Chapter 2

Architectures

Architectural Styles

Software architecture - logical organization of distributed systems into software components

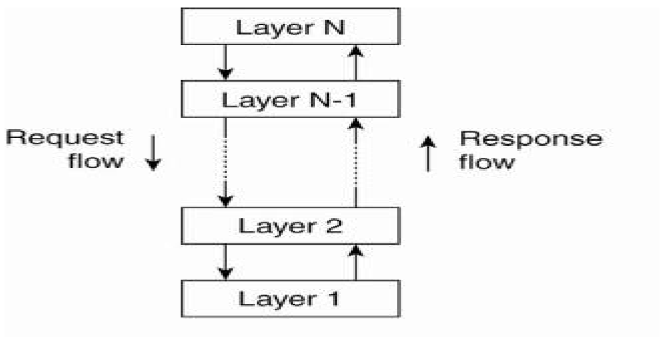
Architectural style:

- Types of components
- The way that components are connected
- The data exchanged between components
- How these elements are jointly configured into a system.

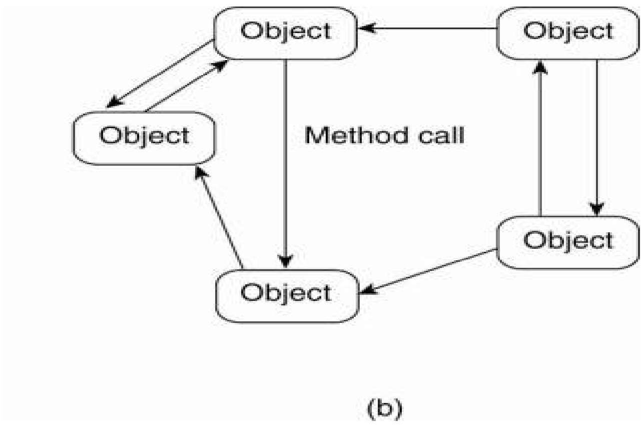
Software component - a modular unit with well-defined, required and provided interfaces that is ***replaceable*** within its environment.

Connector - a mechanism that mediates communication, coordination, or cooperation among componentse.g. a connector can be formed by the facilities for (remote) procedure calls, message passing, or streaming data.

Architectural styles for distributed systems are:
1. Layered architectures
The (a) layered and (b) object-based architectural style.
2. Object-based architectures
· each object corresponds a component
· components are connected through a (remote) procedure call mechanism. (The layered and object-based architectures still form the most important styles for large Softwaresystems).



(a)



3. Data-centered architectures

· processes communicate through a common (passive or active) repository

· e.g.

§ wealth of networked applications have been developed that rely on a shared distributed file system in which virtually all communication takes place through files.

§ Web-based distributed systems are largely data-centric: processes communicate through the use of shared Web-based data services.

4. Event-based architectures

o processes communicate through the propagation of events

o e.g. publish/subscribe systems (Eugster et al., 2003).

§ processes publish events after which the middleware ensures that only those processes that subscribed to those events will receive them.

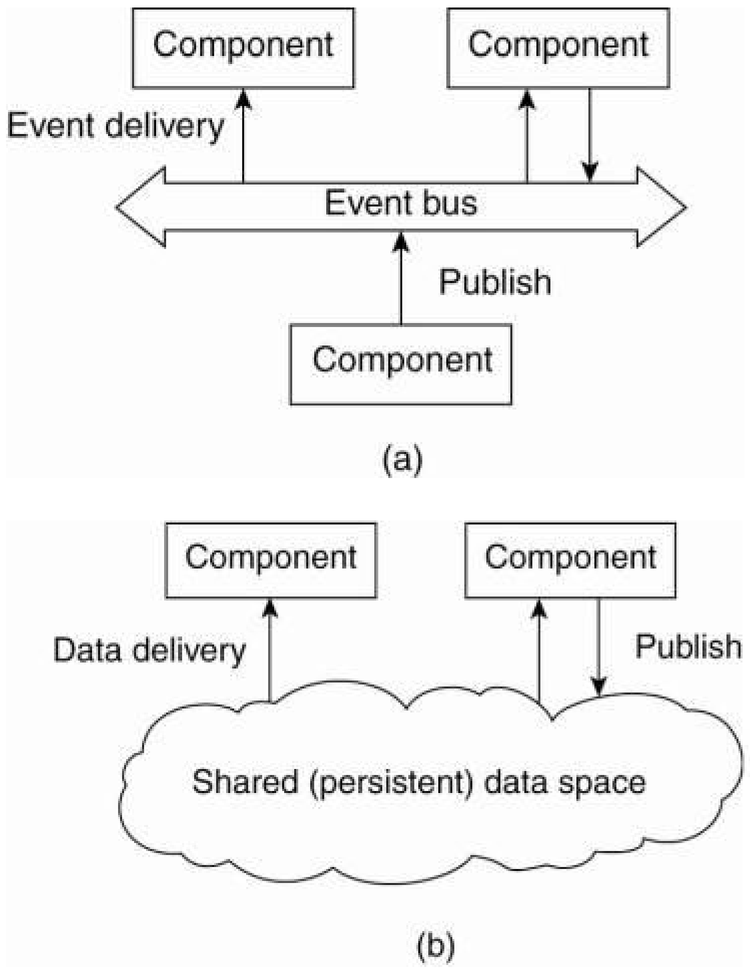
○ advantage - processes are loosely coupled. In principle, they need not explicitly refer to each other. This is also referred to as being decoupled in space, or referentially decoupled.

Shared data spaces – combination of event-based architectures with data-centered architectures.

○ processes are decoupled in time: they need not both be active when communication takes place.

○ many shared data spaces use a SQL-like interface to the shared repository - data can be accessed using a *description* rather than an *explicit reference*, as is the case with files.

The (a) event-based and (b) shared data-space architectural style.



Aim of Architectures: achieving distribution transparency.

System Architectures

Centralized Architectures

Manage distributed system complexity - think in terms of clients that request services from servers.

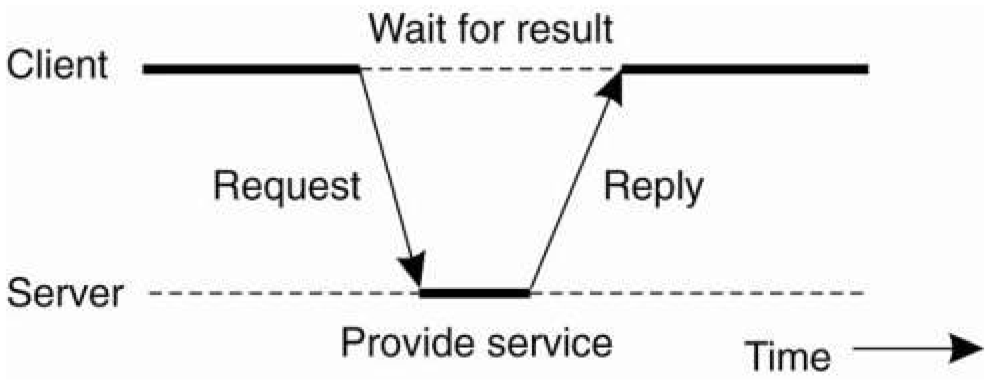
Basic client-server model:

o Processes are divided into two groups:

1. A server is a process implementing a specific service, for example, a file system service or a database service.

2. A client is a process that requests a service from a server by sending it a request and subsequently waiting for the server's reply.

○ General interaction between a client and a server.



Communication - implemented using a connectionless protocol when the network is reliable -> e.g. local-area networks.

1. Client requests a service – packages and sends a message for the server, identifying the service it wants, along with the necessary input data.

2. The Server will always wait for an incoming request, process it, and package the results in a reply message that is then sent to the client.

Connectionless protocol

○ Describes communication between two network end points in which a message can be sent from one end point to another without prior arrangement.

○ Device at one end of the communication transmits data to the other, without first ensuring that the recipient is available and ready to receive the data.

○ The device sending a message sends it addressed to the intended recipient.

○ More frequent problems with transmission than with connection-orientated protocols and it may be necessary to resend the data several times.

○ making the protocol resistant to occasional transmission failures is not trivial.

○ the client cannot detect whether the original request message was lost, or that transmission of the reply failed.

§ If the reply was lost, then resending a request may result in performing the operation twice

· e.g. If operation was "transfer $10,000 from my bank account," then clearly, it would have been better that we simply reported an error instead.

§ When an operation can be repeated multiple times without harm, it is said to be idempotent

○ Often disfavored by network administrators because it is much harder to filter malicious packets from a connectionless protocol using a firewall.

○ e.g. connectionless protocols -The Internet Protocol (IP) and User Datagram Protocol (UDP) are connectionless protocols,

○ Alternative - connection-oriented protocol (TCP/IP -the most common use of IP)

○ not appropriate in a local-area network due to relatively low performance

○ works fine in wide-area systems in which communication is inherently unreliable.

○ e.g. virtually all Internet application protocols are based on reliable TCP/IP connections.

§ whenever a client requests a service, it first sets up a connection to the server before sending the request.

§ The server uses that same connection to send the reply message, after which the connection is torn down.

§ Problem: setting up and tearing down a connection is relatively costly, especially when the request and reply messages are small.

Application Layering

Issues with Client / Server:

○ How to draw a clear distinction between a client and a server.

○ Often no clear distinction.

o e.g. a server for a distributed database may continuously act as a client because it is forwarding requests to different file servers responsible for implementing the database tables.

Since many client-server applications are targeted toward supporting user access to databases, **distinctions may be analyzed in a layered architectural style**:

1. The user-interface level - contains all that is necessary to directly interface with the user, such as display management

o Clients typically implement the user-interface level

o simplest user-interface program - character-based screen

§ the user's terminal does some local processing such as echoing typed keystrokes, or supporting form-like interfaces in which a complete entry is to be edited before sending it to the main computer

o Simple GUI

§ pop-up or pull-down menus are used with many screen controls handled through a mouse instead of the keyboard.

o Modern user interfaces offer considerably more functionality by allowing applications to share a single graphical window, and to use that window to exchange data through user actions.

2. The processing level - contains the applications

o middle part of hierarchy -> logically placed at the processing level

3. The data level - manages the actual data that is being acted on

Example: Internet search engine

· User-interface level: a user types in a string of keywords and is subsequently presented with a list of titles of Web pages.

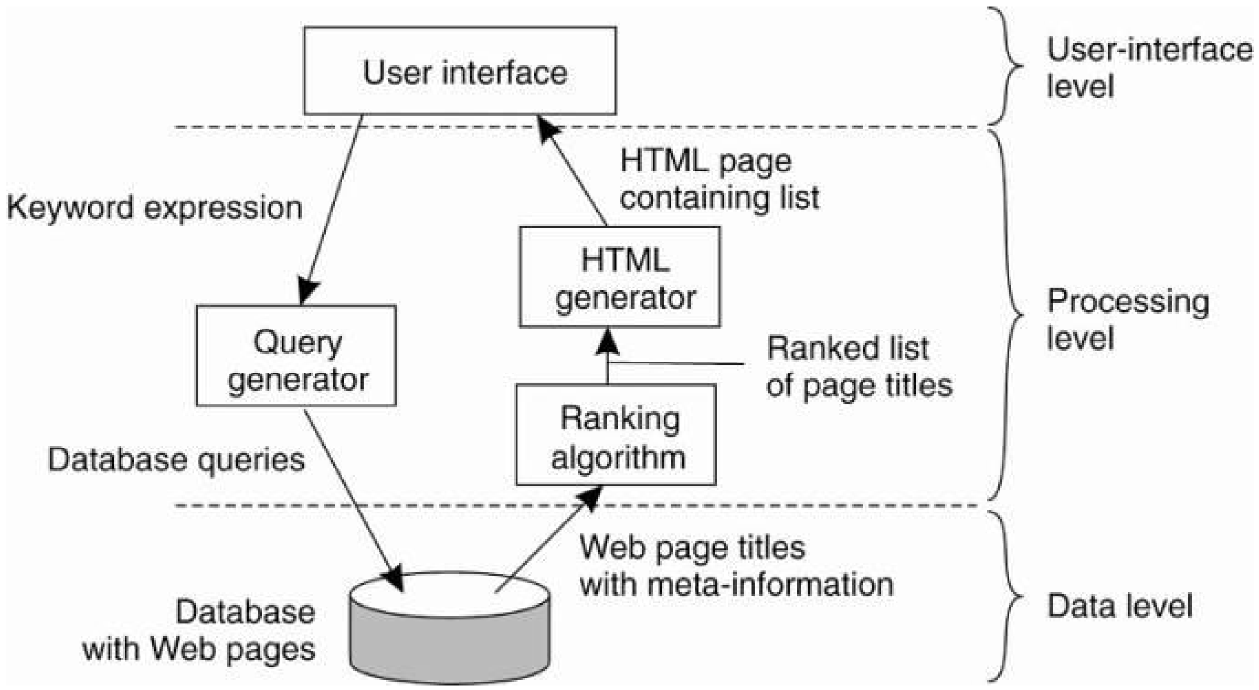
· Data Level: huge database of Web pages that have been prefetched and indexed.

· Processing level: - search engine that transforms the user's string of keywords into one or more database queries.

o ranks the results into a list

o transforms that list into a series of HTML pages

Simplified organization of an Internet search engine into three different layers.



Client-Server Model – Data Level

· Contains the programs that maintain the actual data on which the applications operate.

· Data are often persistent - even if no application is running, data will be stored somewhere for next use.

· Data level consists of a file system, but it is more common to use a full-fledged database.

· Data level is typically implemented at the server side.

· Responsible for keeping data consistent across different applications.

O With databases - metadata such as table descriptions, entry constraints and application-specific metadata are also stored at this level.

· **Relational database** organize most business-oriented data.

○ Data independence is crucial

· data are organized independent of the applications in such a way that changes in that organization do not affect applications, and neither do the applications affect the data organization.

○ Using relational databases in the client-server model helps separate the processing level from the data level, as processing and data are considered independent.

· Other Data base choices –

○ many applications operate on complex data types that are more easily modeled in terms of objects than in terms of relations.

○ implement the data level by means of an object-oriented or object-relational database.

§ built upon the widely dispersed relational data model, while offering the advantages ofobject-orientation.

Multitiered Architectures

Possibilities for physically distributing a client-server application across several machines.

· Simplest organization - two types of machines:

1. A client machine containing only the programs implementing (part of) the user-interface level

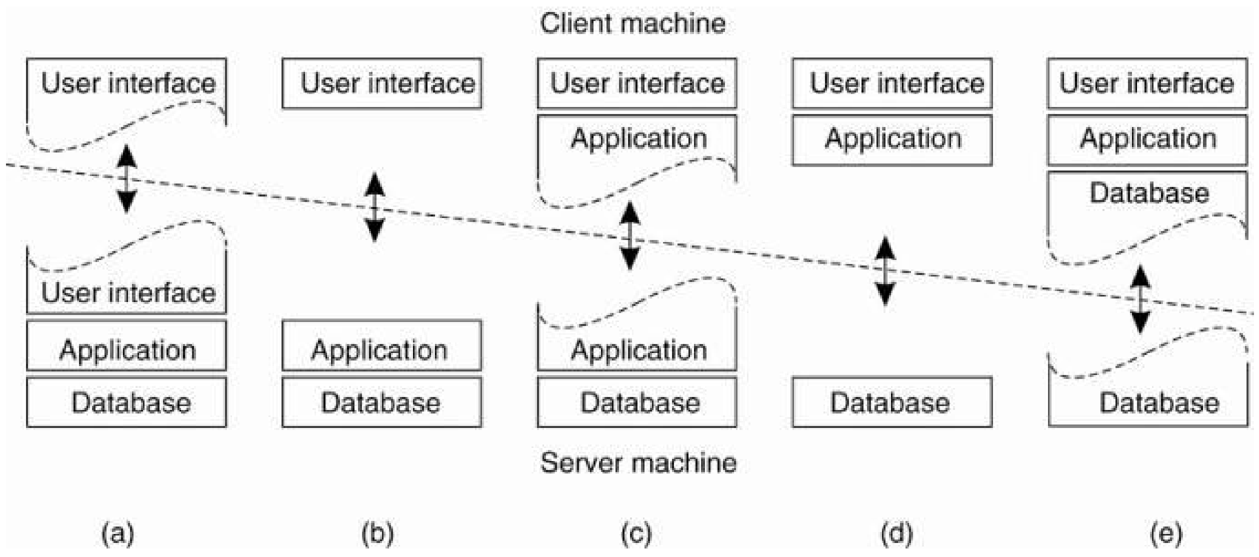
2. A server machine containing the rest, that is the programs implementing the processing and data level

○ Everything is handled by the server while the client is essentially no more than a dumb terminal, possibly with a pretty graphical interface.

· Distribute the programs in the application layers across different machines [Jing et al. (1999)].

○ Two-tiered architecture: client machines and server machines.

○ Alternative client-server organizations (a)–(e).



Cases:

A: only the terminal-dependent part of the user interface on the client machine

B: place the entire user-interface software on the client side

○ Divide the application into a graphical front end, which communicates with the rest of the application (residing at the server) through an application-specific protocol.

○ the front end (the client software) does no processing other than necessary for presenting the application's interface

C: move part of the application to the front end

○ e.g. the application makes use of a form that needs to be filled in entirely before it can be processed

○ front end can then check the correctness and consistency of the form, and where necessary interact with the user

D: used where the client machine is a PC or workstation, connected through a network to a distributed file system or database

○ most of the application is running on the client machine, but all operations on files or database entries go to the server

○ e.g. many banking applications run on an end-user's machine where the user prepares transactions and such

Once finished, the application contacts the database on the bank's server and uploads the transactions for further processing

E: used where the client machine is a PC or workstation, connected through a network to a distributed file system or database

- the situation where the client's local disk contains part of the data

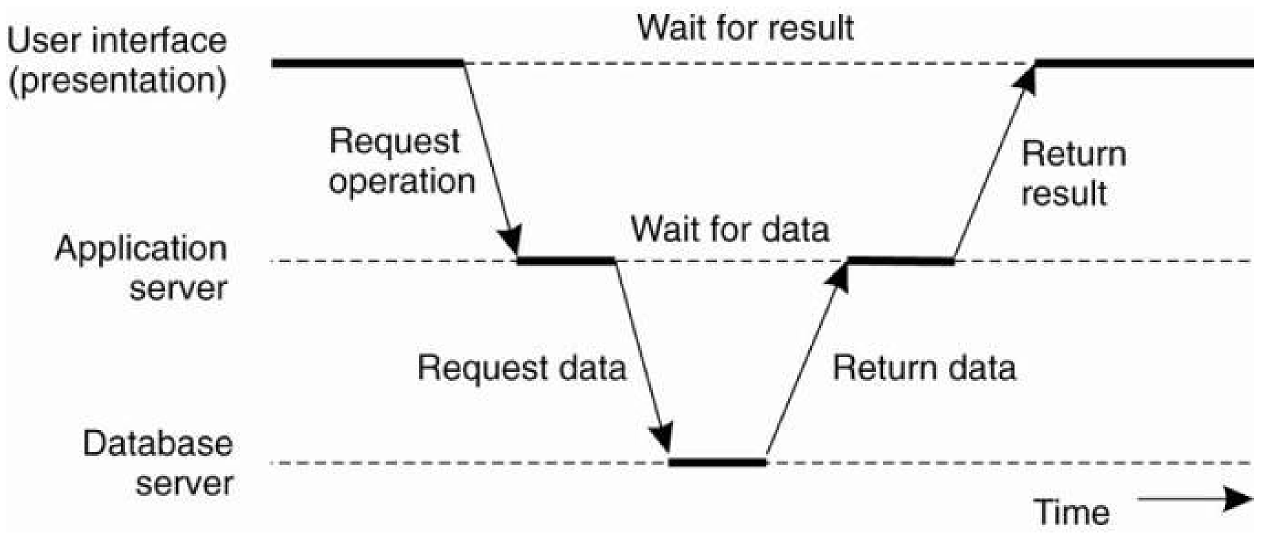
Issues:

Trend to move away from the configurations D and E.

- Although client machines do a lot, they are also more problematic to manage
- Having more functionality on the client machine makes client-side software more prone to errors and more dependent on the client's underlying platform (i.e., operating system and resources).
- From a system's management perspective, having fat clients is not optimal.
- Thin clients in A – C are much easier

Trend: server-side solutions are becoming increasingly more distributed as a single server is being replaced by multiple servers running on different machines.

- a server may sometimes need to act as a client leading to a (physically) three-tiered architecture.
- Example of a server acting as client.



○ Programs that form part of the processing level reside on a separate server, but may additionally be partly distributed across the client and server machines.

○ e.g. three-tiered architecture - organization of Web sites.

§ Web server acts as an entry point to a site, passing requests to an application server where the actual processing takes place.

§ Application server interacts with a database server.

· e.g., an application server may be responsible for running the code to inspect the available inventory of some goods as offered by an electronic bookstore. To do so, it may need to interact with a database containing the raw inventory data.

Decentralized Architectures

Vertical Distribution Architecture

○ Achieved by placing logically different components on different machines

○ term is related to the concept of vertical fragmentation as used in distributed relational databases, where it means that tables are split column-wise, and subsequently distributed across multiple machines

○ Multitiered client-server architectures are a direct consequence of dividing applications into a user-interface, processing components, and a data level. The different tiers correspond directly with the logical organization of applications.

○ Vertical Distribution - organizing a client-server application as a multitiered architecture.

○ Can help manage distributed systems by logically and physically splitting functions across multiple machines, where each machine is tailored to a specific group of functions.

Horizontal Distribution Architecture

○ Client or server may be physically split up into logically equivalent parts, but each part is operating on its own share of the complete data set, thus balancing the load.

e.g peer-to-peer systems.

○ Processes that constitute a peer-to-peer system are all equal.

○ Functions that need to be carried out are represented by every process that constitutes the distributed system.

○ Much of the interaction between processes is symmetric:

○ each process will act as a client and a server at the same time (which is also referred to as acting as a servent).

○ Peer-to-peer architectures - how to organize the processes in an overlay network in which the nodes are formed by the processes and the links represent the possible communication channels (which are usually realized as TCP connections).

○ A Process cannot communicate directly with an arbitrary other process, but is required to send messages through the available communication channels.

○ Two types of overlay networks exist: those that are structured and those that are not. (Castro et al. 2005).

○ Survey paper (Lua et al. 2005).

○ A reference architecture that allows for a more formal comparison of the different types of peer-to-peer systems (Aberer et al. 2005) provide a reference architecture that allows for a more formal comparison of the different types of peer-to-peer systems.

○ A survey taken from the perspective of content distribution is provided by (Androutsellis-Theotokis and Spinellis 2004).

Structured Peer-to-Peer Architectures

○ The P2P overlay network consists of all the participating peers as network nodes.

○ There are links between any two nodes that know each other: i.e. if a participating peer knows the location of another peer in the P2P network, then there is a directed edge from the former node to the latter in the overlay network.

○ Based on how the nodes in the overlay network are linked to each other, we can classify the P2P networks as unstructured or structured.

○ Some well known structured P2P networks are Chord, Pastry, Tapestry, CAN, and Tulip.

○ A structured Peer-to-Peer overlay network is constructed using a deterministic procedure.

○ Most-used procedure - organize the processes through a distributed hash table (DHT). (Hash Table description)

o DHT-based system –

○ data items are assigned a random key from a large identifier space, such as a 128-bit or 160-bit identifier.

○ nodes are assigned a random number from the same identifier space.

○ DHT-based system implements an efficient and deterministic scheme that uniquely maps the key of a data item to the identifier of a node based on some distance metric (Balakrishnan, 2003).

○ When looking up a data item, the network address of the node responsible for that data item is returned.

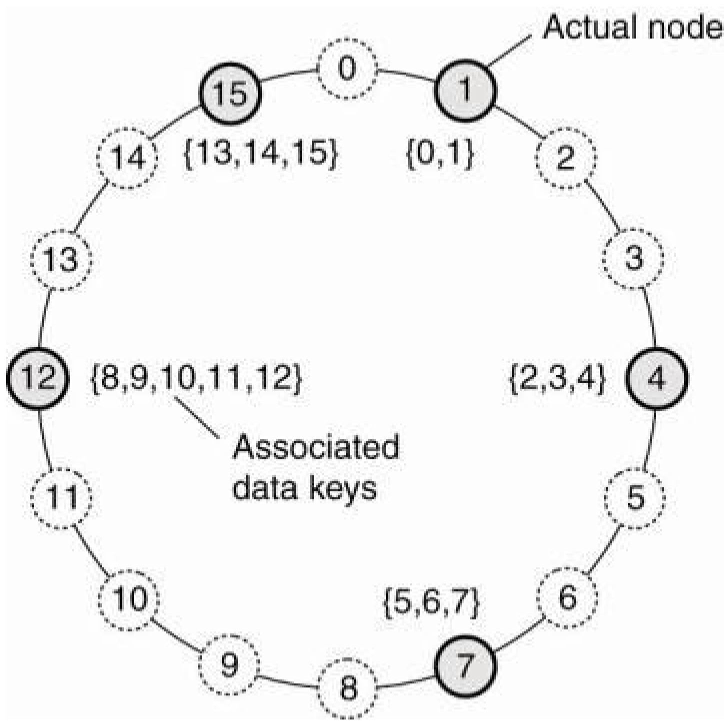
○ This is accomplished by routing a request for a data item to the responsible node.

Example: Chord system (Stoica et al., 2003) the nodes are logically organized in a ring such that a data item with key k is mapped to the node with the smallest identifier idk.

○ This node is referred to as the successor of key k and denoted as succ(k)

○ To look up the data item an application running on an arbitrary node would then call the function LOOKUP(k) which would subsequently return the network address of succ(k). At that point, the application can contact the node to obtain a copy of the data item.

The mapping of data items onto nodes in Chord.



How do nodes organize themselves into an overlay network?

○ Looking up a key does not follow the logical organization of nodes in the ring.

○ Each node will maintain shortcuts to other nodes in such a way that lookups can generally be done in O(log (N)) number of steps, where N is the number of nodes participating in the overlay.

Joining the P2P Network

1. When a node wants to join the system, it starts with generating a random identifier id.

2. Then, the node can simply do a lookup on id, which will return the network address of succ(id).

3. The joining node then contacts succ(id) and its predecessor and insert itself in the ring.

a. This scheme requires that each node also stores information on its predecessor.

b. Insertion also yields that each data item whose key is now associated with node id, is transferred from succ(id).

Leaving the P2P Network

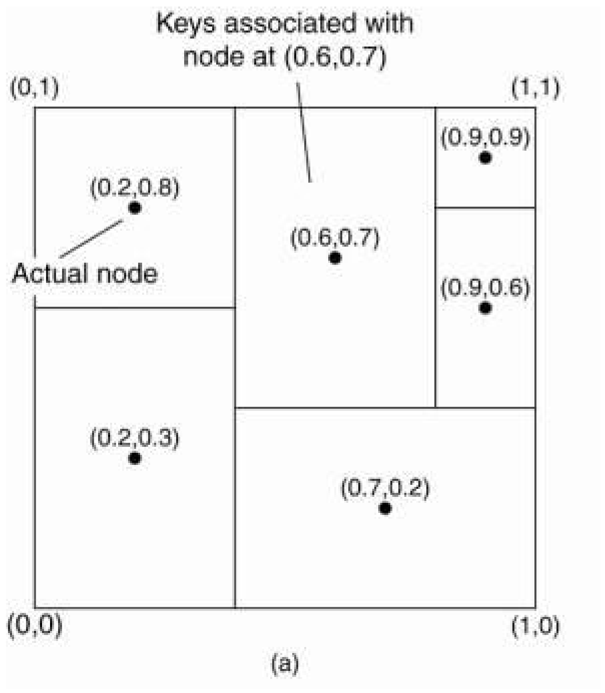
1. node id informs its departure to its predecessor and successor

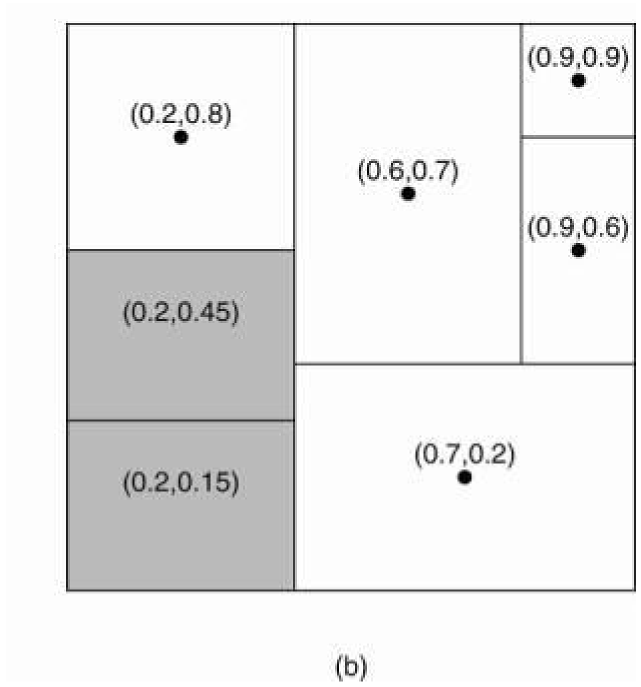
2. transfers its data items to succ(id).

Content Addressable Network (CAN) – (Ratnasamy et al. 2001).

- CAN deploys a d-dimensional Cartesian coordinate space, which is completely partitioned among all all the nodes that participate in the system.
- Example: 2-dimensional case

(a) The mapping of data items onto nodes in CAN. (b) Splitting a region when a node joins.





· Two-dimensional space [0,1]x[0,1] is divided among six nodes.

· Each node has an associated region.

· Every data item in CAN will be assigned a unique point in this space, after which it is also clear which node is responsible for that data (ignoring data items that fall on the border of multiple regions, for which a deterministic assignment rule is used).

Joining CAN

· When a node P wants to join a CAN system, it picks an arbitrary point from the coordinate space and subsequently looks up the node Q in whose region that point falls.

· Node Q then splits its region into two halves and one half is assigned to the node P.

· Nodes keep track of their neighbors, that is, nodes responsible for adjacent region.

· When splitting a region, the joining node P can easily come to know who its new neighbors are by asking node P.

· As in Chord, the data items for which node P is now responsible are transferred from node Q.

Leaving CAN

· Assume that the node with coordinate (0.6,0.7) leaves.

· Its region will be assigned to one of its neighbors, say the node at (0.9,0.9), but it is clear that simply merging it and obtaining a rectangle cannot be done.

· In this case, the node at (0.9,0.9) will simply take care of that region and inform the old neighbors of this fact.

· This may lead to less symmetric partitioning of the coordinate space, for which reason a background process is periodically started to repartition the entire space.

Unstructured Peer-to-Peer Architectures (Risson and Moors, 2006).

An unstructured P2P network is formed when the overlay links are established arbitrarily.

- Such networks can be easily constructed as a new peer that wants to join the network can copy existing links of another node and then form its own links over time.
- In an unstructured P2P network, if a peer wants to find a desired piece of data in the network, the query has to be flooded through the network in order to find as many peers as possible that share the data.
- Main disadvantage - queries may not always be resolved.
- Popular content is likely to be available at several peers and any peer searching for it is likely to find the same thing, but, if a peer is looking for a rare or not-so-popular data shared by only a few other peers, then it is highly unlikely that search will be successful.

- Since there is no correlation between a peer and the content managed by it, there is no guarantee that flooding will find a peer that has the desired data.
- Flooding also causes a high amount of signaling traffic in the network and hence such networks typically have very poor search efficiency.
- Most of the popular P2P networks such as Napster, Gnutella and KaZaA are unstructured.
- Rely on randomized algorithms for constructing an overlay network.
- Each node maintains a list of neighbors constructed in a more or less random way.
- Data items are assumed to be randomly placed on nodes.

Goal - construct an overlay network that resembles a random graph. (survey)

- Each node maintains a list of c neighbors, where each of these neighbors represents a randomly chosen live node from the current set of nodes.
- The list of neighbors is referred to as a partial view.
- Many ways to construct a partial view. (Jelasity et al. 2004).

Superpeers (overview)

Network nodes that maintaining an index of node or acting as a broker for nodes are generally referred to as superpeers.

Unstructured peer-to-peer systems - locating relevant data items can become problematic as the network grows.

- no deterministic way of routing a lookup request to a specific data item -> only technique a node can resort to is flooding the request.
- flooding can be dammed-> alternative -> use special nodes that maintain an index of data items.

Other situations in which abandoning the symmetric nature of peer-to-peer systems is sensible.

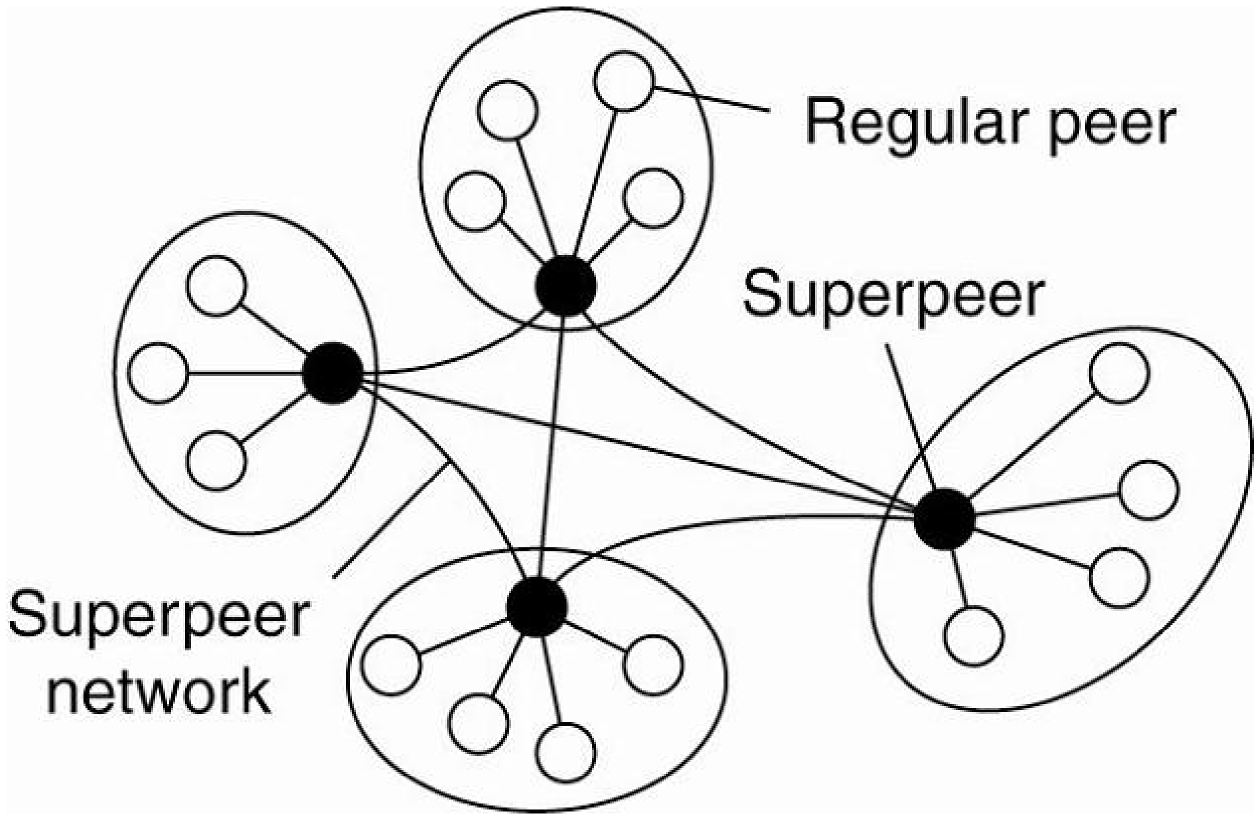
Example: collaboration of nodes that offer resources to each other.

§ in a collaborative content delivery network (CDN), nodes may offer storage for hosting copies of Web pages allowing Web clients to access pages nearby, and thus to access them quickly.

§ A node P may need to seek for resources in a specific part of the network.

§ Making use of a broker that collects resource usage for a number of nodes that are in each other's proximity will allow to quickly select a node with sufficient resources.

A hierarchical organization of nodes into a superpeer network.



- The client-superpeer relation is fixed n many cases: whenever a regular peer joins the network, it attaches to one of the superpeers and remains attached until it leaves the network.
- Expected that superpeers are long-lived processes with a high availability.
- To compensate for potential unstable behavior of a superpeer, backup schemes can be deployed, such as pairing every superpeer with another one and requiring clients to attach to both.

Hybrid Architectures

E**dge-Server Systems**

· Deployed on the Internet where servers are placed "at the edge" of the network.

§ purpose is to serve content, possibly after applying filtering and transcoding functions

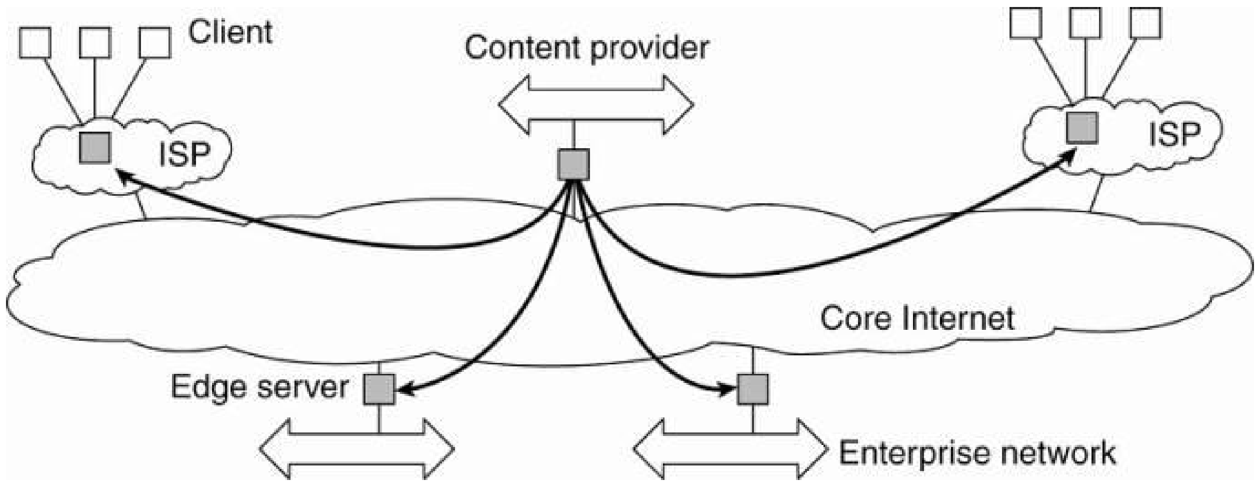
§ a collection of edge servers can be used to optimize content and application distribution

· This edge is formed by the boundary between enterprise networks and the actual Internet

§ e.g, an Internet Service Provider (ISP).

§ e.g. end users at home connect to the Internet through their ISP, the ISP can be considered as residing at the edge of the Internet.

○ Viewing the Internet as consisting of a collection of edge servers.



Basic model - one edge server acts as an origin server from which all content originates.

That server can use other edge servers for replicating Web pages and such (Leff et al., 2004; Nayate et al., 2004).

Collaborative Distributed Systems

○ Hybrid architectures are deployed in collaborative distributed systems.

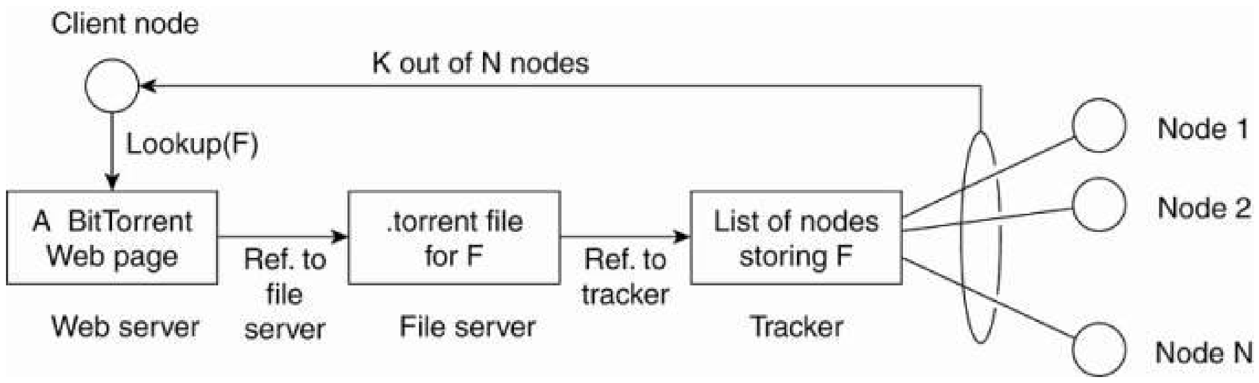
○ Two step process:

1. Join system using a traditional client-server scheme.

2. Once a node has joined the system - use a fully decentralized scheme for collaboration.

Example: the BitTorrent file-sharing system (Cohen, 2003).

- BitTorrent is a peer-to-peer file downloading system.
- An end user downloads chunks of a file from other users until the downloaded chunks can be assembled together yielding the complete file.
- BitTorrent combines centralized with decentralized solutions.
- The principal working of BitTorrent



- Design goal - ensure collaboration.

1. Most file-sharing systems - participants download files only (Adar and Huberman, 2000; Saroiu et al., 2003; and Yang et al., 2005).

File download process:

1. Access a global directory of one of a few well-known Web sites.

○ Directory contains references to what are called .torrent files.

○ A .torrent file contains the information that is needed to download a specific file.

○ It refers to a tracker - a server that keeps an accurate account of active nodes that have (chunks) of the requested file.

○ An active node is one that is currently downloading another file.

○ Many different trackers - but only a single tracker per file (or collection of files).

2. Once the nodes have been identified from where chunks can be downloaded - the downloading node becomes active.

- This node will be forced to help others by providing chunks of the file it is downloading that others do not yet have.

- Enforcement comes from a very simple rule: if node P notices that node Q is downloading more than it is uploading, P can decide to decrease the rate at which it sends data to Q.
- This scheme works well provided P has something to download from Q.
- For this reason, nodes are often supplied with references to many other nodes putting them in a better position to trade data.

System bottleneck is formed by the trackers.

Example: the Globule (paper) collaborative content distribution network.

- Globule strongly resembles the edge-server architecture.
- Instead of edge servers, end users (but also organizations) voluntarily provide enhanced Web servers that are capable of collaborating in the replication of Web pages.
- Each such server has the following components:

1. A component that can redirect client requests to other servers.

2. A component for analyzing access patterns.

3. A component for managing the replication of Web pages.

Architectures Versus Middleware

- Middleware forms a layer between applications and distributed platforms
- Provide a degree of distribution transparency,hiding the distribution of data, processing, and control from applications.
- Where middleware fits in?

Middleware systems follow a specific architectural style

- Object-based architectural style - CORBA
- Event-based architectural style - TIB/Rendezvous

Problems:

- Molding middleware molded to a specific architectural style makes designing applications simpler BUT the middleware may no longer be optimal for what an application developer had in mind.

- Middleware is meant to provide distribution transparency, BUT specific solutions should be adaptable to application requirements.

Solutions:

Good: Make several versions of a middleware system, where each version is tailored to a specific class of applications.

Better: Make middleware systems that are easy to configure, adapt, and customize as needed by an application.

Results:

- Systems are now developed with a stricter separation between policies and mechanisms.
- Led to mechanisms by which the behavior of middleware can be modified (Sadjadi and McKinley, 2003).
- Commonly followed approach:

Interceptors

An interceptor is a software construct that will break the usual flow of control and allow other (application specific) code to be executed.

Example: consider interception as supported in many object-based distributed systems.

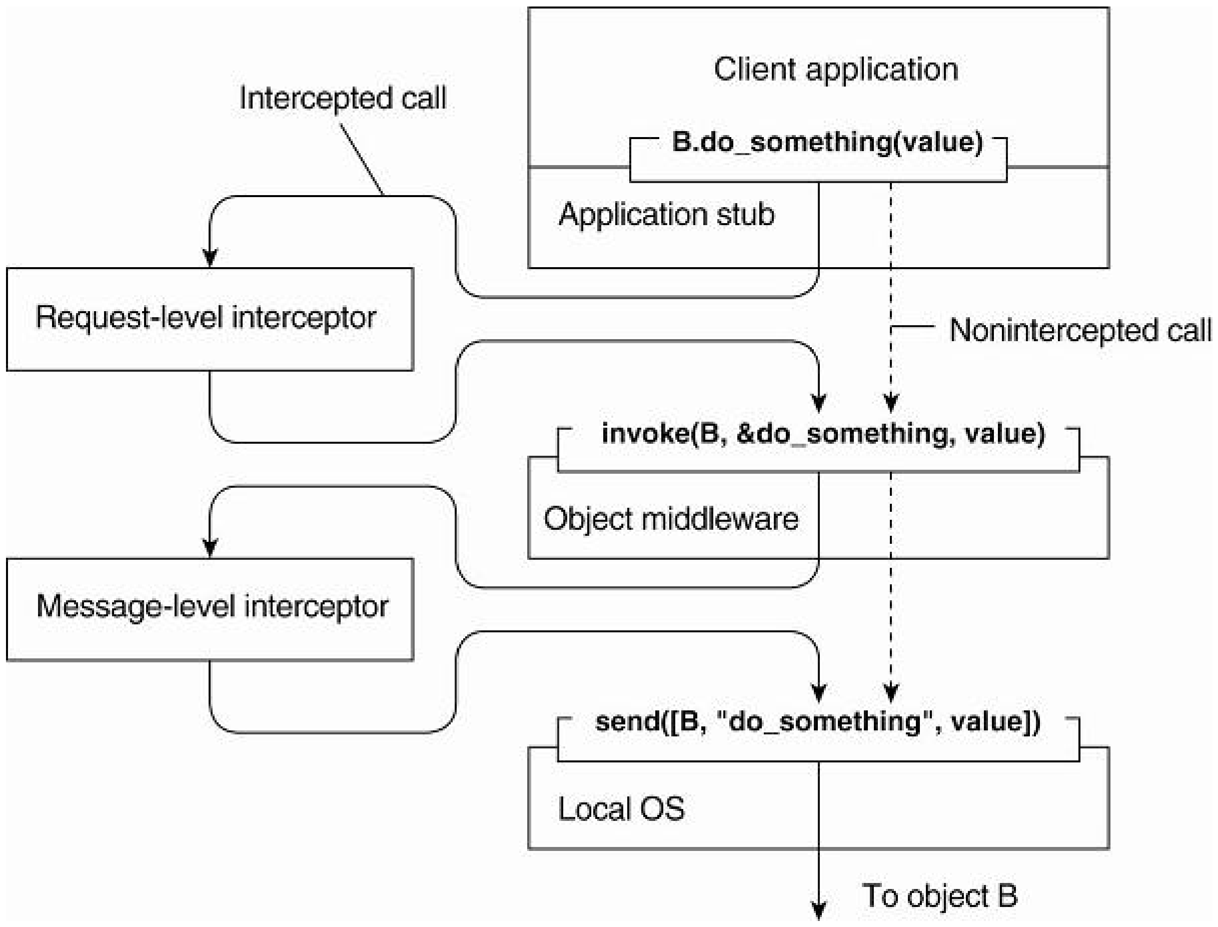
- An object A can call a method that belongs to an object B, while the latter resides on a different machine than A.
- This remote-object invocation is carried out in 3 steps:

1. Object A is offered a local interface that is exactly the same as the interface offered by object B. A simply calls the method available in that interface.

2. The call by A is transformed into a generic object invocation, made possible through a general object-invocation interface offered by the middleware at the machine where A resides.

3. Finally, the generic object invocation is transformed into a message that is sent through the transport-level network interface as offered by A's local operating system.

- Using interceptors to handle remote-object invocations.



Above Figure:

1. After the first step, the call B.do_something(value) is transformed into a generic call such as invoke(B, &do_something, value) with a reference to B's method and the parameters that go along with the call.
2. Assume that object B is replicated.
- Here, each replica should be invoked.
3. Interception helps here the request-level interceptor will call invoke(B, &do_something, value) for each of the replicas.
- Object A need not be aware of the replication of B
- The object middleware need not have special components that deal with this replicated call.
- Only the request-level interceptor, which may be added to the middleware needs to know about B's replication.

- A call to a remote object will have to be sent over the network.

- The messaging interface as offered by the local operating system will need to be invoked.
- At that level, a message-level interceptor may assist in transferring the invocation to the target object.

○**Example**:

§ imagine that the parameter value actually corresponds to a huge array of data.

§ may be wise to fragment the data into smaller parts to have it assembled again at the destination.

§ the middleware need not be aware of this fragmentation; the lower-level interceptor will transparently handle the rest of the communication with the local operating system.

General Approaches to Adaptive Software

§ Environment in which distributed applications are executed changes continuously

§ Changes include:

- mobility
- variance in the quality-of-service of networks
- failing hardware
- battery drainage
- etc.

Adaptive software for middleware

Three basic techniques to come to software adaptation and open research area (McKinley et al. 2004):

1. Separation of concerns

§ separate the parts that implement functionality from those that take care of other things (known as extra functionalities) such as reliability, performance, security, etc

§ cannot easily separate these extra functionalities by means of modularization

§ aspect-oriented software development used to address separation of concerns (Kiczales et al. 1997)

2. Computational reflection

§ the ability of a program to inspect itself and, if necessary, adapt its behavior (Kon et al., 2002).

3. Component-based design

§ Supports adaptation through composition.

§ A system may either be configured statically at design time, or dynamically at runtime.

○ The latter requires support for late binding, a technique that has been successfully applied in programming language environments, but also for operating systems where modules can be loaded and unloaded at will.

2.4. Self-Management in Distributed Systems

Must organize the components of a distributed system such that monitoring and adjustments can be done

Organize distributed systems as high-level feedback-control systems to allow automatic adaptations to changes:

§ Autonomic computing (Kephart, 2003)

§ Self-star systems - indicates the variety by which automatic adaptations are being captured: self-managing, self-healing, self-configuring, self-optimizing, etc.

The Feedback Control Model

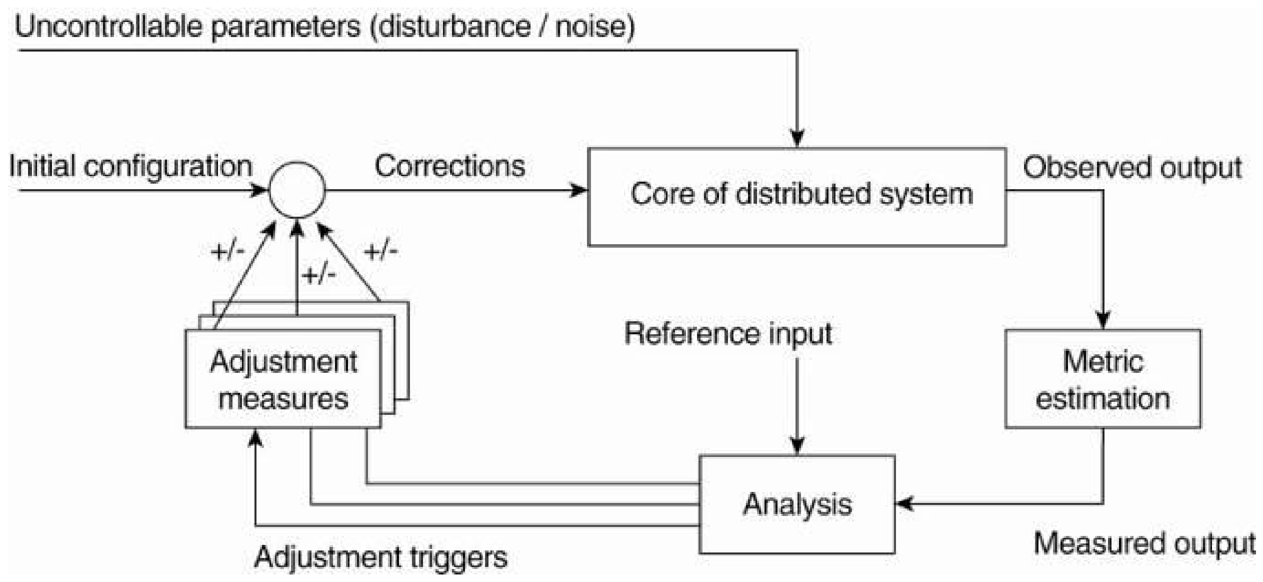
§ Adaptations take place by means of one or more feedback control loops.

§ Systems that are organized by means of such loops are referred to as feedback control systems.

§ Feedback control has since long been applied in various engineering fields, and its mathematical foundations are gradually also finding their way in computing systems (Diao et al., 2005).

§ For self-managing systems, the architectural issues are initially the most interesting.

§ The basic idea behind this organization is :



Three elements that form the feedback control loop:

1. The system itself needs to be monitored, which requires that various aspects of the system need to be measured.

2. Another part of the feedback control loop analyzes the measurements and compares these to reference values. This feedback analysis component forms the heart of the control loop, as it will contain the algorithms that decide on possible adaptations.

3. The last group of components consist of various mechanisms to directly influence the behavior of the system.

· There can be many different mechanisms: placing replicas, changing scheduling priorities, switching services, moving data for reasons of availability, redirecting requests to different servers, etc.

· The analysis component will need to be aware of these mechanisms and their (expected) effect on system behavior.

Example: Systems Monitoring with Astrolabe

- Astrolabe (Van Renesse et al., 2003): a system that can support general monitoring of very large distributed systems.

- Astrolabe is a general tool for observing systems behavior.
- Its output can be used to feed into an analysis component for deciding on corrective actions.

- Astrolabe organizes a large collection of hosts into a hierarchy of zones.

§ The lowest-level zones consist of a single host, which are subsequently grouped into zones of increasing size.

§ The top-level zone covers all hosts.

o Every host runs an Astrolabe process, called an agent, that collects information on the zones in which that host is contained.

§ The agent communicates with other agents with the aim to spread zone information across the entire system.

o Each host maintains a set of attributes for collecting local information.

§ e.g, a host may keep track of specific files it stores, its resource usage, etc.

· Example: Three hosts: A, B, and C grouped into a zone.

Each machine keeps track of its IP address, CPU load, available free memory, and the number of active processes.

§ Each of these attributes can be directly written using local information from each host.

§ At the zone level, only aggregated information can be collected, such as the average CPU load, or the average number of active processes.

o Data collection and information aggregation in Astrolabe.

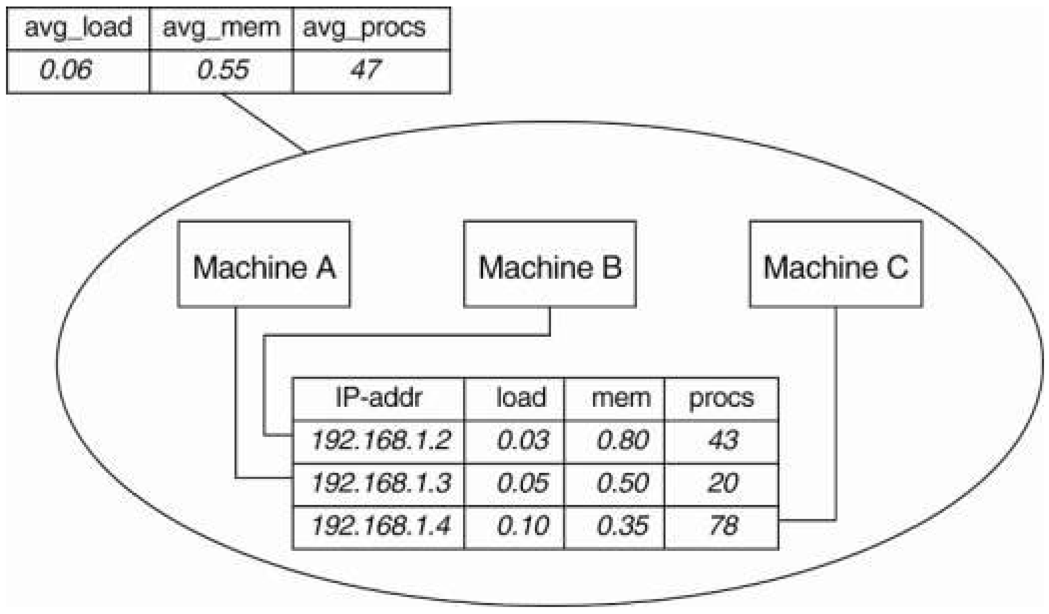


Figure shows how the information as gathered by each machine can be viewed as a record in a database

§ these records jointly form a relation (table).

§ Astrolabe views all the collected data as tables.

· Aggregated information is obtained by programmable aggregation functions, which are very similar to functions available in the relational database language SQL.

e.g., assuming that the host information from above is maintained in a local table called *hostinfo*, we could collect the average number of processes for the zone containing machines A, B, and C, through the simple SQL query:

```
SELECT AVG(procs) AS avg_procs FROM hostinfo
```

· Queries are continuously evaluated by each agent running on each host.

· An agent running on a host is responsible for computing parts of the tables of its associated zones.

· Records for which it holds no computational responsibility are occasionally sent to it through a simple, yet effective exchange procedure known as gossiping.

· An agent will pass computed results to other agents as well.

· All agents that needed to assist in obtaining some aggregated information will see the same result (provided that no changes occur in the meantime).

Example: **Differentiating Replication Strategies in Globule**

Globule relies on end-user servers being placed in the Internet, and that these servers collaborate to optimize performance through replication of Web pages.

○ Each origin server (i.e., the server responsible for handling updates of a specific Web site), keeps track of access patterns on a per-page basis.

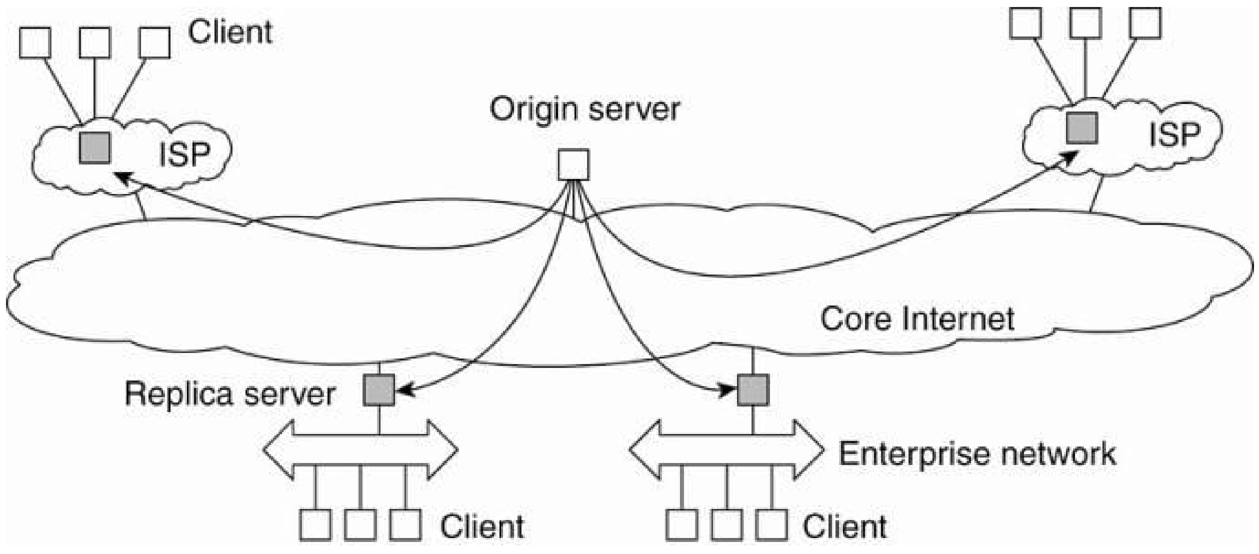
○ Access patterns are expressed as read and write operations for a page, each operation being timestamped and logged by the origin server for that page.

Globule assumes that the Internet can be viewed as an edge-server system.

○ It assumes that requests can always be passed through an appropriate edge server. T

○ Model allows an origin server to see what would have happened if it had placed a replica on a specific edge server.

○ The edge-server model assumed by Globule.



When an origin server receives a request for a page:

o it records the IP address from where the request originated

o looks up the ISP or enterprise network associated with that request using the WHOIS Internet service (Deutsch et al., 1995).

o It then looks for the nearest existing replica server that could act as edge server for that client and computes the latency to that server along with the maximal bandwidth.

O Globule assumes that the latency between the replica server and the requesting user machine is negligible, and likewise that bandwidth between the two is plentiful.

O Once enough requests for a page have been collected, the origin server performs a simple "what-if analysis."

o evaluates several replication policies, where a policy describes where a specific page is replicated to, and how that page is kept consistent.

§ Each replication policy incurs a cost that can be expressed as a simple linear function:

cost=(w1 x m1)+(w2 x m2)+ . . .+(wn x mn)

where mk denotes a performance metric and wk is the weight indicating how important that metric is.

O Typical performance metrics are the aggregated delays between a client and a replica server when returning copies of Web pages, the total consumed bandwidth between the origin server and a replica server for keeping a replica consistent, and the number of stale copies that are (allowed to be) returned to a client (Pierre et al., 2002).

o An origin server regularly evaluates a few tens of replication polices using a trace-driven simulation, for each Web page separately.

O A best policy is selected and subsequently enforced.

Example: Automatic Component Repair Management in Jade

O Jade system detects component failures in clusters of computers built using a component-based approach. (Bouchenak et al., 2005)

o Jade is built on the Fractal component model, a Java implementation of a framework that allows components to be added and removed at runtime (Bruneton et al., 2004).

o A component in Fractal can have two types of interfaces:

o A server interface is used to call methods that are implemented by that component.

O A client interface is used by a component to call other components.

O Components are connected to each other by binding interfaces.

O e.g, a client interface of component C1 can be bound to the server interface of component C2.

O A primitive binding means that a call to a client interface directly leads to calling the bounded server interface.

O Jade uses the notion of a repair management domain.

O A domain consists of a number of nodes, where each node represents a server along with the components that are executed by that server.

O A separate node manager is responsible for adding and removing nodes from the domain.

§ The node manager may be replicated for assuring high availability.

O Each node is equipped with failure detectors, which monitor the health of a node or one of its components and report any failures to the node manager.

§ These detectors consider exceptional changes in the state of component, the usage of resources, and the actual failure of a component. Note that the latter may actually mean that a machine has crashed.

O When a failure has been detected, a repair procedure is started.

O The procedure is driven by a repair policy, partly executed by the node manager.

O Policies are stated explicitly and are carried out depending on the detected failure.

§ e.g, suppose a node failure has been detected - the repair policy may prescribe that the following steps are to be carried out:

1. Terminate every binding between a component on a nonfaulty node, and a component on the node that just failed.
2. Request the node manager to start and add a new node to the domain.
3. Configure the new node with exactly the same components as those on the crashed node.
4. Re-establish all the bindings that were previously terminated.

O Jade is an example of self-management: upon the detection of a failure, a repair policy is automatically executed to bring the system as a whole into a state in which it was before the crash.

- Being a component-based system, this automatic repair requires specific support to allow components to be added and removed at runtime.
- In general, turning legacy applications into self-managing systems is not possible.