PBJ: A Gnutella Inspired File Sharing System

Camden Clements
School of Computing
Clemson University
Clemson, SC 29632
Email: camdenc@gmail.com

Adam Hodges
School of Computing
Clemson University
Clemson, SC 29632
Email: hodges8@clemson.edu

Zach Welch
School of Computing
Clemson University
Clemson, SC 29632
Email: zwelch@clemson.edu

***Abstract*—PBJ is a distributed file sharing system designed on many of the same principles of the gnutella file sharing application. Rather than route search requests through some central entity like Napster and to a lesser extent BitTorrent, PBJ creates a network of connected users. Search requests are broadcast between users and files are downloaded directly between them. This paper details the algorithms behind PBJ, the specifics of its implementation, and a comparison of PBJ to other popular file sharing systems. Future improvements to PBJ are also discussed.**

I. Introduction

Peer to peer (P2P) file sharing is probably one of the most well known and ubiquitous examples of a distributed system. These systems allow users to search for and download files stored on the machines of other users. Systems such as Napster, BitTorrent, and Gnutella are used by millions of people every day to download files. While the content on these systems is not always legal (eg. file sharing of media such as music and movies), P2P file sharing networks are..... File sharing is an ideal case study of distributed systems, and highlight many of the design decisions faced when developing such systems. P2P systems can feature a number of different network topographies that range from having a central server storing all the files (Napster) to a distributed system with almost no centralized elements (Gnutella). In the sections that follow we document and describe PBJ, our P2P file sharing system influenced by Gnutella.

A. Background

1) Napster: A defining characteristic of a P2P file sharing application is the kind of network topology it employs. Napster, one of the first file sharing networks, was a highly centralized system. All online Napster users connected to a central server. Each user would have a folder designated for sharing audio files (only mp3s were transferred on Napster). This central server would handle a user request by searching for related files in the share folders of other users. The central server would then report a list of relevant files and their locations that the user could choose to download. Any files the requesting user chose to download would directly connect with the associated machine for actual file transfer. The "flaw" of Napster's system is that the central server is appropriately named; if the server goes down, then the network ceases to exist because no requests can be processed. This is exactly what happened in 2001 when the inevitable flood of copyright infringement lawsuits were filed against Napster. File sharing applications developed after Napsters shut down attempted to avoid this problem by making the system less centralized.

2) BitTorrent: Most file sharing today uses the BitTorrent protocol. BitTorrent differs from other file sharing systems in that where as Napster and gnutella transfer entire files between nodes, BitTorrent downloads parts of the file from multiple sources. BitTorrent works by having a user obtain a torrent file (usually by downloading it from a website). The local BitTorrent client interprets the contents of the torrent file and connects with a tracker, a server with information about how to find the file. The tracker finds seeders, other BitTorrent clients with a local copy of the file to be transferred. The file also identifies the swarm, a set of clients with a portion of the file, usually in the process of downloading it themselves. The searching client downloads from these sources simultaneously to build the desired file. BitTorrent is especially useful for downloading popular files, since there will be a large pool of seeders and a large swarm. While BitTorrent is clearly more distributed than Napster, since their are multiple central servers instead of a single large server for the entire application. However, if the tracker(s) in a torrent file are taken down, there is no way that torrent file can be used to obtain the actual file.

3) Gnutella: Gnutella (a portmanteau of GNU, the free software project, and Nutella, the chocolate and hazelnut spread) is a decentralized network of interconnected users running a gnutella client called a node. Search requests are broadcast to all of the searching nodes neighbor clients (which gnutella terms peers), which in turn broadcast the request to all of their peers as well. This goes on until the file is found or the request goes a specified number of peer hops without finding a file (much like the time to live field in an IP packet). This does allow for the possibility that a search request may fail to return a file in the network. Later versions of gnutellas network introduced the concept of ultra nodes and ultra peers. Each ultra node is connected to a network of regular nodes and a large number of other ultra nodes, each with its own network of nodes, making gnutella a network of networks. The reasons for this change is that it many more nodes can be reached from any one node in a few hops, making searching more efficient and the system more scalable. Nodes keep track of nodes and ultra nodes they have previously connected with and attempt to reconnect to the network through these nodes.

When connecting for the first time, these nodes must attempt to a set of nodes guaranteed to be in the gnutella network. This means that despite the highly decentralized nature of gnutella, a few centralized elements remain.

II. Methodology

PBJ takes the key qualities of gnutella and uses them as a starting point, rather than implement a version of the gnutella client. LIke gnutella, PBJ contains a network of ultra nodes. In PBJ, each ultra node has a set of nodes. Note that ultra nodes are also nodes in PBJ. If there are k nodes per ultra node, the first node to be added to the network will become an ultra node, the next k nodes will be added to the ultra node. The k+1th node will then become a new ultra node. Ultra nodes in PBJ are given a unique ID. This unique ID is a vital element to building the network. The basic algorithm for creation of the network builds an outward spiral of ultra peers. While this alone would be wildly inefficient in terms of searching and stability, new ultra nodes also connect to ultra nodes deeper within the spiral. This allows search requests to quickly move between otherwise distant ultra nodes. More specifically, the first node created will be given an ID of 0, the second an ID of 1, etc. When a node is created with ID Y, it will attempt to connect to all nodes X, 0 ¡ X ¡ Y, where Y - X = 2Z, Z being an even integer. So for example, node 16 would connect with node 15 (16 - 15 = 1 = 20), node 12 (16 - 12 = 4 = 22), and node 0 (16 - 0 = 16 = 24). To make talking about the network easier, we delineate between kinds of hops between nodes. For example, a hop from 5 to 6 is a 1-hop, where a hop from 5 to 9 is a 4-hop etc.

The PBJ network structure presents several clear benefits. The first clear gain is that the PBJ network is that there is a high level of redundancy in the connections. This means that the nodes are connected in such a way that it is easy for search requests to reach most if not all of the nodes on the network in a few hops. Currently, building the network is done through a predefined gateway at a predefined IP address and port. The gateway keeps track of all the ultrapeers in the system. When a new node tries to join the system, the gateway makes several decisions based on the current status of the PBJ network. First, the gateway decides if the new node will become an ultra node. If so, it will give the new ultra node a list of other ultra nodes to connect to. If the new node is not an ultra peer, the gateway will provide information on the appropriate ultra peer sub network to join. These decisions by the gateway essentially dictate how the network is built. We devised two separate algorithms for building the network; the one we actually implemented is described here and the other is detailed in the section Future Work. As implemented, the first node to enter the network becomes ultra peer 0. If, for example, there are three nodes per ultra node, the next three nodes to connect would become node 0s sub network. The fifth node to join the network would become ultra node 1 and would connect to ultra node 0. In this scheme of building the network, each ultra nodes sub network is filled before the next ultra node is created.

To share files, a user must specify a share directory. All files and sub directories in their share directory will be available to the network for download. When a user wants to search, they enter a keyword and submit a search request. The search request will be sent to the searching nodes ultra node. The ultra node will send the search request to all the other peers in its sub network before broadcasting it to its connected ultra peers. Search requests consist of four major parts - the searching nodes address, the keyword, a search id, and the time to live (ttl). The ttl is initially set to a positive integer. Each time a search request is passed between ultra peers, the request ttl is decremented. Notice that ttl only limits the number of hops among ultra peers, passing a request within an ultra node and its sub network has no effect. If an ultra node receives a search request with a ttl of 0, it will send the request to its sub network and delete the request. If a keyword match is found, the node with the desired file sends a message to the node specified in the search request with its address and the path of the desired file relative to the share folder. As these acknowledge messages come in to the search node, it displays to the user a list of all the files found on the network. The user can then choose one or more files to download. The searching node directly connects to the node with the selected file and downloads it. The search id is sent as an attempt to significantly lessen the search request traffic on the network. The structure of the PBJ network means there are many different paths between two ultra nodes. While this is very useful in keeping the network stable and allowing search requests to quickly reach a large number of files, it also means that the same ultra node will receive and process a request multiple times, sending the message out to all of its ultra peers, who have already received the request. This type of network behavior is extremely inefficient. PBJ attempts to fix this problem in part by having a unique pair of values for each search request. One of these values is the ultra node id the search request originated from and the other is local to each ultra node and represents the number of previous requests sent out by the ultra node. The first request sent from ultra node 6 would have an id pair of (6,1), the second would have (6,2) etc. Each ultra node keeps track of the last several request ids received and if the request id of a new search request matches one in the list, that request is ignored. This scheme vastly cuts down the amount of redundant request traffic and makes the system more efficient as a whole.

An area of contention during development of PBJ was how to correctly handle ultra peer disconnection. Various solutions were proposed, including updating the gateway, electing a node from the sub network to be a new ultra node, and the Ostrich approach (ignoring it). We eventually decided to implement a system of letting the gateway know when an ultra node goes down. Every time the gateway is about to add a new node to the network, it pings its ultra peers. If one or more does not respond, it marks those as missing, and inserts ultra nodes into these positions until they are all filled, and then fills them The point here is to ensure important connections in the network are rebuilt before the ultra nodes are filled. In addition, ultra nodes will occasionally ping their ultra peers

and sub network. If an ultra node does not get a response from an ultra peer, it removes it from the list of connected ultra peers. If an ultra peer cannot connect to a peer in its sub network, it removes that peer from its list and notifies the gateway that a spot has opened up in its sub network. If a node in a sub network cannot reach its ultra peer, it re enters the network through the gateway. This ensures that all lost nodes will be replaced as new ones are added to the system, and that the network will not become fractured with high node turnover.

III. Results

There are a variety of improvements that PBJ could benefit from in future distributions. One previously discussed is changing how nodes join the system. The current gateway method functions well, but its centralized nature is hardly ideal for a system whose design so highlights decentralization. Several alternatives have been discussed. The most likely implementation would be to place the burden of network building on the existing ultra nodes. A node would keep track of its previously connected ultra nodes and attempt to connect to these peers. Depending on its current status, the ultra peer might add the new users or decide to push it towards higher valued ultra peers to attempt placement. Eventually it would either get placed in a sub network or reach the ultra peer with the highest id and become the new highest ultra peer. If none of the previously connected nodes are currently connected to the network, a range of options could be implemented to ensure the user gets into the network including but not limited to : keeping the gateway as a backup, having ultra nodes at known URLs, placing the burden on the user to find the information.

Several different specific algorithms for how exactly the gateway builds the grid were discussed and implemented. The basic algorithm, described above, creates a new ultra node and then fills that ultra nodes sub network before creating a new ultra peer. While this algorithm works acceptably, it ignores several observations about PBJs network. In PBJ, the number of hops it takes to get from node A to node B is highly dependent on the current number of ultra peers in the system. As a simple example, the optimal number of hops from ultra peers 0 to 63 is 5 hops (0-16-32-48-47-63) if ultra node 64 does not exist and 2 hops (0 - 64-63) if ultra node 64 does exist. Indeed, a node whose largest hop was one of the last nodes to be added has the optimal network structure for searching. Our algorithm for taking advantage of this trait slightly alters how the network is built by the gateway. The basic idea behind this algorithm is to have an ultra peer back bone in place first with empty sub networks and then fill in the sub networks. When creating a new network, the first eight (eight is half of sixteen) nodes into the system will become the first 8 ultra peers. The next nodes to join the system will join these ultra peers sub networks. When all eight sub networks have been filled, the gateway then add the next nodes as ultra peers in the system until there are thirty two ultra peers (thirty two is half of sixty four) in the system. This continues so that when the network is full, it grows to half the size of the next even power of 2 before filling in the ultra peers. This ensures that the majority of the peers are close together but have the ability to move farther because there is an ultra peer structure in place to facilitate such movement. Another potential improvement would be to choose ultra peers based on network quality. Ultra peers have to handle the vast majority of the network traffic. It then makes sense that the machines with the strongest network connection should be ultra peers. This would help stabilize the network and improve quality.

Simple things such as an improved GUI and security/privacy features could also be added to make the system more user friendly. Another concern we have is attempting to mitigate is leeching, the practice where by users of a file sharing network do not share any files to be downloaded but still use the network to get files.

IV. Analysis

V. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.