



System Design for Large Scale

Law of Diminishing Returns

SETI@home

Napster

Context

Resume

Gnutella

Early design

Improved

Bloom filters

Distributed Hash Tables

Chord


Kademlia

Properties

Law of Diminishing Returns

Law of Diminishing Marginal Returns

The law of diminishing marginal returns is a theory in economics that predicts that after some optimal level of capacity is reached, adding an additional factor of production will actually result in

 <https://www.investopedia.com/terms/l/lawofdiminishingmarginalreturn.asp>



More entities we have working the same local and the same time, higher is the probability that they will have to synchronize between each other. In other words, after some optimal level of capacity is reached, adding new factors of production adds smaller levels of output.

SETI@home

It was a mission, basically, to find ETs...

The signals from the broad space were captured and there's the necessity to find out if the signals were merely white noise or if there was some information on that. There was a central server that would get these signals and tasks were sent to peers. These peers would analyze if the signal (they would process the information) and return the result to the server. Of course, repeated information would be sent to many peers, in order to verify if the information given by one matches the others.

Nowadays the project does not give tasks to others:

SETI@home

SETI@home is a scientific experiment, based at UC Berkeley, that uses Internet-connected computers in the Search for Extraterrestrial Intelligence (SETI). You can participate by running a free program that downloads and analyzes radio telescope data. Already joined? Log in.

 <https://setiathome.berkeley.edu/>

Napster

Context

With MP3 music could be efficiently encoded and share with existing file copying mechanism, because the codification of the signal considers that the final consumer is human. So, some frequency range would not be encoded. This would decrease the size of musics (it would have 1/10 of the original size).

By this time, people would like to download music. But it wouldn't be possible with a single server. By that time, the servers didn't have enough size to store all the music. The solution was **to have a smaller server that would have list telling what peers have what musics stored**. When person wanted to download a music, the server would simply say where's the peer with it and a **direct link** would be made.



Resume

The **Napster** kept a centralized catalog of music descriptions and references to online users that hosted copies of it.

Due to the presence of firewalls, ability to communicate with the server does not imply capacity of accepting connections from other peers (we have where a problem of double firewall).

This service was shutdown, due to legal attacks from the music industry.

Gnutella

Early design

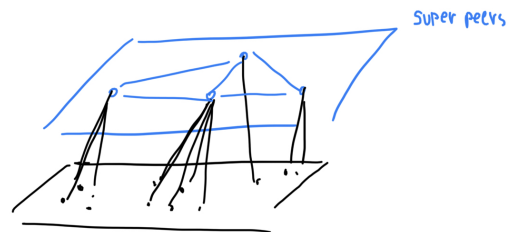
The idea was to have a fully distributed system for file sharing. It uses as base the **Erdos-Renyi**. The idea was to nodes connect to others in a random way. The number of connected nodes can be random.

- The bootstrapping is done by HTTP hosted host caches (that contains a list of peers that the node could connect to join the network). Once joined the network we could start to store nodes in **local host caches**. Due to high churn (rate that clients exit the network), local host caches can quickly become outdated.
- The routing was done by flooding and reverse path routing.
- **PING** and **PONG** messages are used to discover new nodes. **PING** are flooded by the network and the **PONG** follows the reverse path.
- **QUERY** and **QUERY RESPONSES**, are used to make search possibles. The **QUERY** is flooded along the network and are back propagated. The answer grows until the diameter is reached or the maximum number of hops. Once a **QUERY RESPONSE** reaches the original node, this node can use the **QUERY RESPONSE** to connect with the other peer and get the content.
- **GET** and **PUSH** request are used to initiate the file transfer directly between peers. The **PUSH** request was used to solve the problem of **one side firewall**. The **PUSH** request was send to the peer with firewall and this one was responsible for setting the connection. But if we had double firewall, this solution wasn't possible.

Later people realized that actually the **PING** and **PONG** was terrible strategy, because it would not scale. The biggest part of the traffic would be consisted of **PING** and **PONG**. It was in the order of $O(n^2)$.

Improved

Some nodes would have a higher connectivity. People started to prefer connections with nodes that were (in the biggest part of the time) always on. With time, it came with the notion of **super peers**.



- The **super peers** would store a table of contents of the “normal” peers. The **super peers** overlay would act like the standard Gnutella: operations of **PING** and **PONG** would be in the traffic. However, this overlay would protect the lower layer from this kind of messages.
- The **super peer** would have a structure called **bloom filter** to keep track of what were the contents of the connected peers. When there’s a match of content upon the reception of a **QUERY**, the **super peer** would send the **QUERY** to the “normal” peer that had a match.
- There was some redundance in the network, so that if a super peer was attacked, other peers may be promoted to super.

Bloom filters

We get $\text{SHA1}(\text{String}, i)$. We do this operation k times.

For each result, we get the last 8 bits, interpret it as an address and set to 1 the position in an array.

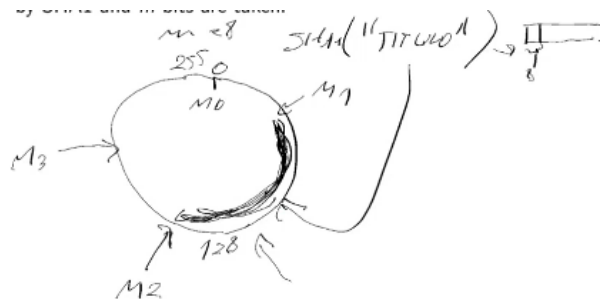
The problem of this approach is that we have a probability of having a **false positive**. But in Gnutella, we don’t actually have a problem with this, because the **super peer** would send the message to the “normal” one, but this one would not have it and thus would not answer.

Distributed Hash Tables

Chord

We have unique IDs in a space from 0 to $2^m - 1$. The nodes IDs and keys are hashed by SHA1 and m bits are taken. Each machine in a ring, must be capable of finding a machine that is responsible for a certain content.

For example, let's suppose we want to store the word `title`. Where should it be stored in the ring? First we need to hash `title` using `SHA1` and by taking `m bits` from the result, we choose the place in the ring to store.



Every node knows a number of m other nodes in the system and to get the value of a certain key, stored in another node, it's always possible to ask the successor for this key. The i^{th} entry in the table of a node indicates the first node s that succeeds the current node in at least 2^{i-1} . A node that searches for a certain key, could ask the closest i^{th} node in the distributed hash table to search the node with the key. In the end, this specific node might use the local connections to reach these. This follows the logic behind the **Watts Strogatz** graph, where we have nodes that have local links, but also tries to create long connections.

In chord nodes keep information of $O(\log n)$ of other nodes.

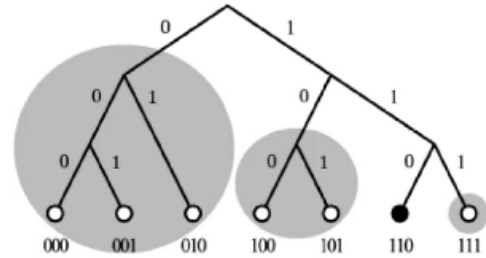
Kademlia

<https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf>

Each node has an id with 160 bits. The id could be, for example, the SHA1 of the ip address.

The distance metric is computed by the **XOR** operation. This is an interesting metric since respects the triangle property.

Unlike chord the alternative next hops can be chosen **for low latency or parallel routing**. The routing table consists in a list of ids, where each entry contains a certain number of nodes that differs in **i** bits from the current node. In other words, the position i , must have bits 0 to $i - 1$ identical to the list owner. It's really easy to find the first entries of the table, but conforms i increases, it becomes more difficult.



The nodes in gray competes for a certain position in **110**'s table.

Properties

- To make a system resilient to errors, we may save for each entry, up to k nodes, where k is around 20.
- When we make lookup, we don't use just one entry to make it, but many entries from the table are used in parallel (this is one difference from chord).
- When there's competition to nodes be part of the table of another, usually the nodes with the highest *uptime* have priority.