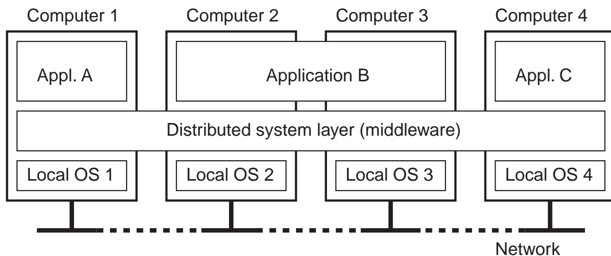
Distributed System: Definition

A distributed system is a piece of software that ensures that:

> *a collection of independent computers appears to its users as a single coherent system*

Two aspects: (1) independent computers and
(2) single system ⇒ middleware.



Scale in Distributed Systems

Observation

Many developers of modern distributed system easily use the adjective “scalable” without making clear **why** their system actually scales.

Scalability

At least three components:

- Number of users and/or processes (size scalability)
- Maximum distance between nodes (geographical scalability)
- Number of administrative domains (administrative scalability)

Observation

Most systems account only, to a certain extent, for size scalability. The (non)solution: powerful servers. Today, the challenge lies in geographical and administrative scalability.

Scale in Distributed Systems

Observation

Many developers of modern distributed system easily use the adjective "scalable" without making clear **why** their system actually scales.

Scalability

At least three components:

- Number of users and/or processes (size scalability)
- Maximum distance between nodes (geographical scalability)
- Number of administrative domains (administrative scalability)

Observation

Most systems account only, to a certain extent, for size scalability. The (non)solution: powerful servers. Today, the challenge lies in geographical and administrative scalability.

Scale in Distributed Systems

Observation

Many developers of modern distributed system easily use the adjective “scalable” without making clear **why** their system actually scales.

Scalability

At least three components:

- Number of users and/or processes (size scalability)
- Maximum distance between nodes (geographical scalability)
- Number of administrative domains (administrative scalability)

Observation

Most systems account only, to a certain extent, for size scalability. The (non)solution: powerful servers. Today, the challenge lies in geographical and administrative scalability.

Techniques for Scaling

Distribution

Partition data and computations across multiple machines:

- Move computations to clients (Java applets)
- Decentralized naming services (DNS)
- Decentralized information systems (WWW)

Techniques for Scaling

Replication/caching

Make copies of data available at different machines:

- Replicated file servers and databases
- Mirrored Web sites
- Web caches (in browsers and proxies)
- File caching (at server and client)

Scaling – The Problem

Observation

Applying scaling techniques is easy, except for one thing:

- Having multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
- Always keeping copies consistent and in a general way requires global synchronization on each modification.
- Global synchronization precludes large-scale solutions.

Observation

If we can tolerate inconsistencies, we may reduce the need for global synchronization, but tolerating inconsistencies is application dependent.

Scaling – The Problem

Observation

Applying scaling techniques is easy, except for one thing:

- Having multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
- Always keeping copies consistent and in a general way requires global synchronization on each modification.
- Global synchronization precludes large-scale solutions.

Observation

If we can tolerate inconsistencies, we may reduce the need for global synchronization, but tolerating inconsistencies is application dependent.

Scaling – The Problem

Observation

Applying scaling techniques is easy, except for one thing:

- Having multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
- Always keeping copies consistent and in a general way requires global synchronization on each modification.
- Global synchronization precludes large-scale solutions.

Observation

If we can tolerate inconsistencies, we may reduce the need for global synchronization, but tolerating inconsistencies is application dependent.

Scaling – The Problem

Observation

Applying scaling techniques is easy, except for one thing:

- Having multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
- Always keeping copies consistent and in a general way requires global synchronization on each modification.
- Global synchronization precludes large-scale solutions.

Observation

If we can tolerate inconsistencies, we may reduce the need for global synchronization, but tolerating inconsistencies is application dependent.

Scaling – The Problem

Observation

Applying scaling techniques is easy, except for one thing:

- Having multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
- Always keeping copies consistent and in a general way requires global synchronization on each modification.
- Global synchronization precludes large-scale solutions.

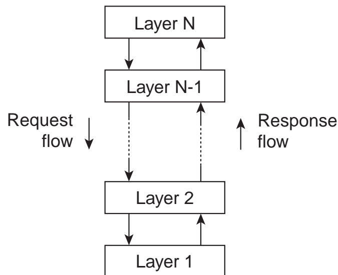
Observation

If we can tolerate inconsistencies, we may reduce the need for global synchronization, but tolerating inconsistencies is application dependent.

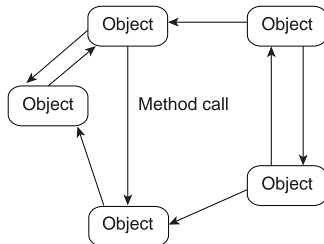
Architectural styles

Basic idea

Organize into logically different components, and distribute those components over the various machines.



(a)



(b)

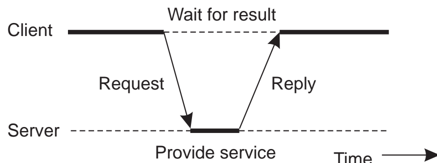
(a) Layered style is used for client-server system
(b) Object-based style for distributed object systems.

Centralized Architectures

Basic Client–Server Model

Characteristics:

- There are processes offering services (servers)
- There are processes that use services (clients)
- Clients and servers can be on different machines
- Clients follow request/reply model wrt to using services



Application Layering

Traditional three-layered view

- User-interface layer contains units for an application's user interface
- Processing layer contains the functions of an application, i.e. without specific data
- Data layer contains the data that a client wants to manipulate through the application components

Observation

This layering is found in many distributed information systems, using traditional database technology and accompanying applications.

Application Layering

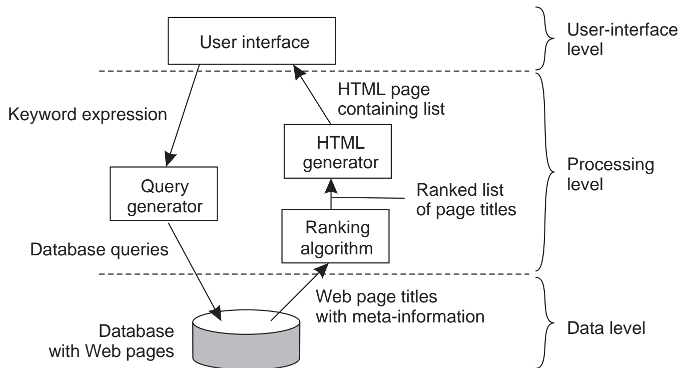
Traditional three-layered view

- User-interface layer contains units for an application's user interface
- Processing layer contains the functions of an application, i.e. without specific data
- Data layer contains the data that a client wants to manipulate through the application components

Observation

This layering is found in many distributed information systems, using traditional database technology and accompanying applications.

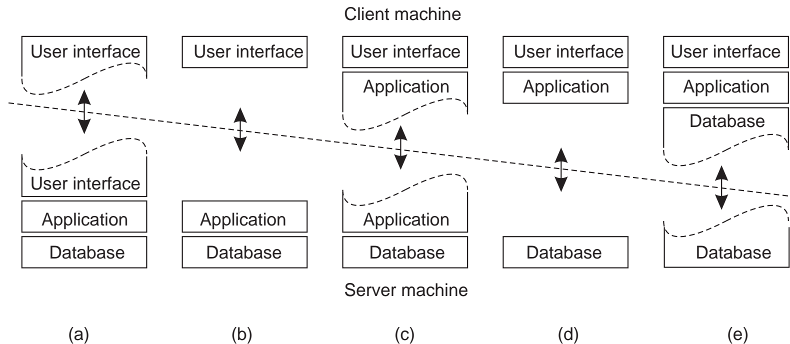
Application Layering



Multi-Tiered Architectures

Single-tiered: dumb terminal/mainframe configuration
Two-tiered: client/single server configuration
Three-tiered: each layer on separate machine

Traditional two-tiered configurations:

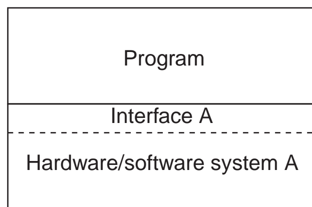


Virtualization

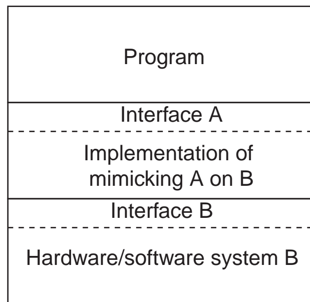
Observation

Virtualization is becoming increasingly important:

- Hardware changes faster than software
- Ease of portability and code migration
- Isolation of failing or attacked components



(a)

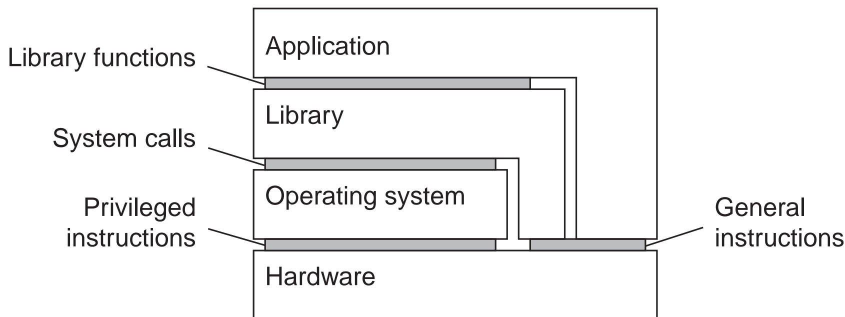


(b)

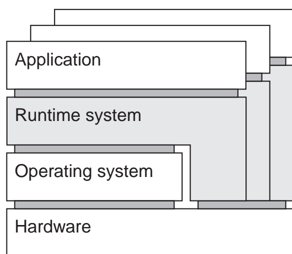
Architecture of VMs

Observation

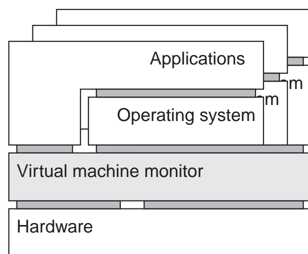
Virtualization can take place at very different levels, strongly depending on the interfaces as offered by various systems components:



Process VMs versus VM Monitors



(a)



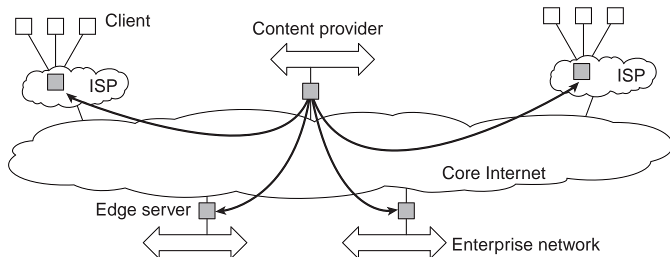
(b)

- Process VM: A program is compiled to intermediate (portable) code, which is then executed by a runtime system (Example: Java VM).
- VM Monitor: A separate software layer mimics the instruction set of hardware ⇒ a complete operating system and its applications can be supported (Example: VMware, VirtualBox).

Hybrid Architectures: Client-server combined with P2P

Example

Edge-server architectures, which are often used for Content Delivery Networks

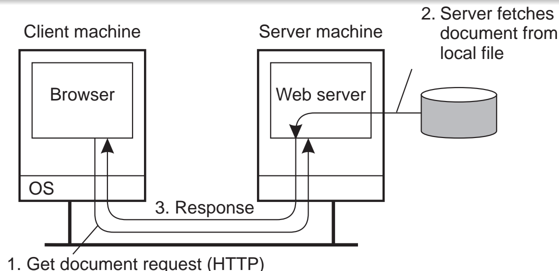


Distributed Web-based systems

Essence

The WWW is a huge client-server system with millions of servers; each server hosting thousands of hyperlinked documents.

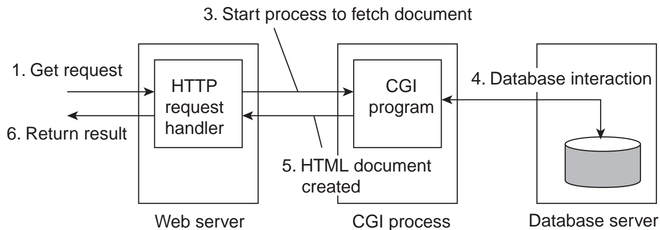
- Documents are often represented in text (plain text, HTML, XML)
- Alternative types: images, audio, video, applications (PDF, PS)
- Documents may contain scripts, executed by client-side software



Multi-tiered architectures

Observation

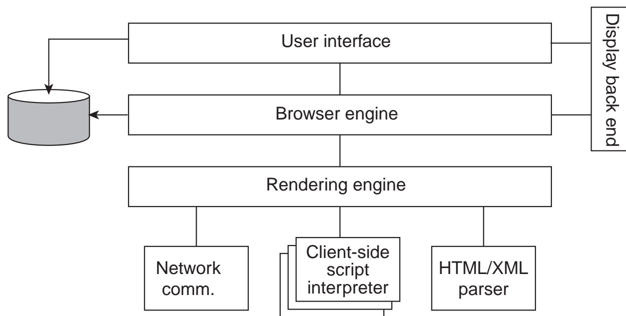
Web sites were soon organised into three tiers.



Clients: Web browsers

Observation

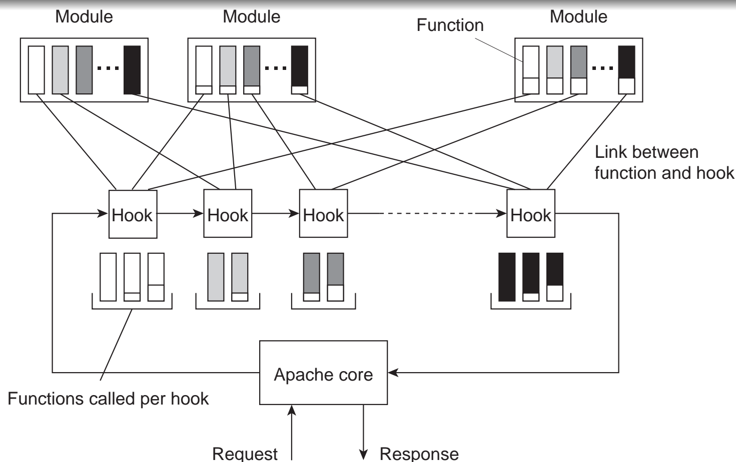
Browsers form the Web's most important client-side sofware. They used to be simple, but that is long ago.



Apache Web server

Observation: More than 52% of all 185 million Web sites are Apache.

The server is internally organised more or less according to the steps needed to process an HTTP request.



Threads and Distributed Systems

Multithreaded Web client

Hiding network latencies:

- Web browser scans an incoming HTML page, and finds that more files need to be fetched.
- Each file is fetched by a separate thread, each doing a (blocking) HTTP request.
- As files come in, the browser displays them.

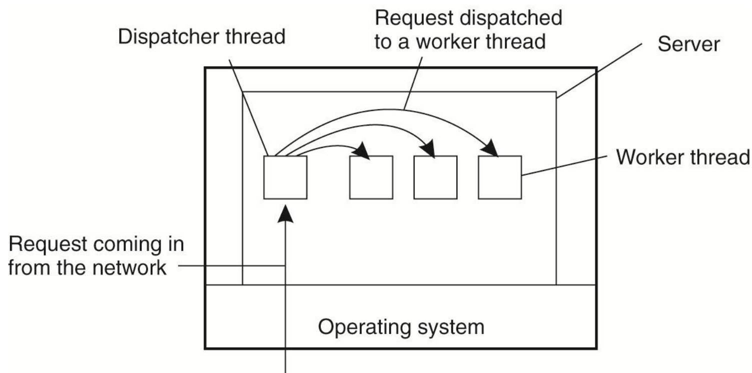
Multiple request-response calls to other machines (RPC)

- A client does several calls at the same time, each one by a different thread.
- It then waits until all results have been returned.
- Note: if calls are to different servers, we may have a linear speed-up.

Threads and Distributed Systems

Multithreaded Servers

Although there are important benefits to multithreaded clients, the main benefits of multithreading in distributed systems are on the server side.



Threads and Distributed Systems

Improve performance

- Starting a thread is much cheaper than starting a new process.
- Having a single-threaded server prohibits simple scale-up to a multiprocessor system.
- As with clients: hide network latency by reacting to next request while previous one is being replied.

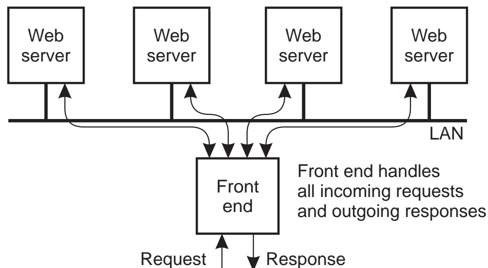
Better structure

- Most servers have high I/O demands. Using simple, well-understood blocking calls simplifies the overall structure.
- Multithreaded programs tend to be smaller and easier to understand due to simplified flow of control.

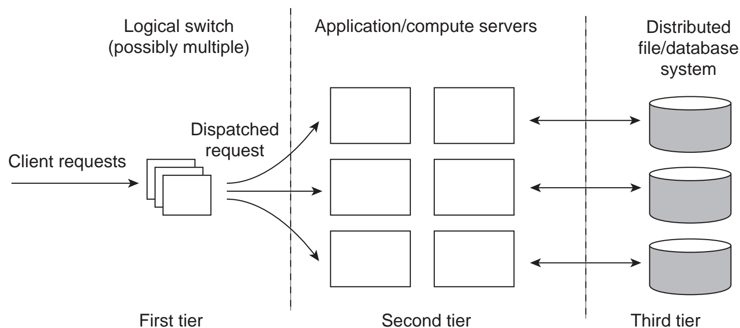
Server clusters

Essence

To improve performance and availability, WWW servers are often clustered in a way that is transparent to clients.



Server clusters: three different tiers



Crucial element

The first tier is generally responsible for passing requests to an appropriate server.

Server clusters

Problem

The front end may easily get overloaded, so that special measures need to be taken.

- Transport-layer switching: Front end simply passes the TCP request to one of the servers, taking some performance metric into account.
- Content-aware distribution: Front end reads the content of the HTTP request and then selects the best server.

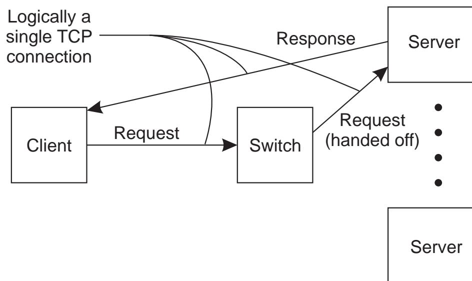
Request Handling

Observation

Having the first tier handle all communication from/to the cluster may lead to a bottleneck.

Solution

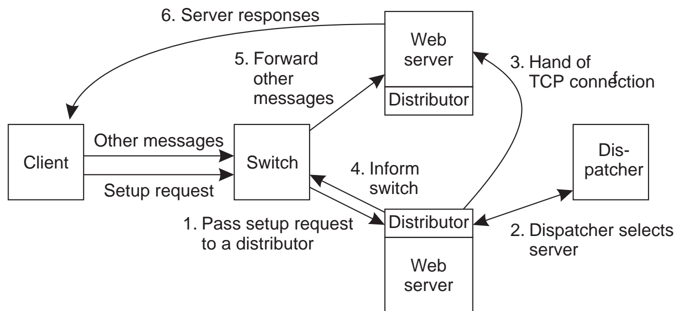
Various, but one popular one is TCP-handoff



Server Clusters

Question

Why can content-aware distribution be so much better?



Naming Entities

- Names, identifiers, and addresses
- Name resolution
- Name space implementation

Naming

Essence

Names are used to denote entities in a distributed system. To operate on an entity, we need to access it at an access point. Access points are entities that are named by means of an address.

Note

A location-independent name for an entity E, is independent from the addresses of the access points offered by E.

Identifiers

Pure name

A name that has no meaning at all; it is just a random string. Pure names can be used for comparison only.

Identifier

A name having the following properties:

- P1: Each identifier refers to at most one entity
- P2: Each entity is referred to by at most one identifier
- P3: An identifier always refers to the same entity (prohibits reusing an identifier)

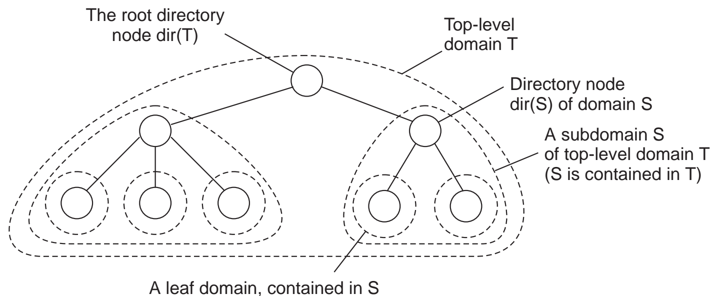
Observation

An identifier need not necessarily be a pure name, i.e., it may have content.

Hierarchical Location Services (HLS)

Basic idea

Build a large-scale search tree for which the underlying network is divided into hierarchical domains. Each domain is represented by a separate directory node.



Name resolution

Problem

To resolve a name we need a directory node. How do we actually find that (initial) node?

Closure mechanism

The mechanism to select the implicit context from which to start name resolution:

- *www.cs.vu.nl*: start at a DNS name server
- */home/steen/mbox*: start at the local NFS file server (possible recursive search)
- *0031204447784*: dial a phone number
- *130.37.24.8*: route to the VU's Web server

Question

Why are closure mechanisms always implicit?

Name resolution

Problem

To resolve a name we need a directory node. How do we actually find that (initial) node?

Closure mechanism

The mechanism to select the implicit context from which to start name resolution:

- *www.cs.vu.nl*: start at a DNS name server
- */home/steen/mbox*: start at the local NFS file server (possible recursive search)
- *0031204447784*: dial a phone number
- *130.37.24.8*: route to the VU's Web server

Question

Why are closure mechanisms always implicit?

Name resolution

Problem

To resolve a name we need a directory node. How do we actually find that (initial) node?

Closure mechanism

The mechanism to select the implicit context from which to start name resolution:

- *www.cs.vu.nl*: start at a DNS name server
- */home/steen/mbox*: start at the local NFS file server (possible recursive search)
- *0031204447784*: dial a phone number
- *130.37.24.8*: route to the VU's Web server

Question

Why are closure mechanisms always implicit?

Name linking

Hard link

What we have described so far as a path name: a name that is resolved by following a specific path in a naming graph from one node to another.

Name-space implementation

Basic issue

Distribute the name resolution process as well as name space management across multiple machines, by distributing nodes of the naming graph.

Distinguish three levels

- Global level: Consists of the high-level directory nodes. Main aspect is that these directory nodes have to be jointly managed by different administrations
- Administrational level: Contains mid-level directory nodes that can be grouped in such a way that each group can be assigned to a separate administration.
- Managerial level: Consists of low-level directory nodes within a single administration. Main issue is effectively mapping directory nodes to local name servers.

Name-space implementation

Basic issue

Distribute the name resolution process as well as name space management across multiple machines, by distributing nodes of the naming graph.

Distinguish three levels

- Global level: Consists of the high-level directory nodes. Main aspect is that these directory nodes have to be jointly managed by different administrations
- Administrational level: Contains mid-level directory nodes that can be grouped in such a way that each group can be assigned to a separate administration.
- Managerial level: Consists of low-level directory nodes within a single administration. Main issue is effectively mapping directory nodes to local name servers.

Name-space implementation

Basic issue

Distribute the name resolution process as well as name space management across multiple machines, by distributing nodes of the naming graph.

Distinguish three levels

- Global level: Consists of the high-level directory nodes. Main aspect is that these directory nodes have to be jointly managed by different administrations
- Administrational level: Contains mid-level directory nodes that can be grouped in such a way that each group can be assigned to a separate administration.
- Managerial level: Consists of low-level directory nodes within a single administration. Main issue is effectively mapping directory nodes to local name servers.

Name-space implementation

Basic issue

Distribute the name resolution process as well as name space management across multiple machines, by distributing nodes of the naming graph.

Distinguish three levels

- Global level: Consists of the high-level directory nodes. Main aspect is that these directory nodes have to be jointly managed by different administrations
- Administrational level: Contains mid-level directory nodes that can be grouped in such a way that each group can be assigned to a separate administration.
- Managerial level: Consists of low-level directory nodes within a single administration. Main issue is effectively mapping directory nodes to local name servers.

Name-space implementation

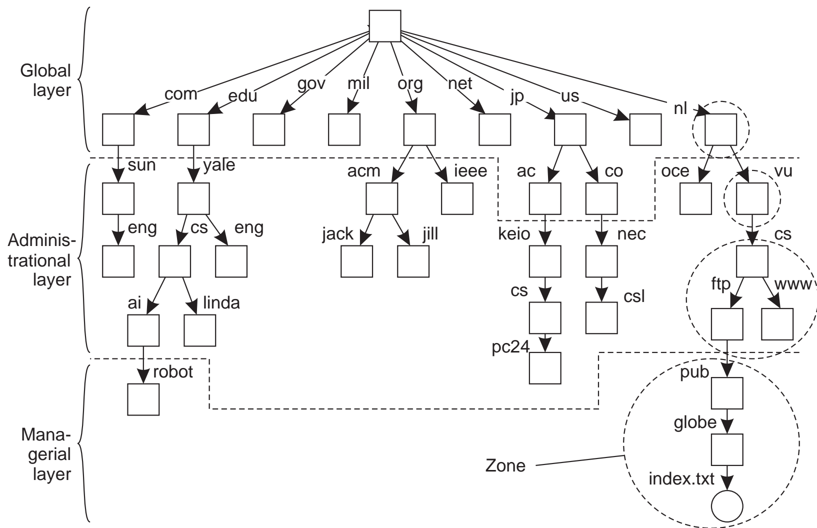
Basic issue

Distribute the name resolution process as well as name space management across multiple machines, by distributing nodes of the naming graph.

Distinguish three levels

- Global level: Consists of the high-level directory nodes. Main aspect is that these directory nodes have to be jointly managed by different administrations
- Administrational level: Contains mid-level directory nodes that can be grouped in such a way that each group can be assigned to a separate administration.
- Managerial level: Consists of low-level directory nodes within a single administration. Main issue is effectively mapping directory nodes to local name servers.

Name-space implementation



Name-space implementation

Item	Global	Administrational	Managerial
1	Worldwide	Organization	Department
2	Few	Many	Vast numbers
3	Seconds	Milliseconds	Immediate
4	Lazy	Immediate	Immediate
5	Many	None or few	None
6	Yes	Yes	Sometimes

1: Geographical scale	4: Update propagation
2: # Nodes	5: # Replicas
3: Responsiveness	6: Client-side caching?

Replication of records

Definition

Replicated at level i – record is replicated to all nodes with i matching prefixes. Note: # hops for looking up record at level i is generally i.

Observation

Let x_i denote the fraction of most popular DNS names of which the records should be replicated at level i, then:

$$x_i = \left[\frac{d^i(\log N - C)}{1 + d + \cdots + d^{\log N - 1}} \right]^{1/(1-\alpha)}$$

with N is the total number of nodes, $d = b^{(1-\alpha)/\alpha}$ and $\alpha \approx 1$, assuming that popularity follows a Zipf distribution:
The frequency of the n-th ranked item is proportional to $1/n^\alpha$

Replication of records

Meaning

If you want to reach an average of $C = 1$ hops when looking up a DNS record, then with $b = 4$, $\alpha = 0.9$, $N = 10,000$ and $1,000,000$ records that

61 most popular records should be replicated at level 0
284 next most popular records at level 1
1323 next most popular records at level 2
6177 next most popular records at level 3
28826 next most popular records at level 4
134505 next most popular records at level 5
the rest should not be replicated

Consistency for mobile users

Example

Consider a distributed database to which you have access through your notebook. Assume your notebook acts as a front end to the database.

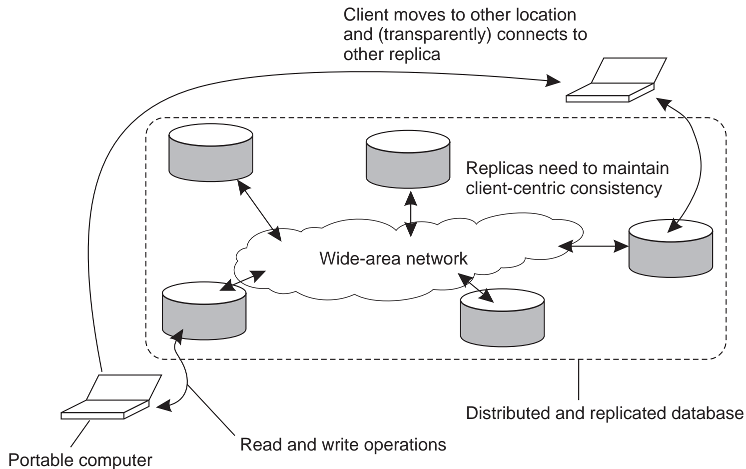
- At location *A* you access the database doing reads and updates.
- At location *B* you continue your work, but unless you access the same server as the one at location *A*, you may detect inconsistencies:
  - your updates at *A* may not have yet been propagated to *B*
  - you may be reading newer entries than the ones available at *A*
  - your updates at *B* may eventually conflict with those at *A*

Consistency for mobile users

Note

The only thing you really want is that the entries you updated and/or read at A, are in B the way you left them in A. In that case, the database will appear to be consistent to you.

Basic architecture



Distribution protocols

- Replica server placement
- Content replication and placement
- Content distribution

Replica placement

Essence

Figure out what the best K places are out of N possible locations.

- Select best location out of $N - K$ for which the average distance to clients is minimal. Then choose the next best server. (Note: The first chosen location minimizes the average distance to all clients.) Computationally expensive.
- Select the K-th largest autonomous system and place a server at the best-connected host. Computationally expensive.
- Position nodes in a d-dimensional geometric space, where distance reflects latency. Identify the K regions with highest density and place a server in every one. Computationally cheap.

Replica placement

Essence

Figure out what the best K places are out of N possible locations.

- Select best location out of $N-K$ for which the average distance to clients is minimal. Then choose the next best server. (Note: The first chosen location minimizes the average distance to all clients.) Computationally expensive.
- Select the K-th largest autonomous system and place a server at the best-connected host. Computationally expensive.
- Position nodes in a d-dimensional geometric space, where distance reflects latency. Identify the K regions with highest density and place a server in every one. Computationally cheap.

Replica placement

Essence

Figure out what the best K places are out of N possible locations.

- Select best location out of $N - K$ for which the average distance to clients is minimal. Then choose the next best server. (Note: The first chosen location minimizes the average distance to all clients.) Computationally expensive.
- Select the K-th largest autonomous system and place a server at the best-connected host. Computationally expensive.
- Position nodes in a d-dimensional geometric space, where distance reflects latency. Identify the K regions with highest density and place a server in every one. Computationally cheap.

Replica placement

Essence

Figure out what the best K places are out of N possible locations.

- Select best location out of $N-K$ for which the average distance to clients is minimal. Then choose the next best server. (Note: The first chosen location minimizes the average distance to all clients.) Computationally expensive.
- Select the K-th largest autonomous system and place a server at the best-connected host. Computationally expensive.
- Position nodes in a d-dimensional geometric space, where distance reflects latency. Identify the K regions with highest density and place a server in every one. Computationally cheap.

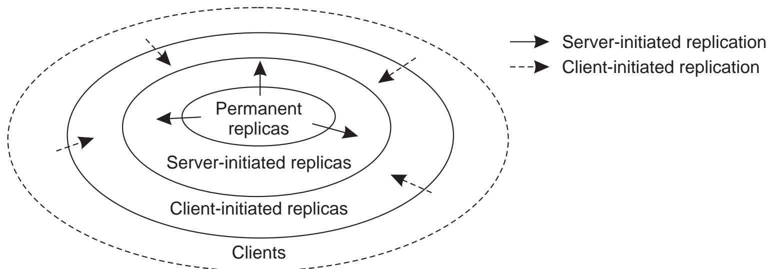
Content replication

Distinguish different processes

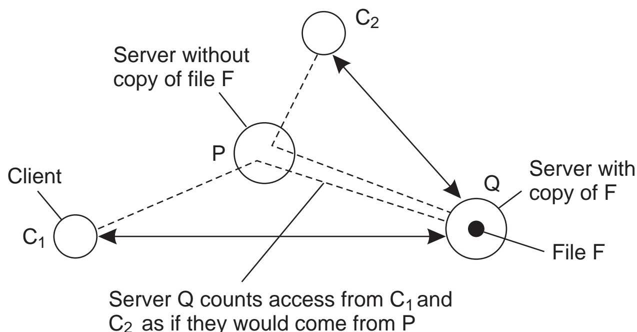
A process is capable of hosting a replica of an object or data:

- Permanent replicas: Process/machine always having a replica
- Server-initiated replica: Process that can dynamically host a replica on request of another server in the data store
- Client-initiated replica: Process that can dynamically host a replica on request of a client (client cache)

Content replication



Server-initiated replicas



- Keep track of access counts per file, aggregated by considering server closest to requesting clients
- Number of accesses drops below threshold $D \Rightarrow$ drop file
- Number of accesses exceeds threshold $R \Rightarrow$ replicate file
- Number of access between D and $R \Rightarrow$ migrate file

Content distribution

Model

Consider only a client-server combination:

- Propagate only notification/invalidation of update (often used for caches)
- Transfer data from one copy to another (distributed databases)
- Propagate the update *operation* to other copies (also called active replication)

Note

No single approach is the best, but depends highly on available bandwidth and read-to-write ratio at replicas.