Frequently asked questions from the previous class survey

- Can vector clocks be reset if the count reaches Long.MAX_VALUE?
- Can vector clocks handle processes failing at runtime?
- In the case of multicasting with Vector clocks, what happens if the message is lost?

Topics covered in this lecture

- Peer to Peer (P2P) Systems
  - P2P middleware and requirements
  - Overlays
  - The P2P lookup problem
  - Implementing DHTs

P2P middleware is designed to orchestrate

- Automatic placement of resources (data items, objects, files, etc.)
- Subsequent location (discovery) of distributed resources

How different P2P generations cope with this issue

- 1st Generation
  - Maintain a centralized index of available files
  - Files are stored at the peers

- 2nd Generation
  - Systems such as Gnutella & Freenet employ partitioned distributed indexes

- 3rd Generation
  - Rely on Overlays
Requirements for P2P systems

- Functional
  - Specific behaviors or functions that must be supported
- Non-functional (or evaluation metrics)
  - Criteria that can be used to judge the operation of a system

Non-functional requirements for P2P systems

- Scalability
- Load balancing
- Dynamic host availability

Functional requirements for P2P middleware

- Locate and communicate with any resource made available to the system
  - Even though resources are dispersed over a large number of nodes
- The ability to add and remove both resources and nodes at will

Non-functional requirements: Load balancing

- Achieved via random placement of resources
- Replicas of heavily used resources are created

Accommodate highly dynamic host availability

- Host computers are free to join or leave at any time
- Provide a dependable service, from unreliable nodes
- As nodes join the system
  - Must be integrated into the system
  - Load must be redistributed to exploit their capabilities
- As nodes leave the system (voluntarily or involuntarily)?
  - Replicate their load and resources
  - Replication levels for some resources must be preserved

Systems that we will look at

- [List of specific systems or topics that will be covered.]
Systems that we will observe closely

- 1st Generation
  - Napster
- 3rd Generation
  - Chord
  - Pastry
  - Tapestry
- Unstructured P2P or 2nd Generation
  - Gnutella and BitTorrent

Napster

- First application in which demand for massively scalable storage and retrieval arose
- Downloading of digital music files
- Became very popular soon after its launch
- At its peak
  - Several million users
  - Thousands swapped music files simultaneously

Key features of the architecture

- Centralized indexes
- Users supplied the files
  - Stored and accessed on their personal computers
- Clients add their own music files to the pool of shared resources
  - Transmit a link to Napster’s indexing service for each available file
  - Shared resources at the “edge of the internet”

Napster Architecture

Overlays

[Used in 3rd Generation P2P Systems]
A distributed algorithm takes the responsibility of locating nodes and objects. This is the routing overlay. Denotes that the middleware is a layer that is responsible for routing requests. From a client to the host that holds the requested object.

But why call it an overlay?

- Denotes that it implements a routing mechanism in the application layer.
- This is different from routing mechanisms deployed at the network level, e.g., IP.
- A logical hop in the routing overlay, encompasses multiple underlying router hops.

What does the routing overlay do?

- Ensures any node can access any object.
- Routes requests through a sequence of nodes.
- Exploits (local) knowledge at each of the intermediate nodes to locate the destination object.
- If there are multiple replicas of objects?
  - Overlay maintains knowledge of all available replicas, and then delivers request to the nearest “live” node.

Overlay vs IP Routing:

- There are several similarities between the two.
- Why have a separate mechanism?
  - The legacy nature of IP.
  - The legacy’s impact is too strong for it to be overcome.
  - Hard to support P2P applications directly.

IP Routing vs Overlay routing: Scale

- IP
  - IPv4 is limited to 2^{32} nodes.
  - IPv6 is 2^{128}
  - But addresses are hierarchically structured.
  - Much of the space is pre-allocated to meet administrative requirements.
- Overlay
  - GUID namespace is very large (2^{128} or 2^{160}).
  - The namespace is also flat allowing for it to be much more fully occupied.
**IP Routing vs Overlay routing:**

**Load Balancing**
- IP
  - Loads are determined by network topology and associated network patterns
- Overlays
  - Object locations can be randomized, so ...
  - Traffic patterns can be divorced from the network topology

**Network dynamics**
- IP
  - Routing tables are updated asynchronously on a best-effort basis
  - Typically on the order of an hour
- Overlays
  - Can be updated synchronously or asynchronously
  - Fractions of seconds

**Fault tolerance**
- IP
  - Redundancy provided by network managers
  - To handle router or network connectivity failure
  - N-fold replication is costly
- Overlay
  - Routes and object references can be replicated n-fold
  - Tolerance of (n-1) failures of nodes or connections

**Target identification**
- IP
  - Each IP address maps to exactly one node
- Overlay
  - Message can be routed to nearest replica of a target object

**Main task of a routing overlay**
1. Routing of requests to objects
2. Insertion of objects
3. Deletion of objects
4. Node additions and removals

**Calculation of Globally Unique Identifiers (GUIDs)**
- This is computed from all or part of the state of the object
- Function delivers a value that is, with a very high probability, unique
  - One way hash functions, such as SHA-1 or MD5 are often used
Why are overlay systems also called Distributed Hash Tables (DHTs)?

- Randomly distributed identifiers are used to determine resource placements and retrievals.

In the DHT model, a data item with GUID X

- Is stored at the node whose GUID is numerically close to X.
- If the replication factor is r≠1, then it is stored at the r hosts whose GUIDs are next-closest to it numerically.

A quick tour of how different P2P systems solve this

- Prefix routing
- Exploiting distance measures

Prefix routing

- Routes for delivery of messages based on values of GUIDs to which they are addressed.
- Narrow search for the next node along the route by applying a binary mask. Selects an increasing number of hexadecimal digits from the destination GUID after each hop.
- Used in Pastry and Tapestry

Exploiting different measures of distance to narrow search for next hop destination

- Chord: Numerical difference between GUIDs of the selected node and the destination node.
- CAN: Uses distance in a d-dimensional hyperspace into which nodes are placed.
- Kadmelia: Uses XOR of pairs of GUIDs as a metric for distance between nodes.

A final note about GUIDs

- These are not human readable.
- Client applications must obtain GUIDs for resources of interest through some indexing service.
- Human readable names or search requests:
  - For e.g., BitTorrent:
    - Web index search leads to a sub file containing details of desired resource GUID.
    - URL of tracker: Host that holds up to date list of network providers willing to supply the file.
The peer-to-peer (P2P) lookup problem

- How do you find a data item in a large collection of peers?
- Lookup must be scalable and decentralized
- Without hierarchy

The lookup problem: Centralized Approach

- Maintain central database
- Maintain table that maps file name to server that holds content
  - NAPSTER
- Problems
  - Reliability
  - Scalability
  - Vulnerability
  - Single point of failure
  - Database bottleneck for all requests
  - Targeted denial of service attacks

The lookup problem: Broadcast

- Flood the network with requests looking for X
- When a node receives the request:
  - Check local repository
  - If it has X, node responds back with a message
- Scaling problems
  - All discovery requests sent to ALL nodes
  - All nodes process every discovery request

Broadcast costs can be reduced by organizing nodes into a hierarchy

- Searches start at the top
- Traverse single path to the node that holds the desired data
- Directed traversal more frugal than broadcast
- Problems
  - Nodes at the top of the tree take larger fraction of load than leaf nodes
  - Requires expensive hardware
  - Loss of tree root (or node close to it) catastrophic

Distributed hash tables

- Few constraints on the structure of the keys
- Requirements
  - Data identified using numeric keys
  -Nodes must be willing to store keys for each other
Storage and retrieval in distributed hash tables

- Data items are inserted and found by specifying a unique key for the data.
- Underlying algorithm must determine which node is responsible for storing the data.

Distributed Storage using DHTs:

- Publishing a file
  - Convert file-name to numeric key
    - Using one-way hash functions like MD5 or SHA-1
  - Call lookup(key)
    - Returns IP address of node responsible for key
  - Send file to be stored at node returned by lookup

Distributed Storage using DHTs:

- Retrieving a file
  1. Obtain name of file
  2. Convert it to a key using one-way hash function
  3. Call lookup(key)
  4. Ask resulting node, from (3), for a copy of the file

Implementing DHTs

- Mapping keys to nodes
- Forwarding a lookup for a key to the appropriate node
- Building routing tables

- Must be load balanced
- Done using one-way hash functions
  - MD5 (128-bit) or SHA-1 (160-bit)
- Ensures that content is distributed uniformly
Implementing DHTs
Forwarding lookups
- Any node that receives query for key
  - Must forward it to a node whose ID is closer to the key
- Above rule guarantees that query eventually arrives at the closest node
- For e.g.:
  - Node has ID 346, and key has ID 542
  - Forwarding to node 495 gets it numerically closer

Distributed hash tables: Identifiers
- Data items are assigned an identifier from a large random space
  - 128-bit UUIDs or 160-bit SHA1 digests
- Nodes are also assigned a number from the same identifier space

Crux of the DHT problem
- Implement an efficient, deterministic scheme to
  - Map data items to node
- When you look up a data item
  - Network address of node holding the data is returned

The contents of this slide-set are based on the following references

Implementing DHTs: Building routing tables
- Multiple nodes participate in locating content
- Each node must know about some other nodes
  - To forward lookup requests
  - SUCCESSOR
    - The node with the closest succeeding ID
  - Other nodes
    - For efficiency in routing