Reducing Traffic and Delays in P2P Systems with Replicated Mutable Files

Andrew Rosen Brendan Benshoof Matt Johnson Anu Bourgeois
Department of Computer Science, Georgia State University
34 Peachtree St NW
Atlanta, Georgia 30303
rosen@cs.gsu.edu

Abstract—**Peer-to-peer networks generate an increasing amount of traffic each year as they become more widely used and as larger files become more common to share. This has led research into how peer-to-peer networks can be used for tasks such as keeping mutable files across a network up-to-date and using other points in the network to store temporary copies of the file to ease the load on the server by diverting traffic. The integrated file replication and consistency maintenance algorithm (IRM) combined the problems into one and examined the effects on a network using Chord. This paper proposes reducing the strain on the file owners by reducing polling and diverting polling traffic to the replica nodes. These changes can be accomplished without decreasing the hit rate at the replica nodes, while also decreasing the overall traffic in the network and average latency.**

I. Introduction

Stuff

II. Background

Not all peer-to-peer networks are equal; there are variety of protocols and methodologies that a networks could implement and this affects what kind of solutions are available to reduce the traffic on the network [?] [?]. The most basic type of network is a structured, centralized network. Peers in this network communicate with a central server to provide their files and to locate other peers that have the files they are searching for. This is structured in the sense that the layout of the overlay network is tightly controlled, in this case by the server(s). While this avoids the problems of routing, it has the same issues of scalability as a client-server layout and is not much of an improvement. An example of this network is the long defunct Napster [?] [?].

On the other side of the spectrum, there are unstructured, decentralized networks. These networks create overlay links between nodes in a random manner. This leads to a very unstructured overlay, but it is one that is very easily constructed. No single node is responsible for the whole of the network; files are located by sending out requests to neighboring peers, which in turn request from their neighbors and so on. Should a file become suddenly popular, this flood of this requests can easily bring some peers to their knees, unable to deal with the high level of traffic [?]. This makes these types of networks also a poor choice for implementation [?].

Modern P2P implementations are hard to classify easily, due to the variety of methods used to create a working network that avoids the weaknesses of the two above network types. Many networks today use a decentralized structured approach to distribute files, where the topology of the overlay is constructed and controlled by the protocol and the information about the network is distributed among the peers. This distribution is typically accomplished by a distributed hash table (DHT). Networks that use a DHT choose specific peers in which to place information about how to find particular files or data. These peers are chosen so that the peer's ID in the network corresponds to the file or data's ID, typically by hashing both ID's and comparing them [?].

In addition, the network topology is distributed among various peers. Each peer has a table consisting of other peers in the network and the means of communicating with them. The contents of this table are also controlled by the protocol. The table handles the routing of requests from one node to another; when a peer receives a request it cannot fulfill, such as information about where to find a particular file, it directs the the request to the node that is ”closest” to the destination of the request. How this works is determined by each protocol [?]. It should be noted that closeness is relative to the algorithm; depending on what identifiers are assigned, a node in New York City might be ”close” to a node in Russia, but ”far” from a node physically located only miles away. Examples of protocols that use these techniques are trackerless BitTorrent [?], Chord [?], and Kademlia [?]. As our work is implemented using Chord, we provide more detail here.

Chord uses hashing and mapping functions to efficiently disseminate information throughout a network, using keys and nodes. Keys can be identifiers for data, such as keywords and tags; the associated data would be the values under this mapping. Chord takes peers in the network and gives them a node ID. Both the keys and the nodes are hashed to produce an identifier of length m, m being some value large enough that the hashing function would be extremely unlikely to assign two different nodes or keys the same hashed identifier. For the sake of clarity, both the node ID and key refers to the corresponding hashed value in the following description.

Chord takes the nodes and structures them into a circle composed of less than 2^m nodes, ordered clockwise by identifier from lowest to highest. Chord then takes the keys and places each in the node that has the same hashed identifier as it or the node with the first identifier that follows this value. The node that takes some key κ is known as the $successor$ of κ, or $successor(\kappa)$.

Since this is a circle of maximum size 2^m, the assessment is done in modulo 2^m. Given K keys and N nodes, each node will end up being responsible for approximately $\frac{K}{N}$ keys.

Each node could then find the information of a key by asking the next node in the circle for the information, who would then pass the request through the circle until the node with the identifier equal or first succeeding the value of the key was found. That node would be then be able to pass the data, should it exist, back to the requesting node. This naive approach is largely inefficient, but provides a base example on how nodes in Chord search for information.

To aid in searching, each node stores the locations of up to m other nodes in the network. Specifically, entry i in the "finger table" of a node with the id of n will be the location of $successor(n + 2^{i-1})\ mod\ 2^m$. When a node wants to find some key, it looks to the entry in the finger table that will get it closest to the key without going past and request that information in the manner described above. Fig **??** [**?**] gives an example network with a node pointing to the other nodes it knows the location of. This approach allows nodes to find a specific key in $O(log_m n)$ time. For a more detailed explanation and psuedocode, please refer to Stoica *et al.*'s paper on the Chord protocol [**?**].