Large Scale Systems Architecture (2)

Grid Computing (M)

Lecture 8

UNIVERSITY
of
GLASGOW



Change of Venue for Future Sessions

- Lectures:
  - Tuesday 12:00 ⇒ F121, Lilybank Gardens
  - Wednesday 12:00 ⇒ F121, Lilybank Gardens

- Tutorials:
  - Friday 11:00 ⇒ 246B, Kelvin Building

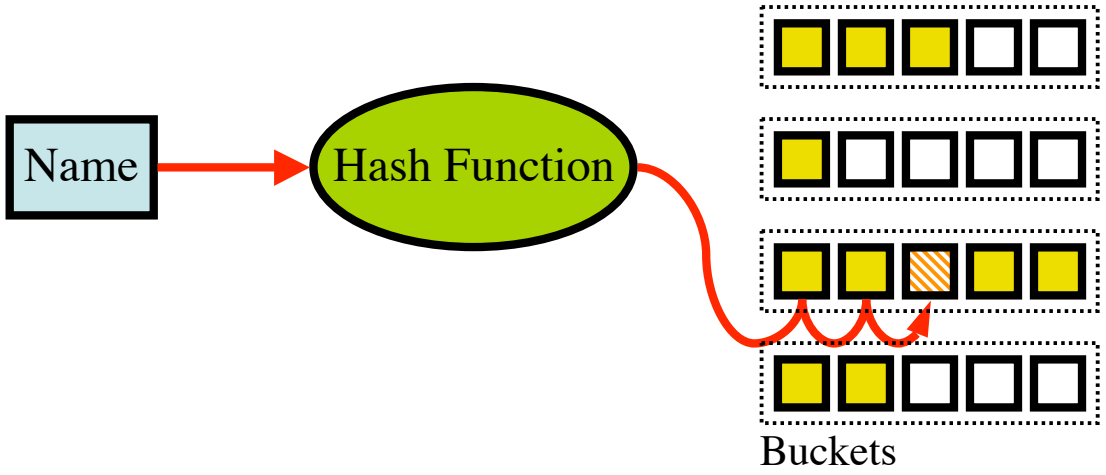
Copyright © 2006 University of Glasgow

Lecture Outline

- The distributed hash table abstraction
  - Concepts
  - Example protocols:
    - Chord
    - CAN
    - Tapestry
- Uses and motivating example systems:
  - OceanStore
  - Astrolabe
- Deployment considerations

Copyright © 2006 University of Glasgow

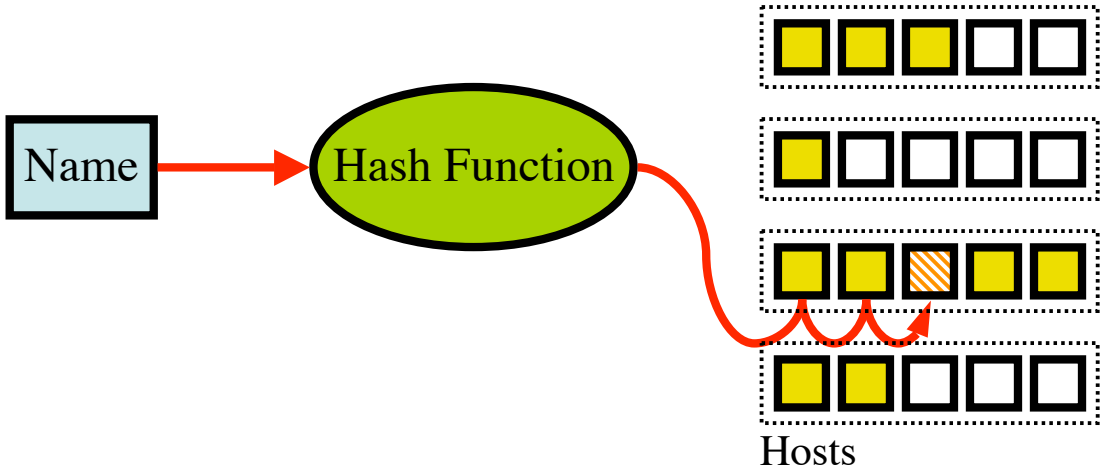
A Traditional Hash Table



- A classical hash table efficiently returns a *value* given a *name*
  - Passes name through a *hash function* mapping it to a fixed bucket address
    - Choice of hash function important, to evenly distribute keys to buckets
  - Iterate through items in the bucket to find value corresponding to the key; return that value
  - Space-time trade off to determine number and size of buckets

Copyright © 2006 University of Glasgow

A Distributed Hash Table (DHT)



- Similar concept:
  - Passes the name through a *hash function* which maps it to a *host*
    - Potentially flat unstructured names; location encoded via hash function
  - Iterate from that host to locate object
    - Relies on a structured protocol to provide pointers to next hosts in the chain

Copyright © 2006 University of Glasgow

Key Properties of a DHT

- Keys are *unstructured*
  - No need for hierarchical names
  - Works with any sort of data
- Data is distributed using a *structured* protocol
  - Each node responsible for a portion of the data space
- Queries are routed efficiently
- No central server or control
  - No node has global state
  - No node has a special position
  - Relies on hash function to provide implicit global knowledge

Copyright © 2006 University of Glasgow

DHT Examples

- Many examples of DHT in the literature, trying to formalize the structure of peer-to-peer name resolution
  - Compared to the many unstructured file-trading systems with ad-hoc name lookup, flooding or centralized schemes
  - Aiming to develop systems that can be reasoned about; have known lookup latency, state requirements, etc.
- Three representative examples:
  - Chord [1]
  - CAN [2]
  - Tapestry [3]

  - Will show basic routing algorithm for each; ignore details of neighbour discovery, handling joins and leaves, etc.
  - Each is a structured peer-to-peer system; but with very different structure

Copyright © 2006 University of Glasgow

Example: Chord

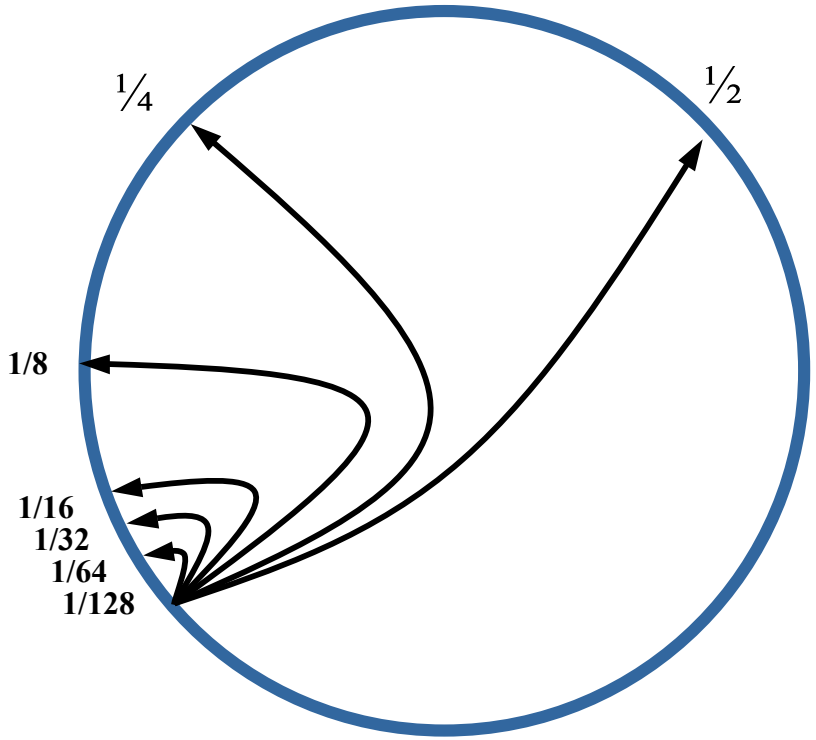
- Distributed name lookup, can be used to build a DHT:
  - Lookup(key) → IP address
  - Chord *does not* store the data

- Nodes and keys identified by hash value:
  - Node ID is hash of IP address
  - Key ID is hash of key
  - Both share the same numeric space
    - 160 bit SHA-1 hashes
    - Flat, uniform, namespace

Copyright © 2006 University of Glasgow

Example: Chord

- A *structured* distributed hash table

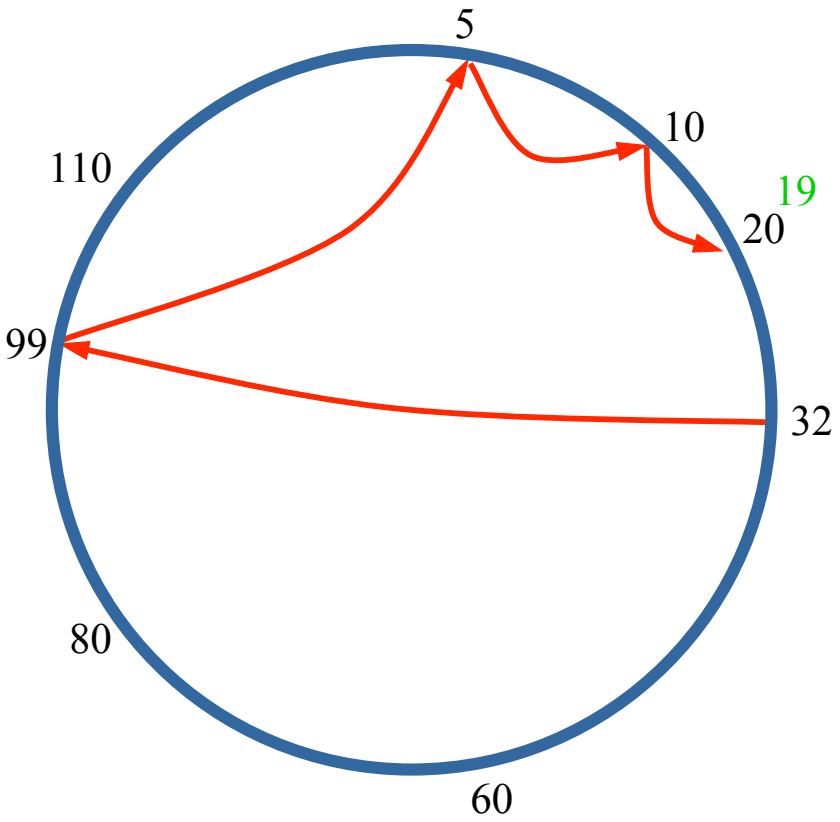


- *N* nodes arranged in a virtual ring
  - Hash values under arithmetic modulo *N*
  - O(*log(N)*) links to other nodes
    - Links to nodes placed $^1/_2$, $^1/_4$, $^1/_8$, $^1/_{16}$, … way around the ring
    - More links to nodes with similar node ID
  - A node manages all keys with key ID less than its node ID, but greater than the previous node's ID, modulo *N*

Copyright © 2006 University of Glasgow

Example: Chord

- Nodes maintain a routing table:
  - (Node ID, IP address) for each link
- Each hop routes queries along the link to the node with the greatest node ID less than key hash (modulo *N*)
  - Each hop halves the distance - in the hash space - to the node with the key
  - Actual network distance unknown at each hop
- Reaches destination in O(*log*(*N*)) hops
  - Efficient lookup in a large space

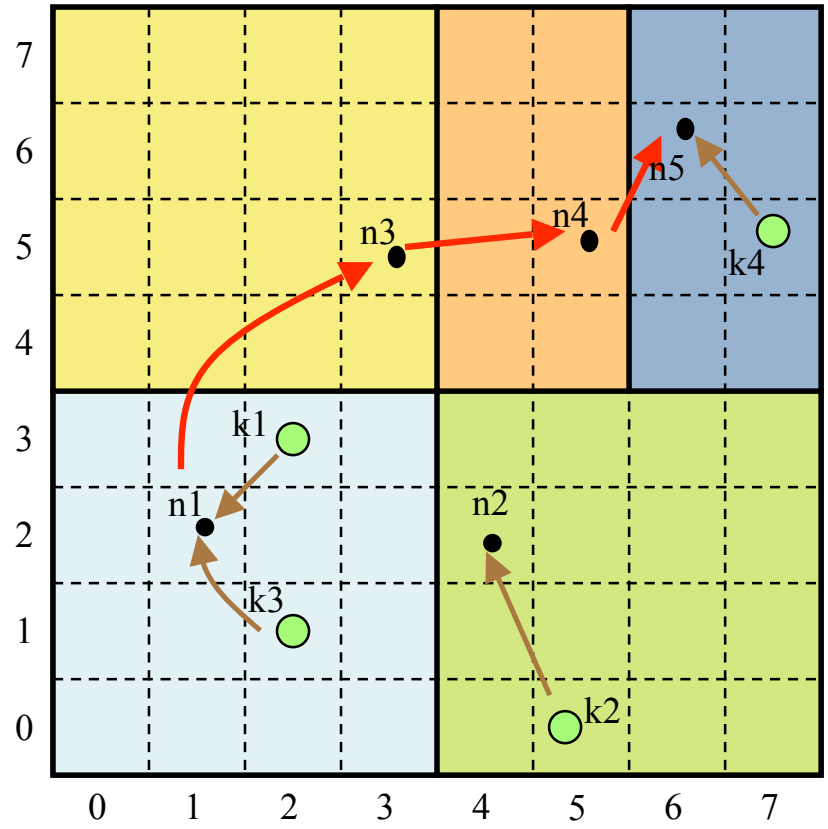


- Robust to node failures; simply choose a different (longer) path around ring

Copyright © 2006 University of Glasgow

Example: CAN

- Uses an *d* dimensional coordinate space on a torus ($d \geq 2$)
- Entire space divided between nodes
  - Each node owns a hyper-rectangle in the space, based on hash of node address
- Key-value pairs are stored in the CAN:
  - Keys mapped to points in coordinate space using a hash function
  - Values stored at the node owning that part of the space
- Each node knows its neighbours
  - *2d* neighbours (one in each direction for each dimension)
- Forward query to the neighbour node towards the location of the key in the coordinate space
  - Reaches destination in $O(N^{1/d})$ hops



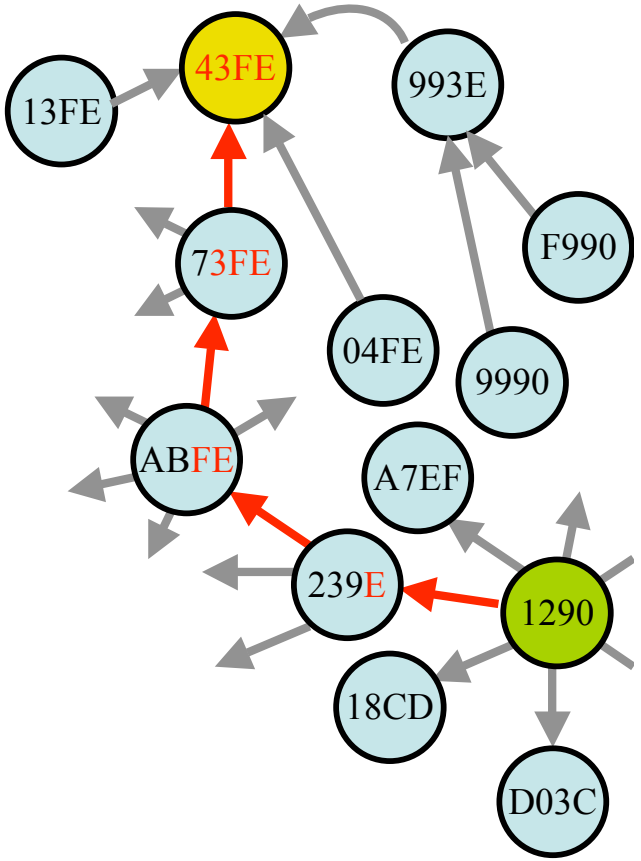
Copyright © 2006 University of Glasgow

Example: Tapestry

- Node and key ID assigned based on hash, converted to digits base *b* (e.g. base 16)
- Global mesh; each node has links to *b* nodes matching each possible suffix of its address
  - *b* $log_b N$ neighbours

- Routes to the *closest* neighbour with longer match to the desired address, digit-by-digit
  - Reaches destination in O($log_b N$) hopes
  - Will match several digits in one hop, if there is a matching neighbour

- Efficient name lookup; topology based routing



Copyright © 2006 University of Glasgow

Comparison: Chord *vs.* Tapestry *vs.* CAN

	Model	**Search Time**	**Node State**
Chord	1 dimensional	O($log\ N$)	O($log\ N$)
CAN	d dimensional	O($N^{1/d}$)	$2d$
Tapestry	Mesh	O($log_b N$)	O($b\ log_b N$)

- Each makes a different trade-off between search time and amount of state required
- Different fault tolerance and robustness properties
- Different degrees of complexity in maintaining the overlay
- Different behaviour when membership changes rapidly

⇒ Alternatives for efficient lookup of unstructured names

Copyright © 2006 University of Glasgow

Uses of Distributed Hash Tables

- A DHT maps from key to value
  - Efficient and location transparent lookup
  - Scalable to very large distributed systems

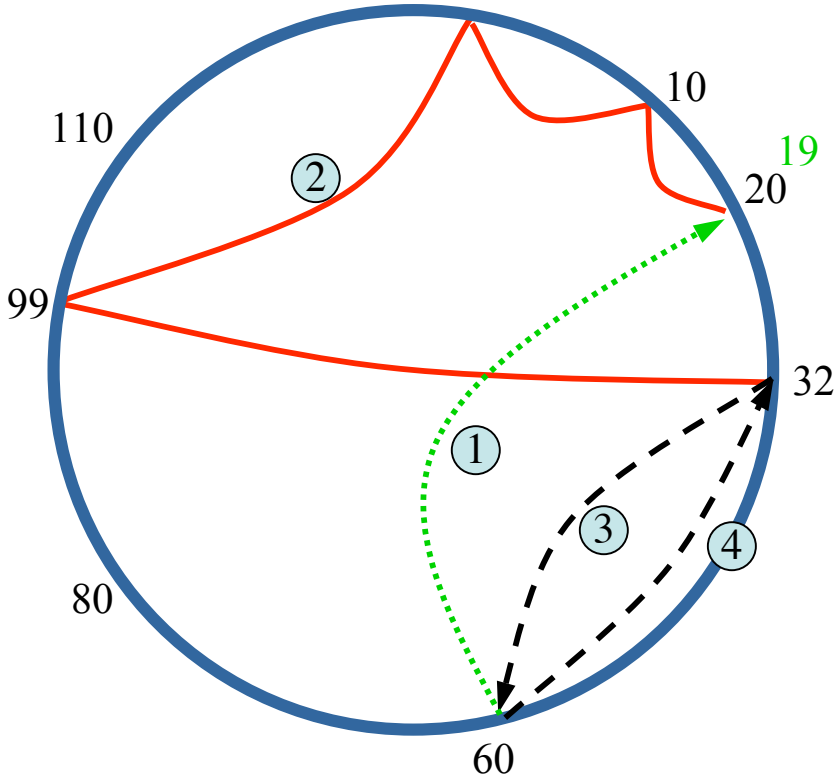
- Can be used for:
  - File sharing and data dissemination
  - Publish/subscribe based event notification
  - Distributed object location
  - Etc.

  - Potential basis for future grid computing systems

Copyright © 2006 University of Glasgow

Peer-to-Peer File Sharing/Data Storage

- A DHT can be used to build *distributed indexes* for file sharing or data dissemination
  - Keys are hashed "filenames", values returned are URLs for the actual data
  - Nodes keep the data, but publish location into the DHT for lookup



1. Name published in the DHT
2. DHT lookup of name, to return the actual location of data item
3. Request data from peer at returned location
4. Data transferred directly

Note: name stored in DHT based on hash function; not data location

Copyright © 2006 University of Glasgow

OceanStore

- An example of a global file system, built using a DHT
  - Aim: support 10 billion users each with 10,000 files
- Un-trusted infrastructure
  - Extensive use of cryptography to ensure privacy; enforce access rules
  - Extensive use of caching for robustness and performance
- Files identified by a hash of the filename + path
  - Files split into blocks, returns pointer to a table of hashes for blocks
    - Blocks indexed by cryptographic hash of contents
    - Blocked pushed somewhere into the network, located using Tapestry
      - Uses the DHT for data storage
    - Robust: can make multiple FEC copies for availability
  - Copy-on-write semantics for block; old versions retained forever
    - Efficient: only changes between versions stored
    - Efficient: files that share content automatically share storage since they hash to the same block, closest replica of the block located by Tapestry

[Lots of details skipped: see the paper 4]

Copyright © 2006 University of Glasgow

Publish/Subscribe and Event Notification

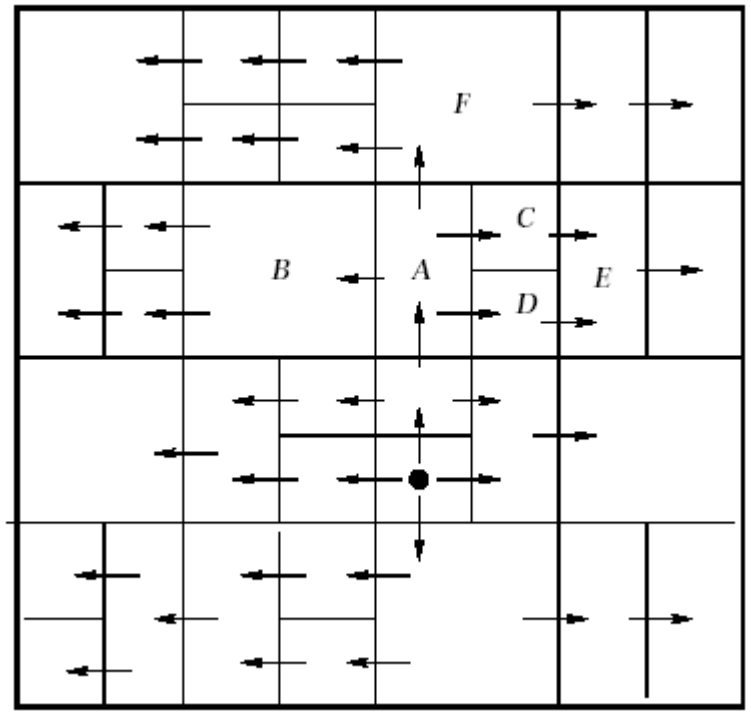
- Want to distribute notifications of events to a group of subscribers in a scalable manner
- Two problems:
  - How to locate the publisher of the events ⇒ DHT name lookup
  - How to notify subscribers of new events ⇒ application level multicast
    - Often built-in to the operation of a DHT using reverse path forwarding down the lookup tree (e.g. Bayeux); very similar to IP multicast

- Examples:
  - Multicast on CAN
  - Scribe on Pastry
  - Bayeux on Tapestry
  - Etc.

Copyright © 2006 University of Glasgow

Example: Multicast on CAN

- Group address hashes to node in the CAN
  - Other nodes contact that node and build a separate mini-CAN
  - Instead of building a multicast distribution tree, build the mini-CAN
- Flood packets from source to all nodes on the mini-CAN
  - Straight forward operation, since CAN has regular structure
  - Easier than a multicast tree; assumes building a CAN is light-weight



- Uses peer-to-peer structure to make flooding efficient

Copyright © 2006 University of Glasgow

Distributed Monitoring and Aggregation

- Problem: How to effectively monitor state of a large system?
  - Distributing full system state very inefficient
  - Full state details not interesting, provided everything working
  - Often sufficient to see a summary of the results
- Examples:
  - Network management
  - Distributed data mining

- Desirable to use a peer-to-peer protocol to distribute and aggregate results
  - Spread processing load across the network
  - Keep state local unless explicitly requested; distribute summaries widely
  - Scalable communication

Copyright © 2006 University of Glasgow

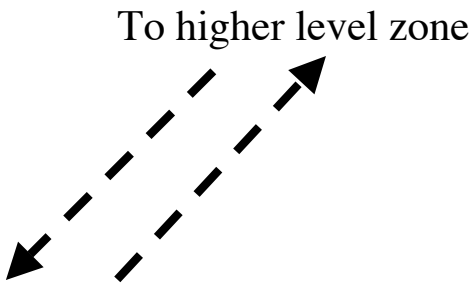
Example: Astrolabe

- Goal is to create a dynamic peer-to-peer database that shows the continuously evolving state of some system [5]
  - Pass SQL queries down through a peer-to-peer tree
    - Each level of the tree is a *zone*
  - Compute results locally
  - Aggregate and summarize data flowing back up the tree
- Approach: “peer to peer gossip”
  - Each machine has a piece of a jigsaw puzzle.
  - Periodically exchange state with a randomly chosen peer from your zone
    - Ensures data diffuses throughout the network with high probability
    - Aggregated data derived at hosts within a zone
  - Periodically query random child zone for it’s aggregated data
- Provides aggregate data for each zone; specific queries can then be issued to find details

Copyright © 2006 University of Glasgow

Example: Astrolabe

Dynamically changing aggregate query output is visible system-wide; specific data on request



Name	Avg Load	Min Load	Max Load
SF	2.6	1.5	4.5
NJ	1.8	0.5	3.2
Paris	3.1	2.4	3.8

SQL query "summarizes" data

Name	Load	Web?	SMTP?	Version	…
swift	2.0	0	1	6.2	
falcon	1.5	1	0	4.1	
eagle	4.5	1	0	6.0	

San Francisco

Name	Load	Web?	SMTP?	Version	…
gazelle	1.7	0	0	4.5	
zebra	3.2	0	1	6.2	
gnu	0.5	1	0	6.2	

New Jersey

Computation done locally; query results flow back up

Copyright © 2006 University of Glasgow

Distributed Monitoring and Aggregation

- Very active research area
- Lots of potential for efficiently managing large scale systems
  - Networks
  - Web server farms
  - Clusters

…no off-the-shelf solutions yet; just research prototypes

Copyright © 2006 University of Glasgow

Deployment Considerations

- Peer-to-peer applications assume network provides transparent end-to-end connectivity
  - The original IP model

- Problem: widespread deployment of NAT and Firewalls breaks this transparency
  - NAT prevents inbound connections; cannot address hosts behind NAT
  - Firewalls can prevent both in- and out-bound connections
  - Makes it difficult to deploy peer-to-peer applications
    - Sometimes intentionally, sometimes unfortunate side-effect
    - Need both political and technical fixes

Copyright © 2006 University of Glasgow

NAT Traversal

- Growing prevalence of NAT fragmenting the Internet
  - Complicates applications since they cannot easily name/access peers
  - Hosts no longer have unique addresses
  - Bidirectional connectivity not assured, may vary by protocol or direction
  - Especially affects protocols with dynamic connections ⇒ peer-to-peer
- Unintentional breakage
  - Working around shortage of IPv4 addresses

- Problem for IP telephony and peer-to-peer communities
  - Numerous solutions ("kludges") under development:
    - STUN
    - TURN
    - ICE

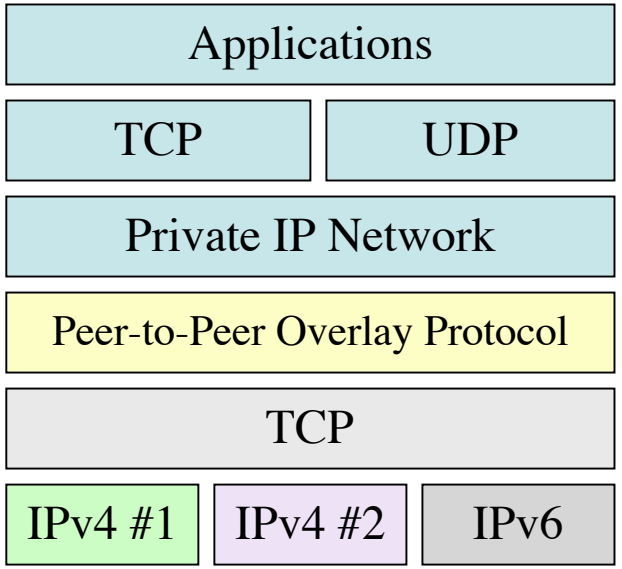
    } Methodologies for detecting presence of NAT, deducing it's behaviour, and establishing connectivity
  - Need signalling for NAT detection and peer-to-peer connection setup

Copyright © 2006 University of Glasgow

NAT Traversal: Overlay Routing

- Seek generic way to traverse NAT boxes; re-establish end-to-end transparent connectivity
  - Implemented as reusable middleware
  - Application independent
  - Allow existing applications to run unchanged
  - Simple to deploy while enforcing security policy

- Possible to use peer-to-peer techniques to build an overlay network; run IP on the overlay
  - Ugly, but solves many problems…



Copyright © 2006 University of Glasgow

Firewalls and Security

- Firewalls *intentionally* break connectivity for security reasons
- Many peer-to-peer applications try to work around this:
  - Dynamically chosen ports
  - Tunnelling in HTTP or other protocols
- *This is bad!*

- Leads to an arms race:
  - Peer-to-peer application evades firewall by tunnelling
  - Firewall gets more sophisticated, looks inside higher level protocol
  - Higher level protocol later modified; can't be deployed because firewalls think the new version is an attempt to tunnel a p2p application
    - E.g. how could we modify HTTP today?

- A social problem; no technical solution

Copyright © 2006 University of Glasgow

Summary

- The distributed hash table abstraction
  - Concepts
  - Example protocols:
    - Chord
    - CAN
    - Tapestry
- Uses and motivating example systems:
  - OceanStore
  - Astrolabe
- Deployment considerations

Peer-to-peer protocols represent interesting design evolution, potentially useful for grid computing systems

Copyright © 2006 University of Glasgow

References

1. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications”, Proceedings of ACM SIGCOMM, San Diego, CA, USA, August 2001.
2. S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, “A Scalable Content-Addressable Network”, Proceedings of ACM SIGCOMM, San Diego, CA, USA, August 2001.
3. B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph and J. D. Kubiatowicz, “Tapestry: A Resilient Global-Scale Overlay for Service Deployment”, IEEE Journal on Selected Areas in Communications, Vol. 22, No. 1, January 2004.
4. J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells and B. Zhao, “OceanStore: An Architecture for Global-Scale Persistent Storage”, Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, USA, November 2000.
5. R. van Renesse, K. P. Birman, W. Vogels, “Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining”, ACM Transactions on Computer Systems, Vol. 21, No. 2, May 2003.

Read to understand the concepts, not all the details

Copyright © 2006 University of Glasgow