CS G623: Advanced Operating Systems

Lecture 3

BITS Pilani
Pilani Campus

Amit Dua

August 9, 2018

Topics already discussed

- Advanced OS
- Types of AOS
- Motivation for Distributed systems
- Components of distributed system
- Goals
  - Access remote resources
    - Resource
    - Why collaboration
    - Price for collaboration
  - Transparency
    - Acc, loca, reloca, migra, repli, concurr, security
    - limitations
  - Open
    - Interface specification complete, neutral
    - Interoperability, portability, extensible
  - Scalable
    - Size, geography, administration
    - Central server, LAN, different organizations
    - Hiding latency, distribution, replication
    - Asynchronous, client side, DNS, Internet, caching, cost of concurrent updation
- Types of DS
  - Cluster and grid computing systems

Topics to be discussed

- Issues with DS
- Message passing vs Remote procedure calls (RPC)

Issues with DS

- Global knowledge
- Naming
- Scaling
- Compatibility
- Process synchronization
- Resource management
- Security
- Structuring of operating system

Global knowledge

- Shared memory
- Up-to-date state of processes and resources
- Global state is known
- Any problematic issue has solutions

- Distributed systems
- Unavailability of global memory
- Global clock
- Unpredictable message delays
- Global state is not known

- What is the state of processes??

- Messages sent and received

- Computer does not know the current and complete status of the global state.

- What is the global state?
- Why do we need global state?

- For garbage collection and
- Distributed deadlock detection

- Global clock is absent
- To order all the events that occur at different times at different computers

- Why we need global clock?

For scheduling

Who should the access to resources and in what order.

Discuss two logical clock schemes

Obtain global state in distributed systems

To arrive at consensus in the absence of global knowledge

Token for mutual exclusion

Token lost

In-transit

Naming

- Naming objects
- Maping logical names to physical addresses

- Lookup table or algorithms

- To avoid single point of failure
- Use replication

- Problems with replication??

- Requires more storage
- Updation synchronization

- Partitioning to minimize loss
- Difficulty in finding name and address of interest

- Structure of names for mapping
- Accessing objects

Scalability

- Scalability should not result in unavailability or graded performance

- E.g. broadcast protocols

- Implementing scarce resource increases cost with scaling

- Storage, communication bandwidth, manpower
- E.g. Storing system's directories in every computer

Compatibility

- Inter-operability among resources in a system

- Binary level: all processes execute the same instruction

Different architecture

Different vendors

- Execution level: same source code can be compiled and executed elsewhere.
- E.g. Andrew

- Protocol level: same protocols
- E.g. SUN NFS naming and authentication

Process synchronization

- Unavailability of shared memory
- Need only single process to access resource at a time
- Mutual exclusion
- Deadlock detection and resolution

Resource management

- Making local, remote resources available
- Effectively and easily
- Hiding location information
- **Data migration** : data brought to location of computation
- Files accessed regardless of local or remote resource
- Distributed file system provide Network transparency
- Access data from another systems memory
- Distributed shared memory should maintain consistency
- Minimize delays

- **Computation Migration:** migrating computation
- Information concerning a remote file directory and receive information
- Better to find info locally and transfer content vs
- Transfer the entire directory

- Only part of computations carried out at different machine
- RPC

- **Distributed scheduling:** transferring process to another computer
- one system is overloaded that originated process
- Maximize overall performance
- Enhanced utilization of computers

Security

- Authentication:
- Authorization:
- Difference???

structuring

- Monolithic kernel

- Collective structure: collection of processes
- Bare minimum at nucleus and
- Memory mgt, file system, scheduling, name services, RPC facilities, time mgt
- Done as independent processes

- Separate policy from mechanism

- Object oriented OS: services implemented as collection of objects

- Authentication : guaranteeing the entity is what it claims
- Authorization: allowing the privileges necessary

Client server model

- Single server responding to clients
- Bottleneck

- Multiple servers can be one option

What is a distributed system

Nearly all systems today are distributed in some way

– they use email

– they access files over a network

– they access printers over a network

– they share other physical or logical resources

– they cooperate with other people on other machines – they access the web

– they receive video, audio, etc

Loosely coupled systems

- Earliest systems used simple explicit network programs
- – FTP (rcp): file transfer program
- – telnet (rlogin/rsh): remote login program
- – mail (SMTP)
- Each system was a completely autonomous independent system, connected to others on the network

Tightly-coupled systems

A "tightly-coupled" system usually refers to a multiprocessor

- runs a single copy of the OS with a single workload queue
- has a single address space
- usually has a single bus or backplane to which all processors and memories are connected
- has very low communication latency
- processors communicate through shared memory

Distributed systems communication primitives

- Message passing
- RPC

Message passing

Primitives are high level constructs used by programs to access underlying communication network.

Affects:

1. Ease of use of system
2. Performance

Message passing:

1. SEND primitive (destination and message)and
2. RECEIVE primitive (source and buffer)

Output of the code

P0

```
String copy (a, hello)
Send (a,1)
String copy(a, bye)
```

P1

```
receive(b,0)
print(received)
```

```
P0
Strcpy(a, hello)                    receive(b,0)
Send(a,1)                           print("%s\n",b)
Strcpy(a, bye)
Send(a,1)
```

Buffering

Buffered: buffer provided by system

Messages are copied 3 times:

1. From user buffer of sender to kernel buffer
2. From kernel buffer of sender to kernel buffer of receiving computer
3. From kernel buffer of receiver to user buffer of receiver

DMA: acts as a bridge between memory and the I/O or network device

Without involving the processor for transferring data between memory and I/O ---either way.

Non-blocking:- sender executes and does not care if the receiver has received or not.

SEND: Control returns to the user process after 1st step.

RECEIVE: Signals intention to receive and provides a buffer

-either check continuously

-signaled by the kernel upon arrival of message

Blocking: sender wait for the receiver to receive the whole date and receiver sends the ACK to the sender

- Predictability of behavior
- Programming flexibility

Synchronous: notion of timing

-2 processes come at same time and communication happens.

Asynchronous: no assumption.

Output of the code

```
P0                                  P1
String copy (a, hello)              receive(b,0)
Send (a,1)                          print(received)
String copy(a, bye)
```

```
P0
Strcpy(a, hello)
Send(a,1)
Strcpy(a, bye)
Send(a,1)
```

```
receive(b,0)
print("%s\n",b)
```

Remote procedure calls

- Pairing of request and response
- Data representation
- Knowing the address of remote machine
- Failures (system or communication)

- Procedure call: transfer the control and data within a program running on a single machine.
- RPC: Client needing a service invokes a procedure at the server.

Client program makes a remote procedure call

C-process suspended.

Transfer the message parameters to S-machine

Computations done at procedure on S-machine

Results communicated to the C-machine

C-machine resumes execution as if it had done a local procedure.

Client and server stub

1. Client makes a local call on a dummy procedure,
2. Constructs a client stub procedure
3. Client stub constructs a message containing identity of remote procedure and parameters to be passed.
4. Server Stub at receiver receives the message
5. Makes a local call to the procedure specified
6. Control returns to server-stub after completion of the procedure
7. Server-stub passes the results and control back to the client stub
8. Client stub returns the results to the client.

Binding

- Binding is a process that determines
  1. the remote procedure and
  2. the machine
- on which it is to be executed upon a remote procedure invocation.

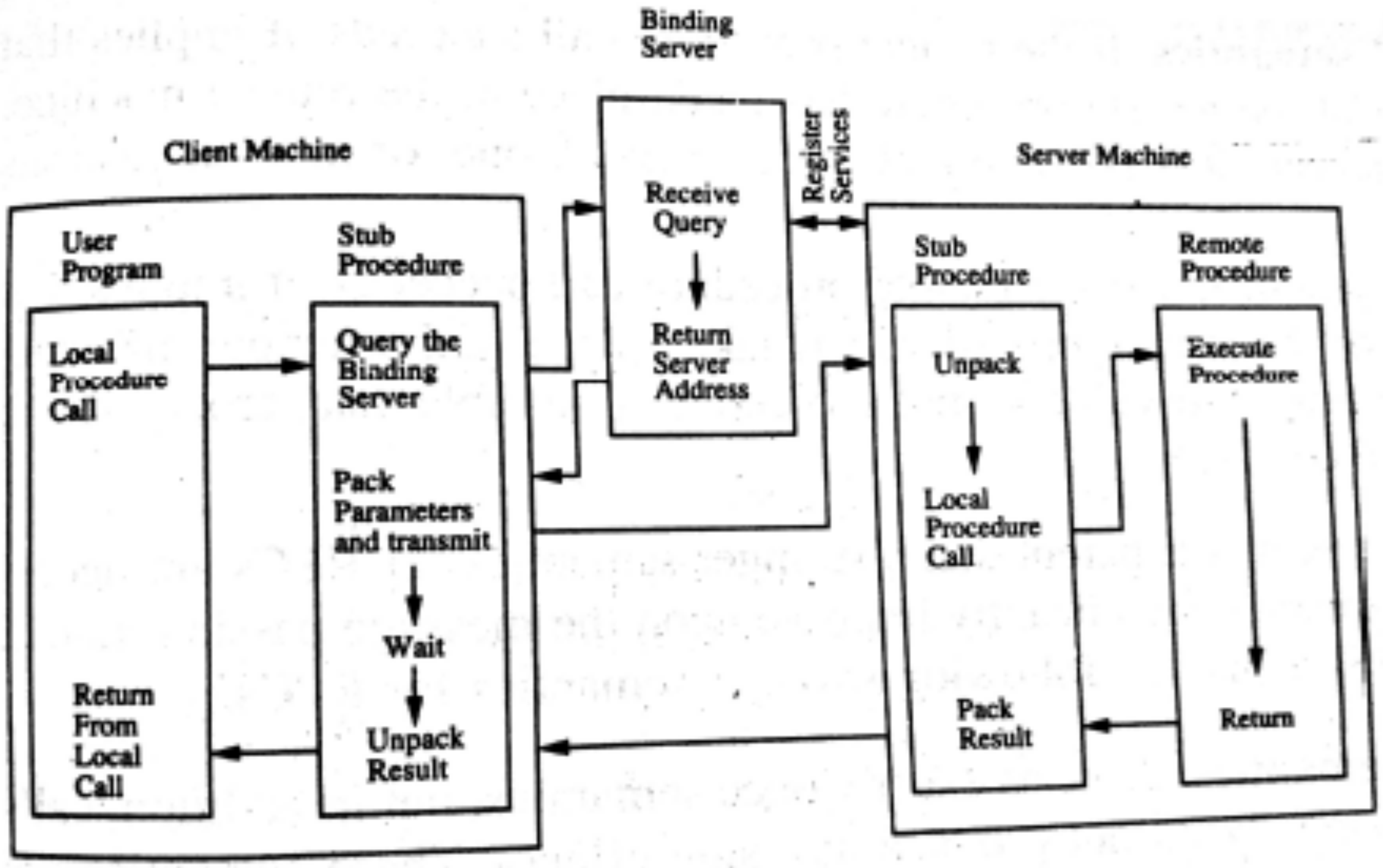
- Client stub procedure obtains the address of the server machine by querying the binding server or
- Client specifies the machine and the services required, and binding server returns the port number for communication with requested.
  - (no location transperancy)

Parameter and result passing

- Convert the parameters and results in appropriate representation, buffer and transmit
- Send the code identifying the format to do conversion of parameters and results.

- Know all representations
- Overhead if both C and S are using same data representation.

RPC


Binding Server
Receive Query
Return Server Address
Register Services
Client Machine
User Program
Local Procedure Call
Return From Local Call
Stub Procedure
Query the Binding Server
Pack Parameters and transmit
Wait
Unpack Result
Server Machine
Stub Procedure
Unpack
Local Procedure Call
Pack Result
Remote Procedure
Execute Procedure
Return

Error handling

- Remote server is slow: No ack in time, 2 copies
- Client crashes and recovers immediately
- At least once: at failure- 0, partial, 1 or more
- Exactly once: at failure, 0,partial, or 1
- At most once: calls do not produce side effects (0 or 1)

Correctness

Call Ci made by machine

Wi corresponding computation at remote machine

C2 happens before C1 (C1→C2) W1 and W2 modify the shared data

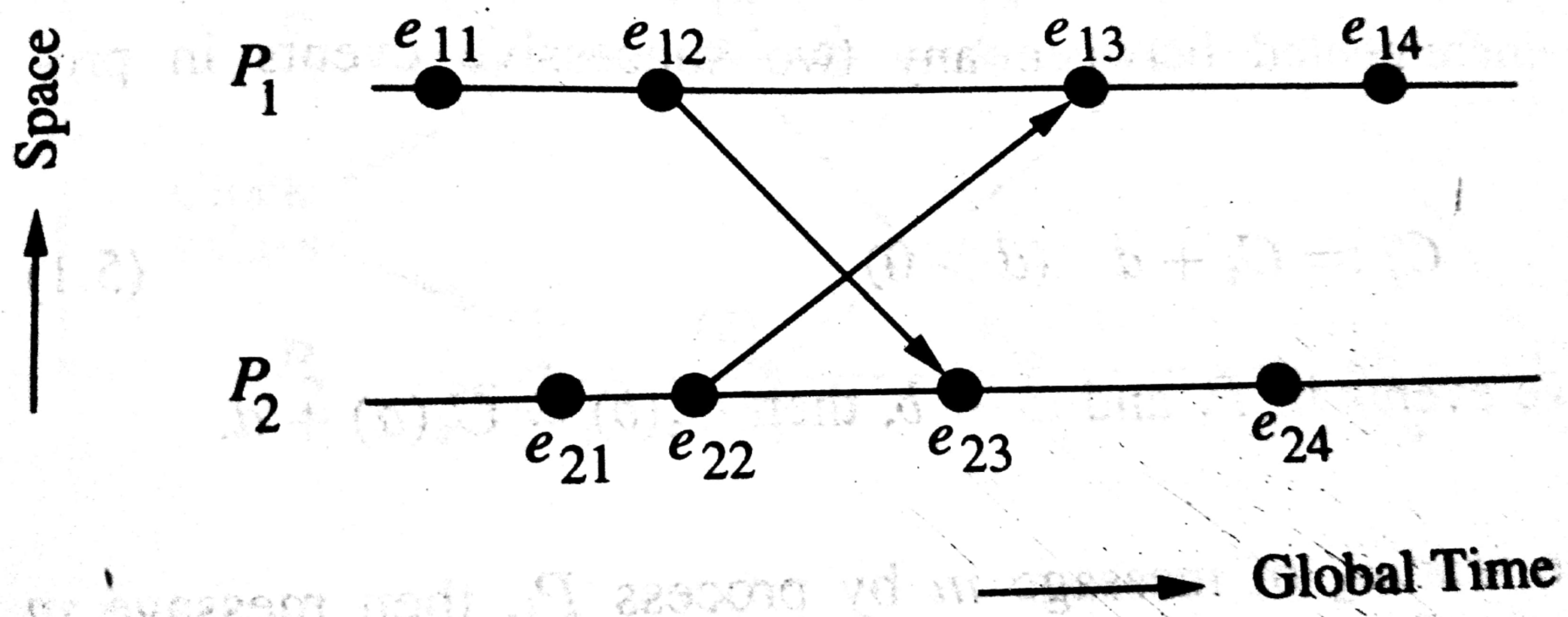
C1→C2 implies W1→W2

Other issues

- Low latency and
- high throughput

Any questions

Logical time



Space
P_1
P_2
e_{11}
e_{12}
e_{13}
e_{14}
e_{21}
e_{22}
e_{23}
e_{24}
Global Time