Distributed File Download Network

Siddharth Rayabharam and Sumant Suryawanshi

Contributing authors: nqr5356@psu.edu; szs7220.psu.edu;

Abstract

With the decreasing cost of cloud storage and its convenience over local storage a lot of files and softwares are now available to download from the internet, but this is an issue since large organizations have limitations on the bandwidth usage and data usage per system due to very heavy network traffic, if a person in such an organization needs to download a large file or software they might not be able to do so. To overcome this we propose a distributed file download network that uses other systems on the same local network as the user to support partial download of files on multiple systems and then regenerate the original file since there is no restriction on the network usage on the local network. This also leads to faster download of large files since partitions of the file can be downloaded on multiple systems in parallel. After testing of this application we see a significant gain in terms of download time for large files, for smaller files the overhead led to very low performance gain.

Keywords: Distributed systems, Peer to peer, file sharing, file download network, peer-assisted file download

1 Introduction

Due to technological advancements, the world has witnessed a massive paradigm shift in the way two machines communicate. Peer-to-peer communication is increasingly being used for a variety of purposes. Data transfer speeds are fast enough to share/store files online. However, many software applications these days generate massive files in order to save their progress. Due to the large number of devices on the network, bandwidth/data usage is restricted in many organizations such as educational institutions, government entities, and private establishments. It is difficult for a user operating on such a network to download huge files from an external network, which may take a very long time or exceed the data use policy imposed by network managers. This may have an impact on the efficacy or productivity of users on such networks.

Segmented file-transfer is a software method that is intended to improve file download speed. It works by simultaneously downloading different portions of the computer file sourced from either multiple servers or from a single server, recombining the parts into the single file requested. The majority of Download Manager applications work in this way. However, we propose a different approach to solve the problem of data usage policy violations. We propose a parallel downloading solution where each segment of the file is downloaded by a different client and reconstructed back as a single file in a client. We will try to achieve this by download each chunk in each node of a local network and transfer chunks to the initiator node using Peer-to-Peer (P2P) [1] transfer. We intend to create a solution which will reduce the bandwidth used per node and improve the overall download time.

The project [2] is created to attain the ultimate goal of reducing the data and bandwidth usage policy violations and achieving faster and safer download of huge files. The paper discusses about the core concepts of operating systems and distributed systems that are used in order to implement the project. This paper gives some details about the literature review or related word that were used for the project in Section 2. In section 3 and section 4, this paper provides detailed explanation about the methodology and architecture details of the project. We later provide some results of the project and introduce the future works in section 6 and section 7, respectively.

2 Literature Review

Peer-to-peer networks are built on top of a physical network architecture using an unstructured virtual overlay network. It began to acquire traction for file sharing. Peer-to-peer file sharing outperforms traditional file sharing. Shetty et al. [3] describe a peer-to-peer file downloading technique in which many clients download parts of the original file. Our project implements the authors' solution, and this paper serves as a foundation for the reference. Furthermore, Min Yang et al. [4] propose a peer-to-peer file sharing scheme based on the network coding PPFEED that can serve as peer-to-peer middleware developed within the web services framework for web-based file sharing applications.

In P2P networks, the peer selection used to work on Trust and Reputation model [5]. Later different techniques like Locality Based Selection Algorithm and P4P have been discovered which are efficient in peer selection without wasting too much bandwidth by sending minimal amount of messages.

Furthermore, a study [6] examined Xunlei, a popular Chinese peer downloading manager, and its architecture and download behavior. In Xunlei, the client starts downloading from a website and establishes a TCP connection with the resource management server. The Xunlei resource management server uses the client's download URL to execute a DNS lookup and find more data resources. The client receives a resource report listing all Xunlei multi-source technology-compatible resources for parallel download. The proposed peer management system is decentralized. A client detects peers using Kademlia protocol and scores them when downloading. The system maintains peer information without a central tracker and is nearly decentralized.

When downloading huge files, there are QoS requirements such as bandwidth and data usage. Sometimes during the peak hours the network prioritizes traffic to a particular sub-network or system. So, the researchers have introduced the concept of the distributed file server to optimize the download process by providing parallelism. Files are kept in a distributed manner on different servers, and these servers serve the request in parallel [7]. Parallelism has proved to be a great relief for the clients to download the file quickly and reliably. Therefore, we try to mimic the above arrangement but in reverse order. The file will be at one remote location and the download happens in parallel among multiple clients.

3 Architecture

3.1 Components

The project mainly consists of two components as shown in the Figure 1

Client Nodes

The client nodes run client application and participate in downloading the chunks of the file. Client nodes consists of admin and helper nodes. The admin node, also known as *Request Initiator* is responsible with communication with different nodes of the system/network. It is also responsible of tracking the progress of the download. Where as, the helper nodes are responsible for the download of the chunks from the server.

Remote Server

The remote server runs server application. The remote server is responsible of creating chunks and initiating peer-to-peer connection between the client nodes. The remote server communicates and notifies about the download only with the admin node. The server also keeps track of the active nodes and also maintains a local cache of all the files that were downloaded.

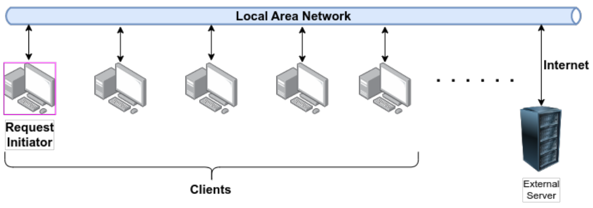


Fig. 1 Architecture

3.2 Technology Stack

The backbone of the project i.e. remote server and client node applications are built on **Python** because of the ease of implementation. We use **Flask** web application framework to create the RESTful applications for the remote server and client nodes. Admin Portal to initiate the download and view the progress of download is built using **Jinja** templates on top of **HTML** and **CSS**. Jinja is used because it is fast, expressive templating engine which is supported by Flask. The remote server also uses **SQLAlchemy Lite** to store the active nodes information, maintain local cache and perform local optimizations.

4 Methodology

In this section, more information is provided about different phases of the file download. The different phases are given below.

4.1 Register Phase

This is the first stage of the download. During this stage, all client nodes begin to run their applications and register their IP addresses with the server as shown in the Figure 2. The remote server's register API is used by the client nodes to register with the remote server. The nodes are not required to provide any data because the remote server can determine the IP address based on the HTTP headers. The server just responds with an OK response to acknowledge that it has detected an active node. The importance of this phase in Pre-download phase 4.3.

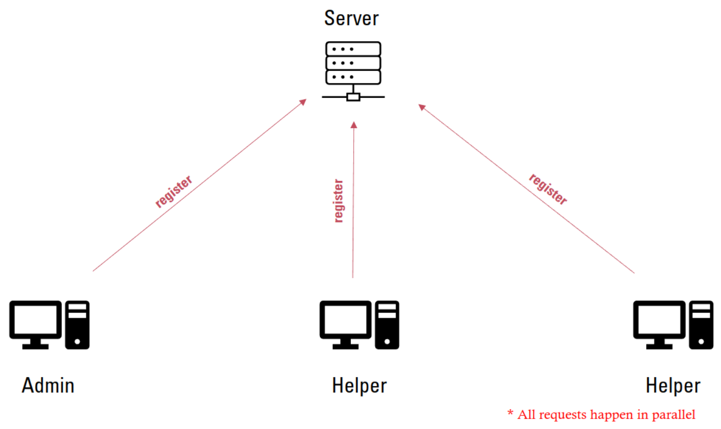


Fig. 2 Data flow between components in Register phase

4.2 URL download phase

Once the user decides to download a file, the user can provide the URL in the Admin Portal. The Admin sends this URL which is provided by the user to the remote server. On receiving a URL from the admin, the server spawns a thread and immediately responds to the admin with an OK. This thread checks the database if this URL has been used in the past. If there is a URL entry in the database, it checks its local storage for the file. If there is no URL entry in the database, it downloads the file into the local storage using the given URL.

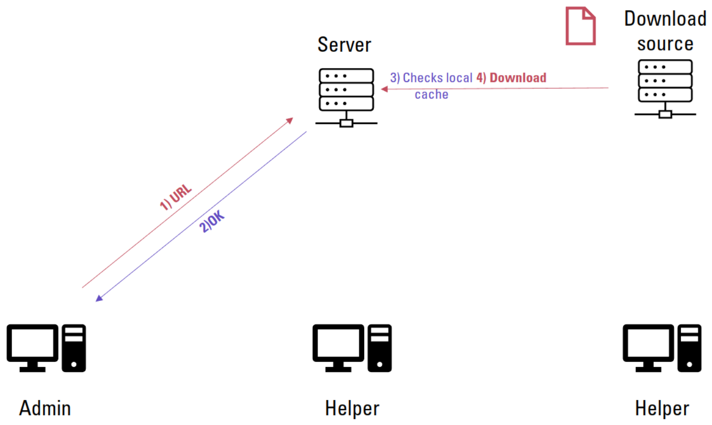
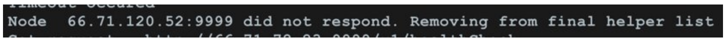


Fig. 3 Data flow between components in URL download phase

4.3 Pre-download Phase

Once the file is available in the local storage, the remote server sends the Ready notification to the admin, this notification contains the request ID of the request and a list of clients registered with the remote-server. The remote-server creates this list based on first 16 bits of the clients IP address and the admins IP address, if the clients are on the same local network they must have same first 16 bits on the IP address. Once the admin receives this list it will check if these clients are still active and if they can participate as helpers in this request, all of these checks will be done in parallel using multi-threading to improve performance. If a client is not available at this time it is removed from the helpers list (refer Figure 4). Once all available helpers are identified their details are sent to the remote-server.



```
Node  66.71.120.52:9999 did not respond. Removing from final helper list
```

Fig. 4 Node removed from the final helpers list

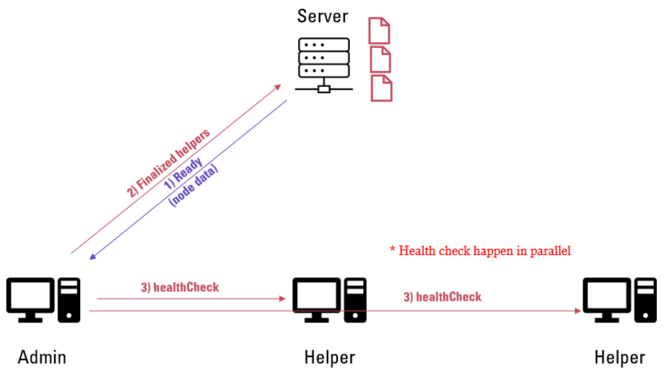


Fig. 5 Data flow between components in Pre-download phase

4.4 Download Phase

After the remote-server receives the list of finalized helpers the remote-server it first calculates the MD5 checksum of the original file then partitions the file in to chunks of equal size and assign one partition to each helper and one to the admin. This mapping along with the checksum is sent back to the admin, the admin relays the information of the assigned chunk to each helper in parallel and meanwhile also begins downloading its own chunk. Once a helper receives its assigned chunk information it begins downloading its chunk from the remote-server.

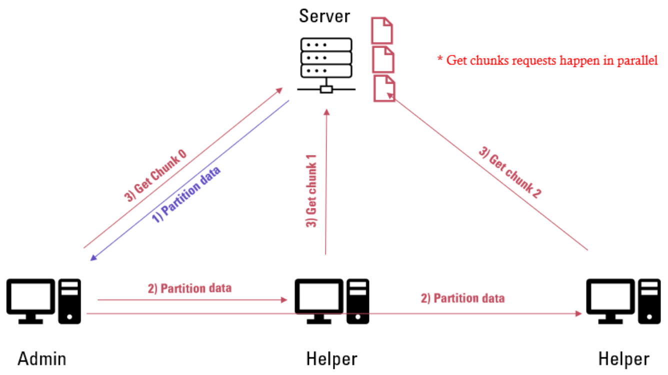
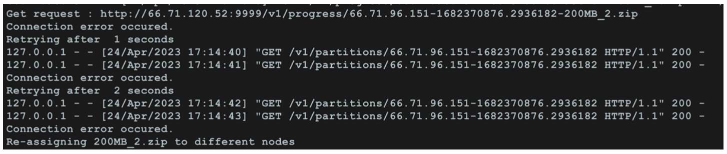


Fig. 6 Data flow between components in Download phase

While the admin and helpers are downloading its respective chunks the admin polls all helpers every second to get the download progress at that helper, if for some reason a helper crashed or becomes unresponsive and there is no progress in last 3 polls the admin will reassign the chunk to another helper with highest progress. Thus this provides some degree of Fault tolerance to the system.



```
Get request : http://66.71.120.52:9999/v1/progress/66.71.96.151-1682370876.2936182-200MB_2.zip
Connection error occured.
Retrying after  1 seconds
127.0.0.1 - - [24/Apr/2023 17:14:40] "GET /v1/partitions/66.71.96.151-1682370876.2936182 HTTP/1.1" 200 -
127.0.0.1 - - [24/Apr/2023 17:14:41] "GET /v1/partitions/66.71.96.151-1682370876.2936182 HTTP/1.1" 200 -
Connection error occured.
Retrying after  2 seconds
127.0.0.1 - - [24/Apr/2023 17:14:42] "GET /v1/partitions/66.71.96.151-1682370876.2936182 HTTP/1.1" 200 -
127.0.0.1 - - [24/Apr/2023 17:14:43] "GET /v1/partitions/66.71.96.151-1682370876.2936182 HTTP/1.1" 200 -
Connection error occured.
Re-assigning 200MB_2.zip to different nodes
```

Fig. 7 Chunk reassigned to recover from crashed/unresponsive helper

4.5 Gather Phase

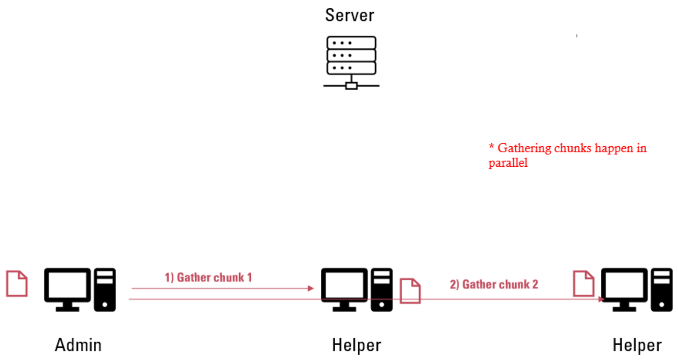


Fig. 8 Data flow between components in Gather phase

Once all chunks have been downloaded at the respective locations, the admin initiates the gather phase. In this phase all chunks are transferred to the admin node on the local network. Once the admin has all the chunks it will stitch back the chunks to generate the original file. To ensure the file integrity the admin calculates its md5 checksum and it is compared to the one calculated by the remote-server. If it matches then the download is complete.

5 Issues Addressed

5.1 Design Issues

One of the major issues we faced was peer discovery, how would the admin find all active and available clients on the local network in an efficient way?. The main problem was that there are a very large number of systems in most organizations, then identifying active available helpers for your download is difficult. Most network discovery tools and packages require admin privileges but such organizations do not give admin access to all users.

The solution we came up with was to register each client at the remote-server once it is active and ready to participate 4.1, at which point the remote-server will save its information in the local database. Then once the admin is ready to look for helpers it will get a list of recently registered client ips with first 16 bits matching. Thus the admin only has to check if those clients are still active. This provides an efficient solution to our problem.

5.2 Implementation Issues

In implementation, the admin pings the helpers at regular intervals to track the download progress of the chunks in the helpers. When admin raises multiple downloads, an helper can download different chunks of different files in parallel.

First issue is to come up with a way to track progress among different request IDs that a helper is downloading chunks for. Similarly, when an admin is tracking multiple requests, then we had to come up with a way to track the progress simultaneously without corrupting the progress data of other requests.

For this, we create a mutli-thread data manager which takes care of sharing the data references among the threads and updating the visibility of different shared variable. When multiple downloads must be tracked, we create a dictionary with key as request ID of the request and then allow all the threads in the application to update the respective values.

6 Results

All the evaluation of the system was done using personal laptops, we setup multiple laptops: 1 for remote-server and 2 to 4 clients. 1 client always acts as the admin and others as helpers. We logged the time taken to recreate the original file at the admin once the file is ready at the server and used this time as our evaluation metric. Initially we wanted to see how the performance of the system would perform for varying number of helpers for same size file. We decided to download a 512 MB file initially without any helpers and then by adding 1 helper next time until we had 4 helpers.

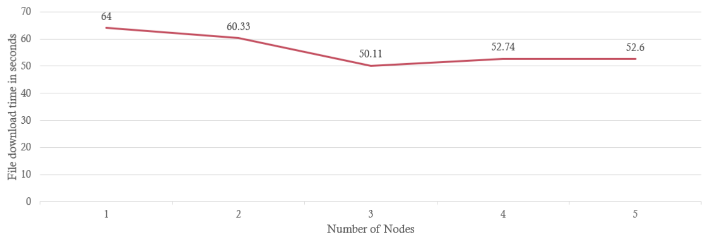


Fig. 9 Performance of the system by varying network size

From the graph we can see that we can see some performance gain as we increase the number of helpers. But the performance plateaus even after adding 3 or 4 helpers. One of the reasons for this can be hardware limitations, since our remote-server is also running on a laptop with just 6 cores the capacity of parallel processing of requests by remote-server is limited. If we host the remote-server on a more powerful system which would be the case of a production level implementation, this can be easily overcome.

Then since we had best performance for 3 clients (1 admin and 2 helpers) we tested the system with 3 clients on varying file size. The baseline for comparison is time to download the file without helpers.

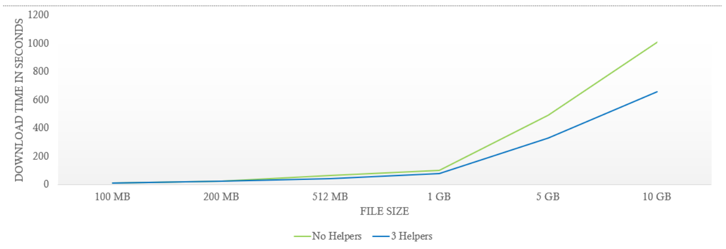


Fig. 10 Performance of system for varying file size

From the graph we can see that for smaller file size there is no significant performance gain, but as the file size is increasing we have significant performance gain.

7 Conclusion and Future Works

We have implemented a distributed file download network which leverages distributed computing paradigms like multi-threading, message parsing, Remote Procedure Calls (RPC), Load Balancing etc. It is highly scalable and reliable solution to download huge files and reduce the data and bandwidth usage by individual systems. This implementation gives satisfactory results in our tests with different sizes of files and different number of nodes. Thus, it can be extended to real world file sharing scenarios.

Further, the performance can be further improved by replicating the remote server to handle the load efficiently and to improve fault tolerance. The robustness of the system can be improved by enabling the helpers to provide the consent when admin does a health check to include the helper in the final list. The download tracking can be improved by providing a better UI for the Admin portal.

References

[1] Wikipedia contributors: Peer-to-peer — Wikipedia, The Free Encyclopedia. [Online; accessed 3-May-2023] (2023). https://en.wikipedia.org/w/index.php?title=Peer-to-peer&oldid=1151890426

[2] Rayabharam, S., Suryawanshi, S.: Distributed File Download Network. https://github.com/maitreya2954/distributed-file-download-network

[3] Shetty, A., Mhatre, S., Sinvhal, N., Devadkar, K.K.: Peer assisted parallel downloading system. In: 2019 International Conference on Communication and Signal Processing (ICCSP), pp. 0650–0655 (2019). https://doi.org/10.1109/ICCSP.2019.8697914

[4] Yang, M., Yang, Y.: Applying network coding to peer-to-peer file sharing. IEEE Transactions on Computers **63**(8), 1938–1950 (2014). https://doi.org/10.1109/TC.2013.88

[5] Wang, Y., Vassileva, J.: Trust and reputation model in peer-to-peer networks. In: Proceedings Third International Conference on Peer-to-Peer Computing (P2P2003), pp. 150–157 (2003). https://doi.org/10.1109/PTP.2003.1231515

[6] Zhang, M., John, W., Chen, C.: Architecture and download behavior of xunlei:a measurement-based study. In: 2010 2nd International Conference on Education Technology and Computer, vol. 1, pp. 1–4941498 (2010). https://doi.org/10.1109/ICETC.2010.5529202

[7] Gianfelici, F.: Measurement of quality of service (qos) for peer-to-peer networks. In: IEEE Symposium on Virtual Environments, Human-Computer Interfaces and Measurement Systems, 2005., p. 6 (2005). https://doi.org/10.1109/VECIMS.2005.1567580