Peer-to-Peer File Sharing Systems: A Survey

Kousshik Raj, Satyam Porwal, Prashant Ramnani, Robin Babu, Shivam Jha

I. Introduction

In the current era, a standalone processing node is an extremely rare occurrence. Rather, we often see a network of interconnected nodes each responsible for a part of the content and resources in the network. Such networks can be primarily grouped into centralized systems and distributed systems. In a centralized solution, a single node is treated as the core of the network and is responsible for processing the entire application locally. On the contrary, in a distributed network, the computation is shared among the different nodes of the network. These distributed systems can be further segregated into a client-server model and a peer-to-peer (commonly known as P2P) model. The client-server model is somewhat similar to the centralized systems in that they both revolve around the core central nodes of the network. In this network, a client requests the 'core' (or the 'server') for content /services without sharing any of its own to the rest of the network. This can be either a flat system, or hierarchical to improve scalability and performance.

Whereas, the P2P model, offers a novel solution as opposed to the traditional client-server systems, which can be extended to a lot of application domains. Here, the network is composed of a large number of heterogeneous and highly dynamic peers (nodes) where they share a part of their own resources such as processing power or storage, or content like softwares and multimedia. These networks can be either fully decentralised ('pure') or have some form of centralisation in it ('hybrid'). Fig.1 illustrates this classification. In this report, we will be focusing on these P2P networks.

What makes P2P networks different from the others is that the participating nodes can both act as clients and servers at the same time. Each node is able to access other nodes in the network without any intermediary. As can be seen from their names, a pure P2P system will not suffer from a random node being pulled out of the network . In the other case, a hybrid network uses some centralised entities to manage the network.

P2P networks are associated with the concepts of resource sharing, decentralisation, and self-organisation. As we have seen, every node of the network shares some of the resources with the network which results in the decentralisation of the network. In a completely decentralised network, there is no entity to globally manage the network and hence the network nodes self organise themselves by the information it obtains locally reachable nodes. A global behavior is obtained as a result of such local operation of the nodes of the network.

II. P2P Architectures

As we have seen, decentralisation is an integral aspect of all P2P networks. But not all such networks have the same amount of decentralisation. Hence, based on the degree of decentralisation we classify the P2P

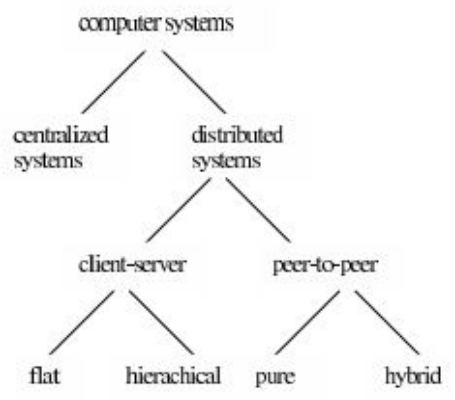

Fig. 1. Classification of systems

networks into two classes:

A. Purely Decentralised

These systems are also commonly called as *pure P2P* systems. This is because they act without any centralized means of control and thereby they exemplify a completely decentralised system. This implies that every node is equivalent to one another and can act as both servers and clients simultaneously. Hence, to differentiate such nodes from nodes of other common networks, these are called 'servant' (taken from SERver and cliENT). CAN, Chord, Freenet, Gnutella, etc. are P2P systems that fall under this class.

The most notable advantage of systems of this class is that they are inherently scalable and fault tolerant. Scalability in a network is usually restricted by the amount of centralized actions necessary and as there are no such limitations in these systems, these networks can grow indefinitely. Moreover, owing to the fact that a loss of one or more peers does not affect the system significantly and can be easily compensated, these systems are inherently fault tolerant as well.

But following the advantages, they also come with detriments. These systems do not have any means to support fast new information discovery which thereby affects the quality of service provided. These systems also lack predictability, as there is no view of the system at the global level.

B. Hybrid Architecture

In hybrid P2P systems, there exists a central entity governing the activities of the system by maintaining a meta-data of the registered users in the network. But even in such a system, the end-to-end interactions happen only between the peers. There are two types of hybrid systems: centralised indexing and decentralised indexing.

In centralised indexing, the central server maintains an index of all the data currently being shared in the network by the active peers. Each peer is connected to the server, to which it sends a query for the required resource. After locating the required resource, data is exchanged in a peer-to-peer fashion. This architecture can be found in Napster. Obviously, the mode of operation of such systems is extremely simple and straightforward. They are very efficient in propagating the finding of new information and thus the searches are quite comprehensive. On the other hand, they are quite vulnerable to malicious attacks and because of the presence of central servers, there is a single point of failure. And unlike the pure P2P systems, they do not support indefinite growth of the network as the central server limits it due its capacity to respond to queries as well as its database size.

In decentralised indexing, we again have a central server to register the users to facilitate the peer discovery process, but unlike the centralised indexing system, here some of the peers assume a more important position than the rest of the nodes in the network. These are collectively called 'supernodes'. These nodes are responsible for indexing the information shared by the local peers connected to that SuperNode and they also proxy the search requests on behalf of these peers. Thus, here the queries are sent to the nearest SuperNode. Kazaa and Morpheus are instances of the decentralised indexing hybrid systems. The interesting fact in such networks is that a peer is automatically qualified to become a SuperNode if they have sufficient bandwidth and processing power. When a new node registers with the central server, the server provides it with the address of one or more SuperNodes it can connect to. Though this system offers a slower information discovery time compared to the centralised indexing system, it reduces the workload on the central server drastically.

III. Discovery Methods

One of the most important parts of distributed peer-to-peer file sharing systems is the discovery mechanism for locating the data within the distributed system.

The first P2P systems, Napster, started with using a centralised structure for this purpose. The second generation of systems, Gnutella, used a flooding-based approach. More recent systems have implemented a Distributed Hash Table (DHT) based approach, BitTorrent, IPFS.

Each of these discovery methods come with different classes of overlay networks: Structured and Unstructured. We will discuss more about this in part IV. For now, let us discuss certain discovery methods:

- **Centralised indexes and repositories**

  Commonly used in hybrid systems, this model the peers to interact with a centralised server which stores all information of location and usage of resources. Whenever a node requests, the central server will process it and find the best peer in its directory that matches the request requirements. The best can be in terms of speed, cost or availability, depending on the needs.

  This model was introduced in Napster, one of the pioneers of the P2P file sharing system. A central directory contains the metadata of all the files, a table of registered user connection information, and a table listing the files that each user holds and shares in the network.

  Fig. 2. shows the resource discovery mechanism in Napster. The client requests the central server and also sends the list of files it maintains. When the server receives the queries it processes it and returns a list of users that hold the matching file. Then the user opens a direct connection to one of the peers and requests and downloads the file.

- **Query Broadcast Flooding**

  Used in purely decentralised P2P models, each peer publishes information about the shared contents in the P2P network. Hence no single node is aware of all the resources in the network. So

resource flooding is done in order to discover a resource, broadcasting is performed, until the request is answered or external constraints are met. The search networks are built in an ad hoc manner, without restricting a priori which nodes can connect or what information can be exchanged.

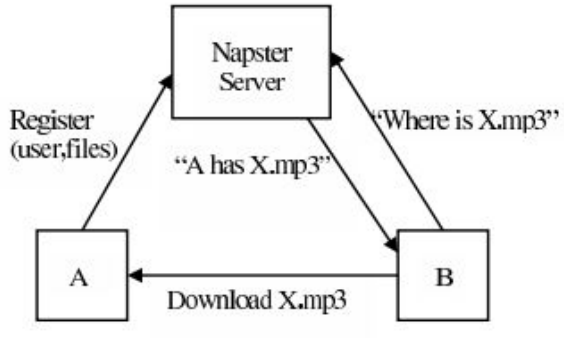

Fig. 2. Resource discovery in Napster

Gnutella implemented a flooding broadcast method to find the files in the network. It works as a distributed file storage system. The protocol implemented by Gnutella used four different types of messages:

1. **Ping**: request for certain host to announce itself
2. **Pong**: Reply to the ping with IP and port of the responding host and details of files shared.
3. **Query**: Details of search request, not limited to search string and minimum speed requirements.
4. **Query hits**: Reply to query message. IP and port of the responding host and details of matching files.

Using a Time-to-Live(TTL)-limiting flooding and loop-avoidance methods for ping and query messages, the method has good efficiency and preserves network bandwidth. Fig. 3. shows flooding based discovery broadcast.

The protocol gives optimal results in a small to average number of peers but does not scale well. Also, accurate discovery of peers is not guaranteed by flooding. Using TTL based approach has its own pros and cons, the network is effectively segmented based on hops and removing TTL would flood the network with requests.

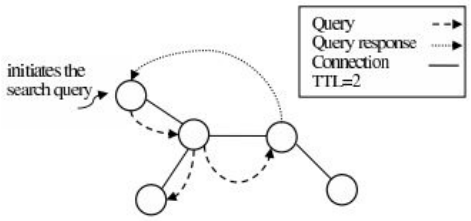

Fig. 3. Resource discovery using Flooding-based broadcast

- **Routing Model / DHT**

This is implemented in more structured systems, where resources are stored using distributed hash tables. In this protocol we try to provide a mapping between the resource identifier and location, in the form of a distributed routing table, so that queries can be efficiently routed to nodes with the desired resources.

We try to reduce the number of P2P hops that are required to locate the resource. The look-up service is implemented to organise in a structured overlay network, and routing messages through overlay to responsible peers.

Some implementations of distributed P2P look-up services:

- **Freenet**:

  Adaptive, loosely structured, decentralised P2P network, which makes query to store and retrieve data items which are identified by location-independent keys.

  The service provided is more of file-storage than file-sharing service. Each peer is assigned a random-ID, each document is assigned doc-ID based on hash of contents and name. The document is routed towards the peer with closest peer-ID.

  On request of a document, the request is forwarded to the node with the most similar peer-ID, on servicing the request, the document is transferred back to the originator, each participating peer keeps a local copy of the same.

- **Chord**

  Decentralised P2P lookup protocol, which stores key/value pairs for distributed data. For each key, a mapping is made to which node stores the key's value.

  In a N-node network, each node maintains routing information about O(*log N*) other nodes, and resolves lookups to these nodes.

- **Content Addressable Network**

  Decentralised and distributed P2P infrastructure which provides hash table like functionality on an internet like scale.

  The basic idea of CAN is to associate each node and item a unique coordinate in a d-dimensional cartesian space. The CAN discovery mechanism is a two part operation - local hashing based look-up of pointer to resource and routing the look-up request to the pointer. The discovery is guaranteed in $O(N^{1/d})$ steps.

- **Pastry**

  Similar to chord, pastry routes a message to node with nodeId that is numerically closest to the key contained in the message from its routing table.

  Pastry minimises the distance each message travels by using network locality into account and other scalar proximity metrics like number of IP routing hops.

IV. P2P Network Structure

P2P networks can be classified in many ways, one of the more common ways to do it is by identifying if the network contains some structure or is more ad-hoc.

Technical meaning of Structured is that P2P overlay network topology is tightly controlled and contents are placed at specific known locations and not spread at random peers, making subsequent queries much more efficient. These structured P2P systems use DHT, where data is placed deterministically using an unique key. The topology in these networks are tightly controlled.

Whereas in unstructured networks, the placement of the data is completely unrelated to the topology and peers in-place. In such cases, neighbours have no information about each other's data, flooding is usually involved in searching in these systems.

Table 1. shows comparison of various P2P mechanisms based on different performance factors.

P2P Mechanism	Unstructured, Pure	Structured, Pure	Centralized indexing hybrid	Distributed indexing hybrid
Scalable	no	yes	no	yes
Flexible	yes	no	no	yes
Robustness	yes	yes	no	yes
Manageable	no	yes	yes	yes

Table 1. Comparision of Various P2P Mechanisms

We will discuss key features of Structured P2P and Unstructured P2P overlay networks, their operational functionalities, evaluate them and discuss various developments in each.

- **Structured P2P Overlay Networks**

  In this classification, the overlay network assigns keys to data and organises its peers into a graph, mapping each data key to a peer. The structured graph ensures efficient discovery of data using the keys. However handling complex queries is not so simple in this case.

  We have already discussed CAN, Chord, and Pastry. Here we will discuss Tapestry in more depth.

  ➔ **Tapestry**

  Similar to Pastry, Tapestry uses decentralised randomness to achieve both load distribution and routing locality. The major difference that can be seen between the two mechanisms is the handling of network locality and data object replication. Tapestry implements a variant of Plaxton-style global mesh like network, with additional features to provide availability, scalability and adaption to failures and attacks.

  The plaxton mesh is a distributed data structure, optimised to support a network overlay for locating named data objects which are connected to one root peer. On top of this tapestry uses multiple roots to avoid single point of failure.

  In Plaxton mesh, each peer can act as server (data object provider), router (forward message), and clients (entity of request). Each node is assigned a unique nodeID, randomly chosen from a large identifier space, using SHA-1 to produce the 160-bit identifier. Object GUIDs are also assigned similarly.

  Local routing maps are implemented at each peer to incrementally route overlay messages to the destination ID digit by digit, similar to longest prefix routing in the CIDR IP address allocation architecture.

  The n^{th} peer that a message reaches, shares a suffix of at least length n with destination ID. To locate the next router,

the $(n+1)^{th}$ level map is used to locate the entry matching the value of next digit in destination ID. This method guarantees any existing peer in the system can be found in $O(log_B N)$ logical hops. Since the peer's local routing map assumes the preceding digits all match the current peer's suffix, the peer only needs to keep a small constant size entry at each route level, yielding a routing map of fixed size.

The lookup and routing mechanism in Tapestry is similar to Plaxton, which are based on matching suffixes in the nodeID. Routing maps are organised into routing levels, where each level contains entries that point to a set of peers closest in distance that matches the suffix for that level. Each node also stores its neighbours.

Objects are located by routing a message towards the root of the object. Each node along the path checks the mapping and redirects the request appropriately. The effect of routing is convergence of nearby paths heading to the same destination. Failure of peers will not cause network-wide failure.

- **Unstructured P2P Overlay Networks**

Unstructured peer-to-peer networks do not impose a particular structure on the overlay network by design, but rather are formed by nodes that randomly form connections to each other.

Since there is no structure globally imposed on these, unstructured networks are easy to build and optimize different regions of the overlay. Since the roles of all the peers in the network are the same, the network is highly robust when the number of peers joining and leaving is high.

The major limitations of the network arises from the unstructured nature as well. Since flooding is involved, a high amount of signaling traffic is caused. Searching for rare data will more likely result in failure. Some unstructured P2P networks have been discussed earlier: FreeNet, Gnutella, FastTrack/KaZaA and BitTorrent are certain examples of the same.

➔ **BitTorrent**

A centralised P2P system which uses central location to manage user's downloads. The file distribution network uses tit-for-tat (peer responds with the same action that its other collaborating peer performed previously) as a method of seeking.

The architecture consists of a central location which is a tracker that is connected to the user when he downloads a *.torrent* file, which contains various meta-data relevant to the file. The tracker keeps track of all the peers who have the file and lookup peers to connect with one another for downloading and uploading. A peer that has completed downloading the file is called seeder and the peer downloading the file is called leecher.

BitTorrent furthermore cuts the files into pieces of fixed size (256KB) so as to track content of each peer. Each seeder announces all of its peers the pieces it has and uses SHA-1 to hash all the pieces; details of encryption are provided in the *.torrent* metadata. An acknowledgement

is sent by leecher to all its peers upon receiving a chunk, this is done to verify data integrity.

Pipelining is implemented to ensure a good TCP performance. When data is transmitted, leechers keep several requests queued up at once in order to get good TCP performance. Requests which cannot be written out to TCP buffer immediately are queued in memory rather than application-network layer.

Choking is a temporary refusal to upload; downloading can still happen and the connection does not need to be renegotiated when choking stops. TCP congestion control behaves very poorly when sending over many connections at once. Choking lets each peer use a tit-for-tat-like algorithm to ensure that they get a consistent download rate. Avoid Choking by fibrillation by changing the peer that is choked once every 30 seconds.

BitTorrent is well-suited for download performance due to its advanced download distribution protocol. The lack of archive functionality in BitTorrent results in relatively short content lifetimes.

Even after almost 2 decades since its inception, BitTorrent generated a substantial amount of traffic: 2.46% of downstream and 27.58% of upstream traffic is generated by the file sharing protocol.

V. Challenges in P2P Systems

P2P systems offer a large number of advantages over traditional Client/Server systems. But they are not silver bullets. They have their own set of problems. Some challenges while dealing with such systems are:

A. Security

In P2P systems the nodes are dynamic and also the peers don't trust each other, and thus achieving a high level of security in P2P systems is more difficult than traditional Client/Server Systems. Normal security mechanisms to protect systems from intruders and attacks such as firewalls can not protect P2P systems since they are essentially globally distributed and also these mechanisms can inhibit P2P communication. Therefore new security concepts are required that allow interaction and processing in P2P systems.

B. Reliability

A Reliable system is a system which can be recovered after some failure or faults. The factors necessary for a reliable system are data replication, node failure detection and recovery, existence of multiple guarantees for location information to avoid a single point of failure and the availability of multiple paths to data.

For providing reliability in unstructured networks, dynamically adding redundant links to the systems is known to work. In this way no single disconnection should cause disconnection of the whole system.

In a structured network, messages need to be routed, and the routing stage has to be updated automatically when a node enters or

leaves the network. To maintain reliability for such conditions, nodes must use part of their bandwidth to maintain the routing states. Further techniques can be used to optimise this so that the cost is manageable in the normal operating conditions.

C. Flexibility

One of the most important services a P2P system provides to its users is the ability to join and leave at their will. Modern P2P systems are characterised by decentralised control, large scale and extreme dynamic environments. Adaptation and self-organisation is very important for such dynamic environments.

In modern unstructured P2P systems like Kazaa, queries are forwarded only to supernodes, which maintain a list with the file names of their connected peers, avoiding overloading all peers of the system.

In standard structured P2P systems, static identifiers are assigned to nodes and based on these distributed data structures are constructed, so the overlay network structure is determined by these identifiers and in turn any self-organization of the system is prevented. Standard structured systems based on distributed hash tables should perform lookups quickly and consistently while nodes arrive and leave the system

D. Load Balancing

The management of the distribution of the data and the computation across the network is very important for the efficient functioning of the network. For this, we employ a Distributed Hash Table (DHT), where all the data that is stored in the network is mapped to a unique identifier. To distribute the load, the identifier space is partitioned into a wide range of subsets and is distributed among the nodes, and each node is responsible for maintaining its portion of the partition space.

Agent based self organizing models can also help in the load balancing among the computing nodes of the p2p network. For example, consider Messor, an Anthill load balancing algorithm. It is a highly adaptable algorithm where the ants move towards highly unbalanced loads and can also redistribute loads to the network from crashed nodes. This is a self organising algorithm and it also efficiently manages the compute power in the network.

VI. Case Study

Now we take a look at some of the most important and popular P2P systems. Many of these systems are still used today, and many have also been designed based on the systems described here.

1. Napster

Napster was a P2P file sharing application mainly made for sharing MP3 songs amongst clients. Napster was one of the first commercial P2P applications, however it is not a true P2P system as a lot of the client requests are directed towards a centralized server, which contains a directory of indices of multiple files. Thus in order for a client to search for a song, it must ping the central server, which returns the possible sites from which the client can download the

song. The P2P aspect in Napster comes in as the actual song transfer occurs directly between the two nodes without direct involvement of the central server.

We discuss some of the properties of Napster:

- **Decentralisation:**

Very less decentralisation due to the presence of central servers that require to be pinged by the client any time it wishes to download a song.

- **Scalability:**

The system is not very easily scalable as the central servers are a clear bottleneck for the system. All traffic from the clients need to enter the servers, thus can cause scalability issues. However it is worth noting that Napster had around 1.5 million users during its prime, thus indicating that it had well replicated servers capable of handling large numbers of clients.

- **Anonymity and Security:**

The clients are anonymous with respect to each other as only the IP address and port of various clients are shared. However the server is aware of all its users, and the songs that each user contains. At the same time the system is somewhat secure as the central server is powerful and thus makes it hard for users to interact with fake IP addresses and ports.

- **Lookup Completeness:**

Lookup is complete as the search for the file is done only at the central server which has a directory of all songs present in all its users.

- **Fault Tolerance:**

The system is not fault tolerant. Any disruption at the central server can cause malfunction of the whole system.

2. Gnutella

Gnutella is also a first generation P2P system like Napster, i.e it was amongst the first few P2P systems that inspired further systems. Unlike Napster, Gnutella turned to a fully distributed architecture. Thus there is no central server. In order for a new client to join the network overlay, it must know the IP Address of at least one client already present in the Gnutella network. To further facilitate this bootstrap operation, the new client receives the immediate neighbours of the client whose address it knows and it further tries to connect with it. The protocol used by Gnutella has been described in the "Discovery Methods" section. Since Gnutella is a true P2P network, it's early protocols used flooding with a TTL (Time-to-Live) for searching. We now discuss some of the important properties of the Gnutella system:

- **Decentralisation:**

The system is highly decentralised and each peer is truly equal in nature.

- **Scalability**:

In theory the Gnutella network should be highly scalable as each peer is equal and ideally there should be no central point for congestion of traffic. However it was observed that flooding of queries hogs the network bandwidth causing huge scalability issues. This also leads to fragmentation of the network into smaller clusters as certain nodes

become unresponsive to huge amounts of network bandwidth utilization.

Furthermore users aren't forced to store or share files. This leads a lot of users simply being present in the networth without any contribution, which leads to huge performance issues and lower probability of being able to find the required files.

- **Anonymity and Security**:

Anonymity is preserved to some extent by virtue of the system being a true P2P system and messages can only come through local nodes. However it is possible for nodes to store information regarding incoming and outgoing traffic that may affect anonymity to some extent.

The security of the system is quite low as the system is prone to unwanted flooding, malicious attacks, hijacking of queries and denial of service attacks.

- **Lookup Completeness:**

The search is incomplete as it may not search across all clients due to the presence of TTL. Thus it is possible for the search query to not yield any result despite the existence of the required file in the system.

- **Fault Resilience:**

The system is quite tolerant to the fault of a node. The system is still capable of communicating with each other without any problem and the system will continue to work correctly without issues, however the faulting of certain nodes can cause performance issues or inability to obtain files present in that node as there is no inherent replication of data in the system.

3. **Gia**
4. **BitTorrent**
5. **Kazaa**
6. **DC++**
7. **Pastry**

VII. Conclusion

System Architecture depends upon the applications requirements. Till now major architectures have been Client/Server Systems. But with the emergence of technologies like Napster, Torrent, IPFS things have started to change. Due to cons of the standard Client/Server model, efforts are being put into the exploration of P2P systems.

In the last few decades, a lot has been done for the development of scalable, fault-tolerant and autonomous P2P systems. Many architectures, each with their pros and cons have emerged. Depending upon the requirements for their particular applications, developers should choose a topology for the platform that matches their needs.

In pure peer-to-peer networks every peer is given equal responsibility irrespective of its capabilities. From this it is evident that this can lead to reduction of performance as less powerful nodes are added. Also, pure p2p systems lack manageability since every peer is its own controller.

Unstructured pure p2p systems use blind flooding for search. This creates the problem of scalability as large scale systems require a large number of messages to be exchanged. Scalability issues can be solved by using structured systems. But in standard structured systems it is hard to maintain the structure required for routing in a very transient node

population, in which nodes join and leave at a high rate.

Pure p2p systems are fault tolerant, since failure of any particular node does not impact the rest of the system. Hybrid p2p systems solve the manageability problem of pure p2p systems, so that the control servers act as a monitoring agent for all the other peers and ensures information coherence. Regarding distributed indexing and centralized indexing systems, drawbacks associated with centralized indexing systems are a single point of failure when the central server goes down and also not being scalable because of the capacity of the server to maintain database and to respond to queries. Distributed indexing systems alleviate these shortcomings by using super-peers. Although super-peer clusters are efficient, scalable and manageable, in order to avoid a single point of failure for the clients in a cluster, some policies of super-peer redundancy should be taken into account. As in the case of failover super-peer, these strategies should be able to take over the job of the primary super-peer.

VIII. References

[1]. To be filled...