The Algorithm of Sharing Incomplete Data in Decentralized P2P

Jin-Wook Seo[†] , Dong-Kyun Kim[††] , Hyun-Chul Kim[†††], and Jin-Wook Chung[†]
[†]Dept.of Electrical and Computer Engineering, Sunkyunkwan University, South Korea

[††]Korea Institute of Science and Technology Information, South Korea

[†††]Dept.of Computer Science, Namseoul University, South Korea

Summary
This paper takes sharing way of incomplete data in order to improve performance in decentralized P2P network. In order to get file in decentralized P2P network in present respond peer that only have complete data. This is to prevent several problems that occur in case use incomplete data. This paper explains about some algorithms and modules to solved this problem. Through this, wish to improve decentralized P2P's performance.

Key words:
Decentralized P2P , sharing incomplete data, improve performance,

1. Introduction

The Internet, a cooperative network formed by millions of hosts spread around the world, is a shared resource. Applications these days, more so than ever, concentrate on 'how to use the network, consume the bandwidth more efficiently, and send packets faster and safer to more distant locations [1]

Such perspective has become to prefer the P2P model, a horizontal network model in the traditional and perpendicularly hierarchical client/server model.

The P2P technology differs from the existing notion of client/server in that PCs are connected among themselves to share resources and thereby making each participant a client and a server at the same time. In P2P overlay network, peers are able to directly share and exchange information without the help of a server. This results in a prompt and secure sharing of network resources and data handling.

When request file that user wants to get files, do to transmit complete data by response in decentralized P2P Network as Gnutella.

if Peer which have incomplete data sends requested data, user can not know is come on what part of complete data. so it occur problems.

and can malicious attack that receives fake files.

This paper takes sharing way of incomplete data in order to improve performance in decentralized P2P network.

2. Related Work

This chapter takes P2P protocols using incomplete data for file sharing in centralized P2P network.

2.1 E-donkey(ED2K)

The eDonkey network (also known as eDonkey2000 network or eD2k) is a decentralized, server-based, peer-to-peer file sharing network used primarily to exchange audio files, video files and computer software. Like most file sharing networks, it is decentralized; files are not stored on a central server but are exchanged directly between users based on the peer-to-peer principle. [2]

The original eD2k protocol has been extended by subsequent releases of both eserver and eMule programs, generally working together to decide what new features the eD2k protocol should support. However, the eD2k protocol is not formally documented (specially, in its current extended state), and it can be said that in practice the eD2k protocol is what eMule and eserver do together when running. As eMule is open-source, its code is open for peer-review of the workings of the protocol (at the program source code level). Examples of eD2k protocol extensions are "peer exchange", "protocol obfuscation" and support for files bigger than 4 Gbytes, among others. The other eD2k client programs, given time, generally follow suit adopting those protocol extensions. [3]

ED2K (E-donkey) protocol uses data for share basically that cuts 972Kb to one piece of data chunk. Do so that can receive at the same time from several peers without receiving data through this from only one peer.

Even if data is incomplete, it is doing thing so that can share this if have data chunks of complete piece.

This work was supported in part by MIC & IITA(N07008 , Invstigation of high-performance File Transfer System through intelligence TCP tunning)

Hash algorithm that use in ED2K's case to use hash algorithm to confirm whether incomplete datum or each data chunk piece is no mistake is MD4.

Files on the eDonkey network are uniquely identified using MD4 root hash of a MD4 hash list of the file. This treats files with identical content but different names as the same, and files with different contents but same name as different.

Files are divided in full chunks of 9,728,000 bytes (9500 x 1024 bytes) plus a remainder chunk, and a separate 128-bit MD4 checksum is computed for each. That way, a transmission error is detected and corrupts only a chunk instead of the whole file. Furthermore, valid downloaded chunks are available for sharing before the rest of the file is downloaded, speeding up the distribution of large files throughout the network. [3]

A file's identification checksum is computed by concatenating the chunks' MD4 checksums in order and hashing the result. In cryptographic terms, the list of MD4 checksums is a hash list, and the file identification checksum is the root hash, also called top hash or master hash.

It is possible for two different chunks or files to have the same checksum and thus appear the same (see Birthday attack), but the chance of that happening is so small that for all practical purposes it never happens, and checksums are considered unique identifiers. [3]

2.2 BitTorrent

If a displayed equation needs a number, place it flush with BitTorrent is a peer-to-peer file sharing (P2P) communications protocol. BitTorrent is a method of distributing large amounts of data widely without the original distributor incurring the entire costs of hardware, hosting and bandwidth resources. Instead, when data is distributed using the BitTorrent protocol, recipients each supply data to newer recipients, reducing the cost and burden on any given individual source, providing redundancy against system problems, and reducing dependence upon the original distributor.

A BitTorrent client is any program which implements the BitTorrent protocol. Each client is capable of preparing, requesting, and transmitting any type of computer file over a network, using the protocol. A peer is any computer running an instance of a client. [4]

To share a file or group of files, a peer first creates a "torrent." This is a small file which contains metadata about the files to be shared, and about the tracker, the computer that coordinates the file distribution. Peers that want to download the file first obtain a torrent file for it, and connect to the specified tracker which tells them from which other peers to download the pieces of the file.

Though both ultimately transfer files over a network, a BitTorrent download differs from a classic full-file HTTP request in several fundamental ways:

- BitTorrent makes many small P2P requests over different TCP sockets, while web-browsers typically make a single HTTP GET request over a single TCP socket.
- BitTorrent downloads in a random or "rarest-first" approach that ensures high availability, while HTTP downloads in a contiguous manner.

Taken together, BitTorrent achieves much lower cost, much higher redundancy, and much greater resistance to abuse or "flash crowds" than a regular HTTP server. However, this protection comes at a cost: downloads take time to ramp up to full speed because these many peer connections take time to establish, and it takes time for a node to get sufficient data to become an effective uploader. As such, a typical BitTorrent download will gradually ramp up to very high speeds, and then slowly ramp back down toward the end of the download. This contrasts with an HTTP server that, while more vulnerable to overload and abuse, ramps up to full speed very quickly and maintains this speed throughout

The peer distributing a data file treats it as a number of identically-sized pieces, typically between 64 kB and 1 MB each. A piece size of greater than 512 kB will reduce the size of a torrent file for a very large payload, but is claimed to reduce the efficiency of the protocol [5]. The peer creates a checksum for each piece, using a hashing algorithm, and records it in the torrent file. When another peer later receives that piece, its checksum is compared to the recorded checksum to test that it is error-free.[6] Peers that provide a complete file are called seeders, and the peer providing the initial copy is called the initial seeder.

The exact information contained in the torrent file depends on the version of the BitTorrent protocol. By convention, the name of a torrent file has the suffix .torrent. Torrent files contain an "announce" section, which specifies the URL of the tracker, and an "info" section which contains (suggested) names for the files, their lengths, the piece length used, and a SHA-1 hash code for each piece, which clients should use to verify the integrity of the data they receive.

Users browse the web to find a torrent of interest, download it, and open it with a BitTorrent client. The client connects to the tracker(s) specified in the torrent file, from which it receives a list of peers currently transferring pieces of the file(s) specified in the torrent. The client connects to those peers to obtain the various pieces. Such a group of peers connected to each other to share a torrent

is called a swarm. If the swarm contains only the initial seeder, the client connects directly to it and begins to request pieces. As peers enter the swarm, they begin to trade pieces with one another, instead of downloading directly from the seeder.

Clients incorporate mechanisms to optimize their download and upload rates; for example they download pieces in a random order, to increase the opportunity to exchange data, which is only possible if two peers have different pieces of the file.

The effectiveness of this data exchange depends largely on the policies that clients use to determine to whom to send data. Clients may prefer to send data to peers that send data back to them (a tit for tat scheme), which encourages fair trading. But strict policies often result in suboptimal situations, where newly joined peers are unable to receive any data (because they don't have any pieces yet to trade themselves) and two peers with a good connection between them do not exchange data simply because neither of them wants to take the initiative. To counter these effects, the official BitTorrent client program uses a mechanism called “optimistic unchoking,” where the client reserves a portion of its available bandwidth for sending pieces to random peers (not necessarily known-good partners, so called preferred peers), in hopes of discovering even better partners and to ensure that newcomers get a chance to join the swarm.[7]

3. Algorithm for sharing partial data

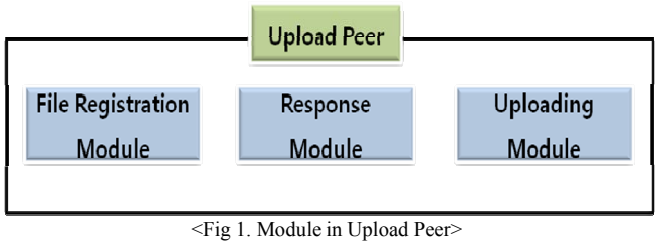
This chapter wish to present way to share incomplete data in decentrilized P2P protocol.

Present pseudo code algorithm in reply with module explanation about peer that share file and peer that request file.

3.1 Upload Peer

Next picture is explaining about modlue that peer that share and uploads file must have.

Peer that do upload can compose by three modules.



<Fig 1. Module in Upload Peer>

File Registration Module is module that register file that user has by oneself to share file. A next table presents algorithm about this module.

<Table 1. File Registration Module Algorithm>

```
A ← sharing data (complete data)
PartSize ← split data size
Part ← partial data from A

File Registration ( A )
1    Hash ( A )
2    IF length ( A ) > PartSize
3      Split ( A , PartSize, part )
4      For t ← 1 to t == length[part]
5        Hash ( part )
6      End for
7    End if
8    Send ( each information)
```

Store hash value of file that user registers using hash algorithm first if register file that want to share by oneself.

And do to divide file as much as data chunk's size that use in P2P protocol.

Save hash value about each data chunk, and each chunk's hash value and hash value for file using P2P protocol each peer inform make.

Do to prevent that fake file or fake data chunk occurs about file that user through this registers.

When Response Module receives search message from other peer, is module that respond in reply.

A next table presents algorithm about this module.

<Table 2. Response Module Algorithm>

```
message ← request message ( include file information )
sharelist ← sharing file list in upload peer

Response ( message )
1    Temp = Find ( message )
2    If temp != NULL & temp == complete file
3      Send ( Response Message )
4    Else If temp != NULL & temp != complete file
5      Send ( sharing file information )
6    Else
```

```
Find ( message )
1  temp = hash value into message
2  For t ← 1 to t == length[sharelist]
3    IF Hash ( sharelist[t] ) == temp
4      return ( sharelist[t] )
5    End if
6  End for
```

Search since peer that receive search message is relevant file among file that share by oneself through hash value for file and file name in message.

If is not, do to send error message that inform that there is no relevant file.

If is, after examine whether relevant file is complete data file, send information message that inform that data chunk of file that if is not perfect data file own has is response message that inform that is part that peer has complete data chunk.

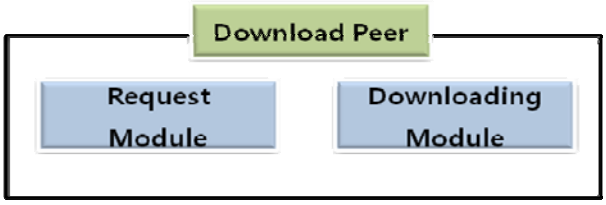
Uploading Module acts role that transmit relevant part if responded other peer requests file.

Because there is information of file to is response mallet, Uploading Module only file to receive download Peer role that send data chunk part that require do.

3.2 Downloading Peer

Next picture is explaining about modlue that peer that share and downloads file must have.

Peer that do download can compose by two modules.



<Fig 2. Module in Download Peer>

Request Module informs and acts role that find peer that have this to peers that connect with own about file that user requires.

Send request message that fill information of hash value and so on in reply with file name that user wants to each peers.

Downloading Module acts role that request data chunk of file to peer that have file that require through search and receive download.

A next table presents algorithm about this module.

<Table 3. Downloading Module Algorithm>

```
message ← response message

Downloading ( message )
1   If file information in message == complete
2     Send ( necessary part ) & download
3   Else
4     boolean t = Find ( necessary part , message)
5     IF t == true
6       Send ( necessary part ) & download
7     End if
8   End if
```

This module that receive message confirms whether peer that have relevant file has one part of data or have whole.

If have whole, own requests necessary data chunk part and this to do download present.

In case have one part that is not whole, confirm whether one part of file that other person peer through information of response message has is necessary part by oneself.

If is part that own has, without doing download request, do download request if have part that own is necessary.

Whether each data chunk's hash value is correct, unite data chunk to single file after is rapidly, and confirm whether also hash value of file is correct and do lest problem should happen.

4. Conclusion

The Internet started off as a perfectly symmetrical network, or a P2P network of users who cooperate with each other. As the Internet became vaster, accommodating an enormous number of users who swarmed to the online world, it created issues such as the bottleneck problem between the local network and the backbone caused by the network overloading between a relatively smaller number of servers and many clients.

Such phenomenon resulted in the emergence of the P2P network, thereby resolving the network difficulties.

This paper takes sharing way of incomplete data in order to improve performance in decentralized P2P network. Such attempt could bring about a P2P network with an improved performance.

References

[1] Nelson Minar & Marc Hedlund, "*Peer to Peer*", oreilly, 1999.
[2] http://www.emule-project.net/home/perl/general.cgi?l=1
[3] Yoram Kulbak & Danny Bickson, "*The eMule Protocol Specification*" , 2005
[4] http://bittorrent.org/introduction.html
[5] http://www.bittorrent.org/protocol.html
[6] Dongyu Qui & R. Srikant , "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks", sigcomm 2004
[7] http://wiki.theory.org/BitTorrentSpecification
[8] Ripeanu, M. *"Peer-to-peer architecture case study: Gnutella network"*, Peer-to-Peer Computing, 2001. Proceedings. First International Conference on 27-29 Aug. 2001 Page(s):99 – 100
[9] M.Bawa, B.F.Cooper, A.Crespo, N.Daswani, "Peer-to-Peer Research at Stanford"
[10] 10. Clip. "The Gnutella Protocol Specification v0.41 Document Revision 1.2"