Distributed Systems

Monsoon 2024

Lecture 14 and 15

International Institute of Information Technology

Hyderabad, India

Reading material:
Karger et al., STOC 1997

How to Locate Data

- A distributed system offers a good abstraction to store and access data.
- Such a system can have commodity machines store data and serve data to other users.
- One way to enable such a system is to provide a simple interface to query the system for data.
- Data itself is stored as key-value pairs.

Distributed Data: Evolution

- The first generation systems that store and serve data usually had some degree of centralization.
- Servers store an index file.
- Example of such a system is Napster.

- The second generation systems moved out of centralization.
- Peers work collaboratively to locate keys and disseminate information.
- Two kinds of systems: structured and unstructured.
- Such systems also known as overlay networks.

Distributed Data: Evolution

- Third generation:
  - In addition to being an overlay, focus is also on other features such as security/anonymity/ and the like.
  - The user and the data server should not be revealing their identity.

A First Generation System: Napster

- A search engine dedicated to MP3 files.
- Users could share MP3 files with each other.
- Each user on connecting to Napster shares the list of the MP3 files they own with the Napster broker.
- A user can search for MP3 files through the broker which maintains an index.
- The actual file transfer happens between the client that has the file and the client that needs the file.
  - Broker not involved in the actual transfer.

Napster

- Multiple broker servers can exist at the same time.
- Several optimizations exist
  - Clients can connect to the least loaded broker.
  - Download also possible from machines that are inside a private network.
- However, the legality of the entire sharing service was heavily up for question.
- Plus, there is no anonymity for any entity.
- Napster was shut down in 2001 due to legal troubles over copyright issues.

Second Generation Systems

- No centralized server.
- Peers have to share information with each other to locate and download shared files.
- Two strategies in general
  - Unstructured networks
  - Structured networks

Distributed Data: Evolution

- Unstructured Systems:
  - Arbitrary set of connections between nodes.
  - Information dissemination via gossiping style protocols.
  - Rumor spreading/ epidemic propagation
- Structured Systems:
  - Connections are well-defined. Logically at least.
  - Known as overlay networks.

Structured Vs. Unstructured

- Structured P2P overlays: These follow a regular topology.
- Example topologies include the hypercube, mesh, butterfly, and such graphs.
- The topology also guides the procedure for object storage and object search.
  - Object storage and search strategies are intricately linked to the overlay structure as well as to the data organization mechanisms.
- Unstructured P2P overlays: These do not follow any regular network topology.
  - Loose guidelines for object search and storage
  - Search mechanisms are ad-hoc, variants of flooding and random walk

Characteristics of P2P Networks

- P2P network is an application-level organization of nodes to form a network.
- Such a network allows nodes to flexibly share resources.
- The name P2P comes from the idea that all nodes are equal; communication directly between peers (no client-server)
- The network allows location of arbitrary objects; no DNS-type service is required.
- The network can be seen as a large combined storage, CPU power, other resources, without scalability costs.
- Dynamic insertion and deletion of nodes, as well as of resources, at low cost.

How to Place Data

- Data identified by indexing, which allows physical data independence from applications.
- Three types of indexing possible
  - Centralized indexing, e.g. like in DNS servers
  - Distributed indexing.
    - Indexes to data scattered across peers.
    - Access data through mechanisms such as Distributed Hash Tables (DHT).
    - These differ in hash mapping, search algorithms, diameter for lookup, fault tolerance, churn resilience.
  - Local indexing.
    - Each peer indexes only the local objects.
    - Remote objects need to be searched for. Used commonly in unstructured overlays (E.g., Gnutella) along with flooding search or random walk search.

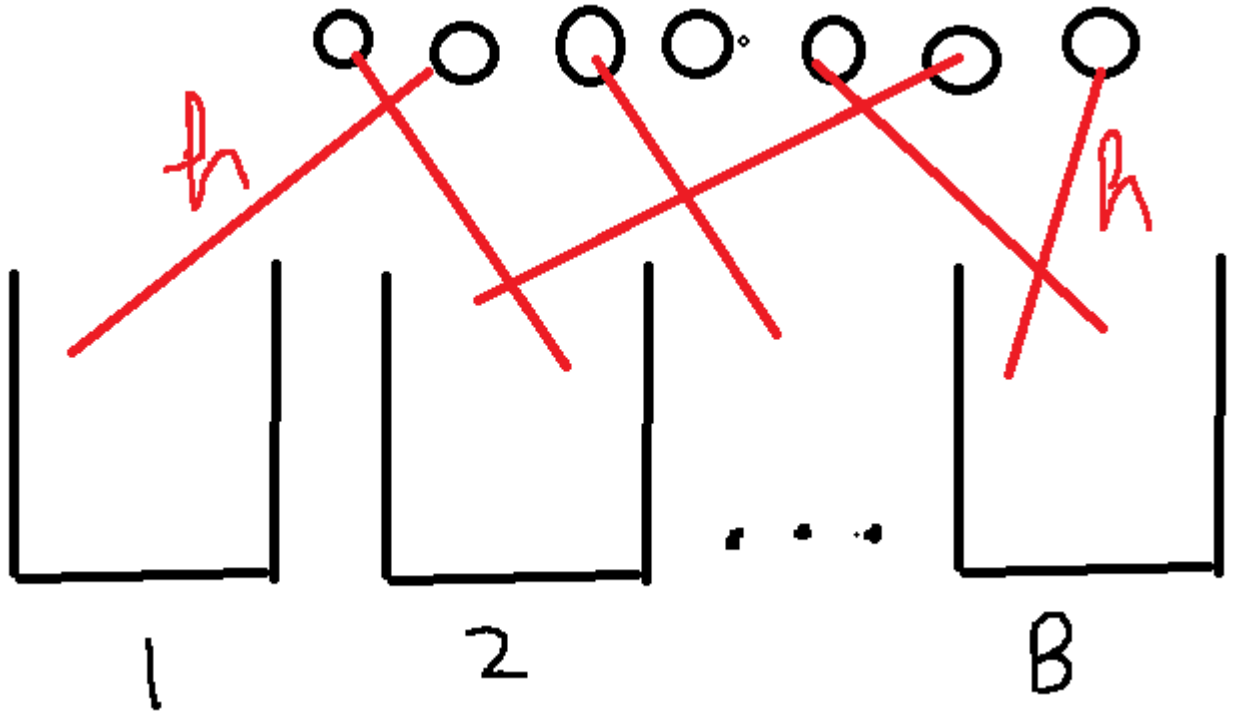
How to Place Data

- Another classification is via semantic vs. semantic-free nature of indexing.
- Semantic indexing is human readable, e.g., filename, keyword, database key.
  - Supports keyword searches, range searches, approximate searches.
- Semantic-free indexing is not human readable.
  - Corresponds to index obtained by use of good hash functions to help in the search process.

How to Place Data – Hashing

- Consider a single server that has many objects and clients that access these objects.
- One often uses caching, and possibly a set of caches, to improve the performance of the server.
- We hope that the objects are distributed uniformly across the caches, so that each cache is responsible for a nearly equal share.
- In addition, clients need to know which cache to query for a specific object.

How to Place Data



- Hashing provides the following solution.
- The server can use a hash function that evenly distributes the objects across the caches.
- Clients can use the same hash function to discover which cache stores a object.

How to Place Data

- The solution however has certain drawbacks.
- Consider what happens when the machines that work as caches change.
- Suppose the hash function uses a function modulo p, e.g. $f(x) = ax + b \pmod p$, where p is the number of machines that cache objects.
- When the number of machines change, the range of the hash function (p in the example) has to change.
- In that case, almost every item would be hashed to a new location.
- Suddenly, all cached data is useless because clients are looking for it in a different location.
- While this information is not current at some client, then clients may even contact a wrong server in their access procedure.

Consistent Hashing – A Solution

- Define a *view* to be the set of caches of which a particular client is aware.
- Consider hash functions that have the following properties:
- Smoothness: When a machine is added to/removed from the set of caches, the expected fraction of objects that must be moved to a new cache is the minimum needed to maintain a balanced load across the caches.
- Spread: Over all the client views, the total number of different caches to which an object is assigned is small.
- Load: The number of distinct objects assigned to a particular cache is small.
- Such hash functions are called consistent hash functions.

Consistent Hashing – A Solution

- The “smoothness” property implies that small changes in the set of caching machines require only small changes in the location of cached objects.
- The “spread” property implies that even in the presence of inconsistent views of the world, references for a given object are directed only to a small number of caching machines.
- The “load” property implies that each cache is assigned nearly the same number of objects.

- Do such hashing schemes exist?

A Construction

- Let I be the set of items and B be the set of buckets.
- A view V is any subset of the buckets B.
- A ranged hash function is a function of the form $f : 2^B \times I \rightarrow B$.
  - Such a function specifies an assignment of items to buckets for every possible view.
  - f(V, *i*) is the bucket to which item *i* is assigned in view V. Written as $f_V(i)$ from now on.
- A family of such ranged hash functions is called as a ranged hash family.
- We call a random range hash function as a function drawn uniformly at random from a ranged hash family of functions.

A Construction

- We consider ranged hash families that have the following properties.

- Balance: A ranged hash family is balanced if, given a particular view V, a set of items, and a randomly chosen function selected from the hash family, with high probability the fraction of items mapped to each bucket is O(1/|V|).
- The balance property requires that the functions in the hash family distribute items among buckets in a balanced fashion.

A Construction

- We consider ranged hash families that have the following properties.
- Monotonicity: A ranged hash function f is monotone if for all views $V_1 \subseteq V_2 \subseteq B$, $f_{V_2}(i) \in V_1$ implies that $f_{V_1}(i) = f_{V_2}(i)$.
- If every function from a ranged hash family is monotone, then the family is called a monotone family.
- The monotonicity property insists that if items are initially assigned to a set of buckets V_1 and then some new buckets are added to form V_2, then an item may move from an old bucket to a new bucket, but not from one old bucket to another old bucket.
- This reflects an intuition about consistency: when the set of usable buckets changes, items should only move if necessary to preserve an even distribution.

A Construction

- We consider ranged hash families that have the following properties.
- Spread: Let $V_1,. . .,V_v$ be a set of views, altogether containing C distinct buckets and each individually containing at least C/t buckets.
- For a ranged hash function f and a particular item i, the spread $\sigma(i)$ is the quantity $|\{f_{V_j}(i)\}_{j=1}^{v}|$.
- The spread of a hash function $\sigma(f)$ is the maximum spread of an item.
- The spread of a hash family is σ if, with high probability, the spread of a random hash function from the family is σ.

A Construction

- The idea behind spread is that there are v people, each of whom can see at least a constant fraction (1/t) of the buckets that are visible to anyone.
- Each person tries to assign an item i to a bucket using a consistent hash function.
- The property says that across the entire group, there are at most $\sigma(i)$ different opinions about which bucket should contain the item.
- Clearly, a good consistent hash function should have low spread over all items.

A Construction

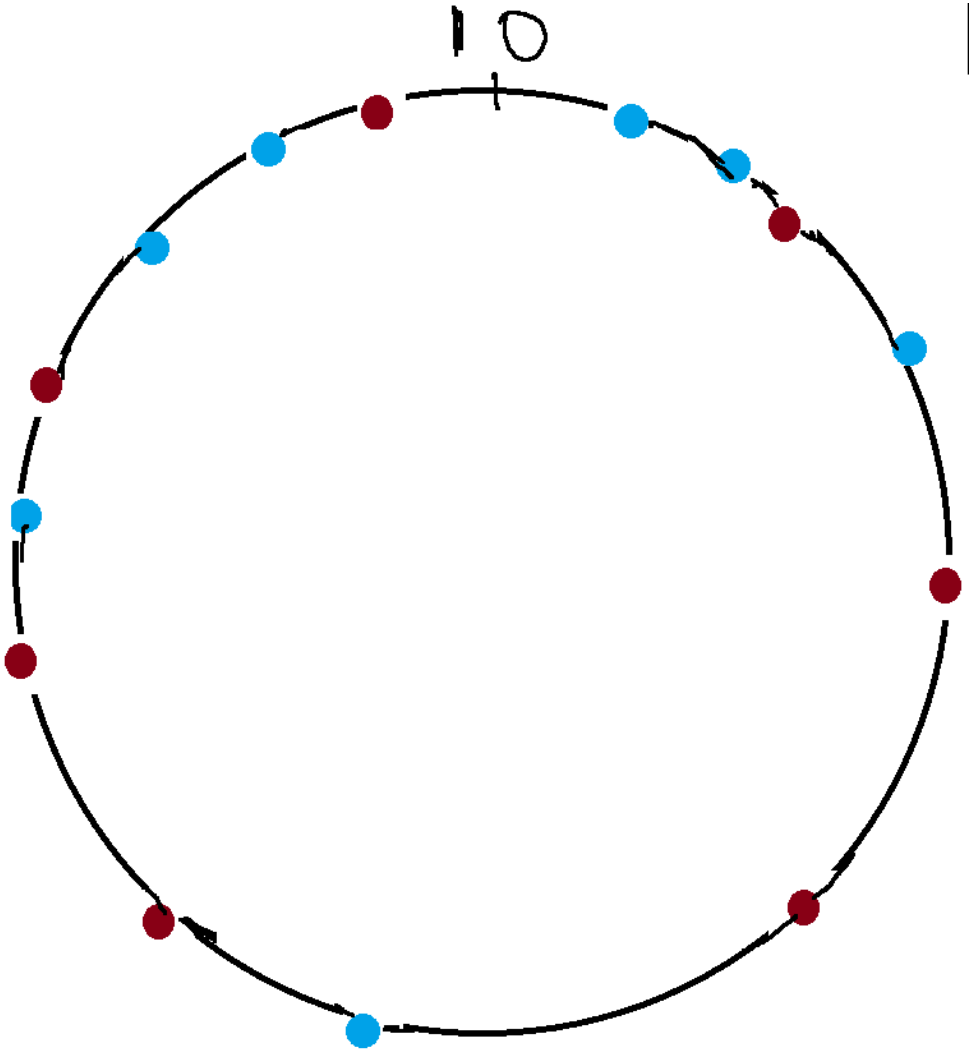
- We consider ranged hash families that have the following properties.
- Load: Define a set of V views as before.
- For a ranged hash function f and bucket b, the load $\lambda(b)$ is the quantity $|U_{v \text{ in } V} f_v^{-1}(b)|$.
- The load of a hash function is the maximum load of a bucket.
- The load of a hash family is λ if with high probability, a randomly chosen hash function has load λ.
- Note that $f_v^{-1}(b)$ is the set of items assigned to bucket b in view V.

A Construction

- The load property is similar to spread.
- The same V buckets are back, but this time consider a particular bucket b instead of an item.
- The property says that there are at most $\lambda(b)$ distinct items that at least one person thinks belongs in the bucket.
- A good consistent hash function should also have low load.

A Construction

- Suppose that we have two random functions r_B and r_I.
- The function r_B maps buckets randomly to the unit interval, [0,1]
- The function r_I does the same for items.
- $f_v(i)$ is defined to be the bucket $b \in V$ that minimizes $|r_B(b) - r_I(i)|$.
- In other words, i is mapped to the bucket "closest" to i.



A Construction

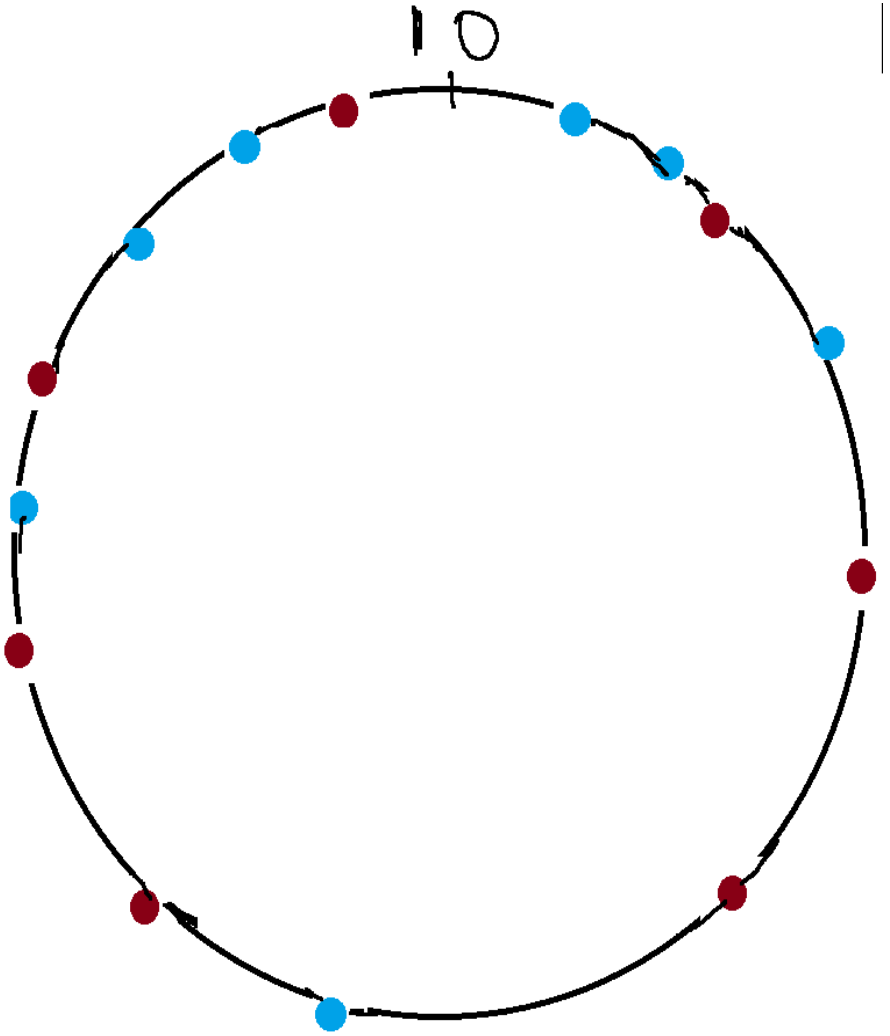
- For the purposes of analysis, we will have each bucket mapped to several points in [0, 1].
- If the number of buckets in the range is always less than C, we will use k.log(C) points for each bucket for some constant k.
- The easiest way to think of this arrangement is to consider replicating each bucket by klog(C) times.
- The function r_B maps each replicated bucket randomly.

A Construction

- The construction above is a ranged hash family that is monotone, has a spread of O(log C), and a load of O(log C), with high probability.
- C is the number of buckets.
- The proof largely relies on using Chernoff bounds and the independence of the functions r_B and r_I.
- Refer to the paper for details.

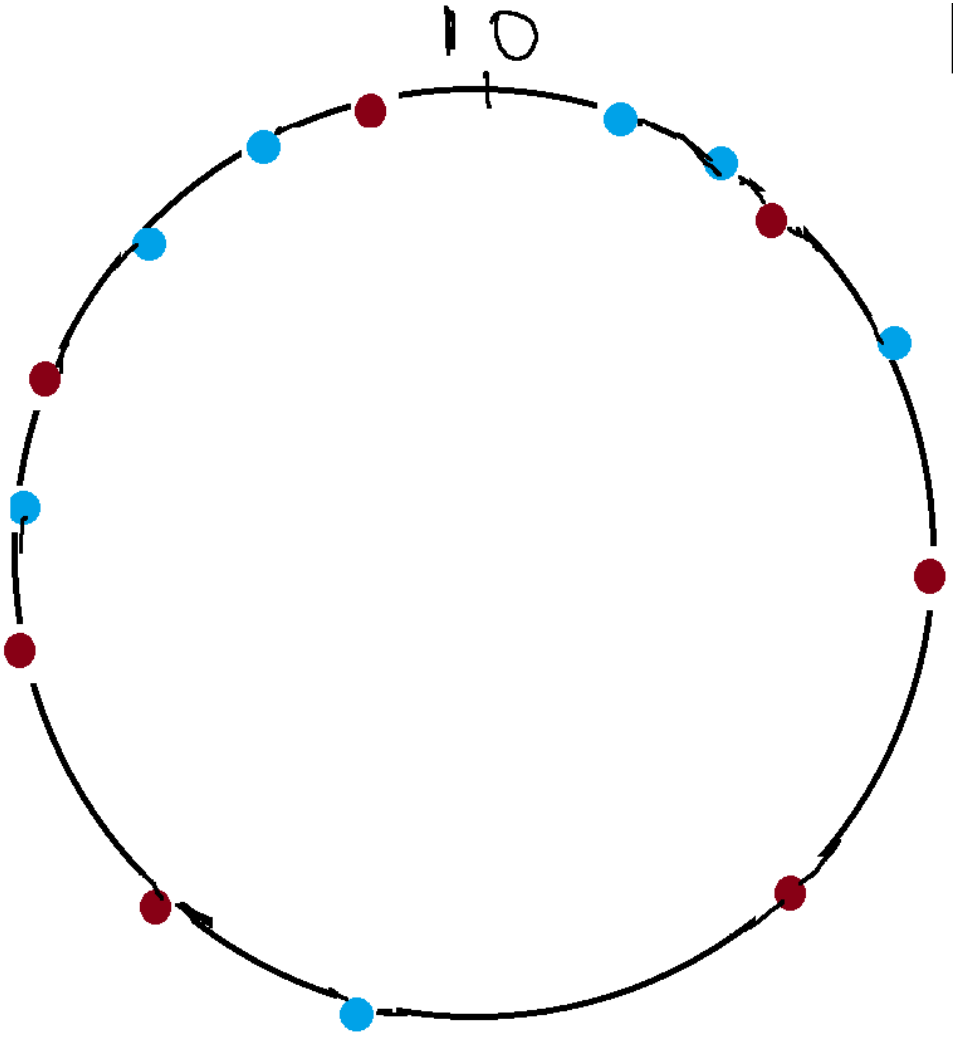
First Example – Chord

- Uses ideas from consistent hashing.
- Node address as well as object value is mapped to a logical identifier in a common space using a consistent hash function.
- When a node joins or leaves the network of n nodes, only 1/n keys have to be moved.
- Two steps involved.
  - Map the object value to its key
  - Map the key to the node in the native address space using lookup



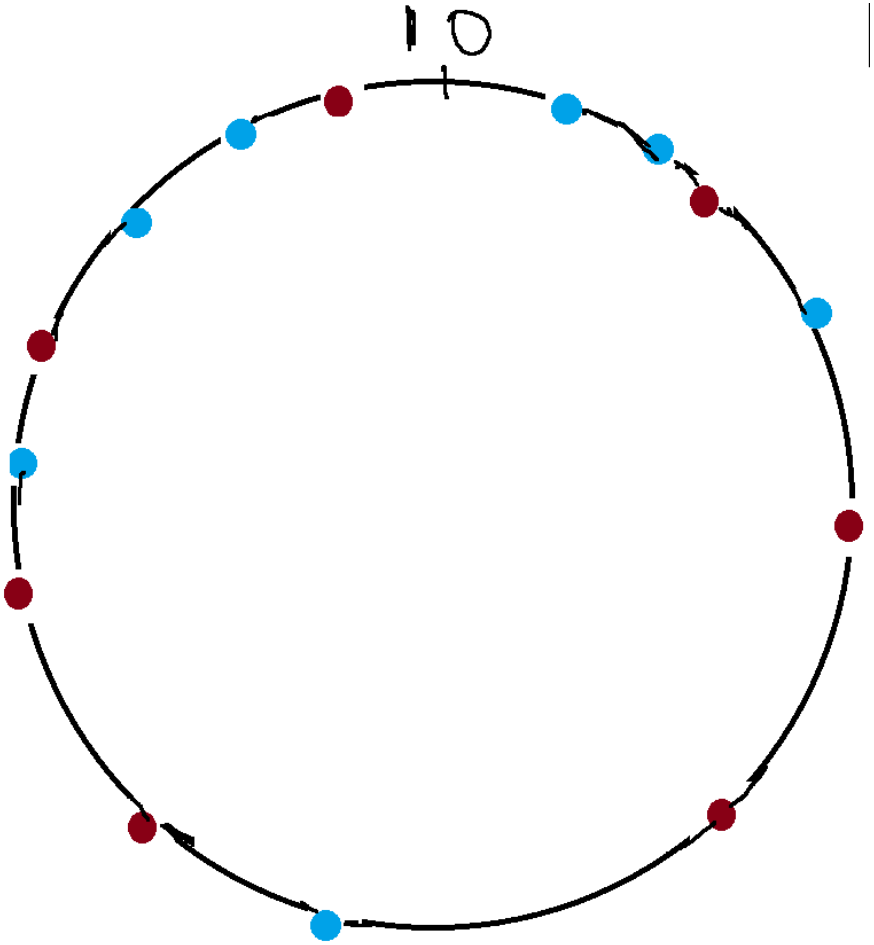
How to Place Data

- Common address space is an m-bit identifier (2^m addresses), and this space is arranged on a logical ring mod(2^m).
- A key k gets assigned to the first node such that the node identifier equals or is greater than the key identifier k in the logical space address.



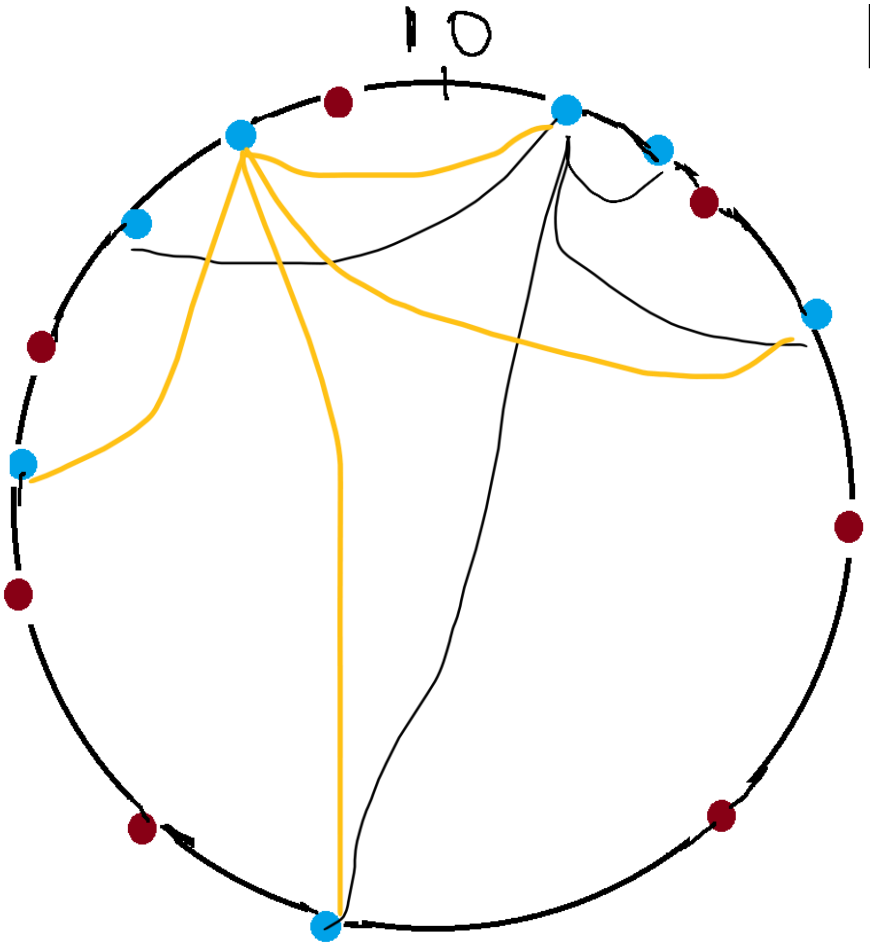
Chord – Lookup

- Each node tracks its successor on the ring.
- A query for key x is forwarded on the ring until it reaches the first node whose identifier y ≥ key x mod(2^m).
- The result is returned to the querying node on the reverse of the path that was followed by the query.
- This mechanism requires O(1) local space but O(n) hops.



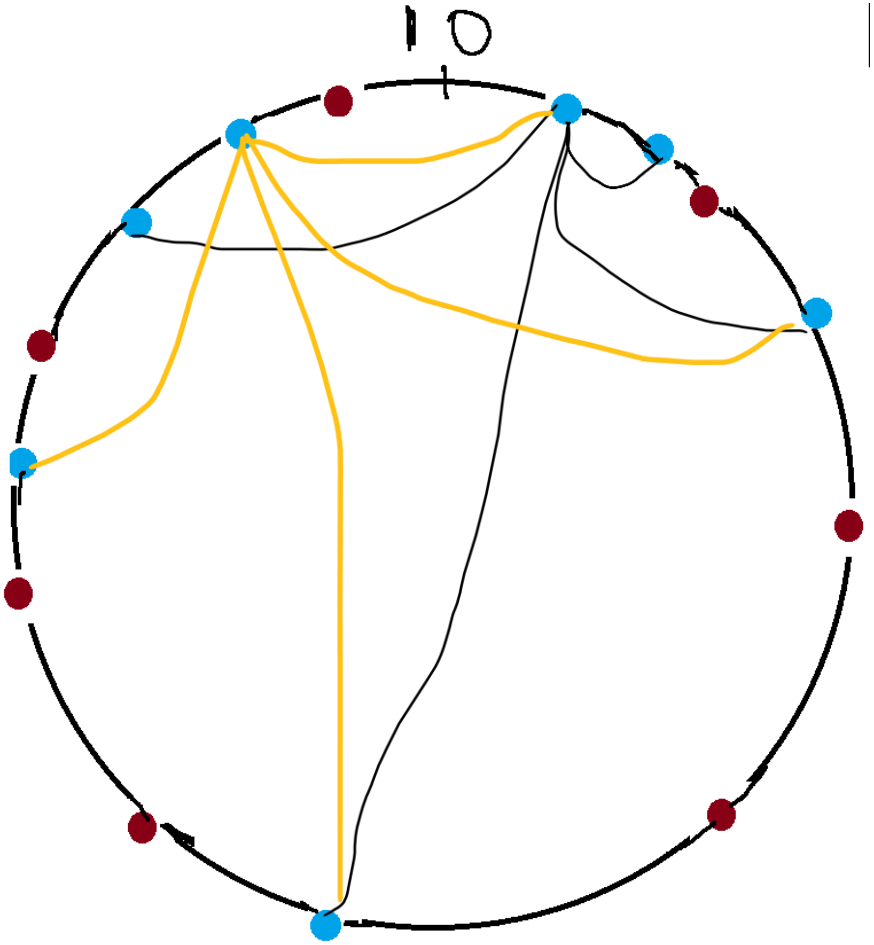
Chord – Improved Lookup

- Each node i maintains a routing table, called the finger table, with O(log n) entries, such that the xth entry ($1 < x \leq m$) is the node identifier of the node succ($(i + 2^{x-1}) \bmod 2^m$).
- This is denoted by i.finger[x] = succ($(i + 2^{x-1}) \bmod 2^m$).
- This is the first node whose key is greater than the key of node i by at least $2^{x-1} \bmod 2^m$.



Chord – Improved Lookup

- Can visualize the process of searching (routing) as a binary search.
- First, contact the peer via the finger table entry corresponding to x = 1.
- If this node is not the one that has the required file, then the next node searched is the finger table entry x = 2.
- Complexity: O(log n) message hops at the cost of O(log n) space in the local routing tables



Chord – Managing Churn

- Processing node join and leave
- When a new node joins, it has to:
  - Find its predecessor and successor
  - Own the items that will now be mapped to the new node.
  - Establish its finger table.
    - Can build its finger table by using the finger table of its successor.
  - When a node leaves, it has to
    - Inform its predecessor and successor
    - Transfer items that it owns to its successor

Other Examples of Overlay Networks

- Other structured overlay network models include
  - CAN – Content Addressable Network
  - Tapestry
  - And many more

Gnutella – An Unstructured Overlay Network

- A joiner connects to some standard nodes from Gnutella directory
- Ping used to discover other hosts; allows new host to announce itself
- Pong in response to Ping; Pong contains IP, port #, max data size for download
- Query messages used for flooding search
- QueryHit are responses. If data is found, this message contains the IP, port#, file size, download rate, etc.
- Path used is reverse path of Query.
- Provisions for handling nodes staying behind a firewall or in a private network.

Gnutella – An Unstructured Overlay Network

- Semantic indexing possible : keyword, range, attribute-based queries
- Easily accommodate high churn
- Efficient when data is replicated in network
- Good if user satisfied with "best-effort" search
- Network is not so large as to cause high delays in search

Search in Gnutella

- Flooding: with checking, with TTL or hop count, expanding ring strategy
- Random Walk: k random walkers, with checking
- Relationships of interest
  - The success rate as a function of the number of message hops, or TTL.
  - The number of messages as a function of the number of message hops, or TTL.
  - The above metrics as the replication ratio and the replication strategy changes.
  - The node coverage, recall, and message efficiency, as a function of the number of hops, or TTL; and of various replication ratios and replication strategies.
- Overall, k-random walk outperforms flooding

BitTorrent

- An unstructured overlay network like Napster but is very popular and navigated the legal challenges carefully.
- The aim of bit torrent is to run a legal network and not circumvent local laws.
- Used to store and forward several types of files including video, OS images, open source packages, datasets, educational content, and so on.

BitTorrent

- For each file, a torrent descriptor file has to be created.
- This descriptor file is also knows as the torrent.
- Contains file metadata and an SHA-1 output of the file contents.
- Once a server hosts a file, it creates the torrent and shares the torrent with other servers.
- The torrent files are tracked by dedicated trackers by storing a mapping between the torrent and the locations where the file is available.

BitTorrent – Mechanism

- BitTorrent supports a swarm of hosts. Every host participates in the system.
- Big files are broken into multiple pieces, each piece is limited in size to 256 KB.
  - File is no longer an atomic unit.
  - This small sized pieces is supposed to improve network usage.
- The pieces are distributed across peers.
- Each peer hosts pieces of different files.

BitTorrent Mechanism

- Having the file at multiple hosts allows a client to download the pieces simultaneously from the hosts.
- The trackers coordinate the transfer of the file along with a mapping of the piece to the hosts.
- Users can use regular search mechanisms that includes Google searches to find torrent descriptor files.
- If a server has a new file, it hosts it. Subsequently, it distributes the torrent file to let client machines and trackers know about it.

BitTorrent – Mechanism

- Nodes in BitTorrent can use multiple strategies to promote the behavior of users in the network.
- The upload bandwidth from a host to a client can be a function of how the client participates in the network.

- Anonymity and privacy are not part of the BitTorrent network design and mechanism.
- The legal onus is on the site that indexes and hosts the torrents.

FreeNet – A Third Generation System

- One of the popular third generation distributed data system is Freenet.
- Recall that third generation systems have to ensure anonymity and privacy of both the sender and receiver of data/files.
- Freenet offers anonymity and deniability as well.
- Think of applications such as dark web, anonymous browsing and hosting.

Freenet Node

- Each node maintains an index and a routing table.
- Routing table entries store the address of some nodes in the network and the files (keys of files) they have.
- Queries are routed to neighbors using flooding with a Time-to-Live (TTL) parameter.
- Identifiers attached to queries so as to prevent cycles while forwarding via flooding.
- Once a node N finds a file, the node has to send the contents to the client.
- However, cannot reveal the identity of N or other intermediaries in the network.

File Transfer

- Anonymity by obfuscation is what is used in Freenet.
- The result of the query follows the same path as the original search request as in Gnutella.
- Each node on the way claims to be the owner of the file -- this hides the real owner.
- Every node on the way caches a copy of the file and creates an entry in its routing table with the key and id of the data source.

Freenet

- A positive side effect is that popular data will tend to get replicated at many nodes, and it is thus easy to locate and fetch it.
- However, on the flip side, routing tables will get bigger because new entries shall get created and added to routing tables all the time.

Summary

- Overlays have been popular due to their peer model
- Truly distributed in that sense
- Structured vs. unstructured
  - Structured based on strong theory
  - Can result in good bounds on routing
  - Consistent hashing is a big development, two decades ago.
- There are several attempts to unify the structure and other processes of structured overlay networks.