Architecture Models of Distributed Systems

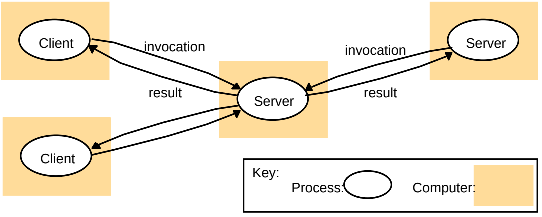
- An architectural model of a distributed system is concerned with the definition and placement of its components and relationship between them. Its goals:
  - Meet present and likely future demands.
  - Make the system reliable, manageable, adaptable, and cost-effective.
- An architectural Model should:
  - Simplify and abstract the functions of individual components
    - Example of an initial simplification is achieved by classifying processes as server process / client process / peer process.
  - Define the placement of the components across a network of computers and patterns for the distribution of data and workloads
  - The interrelationship between the components
    - i.e. functional roles and the patterns of communication between them.
- Examples of architecture models:
  - Client-Sever,
  - Peer-to-peer,
  - Service Oriented Architecture.

Client/Server Basic Model (1)

For a specific service; processes are divided into two groups: servers and clients

- Client:
  - A process that requests service. Clients usually invoked by end users when they require service. A Client usually blocks until server responds.
- Server:
  - A process that provides service and usually with special privileges. A Server usually waits for incoming requests.
  - A Server can have many clients making concurrent requests.

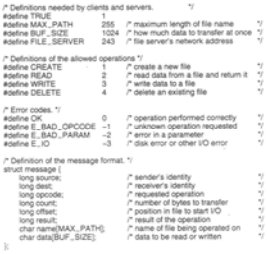
Client/Server Basic Model (2)



- Client process interact with individual server processes in order to access data or resource. The server in turn may use services of other servers.
- Examples:
  - A Web browser is a client to a Web Server which is often a client of file server.

An Example Client and Server (1)

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

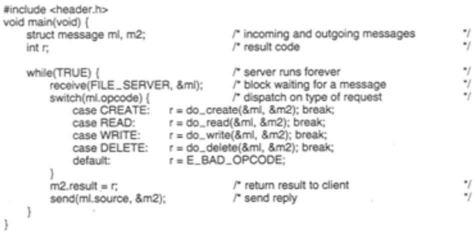


```
/* Definitions needed by clients and servers.                    */
#define TRUE            1
#define MAX_PATH        255   /* maximum length of file name       */
#define BUF_SIZE        1024  /* how much data to transfer at once */
#define FILE_SERVER     243   /* file server's network address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

/* 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        */
};
```

Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

An Example Client and Server (2)

- A sample server.

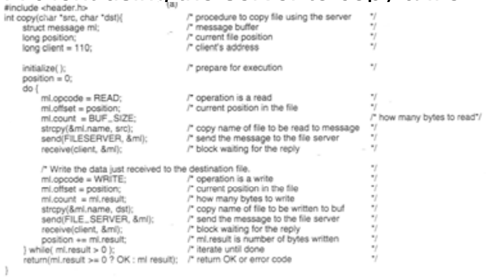


```
#include <header.h>
void main(void) {
    struct message ml, m2;              /* incoming and outgoing messages */
    int r;                              /* result code                    */

    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                     */
    }
}
```

Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

An Example Client and Server (3)

- A client using the server to copy a file.



```
#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   */
}
```

Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

Advantages of the Client-Server Architecture

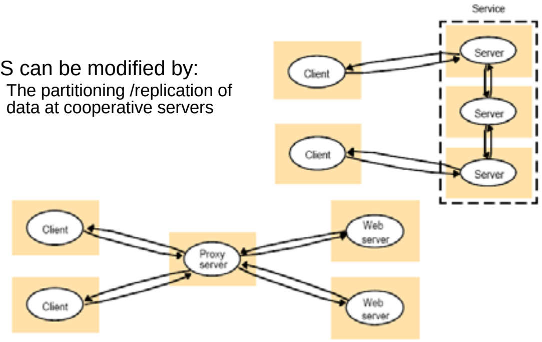
- Efficient division of labor.
- Horizontal and vertical scaling of resources.
- Better price/performance on client machines.
- Ability to use familiar tools on client machines.
- Client access to remote data (via standards) .
- Full DBMS functionality provided to client workstations.
- Overall better system price/performance.

Problems with the Multiple Client / Single Server Architecture

- Server forms a bottleneck.
- Server forms a single point of failure.
- System scaling is difficult.

Variants of Client/Sever Model

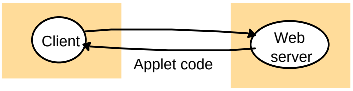
- C/S can be modified by:
  - The partitioning /replication of data at cooperative servers



  - The caching of data by proxy servers or clients

Variants of Client Sever Model: Mobile Code and Web Applets

a) client request results in the downloading of applet code



b) client interacts with the applet

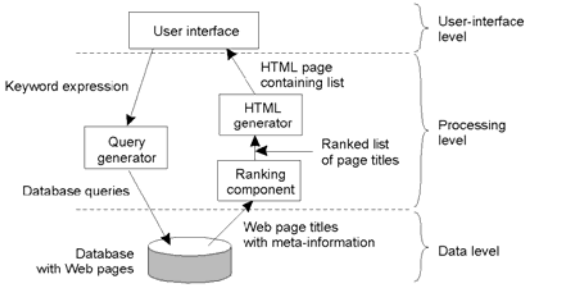
Client
Applet
Web server

- Applets downloaded to clients give good *interactive* response.
- Mobile codes such as applets are potential security threat,
  - The browser gives applets limited access to local resources (e.g. NO access to local/user file system).

Variants of Client Sever Model: Mobile Agents

- **Mobile agent**: A running program (code and data) that travels from one computer to another in a network carrying out an autonomous task, usually on behalf of some other process.
  - Advantages: flexibility, savings in communications cost
- Potential security threat to the resources in computers they visit. The environment receiving agent should decide which of the local resource(s) to allow. (e.g., crawlers and web servers).
- Agents themselves can be vulnerable – they may not be able to complete task if they are refused access.

Application S/W Logical Components

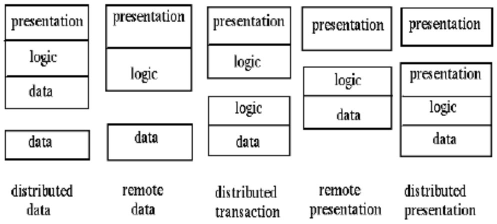


Copywrite: Sanjeev Setia; Distributed Software Systems

- Many applications can be considered to be made up of three software components or logical tiers:
  - user interface, processing (logic) layer, and data layer

Architecture of Application S/W

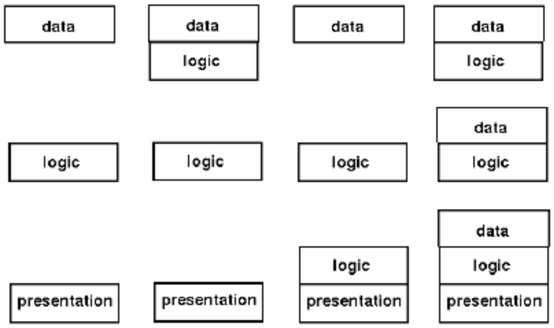
- Client/server architectures
  - single-physical tiered
  - two-physical tiered
  - multi-tiered



e.g. Distributed DB Network File system WWW Telnet X-windows

Gartener Group Configurations

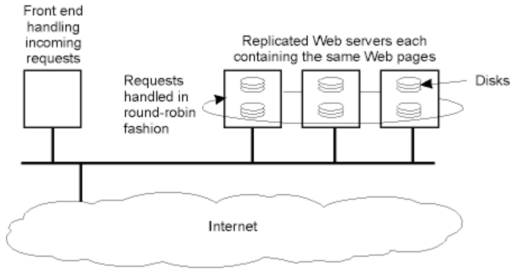
3-Tier C/S Architecture



Advantages of Multi-Tier Architecture

- Frees clients from dependencies on the exact implementation of the database.
- It allows “business logic” to be concentrated in one place.
- Software updates are restricted to middle layer
- Performance improvements possible by batching requests from many clients to the database.
- Database and business logic tiers could be implemented by multiple servers for scalability

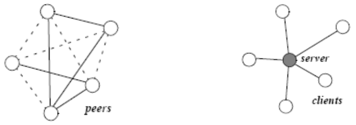
- An example of horizontal distribution of a Web service.


Front end handling incoming requests
Replicated Web servers each containing the same Web pages
Requests handled in round-robin fashion
Disks
Internet

Peer-to-Peer (P2P) Architecture

Definition:

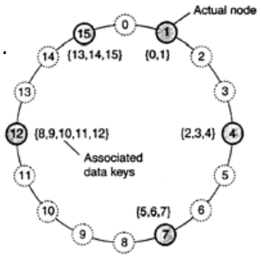
- A P2P computer network refers to any network that does not have fixed clients and servers, but a number of peer nodes that function as both clients and servers to other nodes on the network. (*Wikipedia.org)*
- P2P computing is an alternative to the centralized and client-server models of computing,
  - In its purest form, the P2P model has no concept of server; rather all participants are peers.



- Because accessing these decentralized resources means operating in an *un-trusted* environment of *unstable* connectivity and *unpredictable* IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers.

Structured Peer-to-Peer

- Chord System: nodes are logically organized in a ring.
- Mapping between nodes and the data they own is required.
- Function lookup(k) returns the network address of the node owning k. Lookups can be done in O(log(N)), where N is the number of nodes.



Tanenbaum and van Steen, Distributed Systems: Principles and Paradigms. Prentice-Hall, Inc. 2002

Unstructured Peer-to-Peer

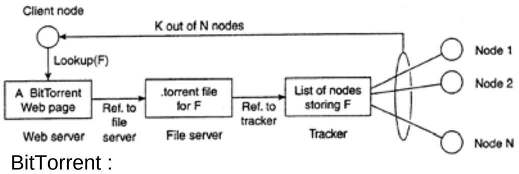
- Rely on **randomized algorithms** for constructing overlay networks that resembles a **random graph**.
- Main idea:
  - Each node maintains a list of neighbors, but that this list is constructed in a more or less random way.
  - Data items are assumed to be randomly placed on nodes.
  - Goal is that each node constructs a partial view of the graph.

Peer-peer applications

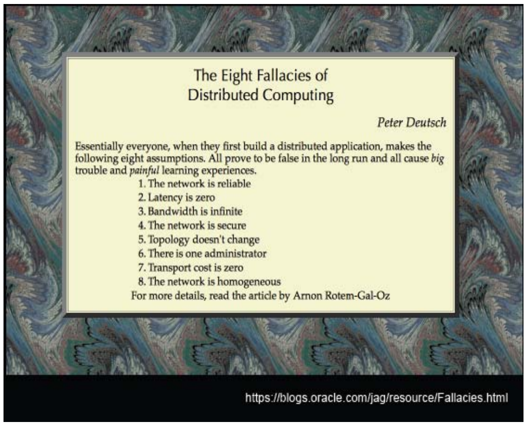
- File sharing
  - Napster, Gnutella, KaZaa.
  - Second generation projects
    - Oceanstore, PAST, Freehaven, FreeNet.
- Distributed Computation
  - SETI@home, Entropia, Parabon, United Devices, Popular Power.
- Other Applications
  - Content Distribution (BitTorrent).
  - Instant Messaging (Jabber), Anonymous Email.
  - Groupware (Groove).
  - P2P Databases.

Hybrid Architecture

- Solution with client-server architectures are combined with decentralized architectures.



- BitTorrent :
  - A centralized server is needed to let the client know about the nodes from which chuncks of the file can be downloaded.
  - Once the client joins the system as a node, a decentralized architecture will be used.



The Eight Fallacies of Distributed Computing

Peter Deutsch

Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause *big* trouble and *painful* learning experiences.

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

For more details, read the article by Arnon Rotem-Gal-Oz

https://blogs.oracle.com/jag/resource/Fallacies.html