Design and Study of a BitTorrent-like File Sharing System

Yuzhou Wang
School of Electrical and Computer Engineering
University of Waterloo
Waterloo, ON, Canada
Email: y2345wan@uwaterloo.ca

Sainan He
School of Electrical and Computer Engineering
University of Waterloo
Waterloo, ON, Canada
Email: s66he@uwaterloo.ca

***Abstract*—Peer-to-Peer(P2P) systems like BitTorrent attracted worldwide Internet users' interest with their ability to share content at high speeds. Efficient content transfer of Bittorrent is achieved by splitting contents into many small pieces, each of which may be downloaded from different peers. However centralized trackers in the original BitTorrent protocol do not benefit from the properties of no single point of failure as in P2P. Decentralized mechanisms such as multi-tracker, distributed hash tables (DHTs) and Peer Exchange (PEX) have thus been proposed for robustness. In this paper, we describe the design and implementation of a BitTorrent-like file sharing system. In our design, we eliminate the traditional tracker and adopt ZooKeeper instead. Our tests focus on fault tolerance and performance of the system. The evaluations are based on measurement of download rate and duration. The experimental results show the system meet the requirements of our design. Some future potential enhancement is also discussed.**

***Index Terms*—BitTorrent; peer-to-peer; distributed; file-sharing; tit-for-tat; ZooKeeper**

I. Introduction

Traditionally, file sharing system is deployed through a client/server network. In this architecture, files are stored in a central server and will be sent to clients if it receives their download requests. However, the number of clients and the bandwidth of the server will affect file transfer speed convincingly. Besides this, if the server fails due to some reasons like unreliable network, the whole system will be down.

In order to reduce the high dependant of file-sharing system on the centralized server, there is a peer-to-peer(P2P) file distribution mechanism proved to be a promising method. No central storage is required, and files can be shared by every workstation in the network. Thanks to peers exchanging their files with each other, the workload is distributed resulting in a great performance.

Nevertheless, a free-riding problem presents. It has been estimated[1] that 2040% of Napster users and up to 70% of Gnutella users shared little or no content. Also, Huberman and Adar[2] found that nearly 50% of responses are returned by 1% of the sharing hosts and that nearly 98% of the responses were returned by 25% of the sharing hosts. The unfairness in free-riding situation degrades the performance of P2P network significantly.

Thus, BitTorrent has been emerged to address this problem using tit-for-tat strategy, which prevents from that selfish peers who do not upload but attempt to keep high download rates. It also assists those peers, who just participate in the network without any data to provide, rapidly become uploaders to others.

To realize fault tolerance and coordination, the main component of BitTorrent known as tracker keeps contact with peers by receiving messages from them when they join or leave the system as well as every fixed period. Therefore, there are a lot of messages flowed in the network, leading to a huge consumption of bandwidth. Taking account of this, we propose a novel BitTorrent-like file sharing system with ZooKeeper.

The rest of the paper is organized as follows. Section II provides a brief overview of the BitTorrent protocol and ZooKeeper. In section III, we present related work and discuss their performance. Section IV describes our design rationale and implementation of BitTorrent-like file sharing system, while Section V shows experimental results under a variety of workloads. We conclude and propose future work in section VI.

II. Terminology overview

A. BitTorrent

BitTorrent is a peer-to-peer(P2P) file sharing mechanism, which is designed to facilitate large file transfers and make it more efficient among multiple peers. A large file is divided into blocks with same size except the last block. Based on this assumption, when peers download blocks from the server, they also upload to each other concurrently. Thus, the system achieves selfscaling as its serving capability grows with peers increasing.

The most significant role in the system is called tracker[3], which keeps track of peers to maintain lists of alive peers, those are corresponding to a certain file, either download or upload it. Since peers send messages periodically to and new one also contact with the tracker, the lists will always keep updated. The contact information of the tracker and detail of a large file is stored in a related metainfo file known as .torrent. A peer is any BitTorrent client which takes part in a download task. It can play two roles in the system, seeder and leecher. Any peer which has and upload a complete file to those seek

to download is a seeder. In contrast, those who send a request to the tracker for downloading and get a response with the list mentioned before to contact with other peers is a leecher. However, it will upload the blocks of file it owns to other leechers.

There are three main policies in this scalable file distribution protocol, Local Rarest First(LRF), Tit-for-Tat(TFT) and Choking policy. As a result of that LRF policy enables peers to choose rarest blocks preferentially rather than at random to download, the system avoids a circumstance where it is likely to take some time for peers to find blocks with few replicas and slow down the download rate. TFT policy is proposed to solve free-riding problem in which some leechers are selfish by only downloading without serving others. The aim of it is to achieve a cooperation that upload bandwidth is exchange with download bandwidth. This strategy is complied by fair trading, whereby peers prefer allocating upload slots within threshold to those also send data at a fast rate in return. As to Choking policy, it is raised to assist TFT to control the number of active connections among peers. Download rate and upload rate are used respectively to determine which remote peer to be choked related to whether the peer has a complete copy of the file or not. These strategies allow BitTorrent to use bandwidth between peers (i.e., perpendicular bandwidth) effectively[4] and handle flash crowds well.

B. ZooKeeper

ZooKeeper is a centralized service deployed to assist large-scale distributed system in process coordination across unreliable networks. It provides naming registry, maintaining configuration and synchronization services to alleviate management complexity.

It is developed to have wait-free property related to shared registers along with driven-by-event one, which improves communication performance of the system. The study of Hunt et al. shows that ZooKeeper is able to handle tens to hundreds of thousands of transactions per second for the target workloads, 2:1 to 100:1 read to write ratio[5], and keeps their linearizability with a lock mechanism at the same time. Therefore, it has been proved to have high throughput and low latency.

As the architecture of ZooKeeper is essentially a shared distributed hierarchical key-value store, the mechanism is fairly robust. Distributed processes contact with each other by obtain information from the shared namespace of registers, which is known as znodes[6]. Zookeeper nodes can be read from and write to by processes, the associated data of which can includes configuration, status information, location information etc.

One of its aims is to make large-scale distributed system achieve fault-tolerant control. This means that if some processes fails, others will be notified if they have set a watcher on znodes. The watch feature also helps to update group membership and start a leader election when old processes leave or new ones join.

III. Related works

In order to increase system's availability, some BitTorrent protocol extensions have been proposed and deployed. These approaches consider different mechanisms for peers to discover other peers including: multi-tracker, Distributed Hash Table (DHT) protocol and Peer Exchange (PEX) protocol.

The first approach is multi-tracker. To avoid overloading trackers and have backup trackers against failures, it allows two or more trackers to track one same torrent instead only one tracker. Every peer that participates in sharing a file can be tracked by one tracker and is a member of one swarm. Multiple swarms tracked by multiple trackers which are associated with one file can coexist in parallel. Multiple trackers improve availability, but the improvement largely comes from a single highly available tracker. The performance of small swarms is sensitive to fluctuations in peer participation. Measurements and analysis have shown that peers in small (less popular) swarms achieve lower throughput on average[7]. In [8], the authors studied the availability of multi-tracker observe the correlated failures of different trackers can reduce the potential improvement from multi-tracker. Besides, the use of multiple trackers can significantly reduce the connectivity of BitTorrent overlay.

It is obvious that limited tracker bandwidth will make an effect on upload/download rate of peers. In order to reduce this impact and accelerate building connection between peers, Andrew and Arvid[9] exploit a trackerless structure using Distributed Hash Table(DHT) protocol. In this protocol, each peer has a DHT node, which maintains contact information of closest nodes/peers in a routing table. In another word, a peer can play a role of tracker, and thus locating those peers related with its requesting file by itself, which simplifies the bootstrapping in the original mechanism. For determining the closeness, they use a XOR-based distance metric in Kademlia[10]. As this novel topology uses parallel, asynchronous queries to update routing table, nodes have enough information so that flexible route queries are guaranteed. When in a fault-prone situation, this system has provable great performance in consistency.

The PEX approach makes use of the communication among peers to share the contact information they have with each other periodically. Though multiple versions of PEX have been implemented, their main idea is that peers keep their neighbors informed about their current contact list. With its decentralized nature, PEX can help the swarm survive much longer in case of tracker failures, thus increasing the fault tolerance of the system. Unfortunately, using PEX does not eliminate the need for a tracker because the peer need to request the tracker to know at least one other peer. According to the experiments study in [11], PEX could improve the download performance - the average reduction of the download time was measured to be around 7%. As the peer needs to send messages containing contact lists to every other neighbor, a trade-off on the frequency of messages sent must be considered. [11] shows that over 80% of PEX messages have a freshness ratio greater than 0.5, but there exists a large degree of redundancy in PEX

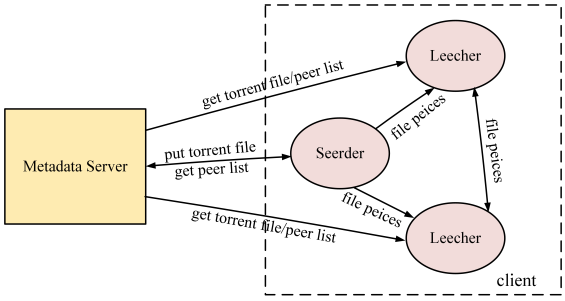

Fig. 1: System Architecture

messages.

IV. Design and implementation

The BitTorrent protocol generally requires the components including tracker, metainfo file containing information about the torrent, original seeder that has the whole content and end user downloaders. Figure 1 illustrates the architecture of our system. The system consists of two main components: metadata server and client.

In our design, we use Apache ZooKeeper to act as a metadata server which provides the information of peer lists of each swarm and metainfo of each file. The top level directories in ZooKeeper consist of /peer and /file nodes. The /peer node is used to hold the hostname and port number information of each peer. The /file nodes maintains the files and swarms. Each file is associated with a descendant node under /file, we store the torrent content in the file node's data. All peers within one file swarm are registered under that specific file node.

A client can be started either as a seeder or as a leecher with a unique peer ID. If the user has a file to share, he can start the client as a seeder. The client will create a metafile and advertise the file to the ZooKeeper. A node with filename as node's name is created under /file and the contents of the torrent information include filename, file size, piece length, pieces will be stored in the node. Meanwhile, the peer is registered under both the /peer node and /file/filename node with the peer ID as the node's name and hostname and port number as the node's data. If the user wants to download a file, he needs to initiate the client as a leecher with the file's name. The client will retrieve the metafile and peer lists associated with the file from the ZooKeeper. The peer is also added to the lists. Then the client can connect to those peers in the returning list to start exchange file pieces. Downloading or uploading multiple files simply involves running multiple client instances.

A. Connection state

A peer must maintain state information for each connection that it has with a remote peer. A peer usually play the roles of both downloader and uploader, Figure 2 shows two finite state machine for these two actions.

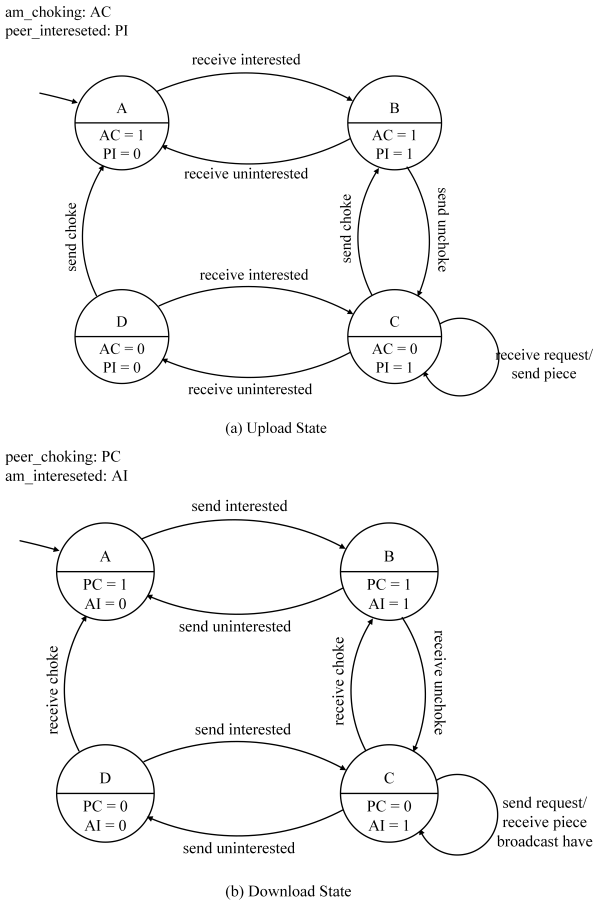

Fig. 2: Connection State

B. Deal with churn

The dynamics of peer participation, or churn, are an inherent property of Peer-to-Peer systems and critical for design. In our system, each client spawns a listening thread to accept new connections from peers joining the swarm later. If receive handshake message successfully, current peer will add the new peer into its peer list. To deal with dropping offline peers, as every node is registered in the metadata server, we set a watch for each peer to monitor the file's node they involve in. If any node in the swarm changes, the peer will be informed and pull the updated peer list from the metadata server with dead peers removed.

C. Choking mechanism

Each peer always unchokes a fixed number of other peers which allows TCP's built-in congestion control to reliably saturate upload capacity[3]. This is decided periodically. In our implementation, a scheduled task is executed at a fixed rate to decide the unchoked sets. For simplifying, we calculate the download rates of all other peers that the current peer provide uploadings to, picks k peers who has the fastest rate. If a peer in the unchoked set is choked previously, then the current peer will send unchoke message to that peer.

D. Piece selection

Selecting pieces to download in a good order is very important for transfer efficiency. For example, if all the peers start to download the first piece from the first bit of the bitfield, it results in all peers having same bits of pieces, and can not exchange files with each other. In our design, we adopted the random first piece algorithm. The peer will check its own bitfield, pick the bit with zero randomly and request that index of piece from other peer.

E. File integrity check

The data of /filename znode has a string called pieces, in which 20-byte hash values related to each piece of the file respectively are concatenated. Secure Hash Algorithm 1(SHA1) produces a hash value for each piece and render it as a hexadecimal number. These SHA1 values, regarded as message digests, give us a favor to check if the file is completely downloaded during carrying on experiments.

V. Experiments and Results

Our system is tested on ecelinux[1-3].uwaterloo.ca. The ZooKeeper service is on snorkel.uwaterloo.ca. The measurements focus on fault tolerance and evaluation of performance. We ran the system with three sizes-small, medium, large- of files of various types. There are 6 peers in this experiment, one is the seeder and the other 5 are leechers. In each test, a peer as seeder is started at the beginning, then different numbers of leechers ran simultaneously or non-simultaneously.

A. Fault tolerance

For fault tolerance, as the centralized tracker is already eliminated in our design, ZooKeeper introduce failures to peers. There are two kinds of potential failures: uploader failure and downloader failure.

After killing several uploaders, existing alive peers get an updated peer list. They close the connection with those died peers and then keep contact with peers still on the list along with exchanging file pieces.

On the other hand, we killed a particular peer who acts as a downloader before finishing and then restarted it to find out if it has capability to recover unfinished download task. When peers finish downloading, we examined the downloaded file by comparing its hash value generated by Secure Hash Algorithm 1(SHA-1) with that of the original file to check the correctness.

The experimental results prove that our implementation is able to handle both situations well.

B. Evaluation of performance

Performance refers to the transfer rate and download duration in terms of ranging swarm size and file size. files are split into fixed-size pieces which are all the same length of 256K except for possibly the last one which may be truncated. Unchoke interval for the choke task is 1s, unchoke slot is set to contain 4 peers. All the tests are run 10 times and all results are averages from 10 runs.

TABLE I: Download Time of Different Swarm Sizes

Test #	size 1	size 2	size 3	size 4	size 5
1	17874	12086	6817	6802	7904
2	14305	11394	5278	7288	6897
3	19312	9793	9747	7229	7901
4	25102	9825	8751	8734	6208
5	23135	10915	11600	6765	5856
6	24998	12018	7847	7138	6299
7	20161	9402	6862	8708	6102
8	25185	15401	9814	7876	7778
9	24632	10248	10264	7834	7323
10	21093	12885	8194	6737	6789
Average(ms)	21579.7	11396.7	8517.4	7511.1	6905.7

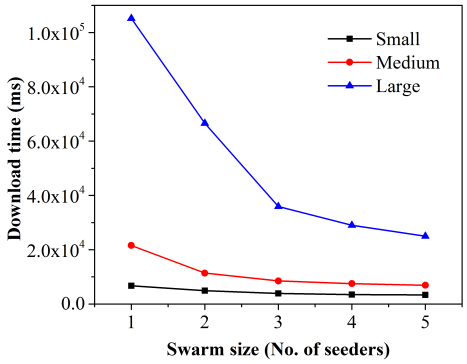

Fig. 3: Download Time vs. Seeder size

1) Performance as a Function of the Swarm Size(number of seeders): This experiment aims to show that BitTorrent performance improves with increasing seeder size. The data points are collected through running experiments with swarm sizes 1 to 5. At each run, one new peer joined a swarm until completing download. The swarm was initially created with one seeder then the rest 5 peers entered the swarm one by one after the preivous peer finishing downloading. Thus, peers that obtain the complete file can be seen as seeders. That is "n seeders-one leecher".

Table 1 shows the measured download time of newly joined peer of 10 runs for the tests. The file size is medium. Figure 3 and 4 represent average download time and average download rates for newly-joined peers. The performance increases as seeder size changed from 1 to 5.

The results show that larger swarm performs better than smaller one. We see that the the performance of average download time and download rate for all 3 sizes files increases as the swarm size increases. It is obvious that the larger sizes the swarms are, the more peers that can provide file pieces to the new peer so that the download can speed up. We also notice that the decrease of download time is dramatical form size 1 to size 3 and then tends to flat. One possible reason for this tendency is that with more providers, the leecher may send redundant requests and messages which affects the efficiency.

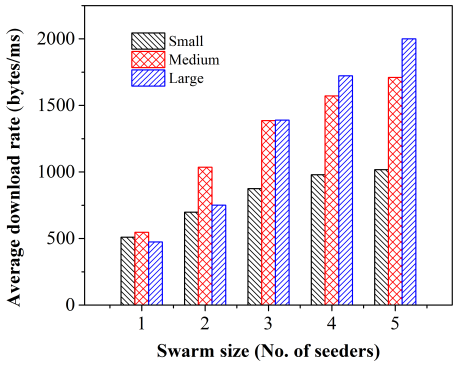

Fig. 4: Average Download Rate vs. Seeder size

TABLE II: Average Download Time of 1 Leecher

file size	medium(ms)	large(ms)
leecher_1	21579.7	105223.1

TABLE V: Average Download Rate of Different Leecher Sizes

file size	size_1	size_3	size_5
medium(byte/ms)	547.17	1149.1	1195.2
large(byte/ms)	474.68	1441.3	1828.6

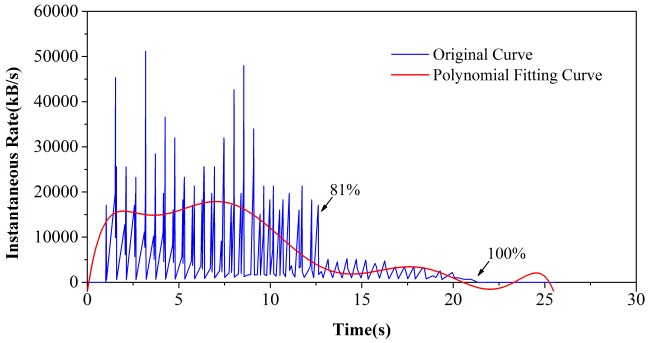

Fig. 5: Estimated Instantaneous Rate vs. Time

2) Performance as a Function of the Leecher Size(number of leechers): In this experiment, the swarm is initiated with one seeder and then we start different number of leecher peers in the same time making their download simultaneous. This is "one seeder-n leecher". The measurement of performance is for 1, 3, and 5 newly-joined leechers respectively. The download time and average transfer rate is shown in table 2-5.

Comparing the results in these tables, we see that the average download time of every leecher peer decreases as the leecher number increases and the average rate increase accordingly. All leechers are started concurrently and they are downloading the same file, the improvement of their performance is due to the advantages of Bittorrent protocol that peers can share pieces with each other during downloading. Another reason is that we adopted the random piece selection algorithm so that the peers are very likely to obtain different pieces at the very beginning, thus they can upload to and download from others simultaneously. This balances the load from the only seeder to all peers.

3) Estimated Instantaneous Rate: The estimated instantaneous rate here is defined as a "windowing" average rate which is the bytes length per piece received divided by the time difference between current piece received and last piece received. We plot the instantaneous rate with time from the download established to download finish in Figure 5. We fitted the curve with polynomial fitting.

We can see that instantaneous rate slows down after about 81% completion. There are two reasons for this tendency. One is that the peer select piece it does not own randomly. In our implementation, we generate a random index number and check this index with local peer's bitfield to decide whether the peer need to request. Thus, when most of the bitfield is already set, it requires more runs to generate an index of bitfield which is not set yet. The other reason is that the peer may send multiple requests for same piece to remote peers and receive redundant content. To speed this up, the peer can send requests for all of its missing pieces to all remote peers and enable cancellation every time a piece arrives.

TABLE III: Average Download Time of 3 Leechers

file size	medium(ms)	large(ms)
leecher_1	10291.7	35817.2
leecher_2	1066.5	34736.4
leecher_3	10467.5	33409.6
Average	10275.2	34654.4

TABLE IV: Average Download Time of 5 Leechers

file size	medium(ms)	large(ms)
leecher_1	9348.7	28307.3
leecher_2	10477.0	27830.9
leecher_3	10561.6	26701.2
leecher_4	9757.1	26273.5
leecher_5	9252.6	27462.6
Average	9879.4	27315.1

VI. Conclusion and Future Work

This paper reveals a BitTorrent-like system, which is a decentralized peer-to-peer file sharing system. This work uses ZooKeeper to replace the metainfo file(.torrent) and tracker in native BitTorrent protocol, thus improving bootstrap and reducing messages flowed in the network.

A new observation of free-riding problem in BitTorrent has experimentally demonstrated[12] by Sirivianos and Chen

regardless of tit-for tat mechanism. When a peer connects to many peers, it increases its chance to be unchoked by leechers and find out seeders. As a result, it may adopt an exploit strategy. Some modifications should be made in the system to allow peers to detect whether a peer tries to become selfish or not before establishing a connection with it.

For the reason that tit-for-tat strategy upload speed to determine which peers will be put in unchoked list, those new entrants without any data have no opportunity to start downloading before some peers finish. Consequently, we will add a new mechanism that enables new peers to upload a piece randomly when they take part in the network. Moreover, an extra unchoked space is offered to give those new peers or peers with a low download/upload rate a chance.

REFERENCES

[1] Saroiu S, Gummadi P K, Gribble S D. Measurement study of peer-to-peer file sharing systems[C]//Electronic Imaging 2002. International Society for Optics and Photonics, 2001: 156-170.

[2] Adar E, Huberman B A. Free riding on Gnutella[J]. First monday, 2000, 5(10).

[3] Cohen B. The BitTorrent protocol specification[J]. 2008.

[4] Bharambe A R, Herley C, Padmanabhan V N. Analyzing and improving BitTorrent performance[J]. Microsoft Research, Microsoft Corporation One Microsoft Way Redmond, WA, 2005, 98052: 2005-03.

[5] Hunt P, Konar M, Junqueira F P, et al. ZooKeeper: Wait-free Coordination for Internet-scale Systems[C]//USENIX Annual Technical Conference. 2010, 8: 9.

[6] Carlos D. Morales. Apache ZooKeeper Description[J]. 2008.

[7] D. Menasche, A. Rocha, B. Li, D. Towsley, and A. Venkataramani. *Content Availability and Bundling in Swarming Systems.* in Proc. ACM CoNEXT, Dec. 2009.

[8] G. Neglia, G. Reina, H. Zhang, D. Towsley, A. Venkataramani, and J. Danaher. *Availability in BitTorrent Systems.* in Proc. IEEE INFOCOM, May 2007.

[9] Loewenstern A, Norberg A. DHT protocol[J]. 2008.

[10] Maymounkov P, Mazieres D. Kademlia: A peer-to-peer information system based on the xor metric[C]//International Workshop on Peer-to-Peer Systems. Springer Berlin Heidelberg, 2002: 53-65.

[11] Wu, Di, Prithula Dhungel, Xiaojun Hei, Chao Zhang, and Keith W. Ross. *Understanding Peer Exchange in BitTorrent Systems.* in Proc. of IEEE Peer-to-Peer Computing (P2P), 2010.

[12] Sirivianos M, Park J H, Chen R, et al. Free-riding in BitTorrent Networks with the Large View Exploit[C]//IPTPS. 2007.