilization of system-wide resources
  - Examples: Condor, DQS
- Code migration (aka *weak mobility)*
  - Shipment of server code to client – filling forms (reduce communication, no need to pre-link stubs with client)
  - Ship parts of client application to server instead of data from server to client (e.g., databases)
  - Improve parallelism – agent-based web searches

Process State

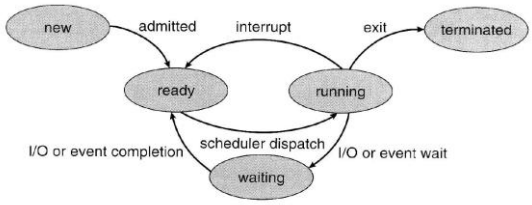


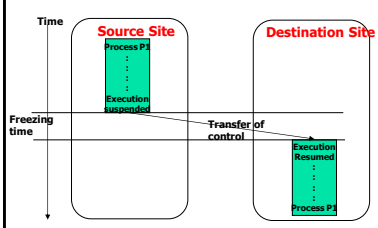
Diagram of process state.

Application Scenarios

Depending on how the request messages are interpreted, there are 4 main application scenarios:

1. **Remote Service**
   - **The message is interpreted as a request for a known service at the remote site.**
2. **Remote Execution**
   - **The messages contain a program to be executed at the remote site.**
3. **Process Migration**
   - **The messages represent a process being migrated to a remote site for continuing the execution.**
4. **Code Migration**
   - **Ship parts of client application to server instead of data from server to client (e.g., databases)**

Process Migration



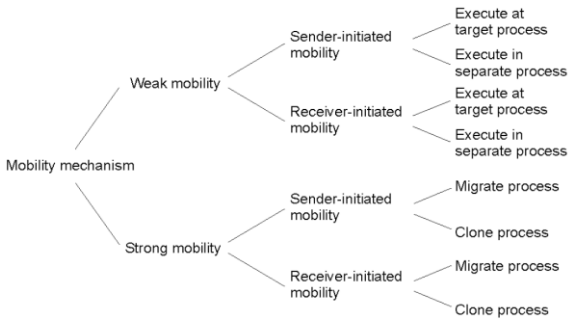
- Selecting a process to be migrated
- Selecting the destination node
- Suspending the process
- Capturing the process state
- Sending the state to the destination
- Resuming the process
- Forwarding future messages to the destination

Process Migration

Benefits

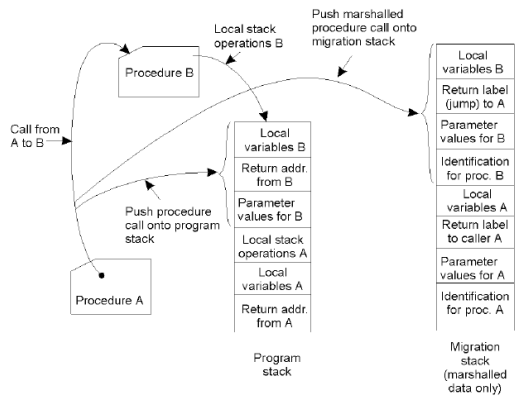
- **Better response time and execution speed-up**
  - Dynamic load balancing among multiple nodes
  - Using a faster CPU
- **Higher throughput and Effective resource utilization**
  - Migrating I/O and CPU-bound processes to file and cycle servers.
- **Reducing network traffic**
  - Migrating processes closer to the resources they are using most heavily.
- **Improving system reliability**
  - Migrating processes from a site in failure to more reliable sites
  - Replicating and migrating critical processes to a remote.

Models for Code Migration



Migration in Heterogeneous Systems

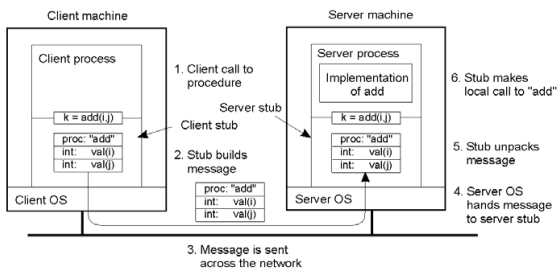
- Systems can be heterogeneous (different architecture, OS)
  - Support only weak mobility: recompile code, no run time information
  - Strong mobility: recompile code segment, transfer execution segment [migration stack]
  - Virtual machines - interpret source (scripts) or intermediate code [Java]



Communications

- Remote Procedure Call
- Message-oriented Communication
- Stream-oriented Communication
- Multicast Communication

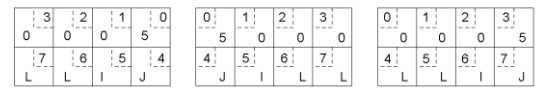
Steps of a Remote Procedure Call (Parameter Passing – identical C/S)



The steps involved in doing a remote computation through RPC in identical c/s machines

Steps of a Remote Procedure Call (Parameter Passing – different C/S)

Example: Consider a procedure with 2 parameters, and integer and a four-character string. Each parameters requires one 32-bit word.



(a) Pentium (original message) **(b) SPARC (after receipt)** **(c) The message after being inverted.**

Parameter Portion of a message built by a client stubs
The little numbers in boxes indicate the address of each byte.

Parameter Specification and Stub Generation

```
foobar( char x; float y; int z[5] )
{
  ....
}
```

a) Procedure

foobar's local variables
x
y
5
z[0]
z[1]
z[2]
z[3]
z[4]

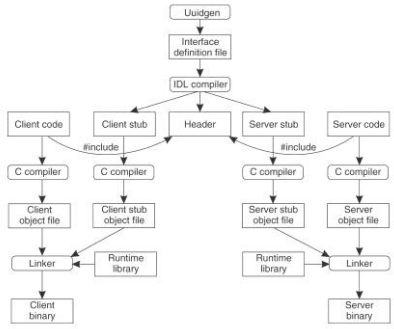
b) The corresponding message.

• **To simplify matters, interfaces are often specified by means of Interface Definition Language (IDL). This interface is compiled into a client stub and server stub.**

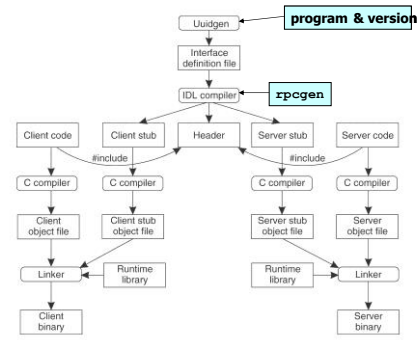
Practical RPC Systems (continued)

- Java RMI (Remote Method Invocation)
  - `java.rmi` standard package
  - Java-oriented approach — objects and methods
- CORBA (Common Object Request Broker Architecture)
  - Standard, multi-language, multi-platform middleware
  - Object-oriented
  - Heavyweight

Implementation Model for DCE



Differences for ONC



Synchronization

- Temporal ordering of events produced by concurrent processes
- Synchronization between senders and receivers of messages
- Coordination of joint activity
- Serialization of concurrent access for shared objects

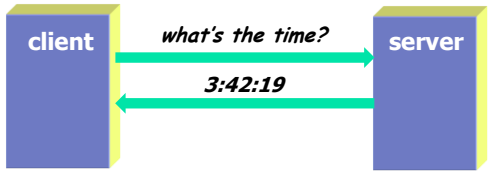
Synchronization Methods

- Two well-known methods:
  - Cristian's algorithm (UTC time-server based)
  - Berkeley algorithm (no UTC signal, but master)

RPC

Simplest synchronization technique

- Issue RPC to obtain time
- Set time



Does not account for network or processing latency

26

Cristian's algorithm

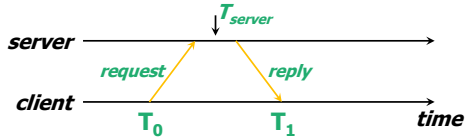
- Client polls time server (which has external UTC source)
- Gets time from server
- Adjusts received time by adding estimate of one-way delay
  - Estimates travel time as 1/2 of RTT
  - Adds this to server time
- Errors introduced not by delays, but by asymmetry in delays (path to server and path from server)

27

Cristian's algorithm

Compensate for delays

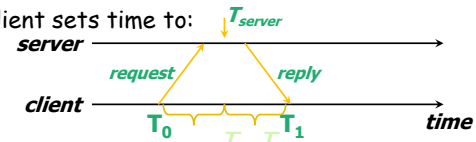
- Note times:
  - request sent: T_0
  - reply received: T_1
- Assume network delays are symmetric



28

Cristian's algorithm

Client sets time to: T_{server}



$\frac{T_1 - T_0}{2}$ = estimated overhead in each direction

$$T_{new} = T_{server} + \frac{T_1 - T_0}{2}$$

29

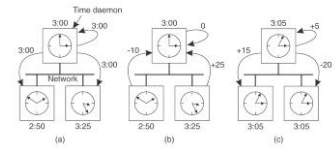
Berkeley Algorithm

- Time master polls clients (master has no UTC)
- Gets time from each client, and averages
- Sends back message to each client with a recommended adjustment
- Clocks are synchronized, but not UTC
- Errors arise when nodes have different delays from master

30

Berkeley Algorithm

- Machines run **time dæmon**
  - Process that implements protocol
- One machine is elected (or designated) as the server (**master**)
  - Others are **slaves**



31

Berkeley Algorithm

- Master polls each machine periodically
  - Ask each machine for time
    - Can use Cristian's algorithm to compensate for network latency
- When results are in, compute average
  - Including master's time
- *Hope: average cancels out individual clock's tendencies to run fast or slow*
- Send offset by which each clock needs adjustment to each slave
  - Avoids problems with network delays if we send a time stamp

32

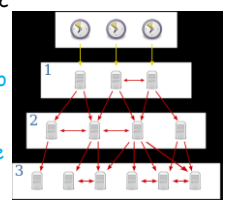
What is NTP?

- Network Time Protocol
  - Used to keep clock's syncronized within your nodes
  - Important for logservers and logfiles
  - Transaction servers
  - Various applications
  - Time critic operations that needed to be syncronized
- Stratum levels
  - Level 1 most accurate, many public exist
  - Level 2 next accurate, often companies timeservers
  - Lower levels
- LW radio carriers as reference
- Important atomic clock servers that supports NTP http://www.eecis.udel.edu/~mills/ntp/servers.html

33

NTP oragnization

- **Stratum 0**: devices such as atomic (caesium, rubidium) clocks, GPS clocks or other radio clocks.
- **stratum 1**: computers attached to Stratum 0 devices. They act as servers for timing requests from Stratum 2 servers via NTP. These computers are also referred to as time servers.



- **Stratum 2**: computers that send NTP requests to Stratum 1 servers
- **Stratum 3**: computers employ exactly the same NTP functions of peering and data sampling as Stratum 2, and can themselves act as servers for lower strata

34

Replication

- Why Replication
- Data-centric consistency models
- Client-centric consistency models
- Replica management
- Consistency protocols

35

Replication

- The art and science of keeping multiple copies of data in a distributed system
- Two reasons:
  - Reliability
  - Performance

36

Example

- DNS (Domain Name Service) allows owner of a domain to replicate name database
- Same two reasons
  - Performance
  - Reliability
- Also
  - Scaling technique

37

Replication

- Data centric
  - I.e., try to keep data consistent across replicas
- Client centric
  - Weaker condition
  - Only maintain consistency for each client separately

Replication

- **Enhancing Services by replicating data**
  - Load Balancing
    - Example: Workload is shared between the servers by binding all the server IP addresses to the service's DNS name. A DNS lookup of the site results in one of the servers' IP addresses being returned, in a round-robin fashion.
  - Fault Tolerance
    - Under the fail-stop model, if up to *f* of *f+1* servers crash, at least one remains to supply the service.
  - Increased Availability
    - service may not be available when servers fail or when the network is partitioned.

P: probability that one server fails= 1 – P= availability of service. e.g. P = 5% => service is available 95% of the time.

P^n: probability that n servers fail= 1 – P^n= availability of service. e.g. P = 5%, n = 3 => service available 99.875% of the time

What are the issues?

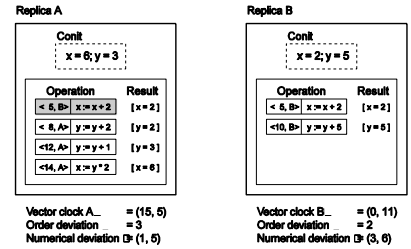
Updating replicas
- how to deal with updated data?
- How are updates distributed?

Replica management
- How many replicas?
- Where to place them?
- When to get rid of them?

Redirection/Routing
- Which replica should clients use?

Scalability ←--CONFILCT---→ management overheads

Example: Conit



- **Conit**: contains the variables x and y:
  - Each replica maintains a **vector clock**
  - B sends A operation [<5,B>: x := x + 2]; A has made this operation **permanent** (cannot be rolled back)
  - A has three **pending** operations => order deviation = 3
  - A has missed **one** operation from B, yielding a max diff of 5 units => (1,5)