Distributed System : Introduction

Houmin Wei

Electronics Engineering & Computer Science
Peking University

北京大學
PEKING UNIVERSITY

October 31, 2017

Outline

Definition of P2P

A P2P(Peer to Peer) System exhibits the following characteristics:

- High degree of autonomy from central servers
- Exploits resources at the edge of the network
  - Storage, CPU cycles, human presence
- Individual nodes have itermittent connectivity

Not strict requirements, instead typical characteristics

Above characteristics allow us to distinguish P2P systems from other similar systems.

Applications of P2P

- P2P File Sharing and content distribution:
  BitTorrent, Napster, Gnutella, KaZaA
- P2P Communication:
  Typical instant messaging setup: Skype
- P2P Computation
- P2P Collaboration

Napster: Overview

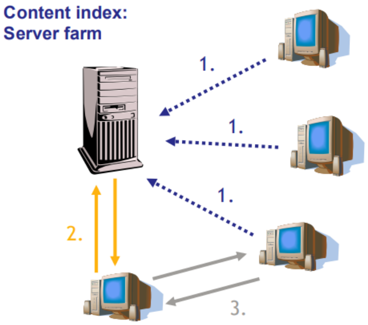
- The first P2P file sharing application(MP3 only)
- Made the term 'peer-to-peer' known(1999, Shawn Fanning)
- Based on central index server(actually a server farm)
- User registers with the central server
  - Give list of files to be shared
  - Central server know all the peers and files in network
- Searching based on keywords
- Search results were a list of files with information about the file and the peer sharing it
  - For example, encoding rate, size of file, peer's bandwidth
  - Some information entered by the user, hence unreliable

Napster: Framework

Original Napster design

1. Peers register with central server, give list of file to be shared.
2. Peers send queries to central server which has content index of all files.
3. File transfers happen directly between peers.

Last point is common to all P2P networks and is their main strength as it allows them to scale well.

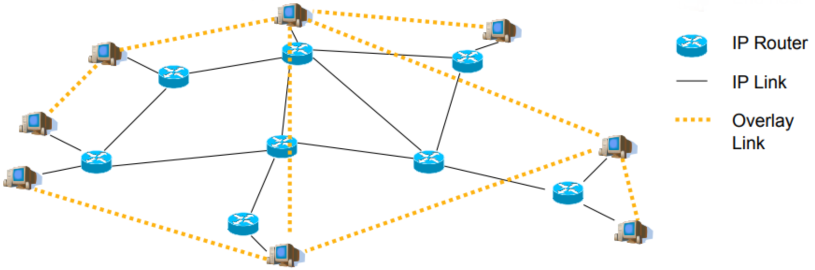


Napster: Discussion

- Pros
  - Consistent view of the network
    Central server always knows who is there and who is not.
  - Fast and efficient searching, Search scope is O(1)
    Central server always knows all available files.
  - Answer guaranteed to be corrent
    Nothing found means none of the current on-line peers in the network has the file.
- Cons
  - Single point of failure
  - Server needs enough computation power to handle all queries
  - Server maintains O(N) State

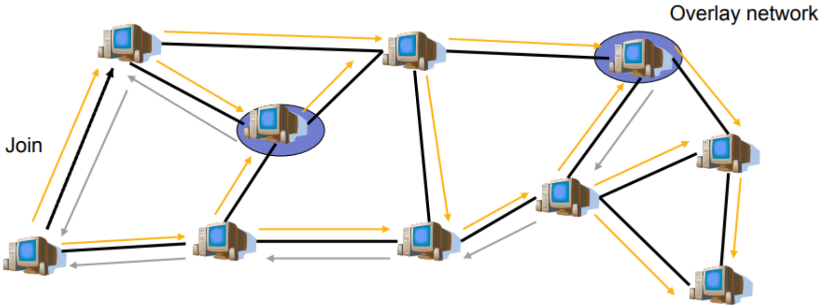
Gnutella: Overview

- Napster is centralize, Gnutella is fully distributed.
- Based on overlay network



A virtual network on top of underlying IP network
- All peers are fully equal, called servents(server + client)

Gnutella: Framework



- To join, peer needs address of one member, learn others
- Queries are sent to neighbors
- Neighbors forword queries to their neighbors(flooding)
- Replies routed back via query path to querying peer

Guntella: Discussion

- Pros:
  - Fully de-centralized
  - Search cost distributed
- Cons:
  - Search scope is O(N)
  - Nodes leave often, network unstable
  - Periodic Ping/Pong consume lots of resources

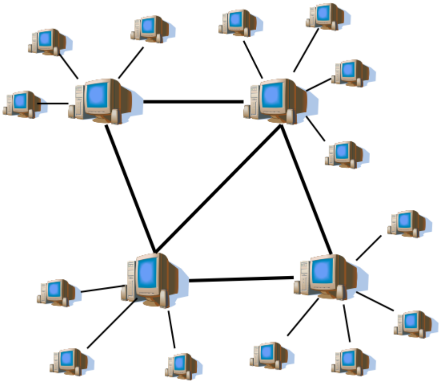
KaZaA: Overview

- Created in 2001
- Two kinds of nodes in KaZaA: Ordinary Nodes, SuperNodes
- ON is a normal peer run by a user
- SN is also a peer run by a user, but with more resources and responsibilities
- KaZaA forms a two-tier hierachy
  top level has only SN, lower level only ON
- ON belongs to one SN
- SN acts as a Napster-like hub for all its ON-children
  keeps track of files in those peers

KaZaA: Framework

Smart Query Flooding:

- Join: on startup, client contacts a SN
- Publish: send list of files to SN
- Search: send query to SN, SN flood query amongst themselves
- Fetch: get the file directly from peers



KaZaA: Discussion

- Pros:
  - Efficient searching under each SN
  - Flooding restricted to SN only
  - Efficient searching with 'low' resource usage
- Cons:
  - Still no real guarantees on search scope or search time

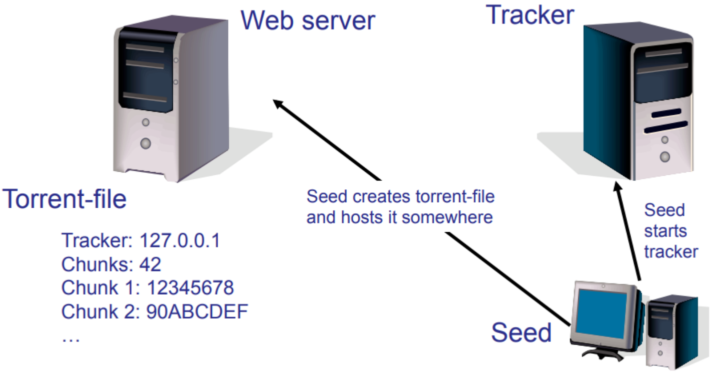
BitTorrent: Overview

2 basic ways to find objects:

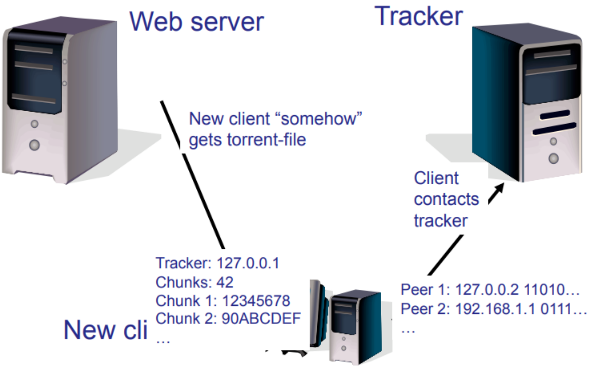
- Search for them with keywords that match objects's description
- Address them using their unique name(cf. URLs in Web)

- Swarming:
  - Join: contact centralized tracker server, get a list of peers.
  - Publish: Run a tracker server.
  - Search: Out-of-band, E.g. use Google to find a tracker for the file you want.
  - Fetch: Download chunks of the file from your peers. Upload chunks you have to them.

BitTorrent: Framework



BitTorrent: Framework



BitTorrent: Tit-for-Tat

- A is downloading from some other people
  A will let the fastest N of those download from him.
- Be optimistic: occasionally let freeloaders download
  Otherwise no one would ever start!

BitTorrent: Discussion

- Pros:
  - Works reasonably well in pratice
  - Gives peers incentive to share resources, avoids freeriders
- Cons:
  - Central tracker server need to bootstrap swarm.
  - What if tracker server fails?

Distributed Hash Tables

In BitTorrent version 4.2.0, BitTorrent introduce Trackerless torrent using DHT.

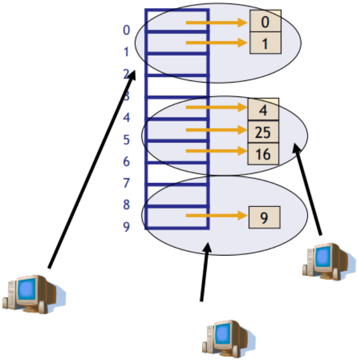
- Actual file transfer process in P2P network is scalable
  File transfers directly between peers
- Searching does not scale in same way
- Put another way: Use addressing instead of searching
- Original motivation for DHTs: More efficient searching and object location in P2P networks
- For a special resource, the tracker record the nodes/peers associated with the resource.
- If the tracker fails, we can lookup the DHT for the nodes/peers info.

Recall Hash Table

- allow insertions, deletions, and lookup in constant time.
- fixed-size array, elements of array also called hash buckets.
- Hash funtion maps keys to elements in the array.
- Properties of good hash functions
  - Fast to compute
  - Good distribution of keys into hash table
  - Example: SHA-1 algorithm

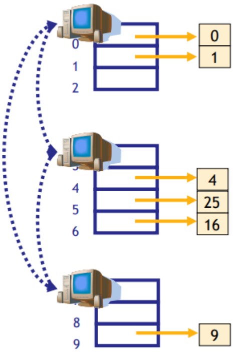
DHT: Idea

- Hash tables are fast for lookups.
- Idea: Distribute hash buckets to peers.
- Result is Distributed Hash Table.
- Need efficient mechanism for finding which peer is responsible for which bucket and routing between them.



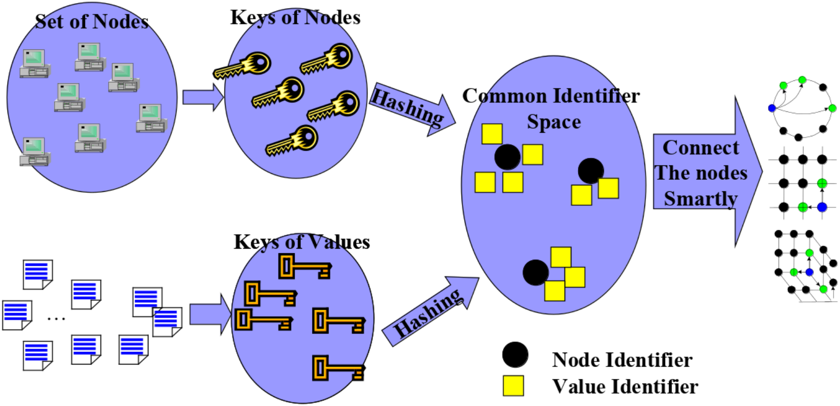
DHT: Principle

- In a DHT, each node is responsible for one or more hash buckets.
  As nodes join and leave, the responsibilities change.
- Nodes communicate among themselves to find the responsible node.
  Scalable Communications make DHTs efficient.
- Hash buckets distributed over nodes.
- Nodes form an overlay network.
  Route messages in overlay to find responsible node.



Structured Overlay Networks/DHTs

Chord, Pastry, Tapestry, CAN,
Kademlia, P-Grid, Viceroy



DHT: Overview

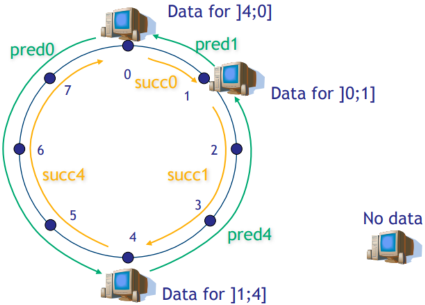
- All DHTs provide the same abstraction
  - Put(key, value)
  - value = Get(key)
- Difference is in overlay routing scheme
  - Chord => ring
  - Kademlia => tree
  - CAN, Tapstry, Pastry ...

Chord: Basics

- Chord use SHA-1 hash function
  - Results in a 160-bit object/node indentifier
  - Same hash function for obejects and nodes
- Node ID hashed from IP address
- Object ID hashed from object name
- SHA-1 gives a 160-bit indentifier space
- organized in a ring which wraps around
  - Nodes keep track of predecessor and successor
  - Node responsible for objects between its predecessor and itself
  - Overlay is ofen called Chord Ring

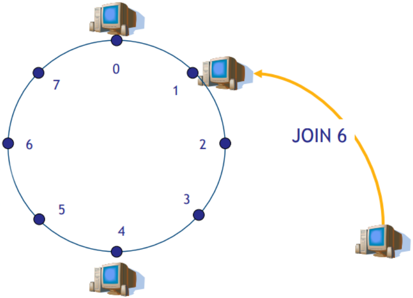
Node Join

- Existing network with nodes on 0,1 and 4
- Hash of new node to join: 6
- Known node in network: Node 1
- Contact Node1

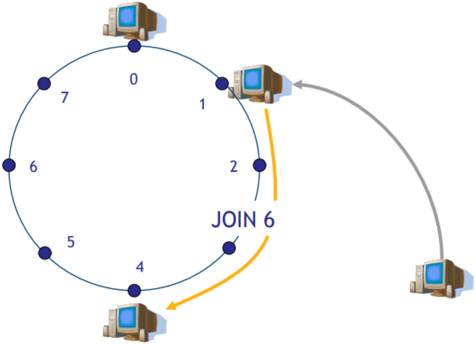


Node Join: Contact Known node

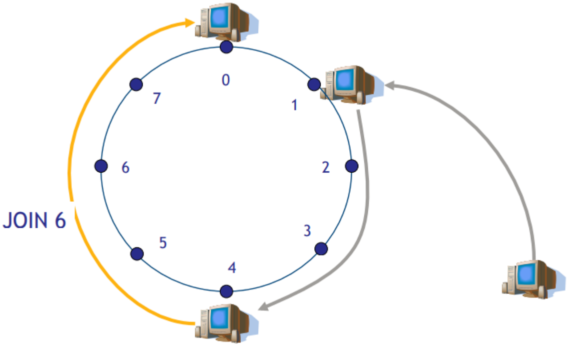
- Existing network with nodes on 0,1 and 4
- Hash of new node to join: 6
- Known node in network: Node 1
- Contact Node1



Node Join: Contact Known node

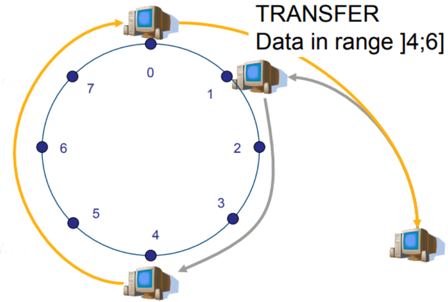


Node Join: Contact Known node

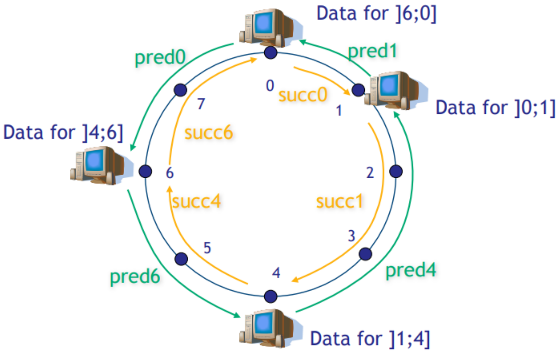


Node Join: Join Successful + Transfer

- Joining is successful
- Old responsible node transfer data that should be in new node
- New node informs Node4 about new successor

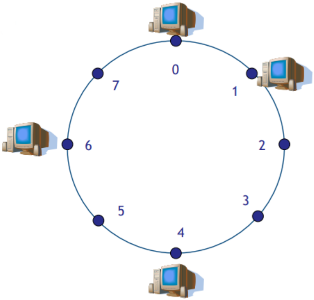


Node Join: All is Done

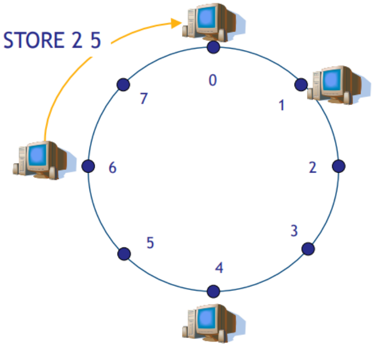


Storing a Value

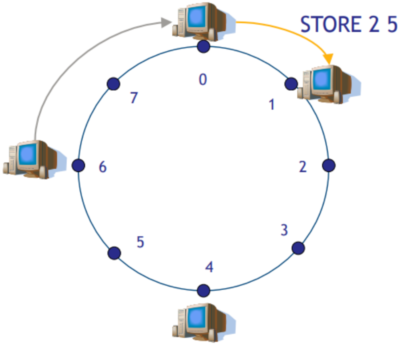
- Node6 wants to strore object with name 'FOO' and value 5
- hash(Foo) = 2



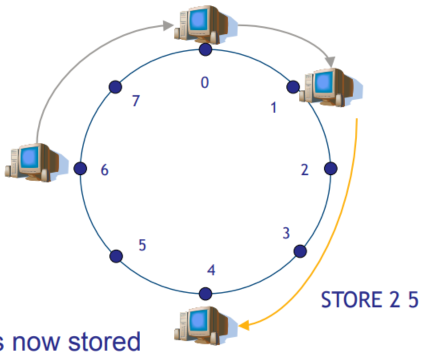
Storing a Value



Storing a Value



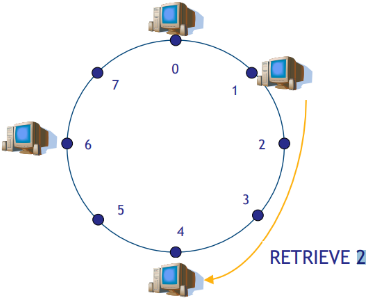
Storing a Value



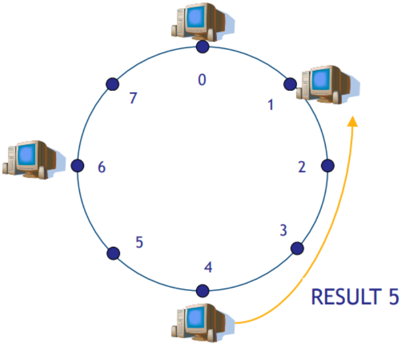
Value is now stored in node 4.

Retrieving a Value

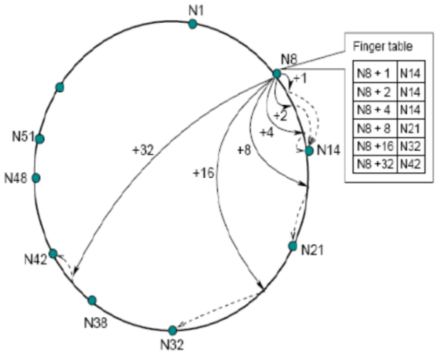
- Node1 wants to get object with name 'FOO'
- $\text{hash}(\text{Foo}) = 2$
- Foo is stored on Node4

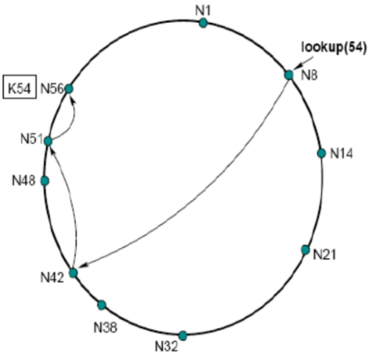


Retrieving a Value



Scalable Key Location: Finger Tables





Row i in finger table at node n contains first node s that succeeds n by least 2^{i-1} on the ring. First finger is the successor.

P2P system ends here.
Let's go back to distributed system.

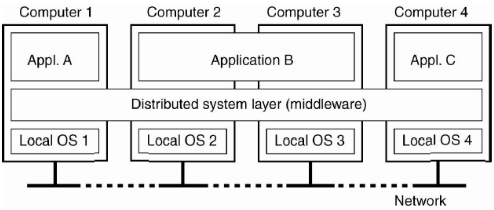
Why Distribution?

- Economics
  - Much better price/performance ratio
- Reliablity
  - One node fails, but the service goes on
- Enhanced performance
  - Tasks can be executed concurrently
- Easier modular expansion
  - Hardware and software resources can be easily added without replacing existing resources.
- Resource Sharing
  - Only one printer, share it over the network

What is a Distributed system?

Definition

A distributed system consists of a collection of autonomous computers, connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system, so that users perceive the system as single, integrated computing facility.



- Resource Sharing
- Transparency
- Scalability
- Concurrency
- Fault Tolerance

Typical Distributed Systems I

- Distributed Storage System
  - Structured Storage Systems
    - MySQL, PostgreSQL
    - Structured data
    - Strong consistency
    - Random access
    - Expansion not good
  - No-Structed Storage Systems
    - GFS, HDFS
    - Manage data using metadata by master
    - Big chunks(like 64MB), replicated copy
    - Fault tolerant automatically.
    - No random access, typically append
    - Not good for real-time system
  - Semi-Structure Storage Systems

Typical Distributed Systems II

- NoSQL(Bigtable, Dynamo, Habse)
- Good Expansion
- Random access(update, read)
- Key-Value Store, No SQL, No ACID

- In-memory Storage Systems
  - memcached, redis
  - based on memory, not disk.
- NewSQL
  - Spanner
  - Use atomic clock to realize syncronization
  - both expansion and SQL

- Distributed Computing System
  - MapReduce like: MapReduce(Hadoop), Spark
  - Graph: GraphLab, Pregel
  - Streaming: Storm
- ...

Topic: Performance

The dream: scalable throughput

Nx servers -> Nx total throughput via parallel CPU/disk/net.
So handling more load only requires buying more computers.

Scaling gets harder as N grows:

- Load im-balance, stragglers.
- Non-parallelizable code: initialization, interaction.
- Bottlenecks from shared resources, e.g. network.

Topic: Fault Tolerance

- 1000s of servers, complex net -> always something broken.
- We'd like to hide these failures from the application.
- We ofen want:
  - Availability – app can keep using its data despite failures.
  - Durability – app's data will come back to life when failures are repaired.
- Big idea: replicated servers.
  If one server crashes, client can proceed using the other(s).

Topic: Consistency

- General-purpose infrastructure needs well-defined behavior.
  E.g. 'Get(k) yields the value from the most recent Put(k,v)'
- Achieving good behavior is hard!
  - 'Replica' servers are hard to keep identical.
  - Clients may crash midway through multi-step update.
  - Servers crash at awkward moments. e.g. after executing but before replying.
  - Network may make live servers look dead.
- Consistency and performance are enemies.
  - Consistency requires communication, e.g. to get latest Put().
  - Strong Consistency often leads to slow systems.
  - High performance often imposes weak consistency on applications.

Later...

- RPC
- Paxos
- Consistent Hash
- Leader Election
- Lamport's Logic Clock
- ...

Talks about crypto.