

CSSE4004-Lecture 2

Architectures of Distributed Systems (cont)

Processes

1

Learning Objectives

After this week you should

- [Architecture] Understand how behaviour of middleware can be modified
- [Architecture] Understand a basic model for self-managing distributed systems
- Understand the role of processes in distributed systems
 - processes, threads in clients or servers
 - processes across multiple systems
- Be able to describe virtualisation models for distributed systems
- Be able to describe reasons and models for code migration

2

Outline

- Architecture (cont)
 - Adaptability and self-management in DS
- Processes
 - Definitions
 - The use of threads in DS
 - Clients/Servers
 - Virtualisation in DS
 - Code migration

3

Adaptability and self-management in DS

- Role of middleware is to provide some degree of distribution transparency
 - Hiding distribution of data, processing and control
- Middleware may have a particular architectural style, e.g.
 - Object-based (CORBA)
 - Event-based (most of middleware built for adaptive, context-aware applications)
- Middleware should be adaptive to meet requirements of various applications

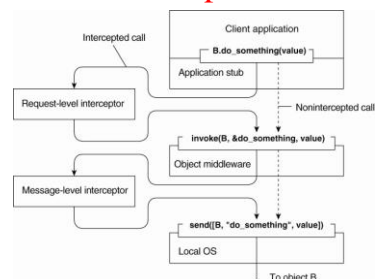
4

Using interceptors to adapt control

- Interceptors change flow of control and allow additional code to be executed
- Many object-based DS use interceptors to change flow of control in the object invocation
 - Requests (object invocations) can be intercepted
 - Messages can be intercepted

5

Interceptors



Ta-St. 2-15. Using interceptors to handle remote-object invocations

6

General Approaches to Adaptive Software

- Separation of concerns
 - E.g., *aspect oriented programming* (not very successful)
- Computational reflection
 - E.g., *reflective middleware*
- Component-based design
 - Adaptation through composition
 - Statically at design time
 - Dynamically at run time
 - Requires support for late binding

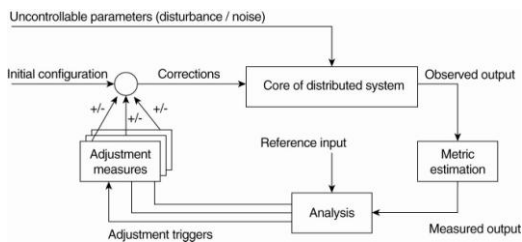
7

Self-management / autonomic computing

- Many distributed systems are complex and require self-management and behaviour adaptation
- Autonomic computing (or *self-**)
 - Self-configuration
 - Self-healing
 - Self-management
 - Context-awareness, etc.
- Adaptation takes places by one or more feedback control loops

8

The Feedback Control Model



Ta-St. 2-16. The logical organization of a feedback control system

9

Outline

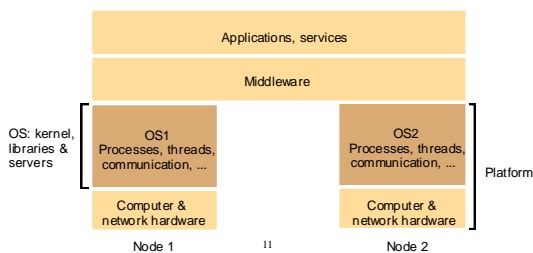
- Architecture (cont)
 - Adaptability and self-management in DS
- Processes
 - Definitions
 - The use of threads in DS
 - Virtualisation in DS
 - Clients/Servers
 - Code migration



10

Rationale

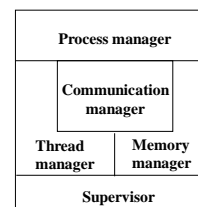
- Build virtual processors in software, on top of physical processors



11

Core OS functionality

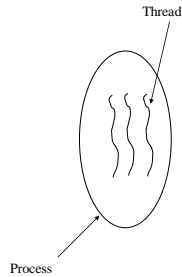
- Process Manager
 - Process lifecycle
- Thread Manager
 - Thread lifecycle
- Communication Manager
 - Communication between threads attached to different process
- Memory Manager
 - Physical/virtual
- Supervisor
 - Dispatching interrupts, traps
 - ...



12

Definitions

- Process:
 - Program being executed
 - Consists of address space and execution state (program counter, stack pointer, processor status word, contents of registers and system-call state)
- Thread (*of execution*):
 - Abstraction of activity within process



13

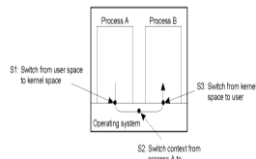
Effect of definitions

- Process:
 - A program, or section of a program, running in its own address space
- Thread
 - A process with less overhead to start, running in the same address space as one or more other threads
- for both, an interruption requires saving *state* – but process more overhead to set up, switch context, etc.

14

Context switching Costs

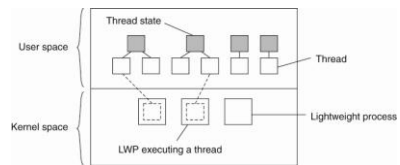
- Threads share address space: OS-independent context switching possible
- Process switch more expensive: involves OS, i.e., trapping to the kernel (switching VM space etc.)
- Kernel-based threads still lighter-weight



15

Thread implementation

- Implementation of threads by Lightweight Processes (LWPs) has advantages compared to user-level and kernel level threads



16

Outline

- Architecture (cont)
 - Adaptability and self-management in DS
- Processes
 - Definitions
 - The use of threads in DS
 - Virtualisation in DS
 - Clients/Servers
 - Code migration



17

Threads in DS ? (client)

- *Multithreaded clients hide network latency*
- Web browser scans incoming HTML page, finds more files to be fetched
 - each file fetched by separate thread, each doing a (blocking) HTTP request
 - as files arrive, browser displays them
- Exploit replicated web services
 - one thread per incoming stream

18

Threads in DS ? (server)

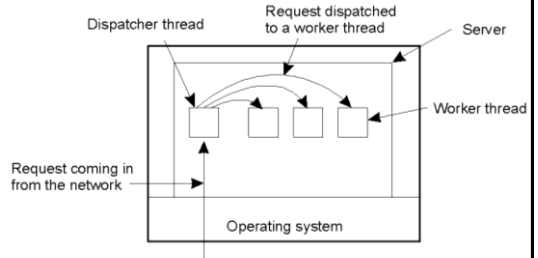
Improve performance, better structure

- Starting thread to handle incoming request much cheaper than starting new process
 - single-threaded server prevents simple scaling to multiprocessor
 - as with client: hide network latency by reacting to next request while previous being transmitted
- Better structure than nonblocking I/O
 - single thread, blocking I/O doesn't scale up
 - simple blocking calls simpler overall structure

19

Multithreaded Servers...

- Multithreaded server organized as dispatcher/worker



...Multithreaded Servers

3 alternatives:

Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls

21

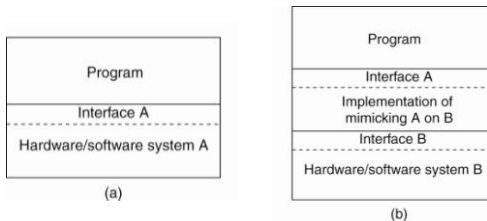
Outline

- Architecture (cont)
 - Adaptability and Self-management in DS
- Processes
 - Definitions
 - The use of threads in DS
 - Virtualisation in DS
 - Clients/Servers
 - Code migration



22

The Role of Virtualization in Distributed Systems



Ta-St. 3-5. (a) General organization between a program, interface, and system. (b) General organization of virtualising system A on top of system B.

23

Architectures of Virtual Machines (1)

Interfaces at different levels:

1. An interface between the hardware and software consisting of machine instructions
 - that can be invoked by any program
2. An interface between the hardware and software, consisting of machine instructions
 - that can be invoked only by privileged programs, such as an operating system

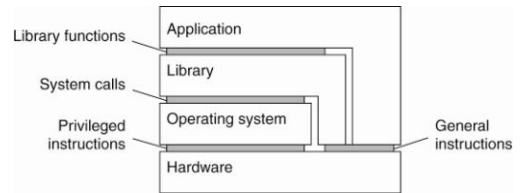
24

Architectures of Virtual Machines (2)

3. An interface consisting of system calls as offered by an operating system
4. An interface consisting of library calls
 - generally forming what is known as an application programming interface (API)
 - In many cases, system calls are hidden by an API

25

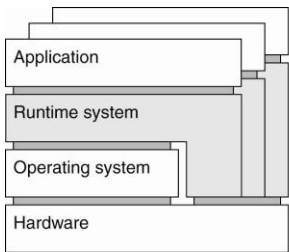
Architectures of Virtual Machines (3)



Ta-St. 3-6. Various interfaces offered by computer systems

26

Architectures of Virtual Machines (4)

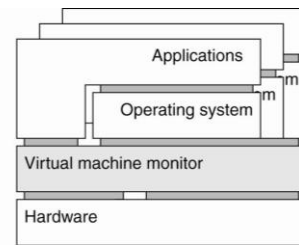


(a)

Ta-St. 3-7. (a) A process virtual machine, with multiple instances of (application, runtime) combinations

27

Architectures of Virtual Machines (5)



(b)

Ta-St. 3-7. (b) A virtual machine monitor, with multiple instances of (applications, operating system) combinations

28

Outline

- Architecture (cont)
 - Adaptability and Self-management in DS
- Processes
 - Definitions
 - The use of threads in DS
 - Virtualisation in DS
 - Clients/Servers
 - Code migration



29

Clients

- Clients are often *thin*
 - Large portion of processing required to support sophisticated user interfaces can be provided by servers
- Clients should support transparencies
 - communication hiding,
 - access transparency (client stub)
 - location, migration, relocation transparencies

30

Servers: General Design Issues...

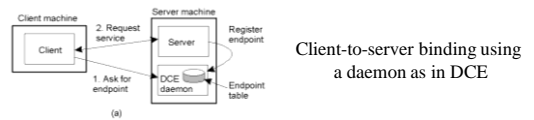
- A server
 - process waits for incoming service requests
 - commonly at specific end-point address (1-to-1 mapping between port and service)
- interrupt server once request accepted
 - out of band data

ftp-data	20	File Transfer [Default Data]
ftp	21	File Transfer [Control]
telnet	23	Telnet
smtp	25	Simple mail transfer
login	49	Login Host Protocol

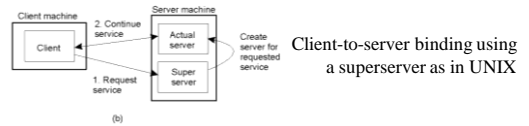
31

...Servers: General Design Issues...

Dynamic binding – no static end-point assigned



Client-to-server binding using a daemon as in DCE



Client-to-server binding using a superserver as in UNIX

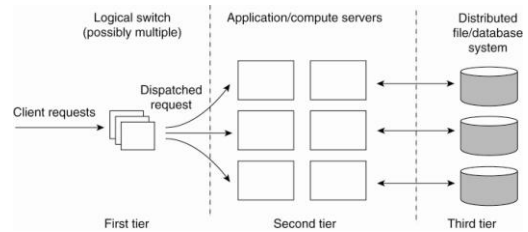
32

...Servers: General Design Issues

- Stateful server** – maintains information on clients
 - e.g., client can cache file locally
 - hard to handle failures
- Stateless server** – knows nothing about clients
 - can change own state without informing clients
 - e.g., web server serves request independently of previous
 - cookie allows client to send state to server

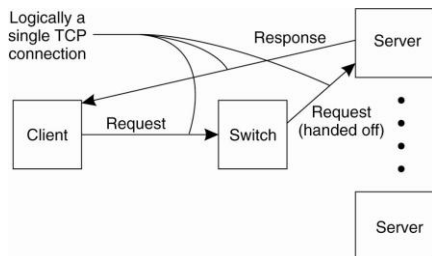
33

Server clusters (1)



34

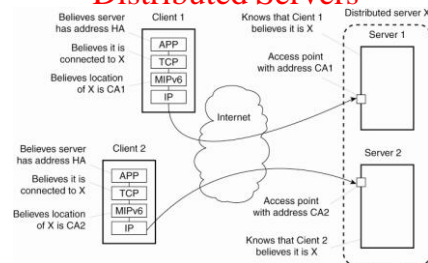
Server Clusters (2)



Ta-St. 3-13. The principle of TCP handoff

35

Distributed Servers



Ta-St. 3-14. Route optimization in a distributed server using MobileIPv6 (HA – home address, CA – care-of address)

36

Outline

- Architecture (cont)
 - Adaptability and Self-management in DS
- Processes
 - Definitions
 - The use of threads in DS
 - Virtualisation in DS
 - Clients/Servers
 - Code migration



37

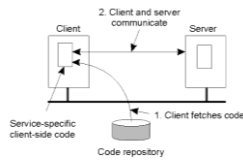
Rationale and Issues

- Pass (executable) code instead of query or result
 - e.g., send search algorithm to search engine
- How to pass local resources?
- How to migrate code in heterogenous systems?

38

Reasons for Migrating Code

- Performance
 - reduce communication
 - load distribution
 - reduce total time
 - parallelism
- Flexibility
- Continue when disconnected
- Dynamically deploy components



39

Problems

- security
 - do you trust someone else's code?
- overhead
 - does the time saved outweigh the time to send resources (code, data, etc.)?
 - is overall effect less or more network load?

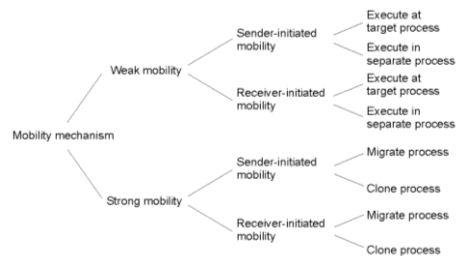
40

Type of migration

- Process consists of Code, Resource and Execution segments
- Weak mobility
 - Move only code and data segment (execute from beginning)
 - simple, especially if code portable, e.g., Java applet
 - Distinguish code shipping (push), code fetching (pull)
- Strong mobility:
 - move component, including execution state
 - migration: move entire object from one machine to other
 - cloning: start clone in same execution state (UNIX `fork`)

41

Models for Code Migration



42

Migration of local resources

- Resources segment has to be changed
 - Reference to opened TCP port
 - URL ?
- Process-resources binding reference type:
 - Binding by identifier: object requires specific instance of resource (e.g., specific database, URL)
 - Binding by value: object requires value of resource (e.g., set of cache entries, libraries – location may vary)
 - Binding by type: object requires only that type of resource available (e.g., colour monitor, printer)

43

Resources-to-machine binding type

- Unattached resources:
 - can easily move with object (e.g., files)
- Fastened resources:
 - can migrate in principle, cost high (e.g., database)
- Fixed resources:
 - cannot be migrated, such as local hardware

44

Actions to be taken when migrating

Resource-to machine binding

	Unattached	Fastened	Fixed
By identifier	MV (or GR)	GR (or MV)	GR
By value	CP (or MV, GR)	GR (or CP)	GR
By type	RB (or GR, CP)	RB (or GR, CP)	RB (or GR)

GR: establish global system-wide reference (e.g., URL)
 MV: move resource
 CP: copy value of resource
 RB: rebind to local resource

45

Migration in Heterogeneous Systems

- Main problem
 - target machine may not be able to execute original code
 - definition of process/thread/processor context highly dependent on hardware, OS runtime system
- Most general
 - use abstract machine implemented on different platforms
- Current solutions
 - interpreted languages running on a virtual machine (Java/JVM, scripting languages)
- New developments
 - Migrate entire computing environment

46

Conclusion

- Autonomic computing became trendy due to complexity of distributed systems
- Processes form basis for communication in DS
- Threads are useful in clients and servers
- Return of resource virtualisation approaches
- Client and servers need to support distribution transparency
- Distributed systems sometimes need to migrate code
- Reading: Chapter 2 (2.3, 2.4)

Chapter 3

47