Slide 1

DISTRIBUTED SYSTEMS (COMP9243)

Lecture 8a: Naming

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

Slide 2

WHAT IS NAMING?

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

➜ Files (`/boot/vmlinuz`), Processes (`1, 14293`), Users (`chak, ikuz, cs9243`), Hosts (`weill, facebook.com`), ...

Examples of naming in distributed systems?
What's the difficulty?

Slide 3

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

Slide 4

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

Slide 5

SYSTEM-ORIENTED VS HUMAN-ORIENTED NAMES

System-Oriented Names:

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

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: URL (`http://www.cse.unsw.edu.au/~cs9243/lectures`)

Slide 6

NAME SPACES

Container for a set of related names

Structure options:

- ➜ Flat (only leaf nodes)
- ➜ Hierarchical (Strictly hierarchical, DAG, Multiple root nodes)
- ➜ Tag-based

Path Names (in hierarchies):

- ➜ Sequence of edge labels
- ➜ Absolute: if first node in path name is a root node
- ➜ Relative: otherwise

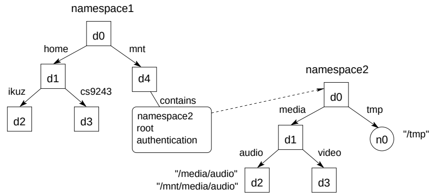
Aliasing:

- ➜ 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 7

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.



- ➜ Combining name spaces
  - `http://www.cse.unsw.edu.au/~cs9243/naming-slides.pdf`
  - Name Spaces: Protocol, DNS, File System

Slide 8

NAMING SERVICES

A naming service provides a name space

Name Server:

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

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

Centralised vs Distributed Naming Service

Slide 9

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

Path Name Resolution:

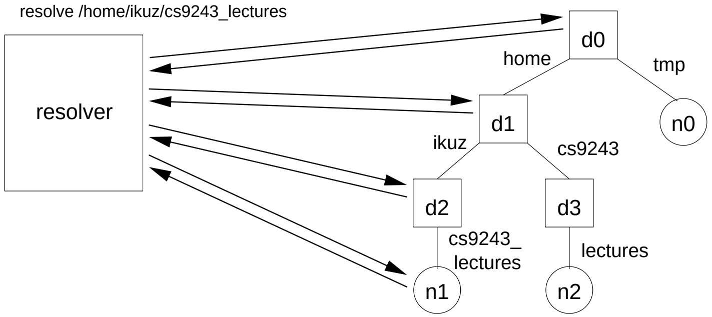
- ➜ Starts at a begin node (first element of the path name)
  - Root node for absolute name
  - Directory node for relative name
- ➜ Ends with data from (or a reference to) the last node (last element of path name)

Resolver:

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

Slide 10

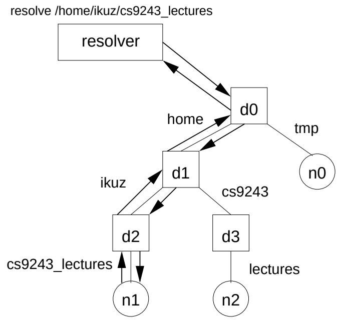
Iterative Resolution:



- ✗ Caching only at resolver
- ✗ Lots of communication

Slide 11

Recursive Resolution:



- ☑ 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

Slide 12

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

Fault Tolerance:

- ➜ Replication

Authoritative Name Server:

- ➜ Name server that stores an entity's original attributes

Slide 13

Partitioning

Split name space over multiple servers

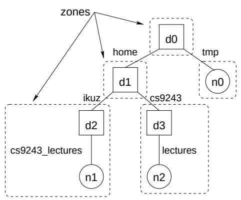
Structured Partitioning:

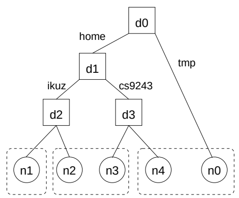
- ➜ split name space according to graph structure
- ➜ Name resolution can use zone hints to quickly find appropriate server
- ☑ Improved lookup performance due to knowledge of structure
- ✗ Rigid structure

Structure-free Partitioning:

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

Slide 14





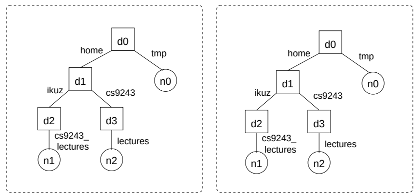
Slide 15

Replication

Copy name space to multiple servers

Full Replication:

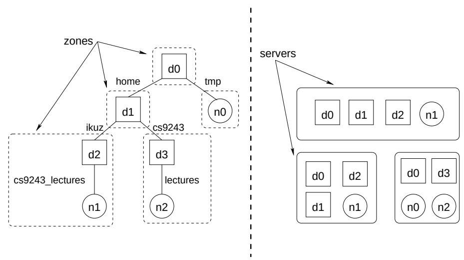
- ➜ copy complete name space
- ☑ Fast performance
- ✗ Size (each server must store whole name space)
- ✗ Consistency (any change has to be performed at all replicas)
- ✗ Administration (who has rights to make changes where?)



Slide 16

Partial replication:

- ➜ Replicate full name servers
- ➜ Replicate zones
- ☑ Improved performance, less consistency overhead
- ☑ Less administrative problems



Slide 17

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
  - Kernel cache: cache kept by kernel
  - User-process cache: separate shared service
- ➜ Cache updates and consistency
  - On use checking
  - Timeout
  - Invalidation
  - Slow propagation

Slide 18

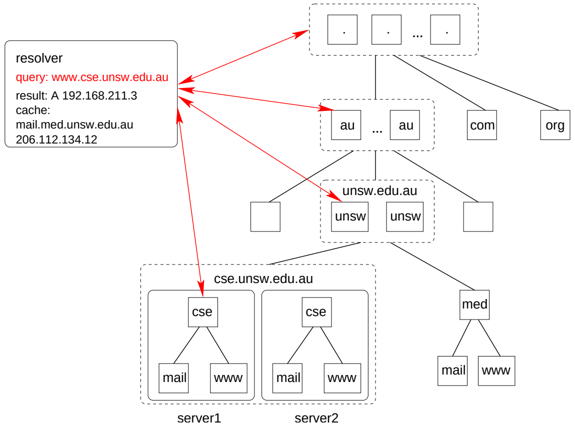
DNS (Domain Name System)

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

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

Slide 19



Slide 20

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 (13 root servers: `A-M.root-servers.net`. Actually 386 replicas using anycast)

Caching:

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

Name Resolution:

- ➜ Query sent to local server
- ➜ If cannot resolve locally then sent to root
- ➜ Resolved recursively or iteratively

Slide 21

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

Attribute-Based Names:

- ➜ Example:`/C=AU/O=UNSW/OU=CSE/CN=WWW Server/Hardware=Sparc/OS=Solaris/Server=Apache`
- ➜ Distinguished name (DN): set of attributes (distinguished attributes) that forms a canonical name for an entity

Slide 22

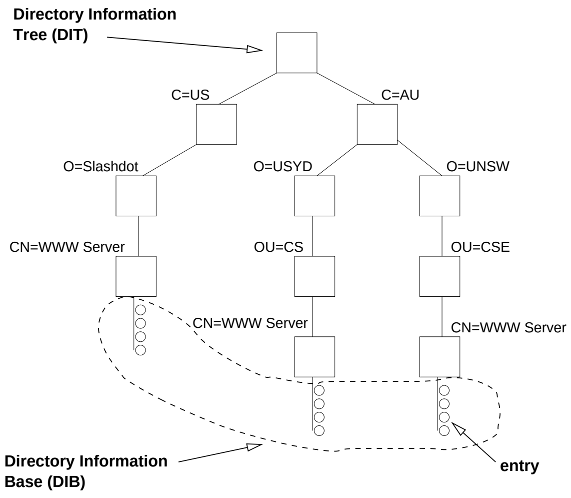
Attribute-Based Naming:

- ➜ Lookup entities based on attributes
- ➜ Example: `search("&(C=AU)(O=UNSW)(OU=*)(CN=WWW Server)")`
- ➜ Attributes stored in *directory entry*, all stored in *directory*

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*

Slide 23



Slide 24

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

Client:

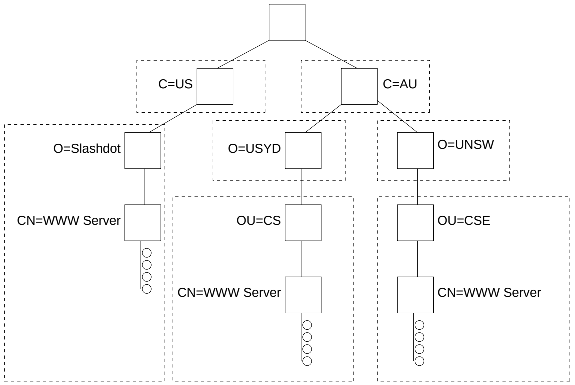
- ➜ Invokes directory service operations

Slide 25

Distributed Directory Service

Partitioning:

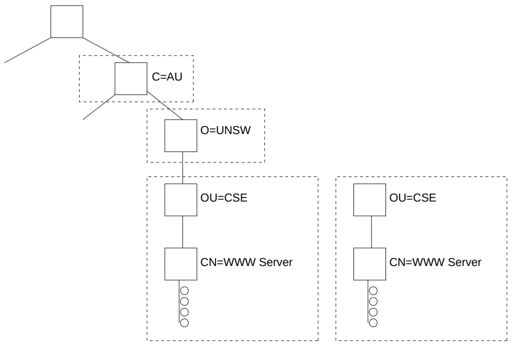
- ➜ Partitioned according to name space structure (e.g., hierarchy)



Slide 26

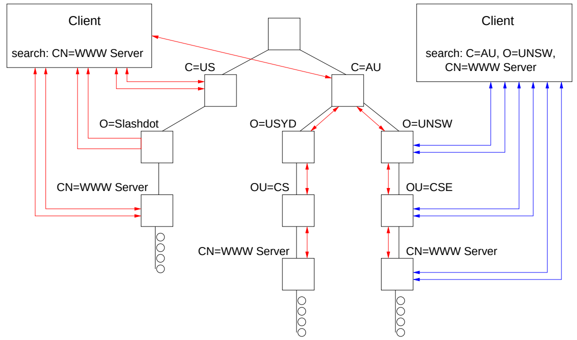
Replication:

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



Slide 27

Searching and Lookup in a Distributed Directory



Slide 28

Approaches:

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

Performance of Searching:

- ➜ 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

Slide 29

X.500 AND LDAP

X.500:

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

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

Slide 30

ADDRESS RESOLUTION OF UNSTRUCTURED NAMES

Unstructured Names:

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

Slide 31

Simple Solution: Broadcasting:

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

Example – ARP:

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

Slide 32

DISTRIBUTED HASH TABLES

Hash table (key value store) as overlay network:

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

Example: look up unstructured host names:

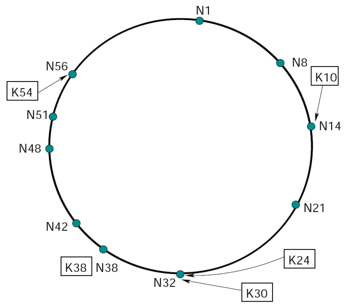
```
put(weill, 129.94.242.49)
put(beethoven, 129.94.172.11)
put(maestro, 129.94.242.33)

address = get(beethoven)
```

- How high is performance cost of lookup?

CHORD: DISTRIBUTED HASH TABLE

Slide 33

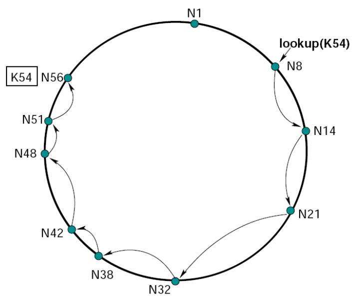
General Structure:



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

Slide 34

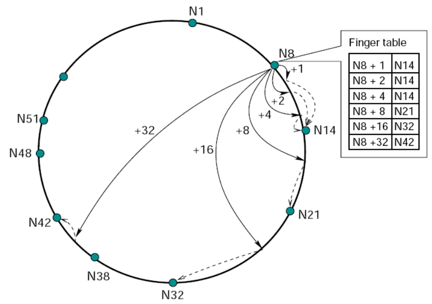
A simple lookup:



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

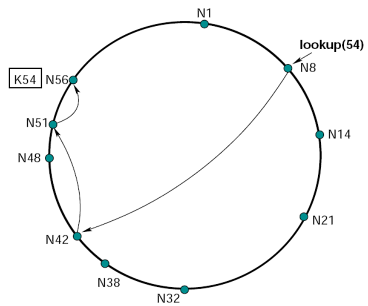
Slide 35

A scalable lookup:



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

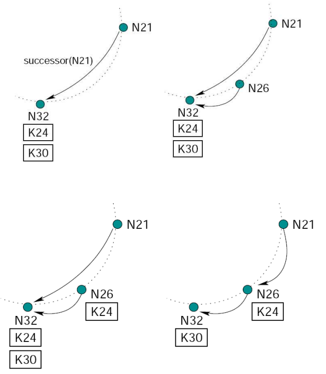
Slide 36



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

Slide 37

Adding a node:



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

Slide 38

Dealing with node failure:

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

Analysis:

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

Slide 39

HOMEWORK

- How could you use a DHT to implement a directory service?
- How could you use a DHT to implement a file system?

Hacker's edition:

- Use an existing DHT implementation to implement a simple file system.
- Implement the DHT yourself

Slide 40

READING LIST

Domain Names - Implementation and Specification RFC 1035 DNS

The Lightweight Directory Access Protocol: X.500 Lite LDAP

Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications Chord