CIS 505: Software Systems
Architectures of Distributed Systems

Insup Lee
Department of Computer and Information Science
University of Pennsylvania

CIS 505, Spring 2007

Software Concepts

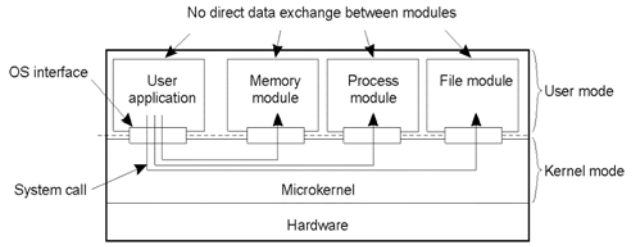
System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

- Software more important for users
  - Connecting users and resources
  - Developing software for distributed systems

System software structures

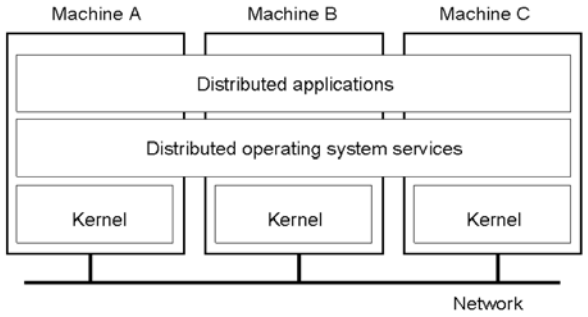
- DOS (Distributed Operating Systems)
  - Uniprocessor OS, Mutliprocessor OS, Multicomputer OS
  - Global IPC, file system interface
  - Distributed Shared Memory
- NOS (Network Operating Systems)
  - Loosely coupled-software on loosely-couple hardware
  - rlogin, rcp, file servers for shared file, etc.
- Middleware

Uniprocessor Operating Systems



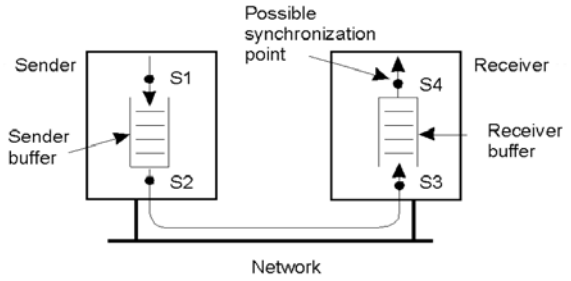
- Separating applications from operating system code through a microkernel.

Multicomputer Operating Systems (1)



- General structure of a multicomputer operating system

Multicomputer Operating Systems (2)



- Alternatives for blocking and buffering in message passing.

Example

- Sender

```
…
print (before send)
send (“my msg”)
print (after send)
```

- Receiver

```
…
print (before receive)
receive (msg)
print (after receive)
```


Multicomputer Operating Systems (3)

Synchronization point	Send buffer	Reliable comm. guaranteed?
Block sender until buffer not full	Yes	Not necessary
Block sender until message sent	No	Not necessary
Block sender until message received	No	Necessary
Block sender until message delivered	No	Necessary

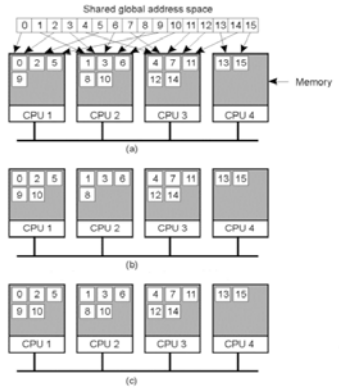
- Relation between blocking, buffering, and reliable communications.
- Dealing with failures.

Distributed Shared Memory Systems (1)

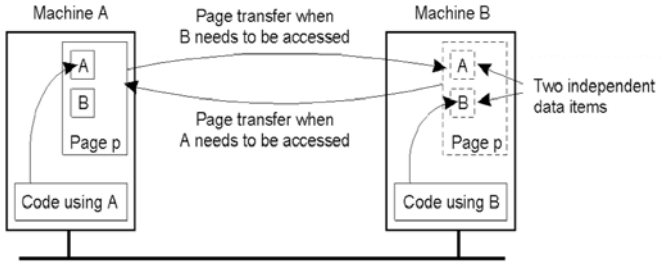
a) Pages of address space distributed among four machines

b) Situation after CPU 1 references page 10

c) Situation if page 10 is read only and replication is used

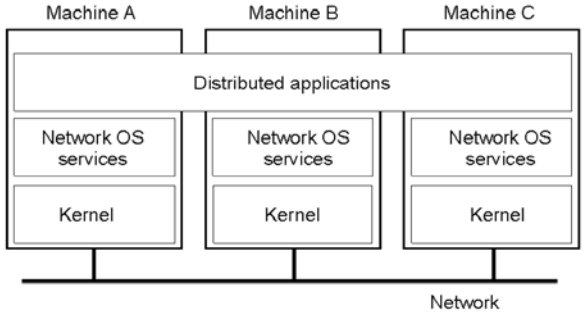


Distributed Shared Memory Systems (2)



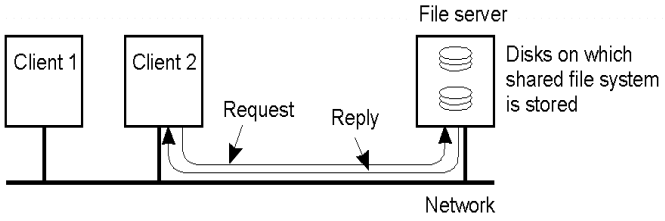
- False sharing of a page between two independent processes.

Network Operating System (1)



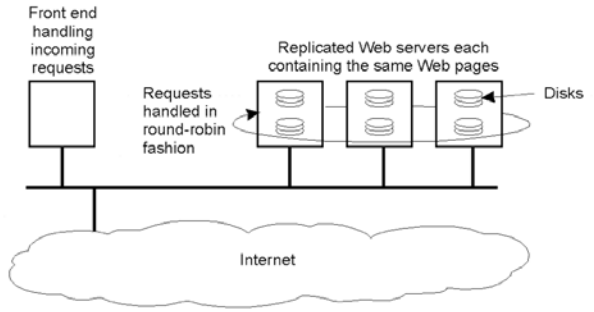
- General structure of a network operating system.

Network Operating System (2)



- Two clients and a server in a network operating system.

Web-sever Architectures

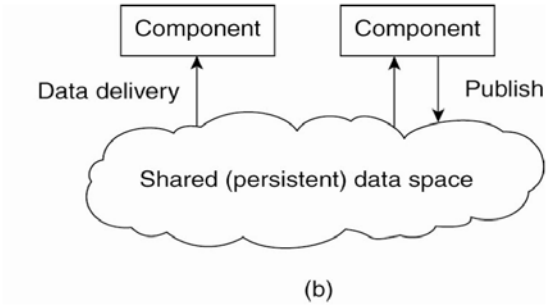


- An example of horizontal distribution of a Web service.

Middleware for Distributed Systems

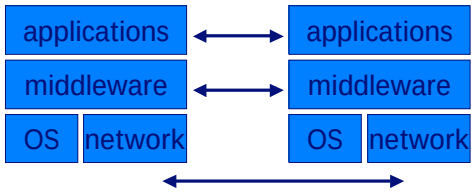
- Middleware is a layer of software between applications and OS that gives a uniform interface
- Central to developing distributed applications
- Different types
  - Document based (world-wide web)
  - File-system based (e.g., NFS)
  - Shared object-based (CORBA)
  - Coordination based (Linda, Publish-subscribe, Jini)

Coordination-based Approach



- Linda: the tuple space
  - In ("fred", 1958), Read ("fred", x), Out("fred", 1958)

Middleware Layer



- In an open middleware-based distributed system, the protocols used by each middleware layer should be the same, as well as the interfaces they offer to applications

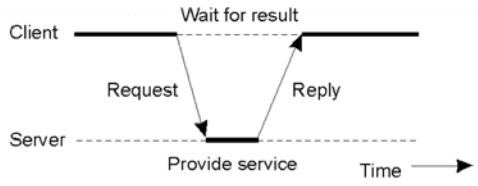
Comparison between Systems

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

System Architectures for Applications

- Centralized Architecture
  - Client-server model
  - Application layering
  - Multitiered architecture
- Decentralized Architecture
  - Peer-to-peer architecture
    - Distributed hash table (DHT)
  - Overlay Networks
- Hybrid Architectures

The Client and Server Model



- General interaction between a client and a server.
- Protocols (e.g., HTTP, SMTP, MIME, etc.)

An Example Client and Server (1)

```
/* Definitions needed by clients and servers.                    */
#define TRUE             1
#define MAX_PATH       255   /* maximum length of file name      */
#define BUF_SIZE      1024   /* how much data to transfer at once */
#define FILE_SERVER    243   /* file server's network address    */

/* Definitions of the allowed operations */
#define CREATE           1   /* create a new file                */
#define READ             2   /* read data from a file and return it */
#define WRITE            3   /* write data to a file             */
#define DELETE           4   /* delete an existing file          */

/* Error codes. */
#define OK               0   /* operation performed correctly    */
#define E_BAD_OPCODE    -1   /* unknown operation requested      */
#define E_BAD_PARAM     -2   /* error in a parameter             */
#define E_IO            -3   /* disk error or other I/O error    */

/* Definition of the message format. */
struct message {
     long source;             /* sender's identity               */
     long dest;               /* receiver's identity             */
     long opcode;             /* requested operation             */
     long count;              /* number of bytes to transfer     */
     long offset;             /* position in file to start I/O   */
     long result;             /* result of the operation         */
     char name[MAX_PATH];     /* name of file being operated on  */
     char data[BUF_SIZE];     /* data to be read or written      */
};
```

- The *header.h* file used by the client and server.

An Example Client and Server (2)

```
#include <header.h>
void main(void) {
     struct message ml, m2;              /* incoming and outgoing messages      */
     int r;                              /* result code                         */

     while(TRUE) {                       /* server runs forever                 */
          receive(FILE_SERVER, &ml);     /* block waiting for a message         */
          switch(ml.opcode) {            /* dispatch on type of request         */
               case CREATE:   r = do_create(&ml, &m2); break;
               case READ:     r = do_read(&ml, &m2); break;
               case WRITE:    r = do_write(&ml, &m2); break;
               case DELETE:   r = do_delete(&ml, &m2); break;
               default:       r = E_BAD_OPCODE;
          }
          m2.result = r;                 /* return result to client             */
          send(ml.source, &m2);          /* send reply                          */
     }
}
```

- A sample server.

An Example Client and Server (3)

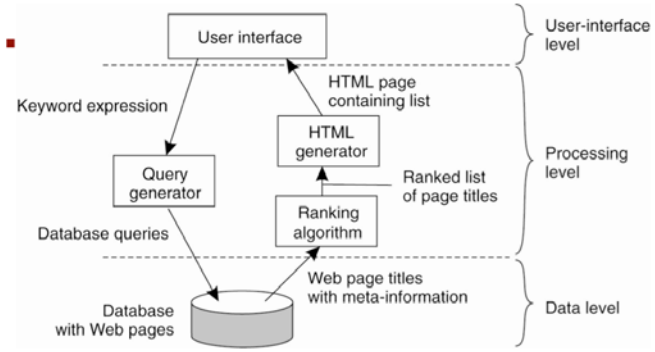
```
#include <header.h>
int copy(char *src, char *dst){          /* procedure to copy file using the server */
     struct message ml;                  /* message buffer                          */
     long position;                      /* current file position                   */
     long client = 110;                  /* client's address                        */

     initialize( );                      /* prepare for execution                   */
     position = 0;
     do {
          ml.opcode = READ;              /* operation is a read                     */
          ml.offset = position;          /* current position in the file            */
          ml.count  = BUF_SIZE;                                                      /* how many bytes to read*/
          strcpy(&ml.name, src);         /* copy name of file to be read to message */
          send(FILESERVER, &ml);         /* send the message to the file server     */
          receive(client, &ml);          /* block waiting for the reply             */

          /* Write the data just received to the destination file.                  */
          ml.opcode = WRITE;             /* operation is a write                    */
          ml.offset = position;          /* current position in the file            */
          ml.count  = ml.result;         /* how many bytes to write                 */
          strcpy(&ml.name, dst);         /* copy name of file to be written to buf  */
          send(FILE_SERVER, &ml);        /* send the message to the file server     */
          receive(client, &ml);          /* block waiting for the reply             */
          position += ml.result;         /* ml.result is number of bytes written    */
     } while( ml.result > 0 );           /* iterate until done                      */
     return(ml.result >= 0 ? OK : ml result);  /* return OK or error code          */
}
```

- A client using the server to copy a file.

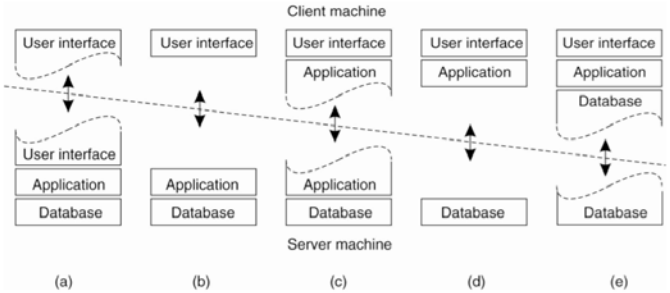
Application Layering



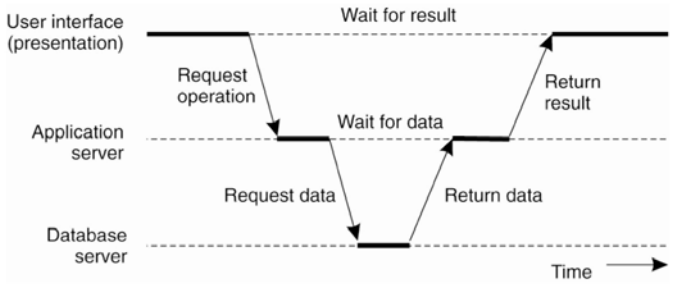
Multitiered Architectures (1)

- 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
- A server machine containing the rest,
  - the programs implementing the processing and data level

Multitiered Architectures (2)

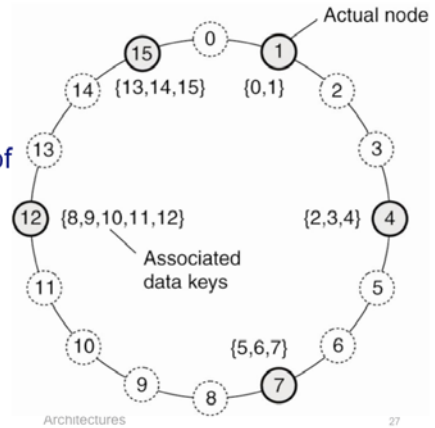


Multitiered Architectures (3)



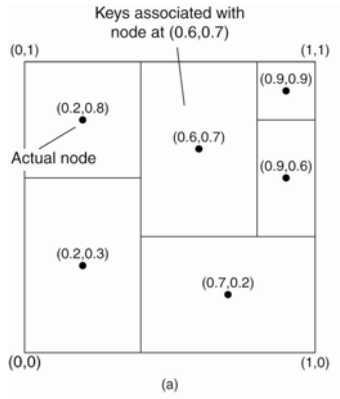
Structured Peer-to-Peer Architectures (1)

- DHT (Distributed Hash Table)
- Ex: the mapping of data items onto nodes in Chord.

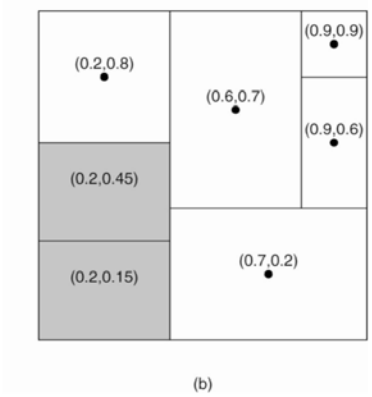


Structured Peer-to-Peer Architectures (2)

- Content Addressable Network (CAN)
- Figure 2-8. (a) The mapping of data items onto nodes in CAN.

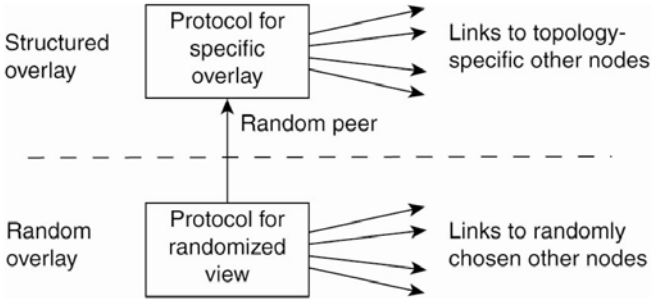


Structured Peer-to-Peer Architectures (3)



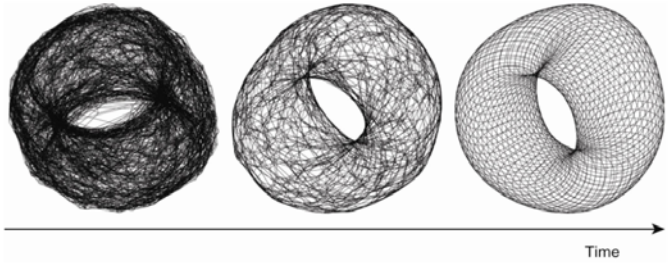
- Figure 2-8. (b) Splitting a region when a node joins.

Topology Management of Overlay Networks



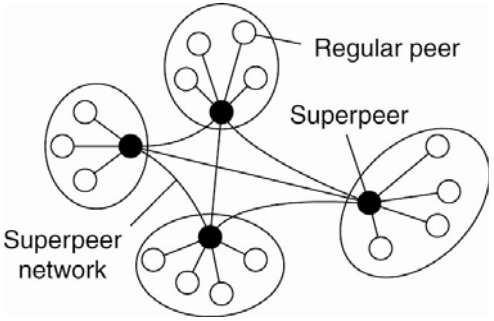
- Structured overlay based on ranking functions
  - E.g., Semantic proximity

Topology Management of Overlay Networks



- Figure 2-11. Generating a specific overlay network using a two-layered unstructured peer-to-peer system [adapted with permission from Jelasity and Babaoglu (2005)].

Superpeers

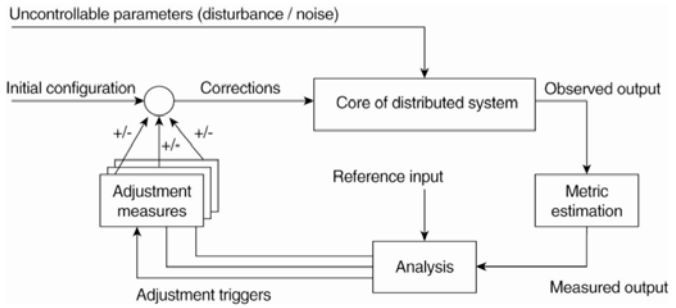


- Figure 2-12. A hierarchical organization of nodes into a superpeer network.

Self-Management of Distributed Systems

- Issues
  - What to monitor
  - How to monitor
  - How to evaluate
  - How to adapt
  - …
- Three steps
  - Monitor
  - Check
  - Steer/Adapt

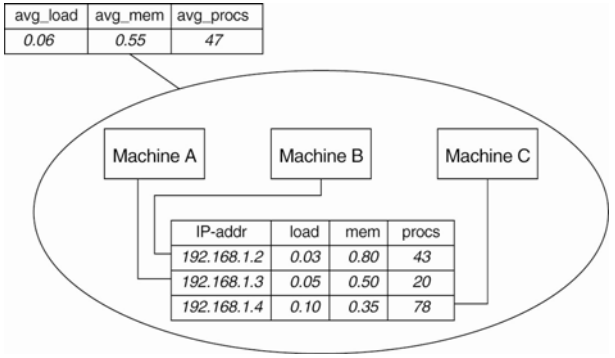
The Feedback Control Model



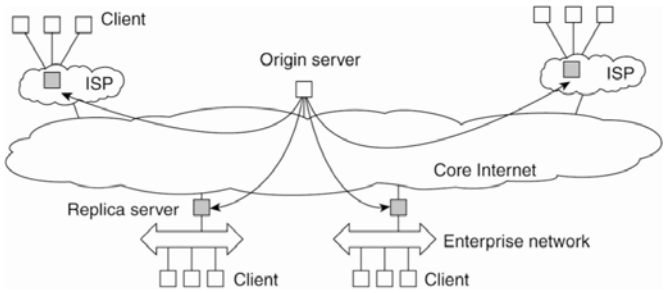
Possible techniques for adaption

1. Changing system parameter values
2. Micro resets
3. Dynamic replacement of components
4. System regeneration by global reset and restriction to essential functionality

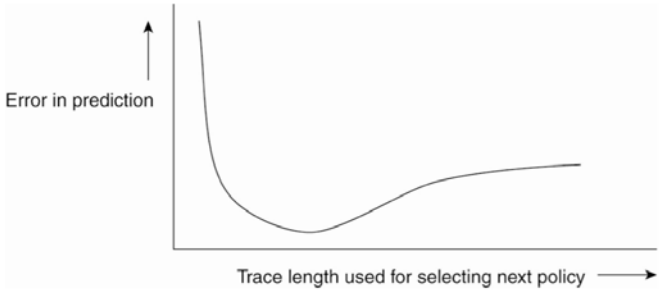
Example: Systems Monitoring with Astrolabe



Example: Differentiating Replication Strategies in Globule (1)



Example: Differentiating Replication Strategies in Globule (2)



- Figure 2-19. The dependency between prediction accuracy and trace length.

Example: Automatic Component Repair Management in Jade

- Steps required in a repair procedure:
- Terminate every binding between a component on a nonfaulty node, and a component on the node that just failed.
- Request the node manager to start and add a new node to the domain.
- Configure the new node with exactly the same components as those on the crashed node.
- Re-establish all the bindings that were previously terminated.