Distributed Systems

(3rd Edition)

Maarten van Steen Andrew S. Tanenbaum

Chapter 05: Naming

Edited by: Hicham G. Elmongui

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 some specific properties

1. An identifier refers to at most one entity.
2. Each entity is referred to by at most one identifier.
3. An identifier always refers to the same entity (i.e., it is never reused).

Observation

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

Broadcasting

Broadcast the ID, requesting the entity to return its current address

- Can never scale beyond local-area networks
- Requires all processes to listen to incoming location requests

Address Resolution Protocol (ARP)

To find out which MAC address is associated with an IP address, broadcast the query “who has this IP address”?

Forwarding pointers

When an entity moves, it leaves behind a pointer to its next location

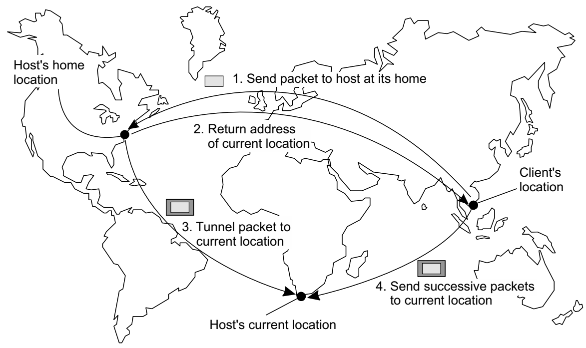
- Dereferencing can be made entirely transparent to clients by simply following the chain of pointers
- Update a client’s reference when present location is found
- Geographical scalability problems (for which separate chain reduction mechanisms are needed):
  - Long chains are not fault tolerant
  - Increased network latency at dereferencing

Home-based approaches

Single-tiered scheme: Let a home keep track of where the entity is

- Entity’s home address registered at a naming service
- The home registers the foreign address of the entity
- Client contacts the home first, and then continues with foreign location

The principle of mobile IP



Home-based approaches

Problems with home-based approaches

- Home address has to be supported for entity's lifetime
- Home address is fixed ⇒ unnecessary burden when the entity permanently moves
- Poor geographical scalability (entity may be next to client)

Note

Permanent moves may be tackled with another level of naming (DNS)

Illustrative: Chord

Consider the organization of many nodes into a logical ring

- Each node is assigned a random m-bit identifier.
- Every entity is assigned a unique m-bit key.
- Entity with key k falls under jurisdiction of node with smallest $id \geq k$ (called its successor $succ(k)$).

Nonsolution

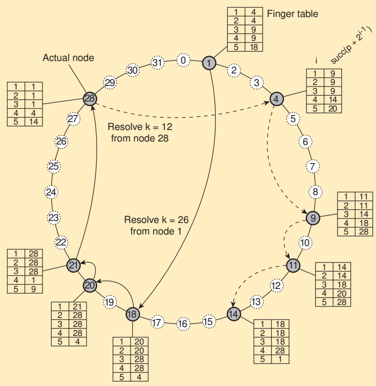
Let each node keep track of its neighbor and start linear search along the ring.

Main Issue in DHT-based Systems

To Efficiently resolve a key k to the address of $succ(k)$.

Chord lookup example

Resolving key 26 from node *1* and key *12* from node *28*

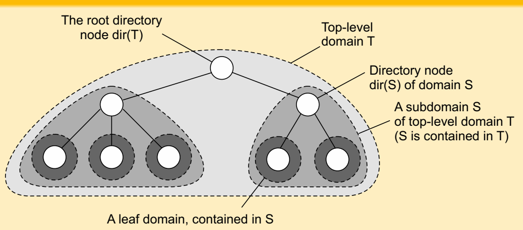


Hierarchical Location Services (HLS)

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.

Principle

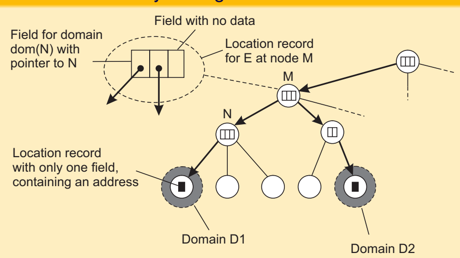


HLS: Tree organization

Invariants

- Address of entity E is stored in a leaf or intermediate node
- Intermediate nodes contain a pointer to a child if and only if the subtree rooted at the child stores an address of the entity
- The root knows about all entities

Storing information of an entity having two addresses in different leaf domains

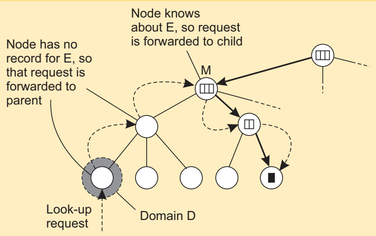


HLS: Lookup operation

Basic principles

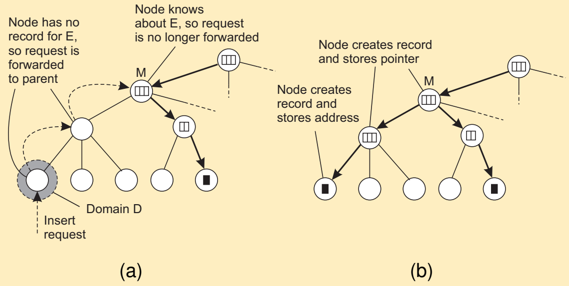
- Start lookup at local leaf node
- Node knows about $E \Rightarrow$ follow downward pointer, else go up
- Upward lookup always stops at root

Looking up a location



HLS: Insert operation

(a) An insert request is forwarded to the first node that knows about entity *E*. (b) A chain of forwarding pointers to the leaf node is created

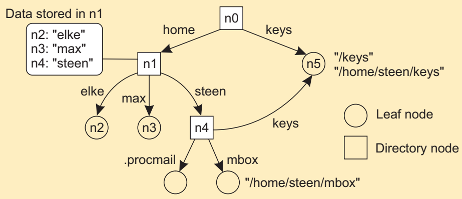


Name space

Naming graph

A graph in which a leaf node represents a (named) entity. A directory node is an entity that refers to other nodes.

A general naming graph with a single root node



Note

A directory node contains a table of *(node identifier, edge label)* pairs.

Name space

We can easily store all kinds of attributes in a node

- Type of the entity
- An identifier for that entity
- Address of the entity's location
- Nicknames
- ...

Note

Directory nodes can also have attributes, besides just storing a directory table with *(identifier, label)* pairs.

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.distributed-systems.net`: start at a DNS name server
- `/home/maarten/mbox`: start at the local NFS file server (possible recursive search)
- `0031 20 598 7784`: dial a phone number
- `77.167.55.6`: route message to a specific IP address

Note

You cannot have an explicit closure mechanism – how would you start?

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.

Soft link: Allow a node *N* to contain a name of another node

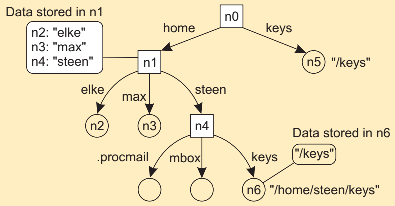
- First resolve *N*'s name (leading to *N*)
- Read the content of *N*, yielding *name*
- Name resolution continues with *name*

Observations

- The name resolution process determines that we read the content of a node, in particular, the name in the other node that we need to go to.
- One way or the other, we know where and how to start name resolution given *name*

Name linking

The concept of a symbolic link explained in a naming graph



Observation

Node *n5* has only one name

Mounting

Issue

Name resolution can also be used to merge different name spaces in a transparent way through mounting: associating a node identifier of another name space with a node in a current name space.

Terminology

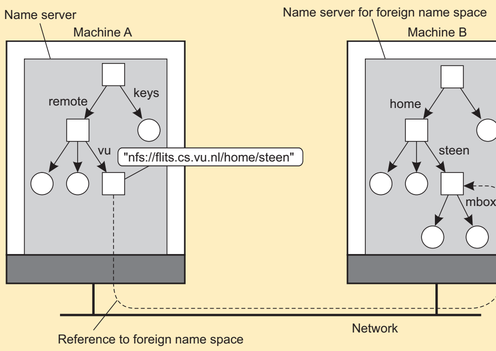
- Foreign name space: the name space that needs to be accessed
- Mount point: the node in the current name space containing the node identifier of the foreign name space
- Mounting point: the node in the foreign name space where to continue name resolution

Mounting across a network

1. The name of an access protocol.
2. The name of the server.
3. The name of the mounting point in the foreign name space.

Mounting in distributed systems

Mounting remote name spaces through a specific access protocol



Name-space implementation

Basic issue

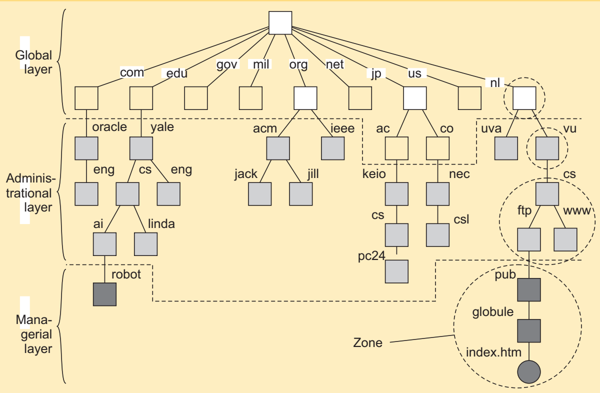
Distribute the name resolution process as well as name space management across multiple machines, by distributing nodes of the naming graph.

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

An example partitioning of the DNS name space, including network files



Name-space implementation

A comparison between name servers for implementing nodes in a name space

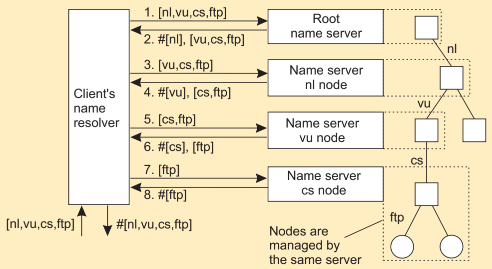
Item	Global	Administrational	Managerial
1	Worldwide	Organization	Department
2	Few	Many	Vast numbers
3	Seconds	Milliseconds	Immediate
4	Lazy	Immediate	Immediate
5	Many	None or few	None
6	Yes	Yes	Sometimes

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

Iterative name resolution

Principle

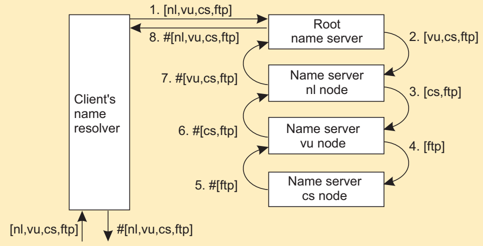
1. $resolve(dir, [name_1, ..., name_K])$ sent to $Server_0$ responsible for *dir*
2. $Server_0$ resolves $resolve(dir, name_1) \rightarrow dir_1$, returning the identification (address) of $Server_1$, which stores dir_1.
3. Client sends $resolve(dir_1, [name_2, ..., name_K])$ to $Server_1$, etc.



Recursive name resolution

Principle

1. $resolve(dir, [name_1, ..., name_K])$ sent to $Server_0$ responsible for *dir*
2. $Server_0$ resolves $resolve(dir, name_1) \rightarrow dir_1$, and sends $resolve(dir_1, [name_2, ..., name_K])$ to $Server_1$, which stores dir_1.
3. $Server_0$ waits for result from $Server_1$, and returns it to client.



Caching in recursive name resolution

Recursive name resolution of [*nl*, *vu*, *cs*,*ftp*]

Server for node	Should resolve	Looks up	Passes to child	Receives and caches	Returns to requester
cs	[*ftp*]	#[*ftp*]	—	—	#[*ftp*]
vu	[*cs*, *ftp*]	#[*cs*]	[*ftp*]	#[*ftp*]	#[*cs*] #[*cs*, *ftp*]
nl	[*vu*, *cs*, *ftp*]	#[*vu*]	[*cs*, *ftp*]	#[*cs*] #[*cs*, *ftp*]	#[*vu*] #[*vu*, *cs*] #[*vu*, *cs*, *ftp*]
root	[*nl*, *vu*, *cs*, *ftp*]	#[*nl*]	[*vu*, *cs*, *ftp*]	#[*vu*] #[*vu*, *cs*] #[*vu*, *cs*, *ftp*]	#[*nl*] #[*nl*, *vu*] #[*nl*, *vu*, *cs*] #[*nl*, *vu*, *cs*, *ftp*]

Attribute-based naming

Observation

In many cases, it is much more convenient to name, and look up entities by means of their attributes ⇒ traditional directory services (aka yellow pages).

Problem

Lookup operations can be extremely expensive, as they require to match requested attribute values, against actual attribute values ⇒ inspect all entities (in principle).

Implementing directory services

Solution for scalable searching

Implement basic directory service as database, and combine with traditional structured naming system.

Lightweight Directory Access Protocol (LDAP)

Each directory entry consists of (*attribute, value*) pairs, and is uniquely named to ease lookups.

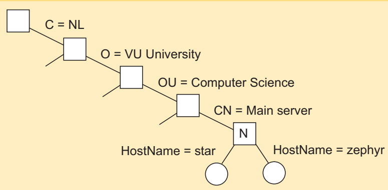
Attribute	Abbr.	Value
Country	*C*	NL
Locality	*L*	Amsterdam
Organization	*O*	VU University
OrganizationalUnit	*OU*	Computer Science
CommonName	*CN*	Main server
Mail_Servers	–	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	–	130.37.20.20
WWW_Server	–	130.37.20.20

LDAP

Essence

- Directory Information Base: collection of all directory entries in an LDAP service.
- Each record is uniquely named as a sequence of naming attributes (called Relative Distinguished Name), so that it can be looked up.
- Directory Information Tree: the naming graph of an LDAP directory service; each node represents a directory entry.

Part of a directory information tree



LDAP

Two directory entries having *HostName* as RDN

Attribute	Value	Attribute	Value
Locality	*Amsterdam*	*Locality*	*Amsterdam*
Organization	*VU University*	*Organization*	*VU University*
OrganizationalUnit	*Computer Science*	*OrganizationalUnit*	*Computer Science*
CommonName	*Main server*	*CommonName*	*Main server*
HostName	*star*	*HostName*	*zephyr*
HostAddress	*192.31.231.42*	*HostAddress*	*137.37.20.10*

Result of `search(''(C=NL)(O=VU University)(OU=*)(CN=Main server)'')`

Distributed index

Basic idea

- Assume a set of attributes $\{a^1, \ldots, a^N\}$
- Each attribute a^k takes values from a set R^k
- For each attribute a^k associate a set $\mathbf{S^k} = \{S^k_1, \ldots, S^k_{n_k}\}$ of n_k servers
- Global mapping F: $F(a^k, v) = S^k_j$ with $S^k_j \in \mathbf{S^k}$ and $v \in R^k$

Observation

If $L(a^k, v)$ is set of keys returned by $F(a^k, v)$, then a query can be formulated as a logical expression, e.g.,

$$(F(a^1, v^1) \wedge F(a^2, v^2)) \vee F(a^3, v^3)$$

which can be processed by the client by constructing the set

$$(L(a^1, v^1) \cap L(a^2, v^2)) \cup L(a^3, v^3)$$

Drawbacks of distributed index

Quite a few

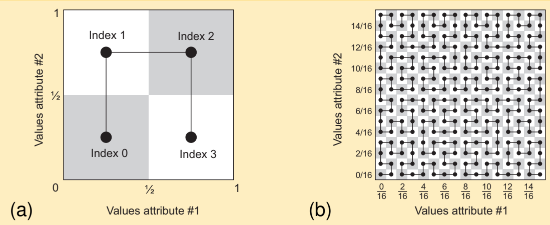
- A query involving k attributes requires contacting k servers
- Imagine looking up "$lastName = Smith \wedge firstName = Pheriby$": the client may need to process many files as there are so many people named "Smith."
- No (easy) support for range queries, such as "$price = [1000 - 2500]$."

Alternative: map all attributes to 1 dimension and then index

Space-filling curves: principle

1. Map the N-dimensional space covered by the N attributes $\{a^1, \ldots, a^N\}$ into a single dimension
2. Hashing values in order to distribute the 1-dimensional space among index servers.

Hilbert space-filling curve of (a) order 1, and (b) order 4



Space-filling curve

Once the curve has been drawn

Consider the two-dimensional case

- a Hilbert curve of order k connects 2^{2k} subsquares $\Rightarrow$ has 2^{2k} indices.
- A range query corresponds to a rectangle R in the 2-dimensional case
- R intersects with a number of subsquares, each one corresponding to an index $\Rightarrow$ we now have a series of indices associated with R.

Getting to the entities

Each index is to be mapped to a server, who keeps a reference to the associated entity. One possible solution: use a DHT.