P2P Caching Service

Mackenzie Zahn, Abdirisak Ibrahim, Justin Mendes, Ali Ahmadi

Summary

This paper will discuss our approach to P2P caching proxy servers referred to as PCS and potential improvements as well as further discuss some related systems. We introduce key concepts and related works first with a high level interpretation of the overall systems. Differences and similarities between the systems and our proposed solution are observed and noted. The benefits of the PCS are noted and discussed throughout the report. Specific design related attributes such as assumptions made in development, the overall system architecture and flow of requests. We describe implementation attributes such as components and requirements, some test cases and the scalability of our system. Lastly we discuss the evaluation of our solution, the usability and the performance of the system. This report outlines the creation of a P2P caching service implementation and how it was accomplished.

I. Introduction

Network congestion is an issue that occurs in many networks today. Many Internet Service Providers utilize the concept of a caching service in order to reduce bandwidth consumption, which makes it less costly to provide consumers with internet access. Caching is a concept that is used in many locations, some of which are at a web client and browser, at an Internet Service Provider, in a caching proxy server located in of an application server, and in the back-end database caching buffer pools (IBM, n.d.). Client side caching is often done through internet web browsers, by saving content on frequently accessed websites. Caching locations are ideally desired to be as close to the client side it can be without security being compromised. This is because the closer the content location is then the shorter distance of requests through the network creating less bandwidth usage overall.

Another concept used widely today are peer-to-peer connections. This widely used concept which is well described by Cope (2002) “in its simplest form, a peer-to-peer (P2P) network is created when two or more PCs are connected and share resources without going through a separate server computer”. By using Peer-to-peer connections the bandwidth usage through a network is greatly reduced by moving all the used bandwidth to be directly between the clients attempting to access each other.

Knowing about these two concepts of Peer-to-peer networks and Memory caching on a network and the benefits that exist, one would think that utilizing the two at the same could be even more beneficial.

The main idea that stemmed this desire to use both of these bandwidth reduction concepts came from real world situations that have been experienced. These experiences were from university classroom environments where a group of students are required to download a file or software tool instructed by the professor. As a result of everyone trying to download the same files at the same time greatly slows down the process, by dramatically increasing the network congestion causing a bottleneck effect for the students download process. Our project aims to solve this issue by accessing a resource from an outside network as few times as practical, and distributed the resource within a local network.

II. Background and Related Work

Other current works that exist which are related to PCS of this report are MiddleMan and Shark.

Middleman is a video caching proxy server technology researched by Soam Acharya and Brian Smith from Cornell University. They describe MiddleMan as “a collection of proxy servers that, as an aggregate, cache video files within a well-connected network” (Acharya & Smith, 2000, p.1). Their goal of their proposed solution is to have multiple clients store small pieces of a video file from the same file, with the intention of having the clients cooperatively stream the data from peer to peer as required. Their implementation allows for a very large cache size as they do not use an individual cache server and instead use all the clients in the network as the cache. MiddleMan concept of memory storing and the flow of network usage heavily relates to our proposed solution with the major differences being that they only store parts of files on clients and our solution stores the whole file. Another difference is that there solution has the clients cooperatively connecting to more than one client node as it iterates through the parts of the video files as needed. In our implementation we have a our clients simultaneously connecting to one closely located client as if it were a server that has all of the desired content for the other users to download. Whenever the bandwidth between the hosting client and the downloading clients reaches a specified constraint a new client would obtain the file from the origin and become another local host of the content.

Shark is a strategy for scaling file servers with the use of a cooperative caching system. Shark is a distributed file system specifically for large scale networks where clients can use each others memory as caches for themself to overall reduce the load on the original file server. Once again Shark like MiddleMan maintains partitions of desired content cached across many different clients. Shark although does not have individual files split up like MiddleMan. Shark is similar to our solution in that it connects clients to each other and accesses and sends whole files. Shark maintains a server that tracks the path names to directories across different machines and “every Shark file system is accessible under a pathname of the form: /shark/@server, pubkey a Shark server exports local file systems to remote clients by acting as an NFS loopback client”(Annapureddy, Freedman & Mazieres, 2005). So similar to our idea, Shark distinguishes between the locations of the content desired by remembering specific paths to different clients on a sort of cache proxy server.

III. Proposed Solution

Solution Name: PCS (Peer-to-peer Caching Service)

A. Assumptions

Some assumptions that were made for the implementation of this concept were:

1. Clients downloading from a host are all part of the same Local Network.
2. UOIT does not already have its own caching server.
3. A strong network connection between clients can be maintained.

The reasoning for assumption 1 is due to the core idea behind our proposed solution is to alleviate bandwidth spread across a network to just be only directly between clients within a close vicinity to each other (potentially same wifi, same router). Assumption 2 must be true in order for PCS to be beneficial for users that would potentially use our idea. As Well as we developed this solution using and testing on UOIT internet. Assumption 3 is made because if the two clients communicating with each other cannot establish a constant and reliable connection to the internet then PCS cannot be made possible.

B. Architecture

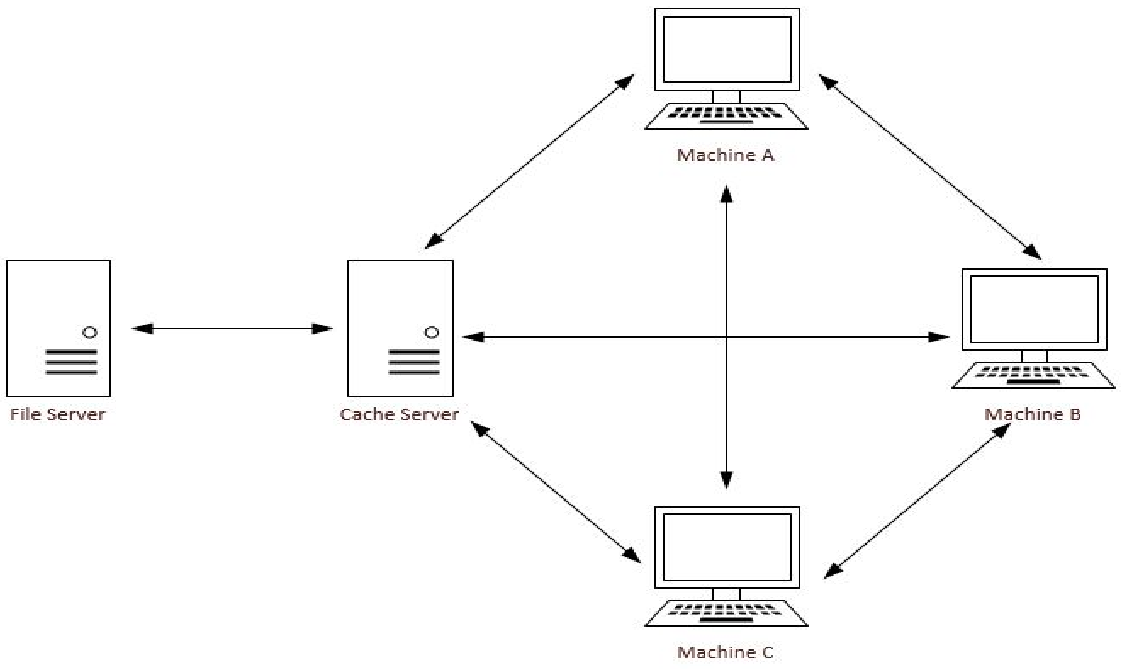

Figure 1.

In Figure 1, the layout of our systems architecture can be seen. The proposed layout consists of an external file server (The Internet) and a cache server which also acts like a proxy to the view of the clients. All the machines (clients) have arrows indicated the interconnectivity between them. The normal flow of events can be seen in Figure 2 below.

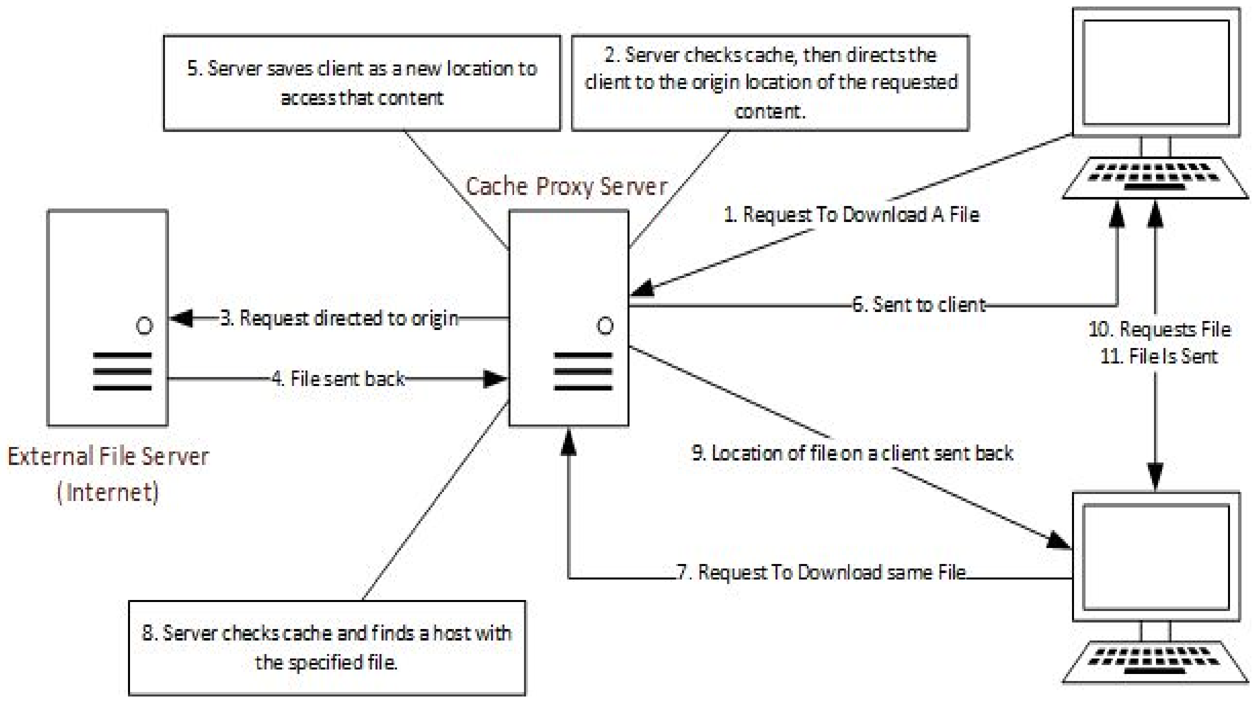
5. Server saves client as a new location to access that content
2. Server checks cache, then directs the client to the origin location of the requested content.
Cache Proxy Server
1. Request To Download A File
3. Request directed to origin
4. File sent back
6. Sent to client
10. Requests File
11. File Is Sent
External File Server (Internet)
9. Location of file on a client sent back
7. Request To Download same File
8. Server checks cache and finds a host with the specified file.

Figure 2.

The regular flow as represented above runs from the initial client requesting to download a file to the cache proxy server. A check is done to see if this file has already been downloaded by any other clients on the network. The client is then directed to the internet to download the file from its origin. The cache server then saves the client's address along with the request. Once the client receives and downloads the file they have become an eligible host. When a second client requests to download the same file the proxy server now checks its cache and informs the second client of the host client. The second client now establishes a connection to the host and downloads the requested file.

C. Implementation

This implementation is intended to serve as a proof of concept prototype rather than a fully fledged, robust application. For the purpose of this prototype, we have assumed that the file transfer procedure is as follows.A request is a string containing only the file name requested, a reply is sent in three separate parts: file name as a String, file size as an int then the file contents as a byte array.

<u>Components:</u>

- **ExternalServer**
  - ExternalServer acts as a basic file server that handles a request formatted as a filename, and replies first with the file name, then file size, then file bytes
  - It will represent an external file source during a use case
- **Server**
  - This is the actual cache server. It will handle all client requests and host the cache.
  - It will run on one machine during a use case
  - It will either forward a request to the ExternalServer, or reply with an address depending on the contents of the cache.
- **Client**
  - This will be the code that runs on a machine that wants to receive a file
  - It may run on any number of machines during a use case
  - It will request a filename from the cache server
  - If it received the file from the cache server, it will begin hosting the file and serving new clients that request the file directly

Requirements:

- ExternalServer must be able to accept a client request, and reply by sending the file requested. The file must be received by the client and become available in said client's file directory.
  - This requirement is necessary to ensure that the the initial process of setting up a client host with the desired file is possible.
  - By meeting this requirement we create a simulated environment where a client can access some arbitrary external server to download a file.
- ExternalServer must continue to run after serving a client, and continue to serve subsequent clients in the same manner.
  - This is needed to handle situations where the current host client is no longer available.
  - By having the simulated external file server running after serving client requests other clients can be directed to download requested files from their origin.
- Client must be able to receive a file and save it in its file directory.
  - This is a major requirement as our whole implemented system revolves around download performance enhancements.
- The cache server must handle Client requests, forward them to ExternalServer, and forward reply back to Client.
  - This requirement is designed to ensure that a client that is the first to request a specific file can do so.

- Cache server must store any requests it forwards to ExternalServer along with the requester's IP in a cache.
    - In order for the other clients to be able to establish peer-to-peer connections with each other, the cache server must be able to remember who has the file and be able to inform the requesting clients accordingly.
- After receiving a file from ExternalServer via cache server, the Client must host the file and serve Clients that request the file.
    - This requirement is necessary as it is an essential feature of our proposed solution.
- If a hosting Client shutdowns for any reason, it must notify the cache Server to remove its information from the cache.
    - Enforcing the client to notify the cache server is vital to error handling faults.
    - Whether the client crashed or has decided to no longer host a file, the cache server is made aware of the situation. All further requests are directed to the external server until a new client host becomes available.
- A hosting Client will handle a limited number of requests then shutdown.
    - This requirement is put in place to control the congestion between peer-to-peer connections. The client host can at max have assist 5 other clients with downloading a file until a new client becomes a host.
- A cached request/IP pair must expire after a predetermined length of time and remove itself from the cache.

- This is to handle situations where a client has downloaded a file and never decides to stop hosting. This is an issue because over time it is more likely that the file version on the host client and the external file server could be mismatched.
- Enforcing this requirement helps with fault tolerance.

D. Test Cases

The following is an example test case. The complete set of test cases is available in a document in the source code folder as was requested. The succession of test cases will show a chronological progression of how each requirement was addressed. These test cases were created throughout development, and repeated until successful.

Test Case: 6

Summary: Verify Client is able to host file after receiving it from ExternalServer through the cache Server and serve new Clients that are sent by the cache server

Requirement: After receiving file from ExternalServer via cache server, the Client must host the file and serve Clients that request the file.

Prerequisites:

1. ExternalServer is running on same machine on port 6969
2. ExternalServer's directory contains test.docx
3. Test.docx is absent from all Client directories

Procedure:

1. Execute Server using localhost:6969 as the address for which it will act as a proxy

2. Execute first Client using localhost:6968(cache server runs on port 6968) with test.docx as requested file
3. Execute second Client using localhost:6968 with test.docx as requested file.

Expected Result: ExternalServer only receives one request. First Client receives request directly from second Client. Test.docx is saved in second Client's directory

Actual Result: ExternalServer only received one request. First Client received a request directly from second Client. Test.docx was saved in second Client's directory

Status: success

Remarks: this test was very valuable because it verified the very basic functionality of the solution.

E. Scalability

The scalability of PCS is theoretically only bound by the physical limitations of the network. Naturally, the more clients on a network, the more congested the network becomes. Many institutions already use caching servers for hundreds or even thousands of devices within a building or campus. With a proper and robust implementation, PCS could be scaled up to similar use cases so long as the network can handle it.

F. Challenges and Solutions

The main challenges that were faced in the development of PCS were handling errors and creating a fault tolerant system. One such challenge was in determining the correct file version. Because PCS requires a user to enter the filename of the resource required, it is difficult to ensure that the files distributed across all clients are the same. This was solved by adding an expiry date to files located in the cache server. The expiry time was initialized when a user is adding a file to the cache server. Once the file has expired, it is removed from the cache server thus ensuring that the file is up to date.

Another challenge we faced was in determining if a client has disconnected from PCS. Initially, when a client was hosting a file and it disconnected from the system, an error occurred if another client attempted to retrieve the file from it. To solve this issue, we implemented a shutdown-hook to detect closure of the client and trigger a function which would remove the client's request from the cache server. This ensures that another client cannot attempt access to it.

IV. Evaluation and Results

A. Usability

The system was implemented using sockets with the Java coding language. The concept of PCS is well displayed but the actual development of this service into a standalone product is still in the preliminary stages. As a result, users require the knowledge to identify the cache server ip and must enter it as an argument when using our solution. The intended use is for many people attempting to download the same content. Efficiency of the overall system was taken into account and as a result the code was written to make the service robust and capable of many scenarios. In order to use the software the external server used for conceptual research must first be running to host the simulated file origin, then the cache server must also be running. The cache server must be aware of the ip address of the external server. The user's must first run the client side code before requesting to download a file. If the client is the first one to download the file it obtains the file from the external server and then it becomes a host for the future requests, replacing the use of the external server.

B. Performance Analysis

It is difficult to properly quantify the performance of our solution using time per request without scaling up our use case to many more clients than we are realistically able to facilitate. Instead we will compare the number of individual requests forwarded to an external server in a normal use case vs a use case where our solution has been implemented.

In a normal use case with N clients, there will be N requests to the external server. As the number of clients scales up, so too does the number of requests and the bandwidth usage and thus the network congestion between the local network and external server. Our solution allows us to cut the number of requests significantly by any factor through distributing the requests within the local network rather than sending them all to the same single server. Using our solution with N clients, instead of having N requests forwarded to a single server, we only have $N/(X + 1)$ where X is the number of clients we allow a hosting client to serve.

However it isn't as simple as only considering the number of requests forwarded to an external server. If we for example set $X = N$, then the original Client that downloads the file first, will host the file for the rest of the N-1 clients which defeats the purpose of trying to mitigate congestion. The way the solution is configured is that each hosting Client may serve 5 additional Clients meaning that for N Clients, we have $N/(5+1)$ requests to the external server. For example, if we have 60 clients, there are only 10 requests forwarded to the external server. This means we have reduced the external requests to 20% of what they would otherwise be.

V. Conclusion and Future Work

To conclude the report it can be seen the our proposed solution PCS properly displays how peer-to-peer systems implemented with caching algorithms can greatly benefit users with faster download speeds due to a reduced bandwidth usage. Although our system successfully implements all operations outlined above, it can be seen that as a conceptual prototype of an intricate bandwidth optimization solution, that to make this a finished product more work needs to be implemented. A potential improvement for our system is to create it for accessing files hosted on the real internet and not a simulated one. In the future if this concept were to be taken beyond where it is now then improvements to security could be implemented. Right now the ip of host clients is set directly to requesting clients and this could be potentially avoided using encryption algorithms. Although a kind of ip security is implemented naturally as the intended use case would likely be a student using a laptop that connects to different networks where their ip would be different often. Another improvement in the future for the system would be to create a way for handling the single point of failure that exists when the cache server crashes. Much effort and works have been spent on the development of this conceptual implementation of a peer-to-peer caching server.

References

1. Acharya, S., & Smith, B. (2000). Middleman: A video caching proxy server. In Proceedings of NOSSDAV.
2. Annapureddy, S., Freedman, M. J., & Mazieres, D. (2005). Shark: Scaling file servers via cooperative caching. In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2 (pp. 129-142). USENIX Association.
3. Cope, J. (2002). What's a Peer-to-Peer (P2P) Network? Retrieved from https://www.computerworld.com/article/2588287/networking/networking-peer-to-peer-network.html
4. IBM. (n.d.). Where caching is performed. Retrieved from https://www.ibm.com/support/knowledgecenter/en/linuxonibm/liaag/cache/pubwascacheperformed.htm