P3 Project Paper

Hybrid P2P Network Overlays

Avinash Kumar
W0971410
Vineet Pandit
W0970126
Kevin Cui
W0883010

2. Introduction

The purpose of this paper is to outline a new scheme of designing P2P networks in order to optimize many important qualities of shared networking, such as fault tolerance and failure recovery, speed, redundancy, quality of service, etc, we propose a variant of a hybridized P2P network, which will combine the more favorable qualities of both structured and unstructured networks.

Peer-to-peer networks have many powerful methods of insuring that connections, that have been established remain so, and that content can be delivered to the users of the network. However, for as many positive qualities that P2P networks have, there are also some drawbacks. In this project proposal, we will attempt to address the issues that P2P networks have, and outline an improved network structure that is adapted from the positive effects of P2P systems, while simultaneously reducing the susceptibility to their disadvantages.

Peer to peer systems can be utilized to create an environment conducive to highly distributed and parallel processing without overuse of the local resources. The right P2P architecture can also display a high degree of fault-tolerance, which is a very desirable property when allowing remote computers to do the heavy lifting. In this project we try to simulate a robust but lightweight P2P network which is capable of tolerating a high number of nodes joining or leaving the network unpredictably, and also storing and retrieving key/value pairs stored on the network. This key/value pair storage can be further enhanced to create a full fledged P2P storage solution on the cloud.

Previous attempts at P2P network design are rooted in the construction of randomly distributed nodes, which were enabled to communicate with their neighbors by utilizing flooding to gain information about one another. P2P networks then progressed towards more structured mappings, such as CAN or Chord network design. However, these structures are relatively expensive because of the need to update all relevant peers. Over an extremely large network, these updates could cause overloads and heavy, unnecessary traffic.

Our approach attempts to break up a P2P network into smaller components. The main component, the core, will be made of a structured P2P network, loosely based off of a Chord network. Smaller branch networks will be connected to this core. These branches will be unstructured networks. This will allow for the structured routing found in the core to combine with the speed and smaller memory footprint that unstructured networks provide.

We also like to propose alternative fault tolerance techniques by coming up with a new algorithm and will show the analysis and result in the defense paper portion.

In summary, the problem that we face with current P2P networks is that many do not employ a hybrid model. Utilizing Greedy for smaller components, while ultimately

structuring the core network around an established network config will allow the network as a whole to operate within a hybridized, optimized environment.

We are investigating within the scope of P2P networks within a cloud-computing, multi-node environment. The concepts outlined in this paper are intended to provide the reader with an innovative framework within which outlines of previous frameworks are visible, but which also avoids the worst of qualities found in both.

3. Theoretical Bases and Literature Review

The problem addressed in this project is being able to store items on to the network and to allowing other nodes to find these items easily. Furthermore, this project also tries to make the P2P network used as efficient as possible but without adding a lot of network overhead of maintaining the structure of a structure P2P overlay network.

Pure P2P networks can be divided into 2 sub categories, namely structured and unstructured.

In structured P2P networks, each node holds a set amount of information. This ensures that the upper bound on the worst case time required to do a find can be mathematically calculated as the information held at each node follows a structure. Some examples of these networks are CAN, Chord, Kad and Tapestry. These networks have different strategies for storing routing information. However, a downside to these structured networks is the overhead to maintain the structure when new nodes join and existing nodes leave. The information of nodes joining and leaving has to be propagated to all the necessary nodes and those nodes have to update their local information accordingly.

In unstructured networks, as the name suggests, there isn't a structure for routing and thus adding or removing nodes is a very simple process. However the disadvantage of these networks is in the find behaviour. In most of these unstructured networks finding a certain item is done by flooding the network with requests with a time to live on each of the request. This search ends when the object is found or the requests time out. Naturally this process is very taxing on the network as the number of requests will keep increasing along with the number of nodes on the network.

There has been research in the direction of merging both the existing overlay networks to try to benefit from the advantages of both in a single hybrid network. These researches have been done using a lot of different combinations of both structured and unstructured protocols. These researches have been in both directions, some creating a structured network of unstructured networks and some opposite. Most of these researches have been able to find a decent compromise in reducing the overheads.

Our approach attempts to break up a P2P network into smaller components. The main component, the core, will be made of a structured P2P network, loosely based off of a Chord network. Smaller branch networks will be connected to this core. These branches will be unstructured networks. This will allow for the structured routing found in the core to combine with the speed and smaller memory footprint that unstructured networks provide.

Our solution differs from others in its implementation of the overlay network topology and node failover algorithms. We will also attempt to devise an optimized solution of hybrid P2P network construction.

In theory we are taking advantages of both structured and unstructured P2P but we don't know for sure our hybrid P2P performs any better in term of speed, fault tolerance, or failure recovery, etc. We will come up with a sound analysis and methodology to prove/disprove our hypothesis.

4. Hypothesis (or Goals)

In this project we intend to present a hybrid P2P network. This P2P network inherently is a structured overlay network that includes a single node from multiple unstructured networks. Our hypothesis is that in this network, although the finding operation will take longer than in a purely structured overlay network, the benefit of having a very low overhead when nodes leave and join unstructured networks will negate the inefficiency of the finds.

5. Methodology

We will develop some tools to measure the performance using some of the aforementioned performance metrics

- Measure the speed of the P2P network -- routing speed (on average), resources search speed, average overlay link latency and network bandwidth consumption
- Calculate the time of faulty node replacement on average
- Calculate the probability of node failure in hybrid P2P network and compare with other solutions
- Plots of number of nodes against average delivery delays and compare with other type of P2P networks

We will be using Java for the tools and the network implementation. Machines in the lab will serve as nodes for our network. We will utilize the LAN already implemented to overlay our network on top of.

6. Implementation

The overall design of this hybrid network is based off of two basic components: membership on the ring (structured) network, and membership on the non-ring (unstructured) network. A node that is located directly on chord network is known as a head, or root node. Any node connected to this node (but not a member of the chord) is this head's family member. In this way, we have created many simulated LANs in which a family consists of a set of nodes where one node is the head of the family, and every other node is connected to only him. To join a head node's family, a new node must hash to the same location (discussed below). It then follows that all nodes within a family hash to the same location on the chord.

When a node sends a request to connect to the network, it only knows about one member of the overall network. This member could be a family head or just a family member, and is in charge of handling the request to bootstrap. This is done by calculating the "desired location" (via a simple hash of IP address and port) of the new node, and then forwarding it along a path until the location has been reached. The request object that is being sent contains a message, in which the new node's ID is stored. This ID could be anything to distinguish nodes, and for the purpose of simplicity we have chosen to represent it as a IP:port address combination.

Once the desired location has been reached, it reads the request to determine the originator of the message: the new node that is requesting to join. It then sends a response to the new node, who has been listening for a response this entire time. This response will contain the desired location, and once the new node receives this, it can read the message and connect to that position. If there is already a node at that location, the new node joins the family, and some of the key/value pairs are sent to it. Otherwise, the new node will take a previously vacant spot, and notify the nodes on either side of it that it has joined. At this point, the node adding process has been completed.

Leaving the network works in much of the same way as joining it. If a node is a family member, it must notify its family head, which then tells all other members of the family. If the leaving node is a family head, it must select a new head, tell its family members that they have a new head, and then notify its predecessor and successor that they each have a new successor and predecessor, respectively.

To store a key/value pair, the key is also hashed to determine its location on the network. The key is then forwarded along a path until it reaches the location, and is stored there. Find works in the same way. The key feature of a chord network is that each node on the chord (in our case, a head of the family) maintains a list known as a "finger table" (see Chapter 7 for more details). The finger table is checked to see which location is closest to the output of the hash, so that the find/store message can jump directly there, rather than traversing the network one subsequent node at a time.

7. Data Analysis and Discussion

In order to test this project, we utilized a lab comprised of multiple IP addresses, linked via a LAN. In order to scale and simulate hundreds of independent computers, we used multiple ports. Outputs were generated based on simple print statements, and we have implemented a very basic shell to send input to the node. The commands include:

store <key> <value>
find <key>
status
fft <number>
leave

The program takes a port and IP:port combination. The port is the node's port, and the IP:port combination is the node on the network, that the new node initially connects to. the "status" command output's the node's status, including neighbors, successor/predecessor, and family head.

Due to the combination of a structured Chord network and an unstructured flooding network, we are able to achieve much better performance than either of the two independently. One of the key concepts is the ability to jump in the chord based on hash values. While this makes finding and storing computationally very fast, the complexities of joining or leaving the network are circumvented by the addition of an unstructured network.

The "fft" command runs a method called "fix finger tables." The finger table is a table that contains the IP:port combinations of the next 2^n nodes on the chord network. This is used to facilitate find and store, making them vastly more efficient than any simply-chained network. However, as nodes join and leave the network, the finger table becomes outdated. In order to maintain its accuracy, the "fix finger tables" method is run periodically. This is a basic tenet of standard chord protocols, as the need to constantly update finger tables for accuracy is a well-known problem. Rather than constantly compute finger tables for each location that a node owns, it becomes much more efficient to run update only for that node at set interval times.

8. Conclusions and Recommendations

The concept of chord networks is implemented to reduce the cost of find and store operations at the expense of add and leave operations. While this may be true for pure chord networks, by implementing hybrid network, we attempt to circumvent this issue.

For the future, a hybridized network can be expanded to include multiple tiers, or roles. Instead of having one head of the family, for example, a secondary head can be selected. Each secondary head in the hybrid network can then form their own structured network, and further reduce find requests.

9. Bibliography

[1]Kai Hwang, Geoffrey C. Fox, Jack J. Dongarra, Distributed and Cloud Computing, Morgan Kaufmann, 2012

[2] S. Androutsellis, D. Spinellis, A survey of P2P content distribution technologies. ACM. Comput. Surv., 2004

10. Appendices

Inputs and outputs:

Starting the hybrid network:

running a single node:

./run_single.sh <port number>

running batch:

./run_batch.sh <# of nodes> <starting port> <IP> <port>

used for running after connecting to <IP>:<port> of node on network

./run_batch.sh <# of nodes> <starting port>

used for running after starting and connecting to <starting port>

Shell:

store <key> <value>

store <value> for <key>

find <key>

find <key>

leave

leave network

fft <number>

fix finger table for <number> hops away