
DISTRIBUTED SYSTEMS (COMP9243)

Lecture 9a: Naming

Slide 1

- ① Basic Concepts
- ② Naming Services
- ③ Attribute-based Naming (aka Directory Services)
- ④ Distributed hash tables

WHAT IS NAMING?

Systems manage a wide collection of **entities** of different kinds. They are identified by different kinds of **names**:

- Files: `/boot/vmlinuz`, `~/lectures/DS/notes/tex/naming.tex`
- Processes: `1`, `14293`
- devices: `/dev/hda`, `/dev/ttyS1`
- users: `chak`, `ikuz`, `cs9243`
- networked hosts: `weill`, `wagner`, `vina01`
- ...

Uniform naming (and access) is a powerful technique:

- UNIX: uniform names for files, devices, sockets, etc.
- `/proc` file system: maps process names to the file names
- Plan 9, Inferno: "All resources are named and accessed like files in a forest of hierarchical file systems"

BASIC CONCEPTS

Name:

- String of bits or characters
- Refers to an entity

Entity:

- Resource, process, user, etc.
- **Operations** performed on entities at **access points**

Address:

- Access point named by an **address**
- Entity address = address of entity's access point
- Multiple access points per entity
- Entity's access points may change

Identifier:

- Name that *uniquely* identifies entity
- Properties:
 - ① Refers to at most one entity
 - ② Entity referred to by at most one identifier
 - ③ Always refers to same entity (i.e. no reuse)
- Allows easy comparison of references

Location Independent Name:

- Independent of entity's address
- Remains valid if entity address changes

SYSTEM-ORIENTED VS HUMAN-ORIENTED NAMES

System-Oriented Names:

- Represented in machine readable form
- Usually 32 bit or 64 bit strings
- Structured or unstructured
- ✓ Easy to store, manipulate, compare
- ✗ Not easy to remember
- ✗ Hard for humans to use
- Example: inode (0x00245dad)

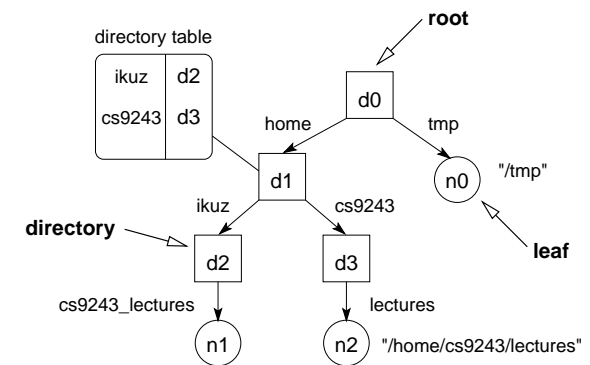
Slide 5

Human-Oriented Names:

- Variable length character strings
- Usually structured
- Often many human-oriented names map onto a single system-oriented name
- ✓ Easy to remember and distinguish between
- ✗ Hard for machine to process
- Example: pathnames
(`/home/cs9243/lectures/naming/slides.tex`)
- Example: URL (`http://www.cse.unsw.edu.au/~cs9243/lectures`)

Slide 6

NAME SPACES



Slide 7

Structure:

- Flat (only leaf nodes)
- Strictly hierarchical
- Directed acyclic graph (DAG)
- Multiple root nodes

Slide 8

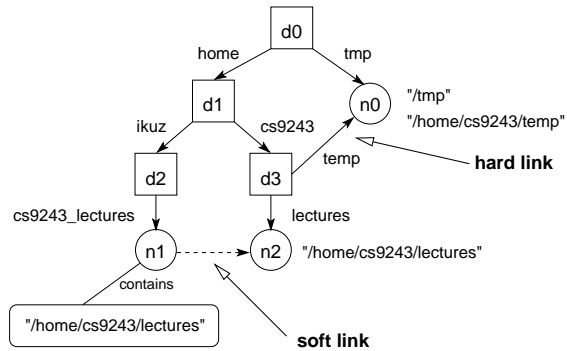
Path Names:

- Sequence of edge labels
- Refers to a leaf node or a directory node
- **Absolute**: if first node in path name is a root node
- **Relative**: otherwise

Aliasing:

- **Alias**: another name for an entity
- **Hard link**: two or more paths to an entity in the graph
- **Soft link**: leaf node stores a (absolute) path name to another node

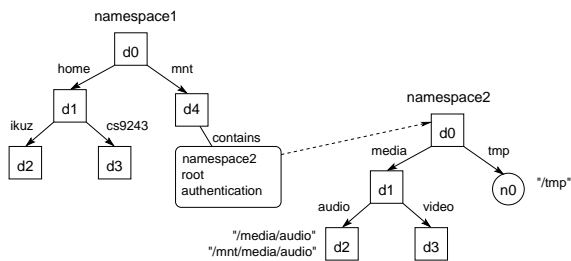
Slide 9



Merging:

- Mounting
 - Directory node stores info about a directory node in other name space
 - Need: protocol, server, path name, authentication and authorisation info, keys for secure communication, etc.

Slide 10



→ Combining name spaces

Slide 11

- Example: URL
- <http://www.cse.unsw.edu.au/~cs9243/naming-slides.pdf>
- Protocol name space
- DNS name space
- File System name space

NAMING SERVICES

A naming service provides a name space

Name Server:

- Naming service implemented by name servers
- Implements naming service operations

Slide 12

Operations:

- **Lookup**: resolve a path name, or element of a path name
- **Add**: add a directory or leaf node
- **Remove**: remove a subtree or leaf node
- **Modify**: modify the contents of a directory or leaf node

Client:

- Invokes naming service operations

DISTRIBUTED NAMING SERVICE

Non-Distributed Naming Service:

- Single name server
- Remote or local

Slide 13

Distributed Naming Service:

- Multiple name servers
- Name space is replicated and/or partitioned
- Try to distribute name space management and name resolution
- Example: DNS

NAME RESOLUTION

The process of looking up a name

Resolution:

- Mapping a name onto the node referred to by the name
- Interested in the data stored by the node

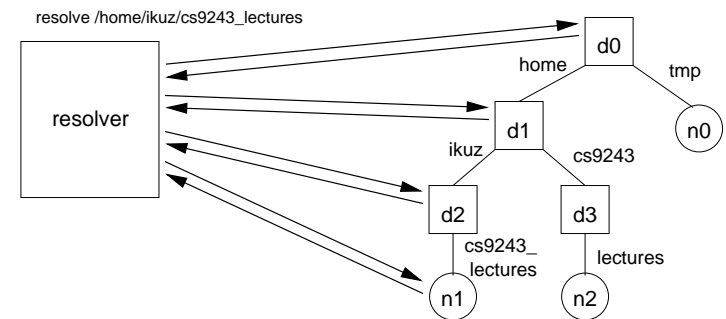
Slide 14

Path Name Resolution:

- Starts at a begin node (first element of the path name)
 - Root node for absolute name
 - Directory node for relative name
- Ends with data from (or a reference to) the last node (last element of path name)

Iterative Resolution:

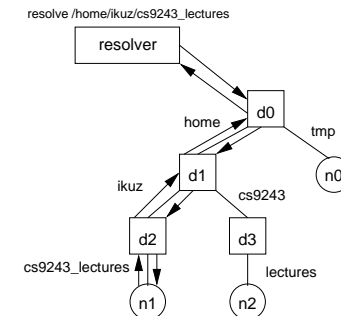
Slide 15



- ✗ Caching only at resolver
- ✗ Lots of communication

Recursive Resolution:

Slide 16



- ✓ Effective caching at name servers
- ✓ Reduced communication (if name servers close together)
- ✓ Name servers can be protected from external access
- ✗ Higher performance demand placed on servers

Resolver:

- Does name resolution on behalf of client
- In client process, in client's kernel, process on client's machine
- In name server
- Separate service

Slide 17 Closure Mechanism:

- Resolution can only take place if we know where to start
- How to find initial node?
 - Have reference to directory node for relative name
 - Obtain reference to root node externally
- Closure mechanism is always implicit

NAMING SERVICE IMPLEMENTATION ISSUES

Performance and Scalability:

- Limit load on name servers
- Limit communication required
- Partitioning: split name space over multiple name servers
- Replication: copy (parts of) name space on multiple name servers

Slide 18

Fault Tolerance:

- Replication

Authoritative Name Server:

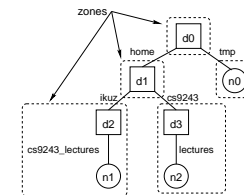
- Name server that stores an entity's original attributes

PARTITIONING

Split name space over multiple servers

Structured Partitioning:

- split name space according to graph structure
- Name resolution can use zone hints to quickly find appropriate server
- Example: associate path name prefixes to zones
 - ✓ Improved lookup performance due to knowledge of structure
 - ✗ Rigid structure

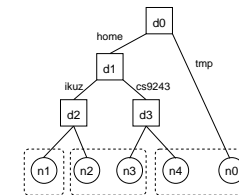


Slide 19

Structure-free Partitioning:

- content placed on servers independent of name space
 - ✓ Flexible
 - ✗ Decreased lookup performance
 - ✗ increased load on root

Slide 20



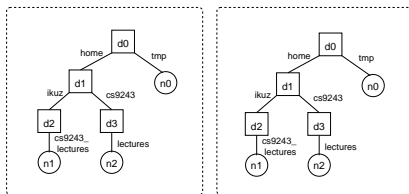
REPLICATION

Copy name space to multiple servers

Full Replication:

- copy complete name space
- ✓ Fast performance
- ✗ Size (each server must store whole name space)
- ✗ Consistency (any change has to be performed at all replicas)
- ✗ Administration (who has rights to make changes where?)

Slide 21



Caching:

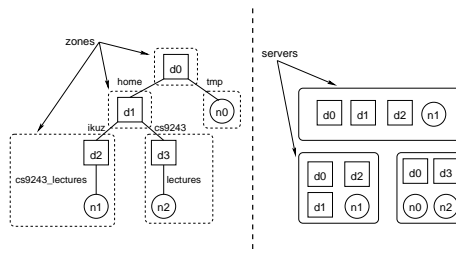
- Cache query results
- ✓ No administrative problems
- Types of caches:
 - Directory cache: cache directory node information
 - Prefix cache: cache path name prefixes
 - Full-name cache: cache full names
- Cache implementations:
 - Process-local cache: in address space of process
 - ✗ no sharing of cache
 - Kernel cache: cache kept by kernel
 - ✓ sharing, no start up misses
 - ✗ kernel bloat
 - User-process cache: separate shared service
 - ✓ sharing of cache
 - ✗ slow access

Slide 23

Partial replication:

- Replicate full name servers
- Replicate zones
- ✓ Improved performance
- ✓ Less administrative problems
- ✓ Less consistency problems

Slide 22



Cache Updates and Consistency:

- On use checking
- Timeout
- Invalidation
- Slow propagation

Slide 24

DNS (DOMAIN NAME SYSTEM)

Structure:

- Hierarchical structure (tree)
- Top-level domains (TLD) (.com, .org, .net, .au, .nl, ...)
- *Zone*: a (group of) directory node
- *Resource records*: contents of a node
- *Domain*: a subtree of the global tree
- *Domain name*: an absolute path name

Slide 25

Type of record	Associated entity	Description
SOA	Zone	Holds information on the represented zone
A	Host	Contains an IP address of the host this node represents
MX	Domain	Refers to a mail server to handle mail addressed to this node
SRV	Domain	Refers to a server handling a specific service
NS	Zone	Refers to a name server that implements the represented zone
CNAME	Node	Symbolic link with the primary name of the represented node
PTR	Host	Contains the canonical name of a host
HINFO	Host	Holds information on the host this node represents
TXT	Any kind	Contains any entity-specific information considered useful

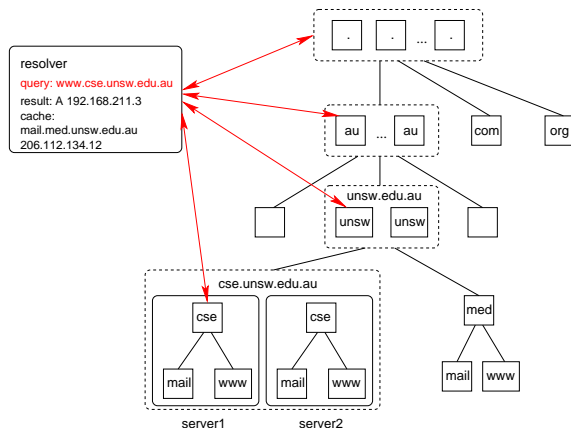
Partitioning:

- Each zone implemented by a name server

Replication:

- Each zone replicated on at least two servers
- Updates performed on *primary*
- Contents transferred to *secondary* using *zone transfer*
- Higher levels have many more replicas (12 root replicas)

Slide 27



Slide 26

Caching:

- Servers cache results of queries
- Original entries have time-to-live field (TTL)
- Cached data provided until TTL expires
- Cached data is non-authoritative

Slide 28

Name Resolution:

- Query sent to local server
- If cannot resolve locally then sent to root
- Resolved recursively or iteratively
- Recursive is optional for name server implementations

LDAP & ATTRIBUTE-BASED NAMING

White Pages vs Yellow Pages:

- White Pages: Name ➔ Phone number
- Yellow Pages: Attribute ➔ Set of entities with that attribute
- Example: X.500 and LDAP

Slide 29

Attribute-Based Names:

- Example: /C=AU/O=UNSW/OU=CSE/CN=WWW Server/Hardware=Sparc/OS=Solaris/Server=Apache
- Compared to conventional naming: `www.cse.unsw.edu.au`
- Distinguished name (DN): set of attributes that forms a canonical name for an entity
- Distinguished attribute: an attribute that is part of a distinguished name (aka Relative DN)

Attribute-Based Naming:

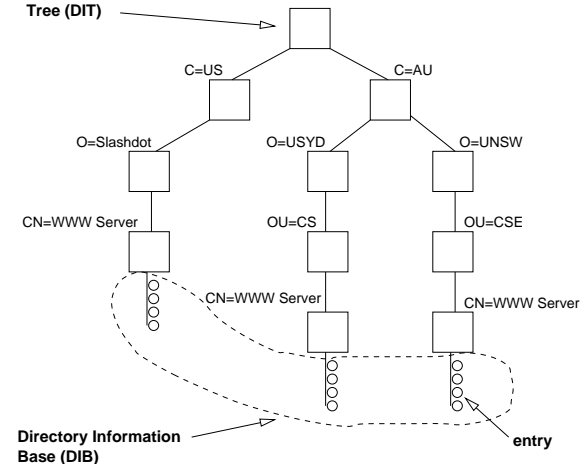
- Lookup entities based on attributes
- Example: `search("&(C=AU)(O=UNSW)(OU=*)(CN=WWW Server)")`
- Attributes stored in *directory entry*
- Collection of all directory entries forms *directory*
- Directory implemented by *directory service*
- **Directory Information Tree** (DIT) logical structure of information in directory (naming graph)
- **Directory Information Base** (DIB) contents of the directory

Slide 30

Name Space:

- Flat: no structure in directory service
- Hierarchical: structured according to a hierarchy
- *Distinguished name* mirrors structure of name space
- All possible attribute types and name space defined by *schema*

Directory Information Tree (DIT)



Slide 31

DIRECTORY SERVICES

A directory service implements a directory

Operations:

- **Lookup**: resolve a distinguished name
- **Add**: add an entity
- **Remove**: remove an entity
- **Modify**: modify the attributes of an entity
- **Search**: search for entities that have particular attributes
- Search can use partial knowledge
- Search does not have to include distinguished attributes
- Most important qualities: allow browsing and allow searching

Slide 32

Client:

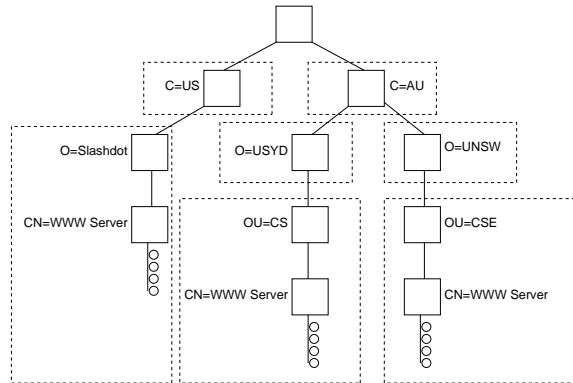
- Invokes directory service operations

DISTRIBUTED DIRECTORY SERVICE

Partitioning:

→ Partitioned according to name space structure (e.g., hierarchy)

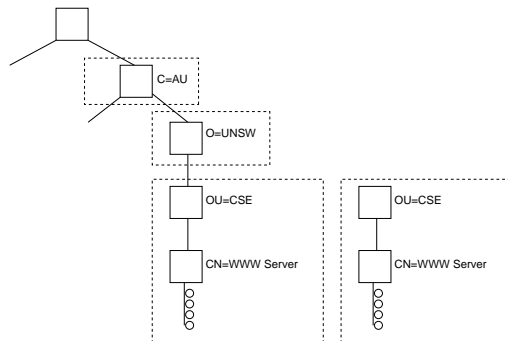
Slide 33



Replication:

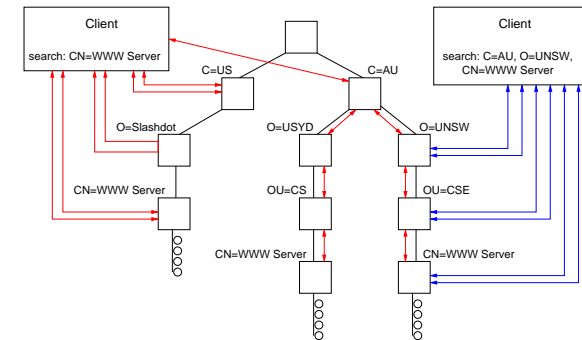
- Replicate whole directory
- Replicate partitions
- Read/Write and read only replicas (e.g. primary-backup)
- Catalog and cache replicas

Slide 34



SEARCHING AND LOOKUP IN A DISTRIBUTED DIRECTORY

Slide 35



Approaches:

- Chaining (recursive)
- Referral (iterative)
- Multicasting (uncommon)

Performance of Searching:

Slide 36

- Searching whole name space: must visit each directory server
- ✗ bad scalability
- Limit searches by specifying *context*
- Catalog: stores copy of subset of DIB information in each server
- **Main problem: multiple attributes mean multiple possible decompositions for partitioning BUT only one decomposition can be implemented**

X.500 AND LDAP

X.500:

- ISO standard
- Global DIT
- Defines DIB, DIB partitioning, and DIB replication

Slide 37 LDAP (Lightweight Directory Access Protocol):

- X.500 access over TCP/IP
 - X.500 is defined for OSI Application layer
 - Textual X.500 name representation
 - Popular on Internet
 - Also X.500 free implementations (e.g. openldap)
 - Used in Windows for Active Directory
-
-

ADDRESS RESOLUTION OF UNSTRUCTURED IDENTIFIERS

Unstructured Identifiers:

- Slide 38**
- Practically random bit strings
 - Example: random key, hash value
 - No location information whatsoever
 - *How to find corresponding address of entity?*
-
-

Simple Solution: Broadcasting:

- Resolver broadcasts query to every node
- Only nodes that have access point will answer

Example – ARP:

Slide 39

Protocol to resolve MAC addresses from IP addresses.

- Resolver broadcasts:
Who has 129.94.242.201? Tell 129.94.242.200
 - 129.94.242.201 answers to 129.94.242.200:
129.94.242.201 is at 00:15:C5:FB:AD:95
-
-

DISTRIBUTED HASH TABLES

Hash table as overlay network:

- `put(key, value), value = get(key), remove(key)`

Example: look up unstructured host names:

Slide 40

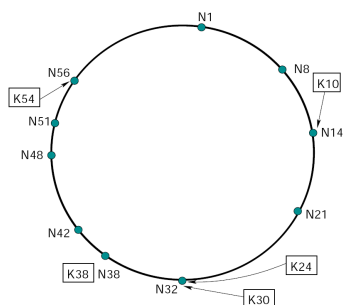
```
put(weill, 129.94.242.49)
put(beethoven, 129.94.172.11)
put(maestro, 129.94.242.33)
```

```
address = get(beethoven)
```

- How high is performance cost of lookup?
-
-

CHORD: DISTRIBUTED HASH TABLE

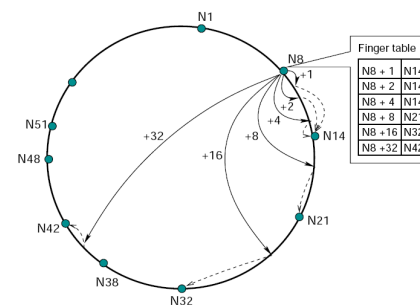
General Structure:



Slide 41

- keys and node IP addresses mapped to identifier
- consistent hashing (SHA-1 m-bits)
- key assigned to first node with $id > key \rightarrow \text{successor}(key)$

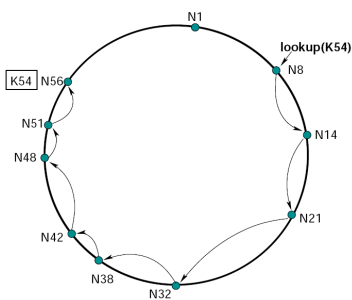
A scalable lookup:



Slide 43

- routing table at every node: *finger table*
- i th entry is $\text{successor}(n + 2^{i-1})$
- $\text{finger}[i]$ is successor

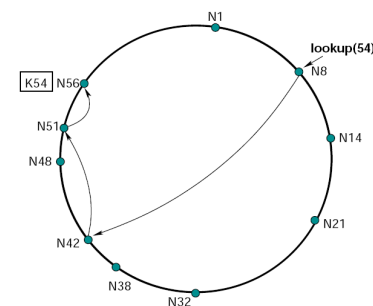
A simple lookup:



Slide 42

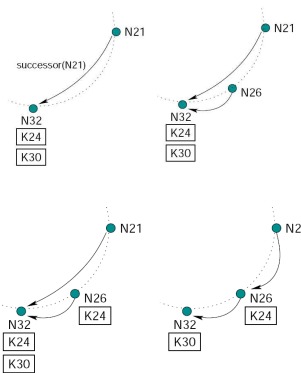
- use successors function
- recursive RPCs until node with key is found
- $O(n)$ cost

Slide 44



- lookup greatest node id in table $< k$
- ask it to lookup the key
- exponentially smaller jumps

Adding a node:



Slide 45

- stabilize: ensure successor pointers up-to-date
- fix_fingers: ensure that finger tables updated

Dealing with node failure:

- successor list: r successors to handle $r - 1$ failures
- higher level must handle loss of data relating to failure

Slide 46 Analysis:

- finger table size: $O(\log n)$.
- $O(\log n)$ nodes contacted for lookup
- $1/2 \log n$ average

Using a DHT:

- Search index
- Database
- File system
- Shared mirrors/storage

Slide 47

