P2P Resource Pool

Ali Tariq **Amit Baran Roy** **Siddharth Lanka** **Tanmai Gajula**

Department of Computer Science,
University of Colorado Boulder,
Email: {Ali.Tariq, Amit.Roy, taga5342, Sai.Lanka}@colorado.edu

Abstract

Peer-to-peer Computing has been one of the emerging areas of research for the last decade. Ever since the emergence of peer-to-peer file sharing systems, researchers have been trying to extend this work to a more generic resource sharing abstraction. There have been considerable efforts to harness the benefit of peer-to-peer networks; scalability, reliability and low-cost ownership - such as Avaki and Seti@home, which try to use large scale networks for compute resources. In this work, we aim to build a peer-to-peer resource sharing abstraction service that looks at resource layers of the peers in the systems and tries to provide seamless resource sharing channels.

Keywords: Peer-to-peer Computing, resource sharing, P2P file sharing

1 Introduction

Peer-to-peer networks have become one of the emerging areas of research in recent years. The main advantages being scalability;improving scalability by avoiding dependency on centralized points - simplicity; eliminating the need for costly infrastructure by enabling direct communication among clients - and reliability; data replication and fault-tolerance - has kept it very desirable for the existing large scale network today. Peer-to-peer network are no-longer restricted to file-sharing platform, the resource further encompass compute power, data storage, network bandwidth and presence (computers, humans and other resources).

Peer-to-peer architecture refers to a class of systems that engage distributed resources to execute a function in a decentralized manner. For a more generic overview, P2P is about; giving to and taking from the peer community. In peer-to-peer networks, peers depending on each other for getting information, resources, forwarding requests, etc. Each peer can serve as both, a client and a server. A computer part of the peer-to-peer network can initiate a query (as a client) for some data or routing information and any other computer, again part of the same peer-to-peer network can respond (as a server) to facilitate the client.

Peer-to-peer Computing is still a relatively new field trying to emerge as an alternate to centralized and server-client model of computing. P2P term is often confused with other terms such as distributed computing [Coulouris et al. 2001], grid computing [Foster and Kesselman 1999], and adhoc networking [Perkins 2001], however it is a completely different architecture whose main goal is only to satisfy the needs of its users. In general, due to the immense benefits of P2P networks people started adopting such channels that has lead to proportionate research in the area but with the wide-spread use, came the mixture/-confusion of terminologies as well as new security concerns in this domain. Lack of accountability precautions in the peer-to-peer networks had increased the abuse-case scenarios for the users such as free-loading, unfair resource usage, etc. The most prominent security concern came from the anonymity - provided to the users which made it extremely difficult for accountability reasons.

Peer-to-peer has seen a lot of growth in various aspects of the existing IT domain. In this work, we plan to provide a generic higher level abstraction layer service that could enable seamless resource sharing between the peers with peer-to-peer architecture. We aim to create a P2P resource pool, comprising of all the users in the network - which would be equally accessible by every peer. Our work will currently focus on providing storage level resource pool where peers can save the data in the pool; the files would be saved in smaller chunks throughout the network to ensure consistency and fault-tolerance, and the ownership of the files will be tracked by co-ordinator(s). Later we would want to extend this work to include compute resources as well. Major contributions of our work include: 1) a generic high level level abstraction platform to provide various resource sharing options, 2) a proof-of-concept of the proposed architecture.

Peer-to-peer has raised a lot of security concerns for the users and the accountability concerns for the organizations, mainly stemming from the anonymity features of the P2P networks. In general, P2P generic resource sharing is still a technology in early stages of development that still requires more rigorous effort to make a more generic and safe platform for the users.

2 Related Work

Peer-to-peer architecture has seen a lot of growth which has further ignited various research areas in this domain. One of the most mature area in P2P remains file-sharing systems. Peer-to-peer distributed file systems are based on the idea that we can replicate our personal files and save these

replicas in the shared spaces from all other network users, as well as sharing of resources such as hard disk, processing and network bandwidth etc. Researchers have proposed storage model based on p2p network [7] which decentralizes file storage and its location is completely transparent to the user, as opposed to many of the other distributed file systems which offer lower levels of transparency. P2P file-sharing systems heavily rely on the data-replication and redundancy strategies[10] to comply with the dynamic nature of the network. One recent work[13] further improves replication consistency by providing a dynamic data replenishment algorithm to ensure data safety for its peers.

Unstructured p2p networks provide benefits of decentralization as well as ease/convenience of nodes joining and leaving the network. But due to lack of a centralized server / entity , they usually adopt flooding algorithms in order to locate resources in the network. This leads to a heavy burden on network bandwidth and low search efficiency due to large number of query messages. Several improvements have been proposed in literature including forwarding based, topology based and caching based algorithms. [5] Some of the more recent work builds over these algorithms such as the active index caching search algorithm which maintains indices caches for each of the nodes on the network for storing information regarding shared resources of the nodes, neighbouring nodes, recently accessed nodes and those that actively publish their indices and improves search efficiency.

Super peer overlay provides an efficient way to run distributed applications on peer to peer networks. The basic idea is to assign nodes with high capacity (a combination of available computational resources, network connections, and lifespan in the network) as super peers which connect to other super peers in the network. The remaining peers (client peers) connect to these super peers, and communicate with other peers through these super peers, which shields these lower capacity nodes from massive query traffic, thus improving the performance of the applications. [6] The super peer selection process includes periodically rebuilding super peer overlay to achieve efficiency and robustness to yield better performance over a dynamic p2p network environment which is generally characterized by node failures and addition of new nodes. In [2], a set theory based model was implemented to share mobile mp3 songs based on region using p2p and adhoc networks technologies. The set theory algorithm is used to detect common mp3 songs of the public playlists shared by peers at a given region. These common songs can be streamed over the air without downloading by any other peer in the same region. Region is identified using GPS coordinates which are attached to the songs of the playlists. GPS coordinates are used to show peers what specific songs are being listened by other peers in a given region.

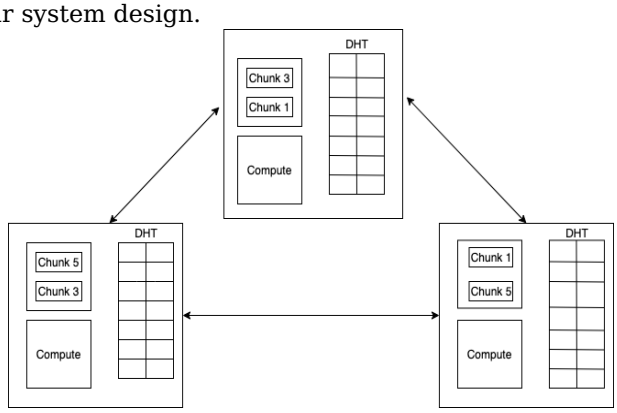
P2P computing became a commercial success since it's rise in popularity in 2000's but factors like lack of robustness, flash crowd, free riders, leavers/offline peers and popularity of cloud computing are leading to its decline lately. A potential future of P2P computing is P2P cloud where peers can make use of available gadgets at home to contribute to cloud and offer services to customers by normal billing [8].

High throughput computing like P2P computing needs efficient job scheduling so that high demanding job gets sufficient resources. An efficient time balanced job scheduling algorithm was implemented [4] that makes use of the divisibility property of jobs. Based on the completion time of indivisible and divisible jobs, the algorithm assigns them to semi occupied workers or else idle workers. Then it assigns the remaining divisible jobs to workers in the same way. The algorithm is adaptive, dynamic and scalable.

Resource sharing in a p2p network is challenging to package it into an efficient solution. [1] provides a p2p learning framework with live streaming of video content and features such as a discussion forum, marking annotations and file sharing integrated into one platform. A unique aspect of this paper is the addition of a shareable whiteboard feature which can be used in tandem with live streaming. They use a chord-like Distributed Hash Table (DHT) for the file sharing and searching features. Distributed directory services are explored in [3] which uses a system on mobile devices(SyD) middleware to implement such a service in a consistent fashion. They provide a framework to build applications on such peer-2-peer networks. They further explore this idea through an application called the inter-vehicle communication application that notifies the driver of available parking spots in a parking lot, traffic 1 mile ahead and other such useful metrics. These applications are a perfect example where a client can request data from it's own local cache or from a neighbouring node to get relevant up-to-date data. Information discovery presents a view of the resources that are available on a p2p network. This is useful for a p2p network with hetereogenous resources to identify the resources available on the network. [9] uses a clustering algorithm to efficiently summarize the information on the resources on a P2P network.

3 System Design

This section details the individual components of our system design.



Architecture diagram

3.1 Abstraction Layer

The purpose of the abstraction layer is to provide the users with a generic interface to share multiple

resources with the rest of the peer-to-peer community. These resources can include but not limited to Storage, Compute, network-bandwidth. The idea is to have a generic enough platform where users can opt to share any specific type of resource at any point in time with small changes in configurations. The abstraction layer shows a collective pool of resources on the network, for each user and is suppose to provide a seamless transition between various resources being shared on the peer-to-peer community.

3.2 Coordinator

The coordinator(s) are the main nodes working to keep the P2P networks alive and working properly. The jobs of the coordinator(s) can be further divided as follows:

3.2.1 Job Placement

In our model of the P2P resource pool, multiple users will be triggering jobs which in case of file sharing, can be a simple file retrieve (read) or a file store (write) operation. The coordinators will be responsible for assigning the jobs to the available nodes using DHT. Then DHT will be ensuring that all the jobs are evenly distributed to the nodes. A key is passed through a hash algorithm that serves as a randomization function. This ensures that each node in the network has an equal chance of being chosen to store the key/value pair. Because DHT nodes don't store all the data, there needs to be a routing layer so that coordinators can locate the node that stores a particular key. Therefore, the process of finding or storing a key/-value pair requires contacting multiple nodes.

3.2.2 Tracking Jobs

Coordinator nodes record information regarding the peers in the network, status updates, the locations of the file contents / chunks of data and resources (storage space, computing power) as well as the progress on the current jobs, that have been shared among the peers. Coordinators would also be responsible for communicating with other tracker nodes in the network to exchange information regarding the list of peers maintained by each of them.

3.3 Replication/Redundancy

The system provides a collective view of resources which in the case of file-sharing, further requires fault-tolerance measures such as data-replication. Replication of files is useful especially while sharing storage. Each file is split into chunks and these chunks are replicated across peers. The job of the replication layer is to maintain multiple copies of the chunks to account for high availability. This layer works in tandem with the DHT to identify the nodes where the replicated chunks could be stored. This can be done with a background process that provides a consistent view of the replicated chunks for both write and update workloads.

4 Implementation

We have a centralized server which holds all metadata regarding the network. We make use of Redis a key-value store to store all peer and file related information in the network. We have a flask server on top of Redis to handle operations such as node joins, file metadata information etc. Each coordinator is a flask server that talks to the Redis server to fetch information about a particular file. When a file is requested Redis server sends the IP of the peer containing the file. Once the IP information is received the coordinator makes a REST call to the peer to download the file to the disk. Similarly, to store a file a peer makes a request to the Redis server asking for an IP where this file could be stored in the network. Once the IP is received the peers communicate with each other and exchange file related information. All exchanges between the peers use binary data for the file transfers. Flask is a HTTP framework to build web applications and the coordinator and the Redis servers make use of flask to communicate with each other. We use the requests module in python to make HTTP calls between services. Files are stored in the hash table using a specific notation separated by a colon. Any peer can request the server for all the files using a redis query scan on the table. The coordinator is also responsible for splitting the files into chunks and replicate across the network with multiple peers. Each peer will also have a tracker program which keeps note of all job related information and handle retries.

4.1 Coordinator

This paragraph talks about operations supported by the server. One endpoint handles node joins in the network. This endpoint receives an IP and adds this IP to the peer list. The server should also prune this list when a peer leaves the network. We capture SIGINT signals on the coordinator process and send a request to the server when the peer leaves the network. The server also provides a query layer and handles multiple requests from clients which use this server to access information stored in the hash table. Few such operations are to find the active nodes and to find all the files in the network. The servers provides these two functionalities as two endpoints. Server handles updating a key-value pair when a new file is uploaded to the network and also when file metadata is requested by a peer. There are two endpoints called get and set to handle these two operations. Since the peers directly speak to each other during file transfers, the coordinator supports two endpoints to send and receive file data in binary format. Currently, to retrieve a file the peer first requests the server to find the IPs of all the nodes that have access to the files. A random IP is chosen from this list to request the file from that specific peer. We transfer base64 encoded strings and convert these strings back to binary data when writing the contents of the file to disk.

4.2 Resource Sharing

One aim of this project is to provide as abstraction layer for the cumulative resources in the pool,

to the users. This required every new user joining the network, to convey the amount to candidate resources to the coordinator. In order to get the available resources, users make use of python3 subprocess module to query the system using python run-time. The coordinator keeps track of the total resources available in the pool which are updated after any change in the number of nodes in the network.

4.3 File Splitting

For implementing file splitting, we modified our initial implementation of node join. So now when a peer joins the network, we are storing both the IP and available disk space information of that peer in the Redis Server. The file splitting algorithm is based on the concept of Round Robin. So, when any node calls the file 'api/store' REST api, we are retrieving all the available nodes and corresponding free space information from Redis Server. We are adding all the nodes which has sufficient free space in the round robin ring. For file splitting, we are setting chunk size as 1MB. First, we are splitting all the available files to store into respective chunks of size 1MB each. Then based on the round robin approach, we are storing each of those file chunks in corresponding nodes of the ring. Also, we are updating the Redis Server with the chunk numbers and the corresponding IP those chunks are stored for each original file. This information is used while retrieving the chunks.

4.4 File Replication

For protection against failure of nodes and high availability, we added replication of file chunks to our implementation design. We introduced a replication factor into Redis Server key-value store, which is configurable on startup. This will set the number of replicas of each file chunk to be distributed across the coordinators. Whenever a node requests to upload a file in the network, the file is split into chunks each of which are replicated on the requesting peer's local system according to the replication factor. The file chunks are distributed across the current list of available nodes (those which have sufficient free space to store the chunks). In case of a replication factor of 1, the file chunks are each transferred to the available nodes in the network in a round robin fashion. For a replication factor > 1, we randomly choose a number of nodes from the available list of peers equal to the replication factor, and transfer the replicas of each of the file chunks accordingly.

4.5 File Merge

Whenever a node requests for a file, it invokes the 'api/fetch/<FILENAME>' REST API call on the Redis server which returns the list of nodes containing chunks of the requested file. The received information contains the locations of each of the file chunks and replicas on the respective nodes. The requesting node then proceeds to contact each of these nodes via the 'api/fetch_content' REST API call and requests for the chunks in sorted order. Here, for each file chunk, the node contacts all the peers that have its replicas, until it gets a successful reply from any one of nodes. Once the node receives all the chunks of the file from the peers, these chunks are aggregated together according to their respective chunk numbers which specifies the order in which the file needs to be rebuilt. This is done on the requesting nodes local system.

4.6 Node Leave

We implemented the node leave functionality for a peer which decides to leave the network. If a peer decides to leave the network, first we are transferring all the chunks which it currently holds to a randomly selected node in the network which is still connected and have sufficient space. After the chunks are transferred, we are also updating the Redis server where we remove the chunk numbers associated with the node IP that is leaving the network and then updating those chunk numbers for the random IP to which it was transferred. Finally, we are invoking '/api/leave' REST call to remove the node entry from the list of available IPs connected in the network and updating the key-value store in the Redis server.

Using the implementation above, we completed a working proof-of-concept of the proposed design. Now we will provide the detailed evaluation of our implementation, in the section below.

5 Evaluation

5.1 Premliminary

```
{
    "keys": [
        "file:sample.pdf"
    ]
}
```

Fig 1:- Files in the P2P network

```
{
    "key": "nodes",
    "value": [
        "127.0.0.1:5001",
        "127.0.0.1:5002"
    ]
}
```

Fig 2:- List of peers in the network

```
{
    'nodes': [
        '127.0.0.1:5001',
        '127.0.0.1:5002',
        '127.0.0.1:5003'
    ]
}
```

Fig 3:- Node Join operation of a new peer

```
URL to retrieve files stored in network:
    http://<ip>/api/fetch/file:sample.pdf
{
    "message": "File downloaded to disk"
}
```

Fig 4:- Success Response once file is retrieved from the peer

5.2 Extension

```
{
    "key": "nodes",
    "value": [
        "10.128.0.9:5000",{'total': 9, '
            used': 1, 'free': 7},
        "10.128.0.10:5000",{'total': 9, '
            used': 1, 'free': 7},
        "10.128.0.11:5000",{'total': 9, '
            used': 1, 'free': 7},
        "10.128.0.12:5000",{'total': 9, '
            used': 1, 'free': 7}
    ]
}
```

Fig 1:- List of peers in the network with disk metrics in GB. Peers join by API call 'api/join'.

```
{
    "key": "file:input_file.txt",
    "value": [
        "10.128.0.9:5000":[3,5,6,7,9,10,12
            ,15,19,20,21],
        "10.128.0.11:5000":[1,2,4,5,7,8,9,
            10,11,13,14,15,16,17,18,19,21],
        "10.128.0.12:5000":[1,2,3,4,6,8,11
            ,12,13,14,16,17,18,20]
    ]
}
```

Fig 2:- A file uploaded by peer IP 10.128.0.10:5000 using API call 'api/store' and chunks got replicated with a replication factor of 2 in other peers. The chunk numbers are stored in an array for each IP as shown and replicated twice.

```
{
    "key": "file:input_file.txt",
    "value": [
        "10.128.0.9:5000":[],
        "10.128.0.11:5000":[1,2,3,4,5,6,7,
            8,9,10,11,12,13,14,15,16,17,18,
            19,20,21],
        "10.128.0.12:5000":[1,2,3,4,6,8,11
            ,12,13,14,16,17,18,20]
    ]
}
```

Fig 3:- Peer with IP 10.129.0.9:5000 decides to leave the network using API call 'api/leave'and its chunks got transferred to another peer with IP 10.128.0.11:5000 as shown.

```
{
    "key": "nodes",
    "value": [
        "10.128.0.10:5000",{'total': 9, '
            used': 1, 'free': 7},
        "10.128.0.11:5000",{'total': 9, '
            used': 1, 'free': 7},
        "10.128.0.12:5000",{'total': 9, '
            used': 1, 'free': 7}
    ]
}
```

Fig 4:- Updated node list of the network after peer with IP 10.129.0.9:5000 left the network.

6 Future Work

This abstraction layer could be extended to include sharing of other resources such as compute and network bandwidth. We present this paper with an example for storage but this model could really be extended to sharing compute for training cpu intensive workloads. Another area where we could experiment is decentralizing the coordinator by using a DHT and arrange the nodes in a ring structure in the network. We could experiment with different deployments of the coordinator both centralized and in a DHT fashion. If a centralized approach is chosen, replication of the redis abstraction layer is useful to handle failures. The incentives w.r.to the resources shared could be modeled based on high contributing nodes i.e., nodes that contribute maximum resources to the p2p network could gain better incentives while accessing other resources. An entire monitoring stack of software that can track the status of resource sharing operations to better understand the statistics on the resources shared and consumed.

7 Conclusion

Peer-to-peer architecture is extremely beneficial and still hold high value against various limitations of modern resource sharing system. In our implementation we show how p2p can be used to increase resource utilization and reuse without incurring any substantial cost. Furthermore, the flexibility in degree of resource sharing and freedom of contribution (no contractual agreement) makes it very desirable to deploy. Although our design only provides abstraction layer for disk storage, the same concepts can be further extended to other resources such as compute and internet-bandwidth which can make it far more beneficial and popular for the users. Our project repository can be accessed at https://github.com/nomad1072/DS-project

References

[1] Nauman Shah *P2PeL: a peer-to-peer computing frame-work for eLearning*, 2002.

[2] M. Wiberg *FolkMusic: a mobile peer-to-peer entertainment system*, 2004.

[3] Sunetri Priyanka Dasari *Peer-to-Peer distributed syd directory synchronization in a proximity-based environment*, 2007.

[4] Moon Yh., Nah Jh., Youn Ch. *"A Time Balanced Job Scheduling Using Divisibility Property for High Throughput Computing in Hybrid Peer-to-Peer Networks"*, 2008.

[5] Pingjian Zhang, Sanqing Li *An Active Index Caching Search Algorithm for Unstructured P2P Networks* , 2009.

[6] Meirong Liu, Erkki Harjula and Mika Yliantila *An efficient selection algorithm for building a super-peer overlay*, 2012.

[7] Gerardo García-Rodríguez, Francisco de Asís López-Fuentes *A Storage Service based on P2P Cloud System*, 2014.

[8] Kisimba P., Jeberson W. *"Future of Peer-To-Peer Technology with the Rise of Cloud Computing"*, 2017.

[9] Eduardo Huardo *"Summary Creation for Information Discovery in Distributed Systems"*, 2011.

[10] Zhang Jian-lin, Zhang Jian-lin *"Study on Redundant Strategies in Peer to Peer Cloud Storage Systems"*, 2010.

[11] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose *"Peer-to-Peer Computing"*, 2002.

[12] Fetahi Wuhib, Rolf Stadler *"Allocating Compute and Network Resources Under Management Objectives in Large-Scale Cloudsg"*, 2013.

[13] Kien Nguyen, Thinh Nguyen, Yevgeniy Kovchegov *"P2P Distributed Data Replenishment"*, 2011.