Implementation of Cooperative Caching Algorithms Using Remote Client Memory

Darshan Kapadia[1], Nakul Nagpal[2]

[1]Department of Computer Science,
B. Thomas Golisano College of Computing and Information Sciences
,Rochester Institute of Technology
,Rochester, New York
Address
[1]darshanrkapadia@gmail.com
[2]nkn5603@rit.edu

***Abstract*— As technology is advancing, processor's speed is increasing much faster than disk access speed. So it becomes necessary to decrease the number of disk accesses by the distributed file system to gain overall performance improvement of the system. As network speed has increased tremendously in recent time, accessing data from remote client machine memory is faster than accessing data from the disk. Thus file caches from number of client machines connected over a high speed network can be combined together to form a global cache, this is called cooperative caching. In cooperative caching when data is requested and if the data is not present on client's local cache, the request is satisfied by another client machine cache if possible. In a typical client server based distributed file system there are 3 levels in memory hierarchy: server disk and server memory, client memory. Cooperative cache can be seen as a fourth level of cache in the distributed file system. The project will implement the following five different read only cooperative caching algorithms using remote client memory: Direct Client Cooperative caching, Greedy Forwarding Algorithm, Centralized Coordinated Cache Algorithm, N-Chance Forwarding Algorithm and N-Chance Forwarding with Centrally Coordinated Cache Algorithm. The project will then compare the performance of the algorithms based on different evaluation metrics such as average read performance, number of disk access, global and local hit ratio. These algorithms though simple in implementation and without major architecture changes can provide really good read performance by decreasing the number of disk accesses and hence reduce total access time.

***Keywords*—Cooperative Caching,Remote,client,Memory, processor's speed,disk access.**

I. Introduction

Cooperative Caching

In Distributed File Systems the amount of data stored in the system is very large. Distributed File Systems should be able to fulfill the data request made by the application as quickly as possible. The main two problems that affect the performance are disk access and network latency to transfer the data from server. Accessing data from remote storage disk is an expensive operation (in terms of time and network usage) and should be avoided as much as possible. There are many algorithms such as First in First Out, Random Replacement, Least Recently Used etc. for cache replacement on individual clients. As these algorithms manage the local client cache greedily they do not take advantage of the data present on other clients' caches [3]. Cooperative caching provides a solution to this problem. In cooperative caching all the clients' cache memory can be managed and used as a global resource. If the data requested is not present in local client cache it can be looked up and transferred from other client's cache. This technique requires data transfer from remote client cache memory but avoids disk access. This approach is faster than disk access in terms of time taken to look up and transfer data. Local client caches can still be maintained using Least Recently Used algorithm. There are many cooperative caching algorithms by which the global cache resource can be maintained. Cooperative caching adds an additional level to the memory hierarchy to look up data. Cooperative Caching Algorithms can provide high local and global hit rates as they don't affect the local client cache but still can use other clients' caches as a global resource.

II. Problem Statement

A. Existing System

In existing Distributed File Systems, there are three levels of memory hierarchy: client cache, server memory and disk. Attempts to reduce the amount of disk access to get overall performance improvement are made by implementing various replacement algorithms on the client cache memory such as First In First Out, Random replacement, Frequency Based Replacement, Least Recently used etc. These algorithms do provide high local hit rates and reduce disk access by some amount. Still clients are unaware of other clients' caches, so even if data is present in other clients' caches they cannot request it, resulting in disk accesses. Also due to lack of coordination between clients there may be many duplicate blocks in the system. The existing work consists of implementation of four Cooperative Caching Algorithms using Remote Client memory on xFS file System [1]. xFS is a server less File System having number of Clients and a Manager. The work shows the comparison of the four algorithms in the server less Distributed File System environment. In contrast, this project implements the four Cooperative Caching Algorithms using Remote Client memory to improve performance in a Master-Slave Architecture Distributed File System and then proposes a new Cooperative Caching algorithm.

B. Proposed Solution

With cooperative caching, the overall global cache hit ratio increases since there is an addition of remote client memory to system memory hierarchy. Also the overall performance of the system will increase since there are fewer disk access operations. Cooperative Caching is also cost effective as it is an alternative to having a large server memory [1].

The project will implement the following five cooperative caching algorithms on Master-Slave Architecture Distributed File System and compare them based on different evaluation criteria:

1. Direct Client Cooperative Caching Algorithm [1].
2. Greedy Forwarding Algorithm [1].
3. Centralized Coordinated Caching Algorithm [1].
4. N-Chance Forwarding Algorithm [1].
5. N-Chance Forwarding with Centrally Coordinated Cache Algorithm.

III. System Architecture

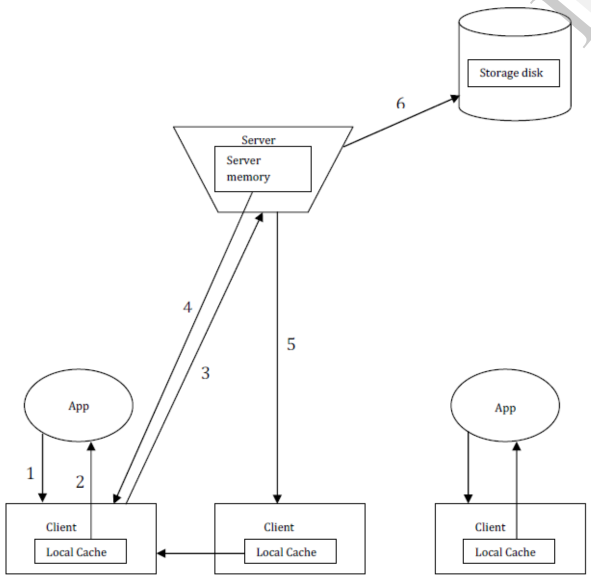

Fig 1: Master-Slave Architecture of Distributed File *System*.

The project will implement a modification of the xFS Architecture [6][7]. It will contain a number of clients and a server connected over a high speed Local Area Network (LAN). Server will have access to the storage disk which contains all the files present in the system.

The general steps taken by the system when data is requested by the application are as follows:

1. Application requests a block of data from the client.
2. The client looks for the desired block in its local cache memory. If found it returns the data back to the application.
3. If not found, the client forwards the request to the server.
4. The server looks for the desired block in the server memory. If found it returns the data back to the client.
5. If not found, the server looks for the data in the global cache. If it finds the data in the global cache it forwards the request to the client containing the data. The data transfer takes place between the clients.
6. If not found the server performs a storage disk access and returns the data back to the client. This is an expensive operation.
7. Once the client gets the block it puts it in its LRU based local cache.

How the data is managed and looked up in the global cache, what portion of memory is dedicated to local and global cache and how forwarding of data requests takes place will be different for all the algorithms. All of five algorithms to be implemented will try to avoid disk access since it degrades the performance of the entire file system.

IV .Cooperative Caching Algorithms

4.1 Direct Client Cooperation Algorithm

Direct Client Cooperation is the simplest algorithm for cooperative caching as it can be implemented with minor modifications to the existing distributed file system architecture. Direct client cooperation algorithm allows a client to use other clients' caches as a backup storage. In this case when an active client's cache is full, the client looks for an idle client and starts forwarding the block entries in idle client's cache. The active client uses idle client's cache in the same way as it uses its own cache. The problem of this algorithm is that clients are unaware of other clients' cache. Due to this, in case of a cache miss, the client will need to send the request to the server to get the desired block even if the copy of the same block may be present in another client's cache. Also this may result into disk access on server side if the server does not have the desired block in its memory which can be a really expensive operation. As we will see performance improvement of this algorithm is much less compared to the other cooperative caching algorithms. Also a design issue has to be considered in case the idle client becomes active. In that case the idle client may decide to flush the entries of the other client or may forward it to another idle client's cache. For simplicity of implementation and results interpretation the project will not consider other client's interference, instead the project will just double the amount of cache space.

4.2 Greedy Forwarding Algorithm

In Greedy Forwarding Algorithm all the clients' caches are used as an aggregated global shared resource. So a client's read request may be fulfilled by any client cache present in the system. Still this algorithm does not attempt to coordinate the contents of the client cache. So each client manages its cache greedily without having any knowledge of another client's cache. In this algorithm when a read request is made the client first looks into its own local cache and if the block is not present it forwards the request to the server. The server looks for the desired block in its cache; if present it supplies the data back to the client. If the desired block is not present in the server's cache, the server looks up the desired block in other clients' caches. If another client has the copy of the desired block, the server forwards the read request to the client containing the data. The data transfer then takes place directly between the client having the data and the requesting client. Since the data is not transferred back to the server and as server only needs to forward the request, the load on server is reduced. Once the client gets the block it puts it in the local cache. If the client's cache is full it evicts the Least Recently Used block from the cache. For this algorithm, a data structure has to be maintained such as a hash table which maps the clients and the contents in their caches. The server can look up this table to find the client and forward the request. Greedy Forwarding Algorithm should increase the global hit ratio since the request is now not only fulfilled by single client cache or server but can be fulfilled by all clients' caches present in the system.

4.3 Centralized Coordinated Cache Algorithm

Centralized Coordinated Cache algorithm adds coordination to the greedy forwarding algorithm. Every client's cache is divided into two parts. One part (local part) is managed greedily by the client whereas the other part of the cache is coordinated globally by the server. The server uses this cache as an extension to its central cache. Once a block is requested, the client first searches for the block in the local part of cache. If not found the client forwards the requests to the server. The server looks up the globally coordinated portion of the client's cache for desired block. If found the server forwards the request to the client having the data. If not found in the globally coordinated cache, the server performs a disk access operation. Once the client gets the block it puts it in the local cache. The server manages major portion of the client's cache and remaining portion of it is managed greedily by the client. The project will vary this ratio to find the optimum partition ratio. Server uses a global replacement algorithm to manage the global part of the client's cache. When a server removes a block from its local cache it replaces the least recently used block in globally shared cache with this block. The server renews the entry of the block in the LRU list. The main advantage of this algorithm is that it will achieve high global hit rate since there is a very large globally shared memory. The disadvantage of this algorithm is that because the local cache size is reduced, the local hit ratio is affected. Also since the server is required for forwarding majority of the requests the load on the server also increases.

4.4 N-Chance Forwarding Algorithm

N-Chance Forwarding algorithm is an extension to the greedy forwarding algorithm. N-Chance forwarding adjusts the portion of the client's cache that is managed cooperatively based on the activity of the client. N-Chance Forwarding Algorithm modifies the greedy forwarding algorithm so that it caches singlets. Singlets are blocks that are present exactly in one client's cache in the entire distributed file system. In the previous three algorithms if there is block which is present in exactly one client's cache and if that block is removed from the system, the next request to access this block will result into disk access which is very expensive. N-Chance forwarding algorithm addresses this problem. In N-Chance forwarding algorithm when the client is about to discard a block from its cache, it checks to see whether that copy is the last copy of the block in the entire system. If it is, it sets the recirculation count of that block to be 'n-1' and forwards it to any random peer in the network. The receiving client puts this block in its Least Recently Used cache as if the block just had a hit. When this client is about to evict this block from its cache it decrements the recirculation count and follows the same above procedure. When the recirculation count on the block reaches 0 it is simply discarded from the system. If the block is read anytime during the process the recirculation count is set back to 'n'. In this algorithm the active client will tend to discard blocks from it cache faster than idle clients. Idle clients on the other end will accumulate the singlets which is good as it provides a greater global hit ratio. Since this algorithm does not modify the amount of local cache it provides a good local hit ratio. Also since by protecting singlets the algorithm provides a good overall global hit ratio.

4.5 N-Chance Forwarding with Centrally Coordinated Cache Algorithm

The project proposes a new algorithm which is a modification to the Centrally Coordinated Cache and N-Chance Forwarding Algorithm. In Centrally Coordinated Cache Algorithm the client cache is divided into 2 parts: One part is managed greedily by the client and the other part is managed globally by the server. The global part of client's cache occupies major portion and local part is the rest. This decreases the local hit ratio significantly (as size of the local cache has decreased) but increases the global hit ratio. In N-Chance Forwarding Algorithm the singlets are re-circulated in the system before they are discarded thus increasing the chances of a global hit. Instead of 80/20 partitioning of the client's cache as in Centralized Coordinated Cache Algorithm, we partition the client's cache in 50/50 ratio and implement N-Chance Forwarding in locally managed portion of client's cache. Since we have only used 50% of client's cache for globally shared cache and used 50% for local cache, we do not expect that the global hit ratio will be affected significantly. Also because of N-Chance Forwarding in the remaining 50% the local managed cache, the local hit ratio should also be good. Thus this algorithm will try to use best of both the algorithms.

V. Simulation Methodology

5.1 Test Environment

The testing environment is simulated using Java RMI and Prof. Alan Kaminsky's Computer Science Course Library [8]. The simulation environment consists of a Master – Slave model of the distributed file system. The Server and Clients in the system are remote objects and communicate via remote method invocation. All the caches in the system, Server cache and Clients' caches use Least Recently Used algorithm for replacement of cache blocks. The Server has access to the disk which contains all the files. The simulation consist of Application, again a Remote Object which performs read operations on clients in the system and notes the average time to read the block, global hit and local hit ratio. The management of the cache blocks in the system is controlled differently based on the 5 algorithms. The various simulation parameters used to implement and test the system are as follows:

- Client Cache Size: This represents the size of cache available at each client (will be same of all clients).
- Server Cache Size: This represents the size of cache memory available at Server.
- Number of Read Requests: This is the total number of read requests made by the Application.
- Client Local Memory Access Time: This is the time taken by client to perform read from its local cache
- Remote Client Memory Access Time: This is the time taken by read request when a block is read from remote client. It is the sum of time to forward request and time to transfer data from one client to other.
- Server Cache Memory Access Time: This is the time taken to transfer block from Server cache memory to the client.
- Disk Access Time: This is the time it takes to perform read from disk and transfer it to the client.

5.2 Evaluation Criteria

To compare the performance of the five cooperative caching algorithms mentioned above the following evaluation metrics are used:

- Average Block Read Time: Time it takes on average to perform a block read operation by the algorithm.
- Number Of Disk Access: Total number of disk access required to perform the read request by the Application
- Global Cache Hit: Total number of blocks read from the global cache. It represents the number of blocks read from global cache.
- Local Cache Hit: Total number of blocks read by client from its local cache.
- Number of Clients: Total number of clients in the system.

VI.Test Cases

All the five algorithms are tested under the following test cases. While testing each case, the simulation parameters such as client's cache size, server cache size, number of blocks read and time required for performing read of blocks are kept constant across algorithms. Also the same random seed is used across algorithms so that same files are read in all simulations.

6.1 Number of blocks in disk is less than total available cache after n clients.

The simulation parameters for this test case are as follows:

Number Of Files	1500
Blocks Per File	5
Total Blocks On Disk	7500
Total File Reads	4000
Client Cache Size	750 blocks
Server Cache Size	2000 blocks
Local Memory Access Time	0.25 msec
Server Memory Access Time	1.05 msec
Remote Client Memory Access Time	1.25 msec
Disk Access time	15.85 msec

Table 1: Simulation parameters for test case 1.

With the simulation parameters in table 1, after about 8 clients in the system, the sum of the clients' cache and server cache is almost equal to the total blocks in the system. Because of this it is possible that all the blocks on the disk may be cached in client or server cache.

The performance of the 5 algorithms under the above test case is as follows:

1. Direct client cooperation algorithm

Number of Clients	Disk Access (Number of blocks)	Local Cache Hit (Number of blocks)	Average Access Time (msec)
4	13820	4140	11.1112
8	13280	4285	10.7058
12	13120	4090	10.5952
16	13002	4178	10.50436
20	13245	3865	10.6967

Table 2: Readings for Direct Client Cooperation Algorithm

2. Greedy forwarding algorithm

Number of Clients	Disk Access (Number of Blocks)	Local Cache Access (Number of Blocks)	Remote Client Cache Access (Number of Blocks)	Average Access Time (msec)
4	9995	2020	4030	8.4058
8	6694	1918	7288	5.99972
12	4470	2016	9014	4.3673
16	3350	1877	10097	3.55489
20	2328	1997	10831	2.80115

Table 3: Readings for Greedy Forwarding Algorithm

3. Centrally coordinated cache algorithm with 80 global/20 local cache partition.

Number of Clients	Disk Access (Number of Blocks)	Global Cache Access (Number of Blocks)	Local Cache Access (Number of Blocks)	Average Access Time (msec)
4	8134	6532	554	7.11232
8	1940	12380	568	2.58668
12	0	14288	465	1.17428
16	0	14210	386	1.17666
20	0	14197	425	1.17497

Table 4: Readings for Centrally Coordinated Cache Algorithm

4. N-Chance forwarding algorithm with recirculation count = 2

Number of Clients	Disk Access (Number of Blocks)	Local Cache Access (Number of Blocks)	Remote Client Cache Access (Number of Blocks)	Average Access Time (msec)
4	7995	2053	5921	6.94339
8	913	2059	12345	1.76671
12	217	2108	13148	1.25774
16	92	1911	13331	1.17495
20	32	1986	13205	1.12629

Table 5: Readings for N-Chance Forwarding Algorithm

5. N-Chance forwarding with centrally coordinated cache algorithm with 50 global/50 local cache partition.

Number of Clients	Disk Access (Number of Blocks)	Global Cache Access (Number of blocks)	Local Cache Access (Number of blocks)	Remote Client Cache Access (Number of Blocks)	Average Access Time (msec)
4	7912	3148	1062	3878	6.93266
8	2286	4959	1100	6871	2.81594
12	405	6392	1131	7518	1.44356
16	166	6866	1024	7227	1.27281
20	107	7000	1011	7283	1.23157

Table 6: Readings for N-Chance Forwarding With Centrally Coordinated Cache Algorithm

The table below shows the readings when the percentage of Global Cache and Local Cache is varied in Centrally Coordinated Cache when number of

Global Cache (%)	Local Cache (%)	Disk Access (Number of Blocks)	Global Access (Number of Blocks)	Local Cache Access (Number of Blocks)	Average access time (msec)
50	50	8586	5650	1180	7.41294
60	40	7394	6800	950	6.55156
70	30	6175	8210	730	5.6724
80	20	5044	9405	485	4.85721
90	10	3967	10530	300	4.07888

Table 7: Readings for centrally coordinated cache by varying local cache/global cache proportion.

The graph in figure 2 shows the number of blocks accessed from the disk when the number of clients is increased from 4 to 20. Total number of blocks read= 20000.

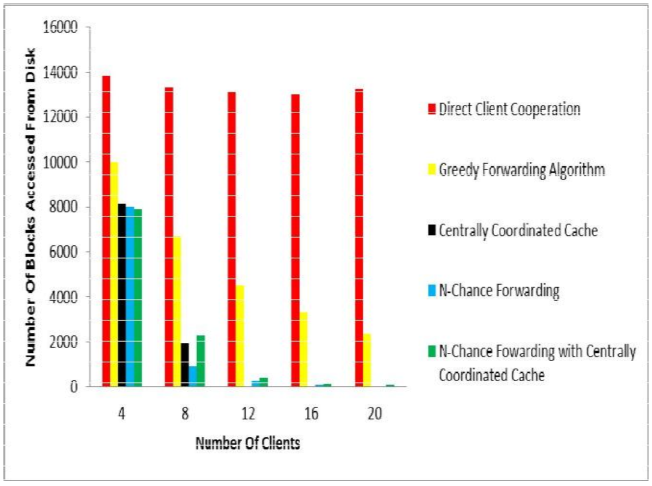

Figure 2: Graph of Number of Clients vs. Number of blocks accessed from disk for 5 algorithms

We can see from the above graph that as the number of clients is increased in the system the number of blocks read from the disk reduces for all the algorithms except the Direct Client Cooperation Algorithm. Also the number of disk accesses by Greedy Forwarding algorithm decreases with increase in clients but not as much the Centrally Coordinated Cache, N-Chance Forwarding and N-Chance Forwarding with Centrally Coordinated Cache algorithm. When number of clients is greater than 8 the disk access by the Centrally Coordinated Cache, N-Chance Forwarding and N-Chance Forwarding with Centrally Coordinated Cache algorithm is much less. This is because almost all the blocks in the system are cached in either clients' cache or server cache.

The graph in figure 3 shows the average access time to read a block for 5 algorithms when the number of clients is increased.

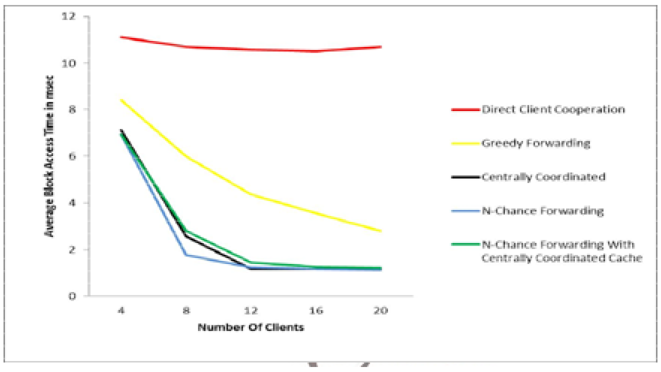

Fig3: Graph of Number of Clients vs. Average Block Access time for 5 algorithms

It can be seen that when the number of clients are increased it does not have any effect on the average access time by Direct Client Cooperation. Also the average access time to read a block in Direct Client Cooperation is much greater than the other 4 algorithms. The average access time of greedy forwarding algorithm decreases but not as much the other 3 algorithms. When number of clients are greater than 8 the average access time of Centrally Coordinated Cache, N-Chance Forwarding and N-Chance Forwarding with Centrally Coordinated Cache algorithm is almost same and remains constant. This is because all the blocks in the system are cached either in clients' cache or server cache.

The graph in figure 4 shows the distribution of how the 20000 blocks are read in the five algorithms. The distribution shows the number of blocks read from disk, server cache, global cache, client local cache and remote client's cache.

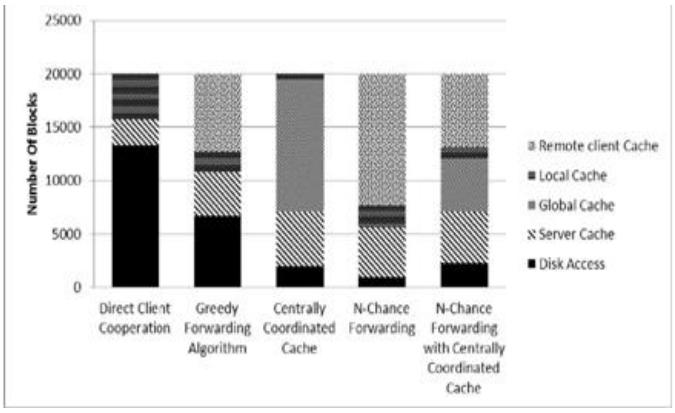

Fig 4: Graph of distribution of location of blocks for the 5 algorithms

The graph in figure 5 shows the average access time to read a block for all the five algorithms when number of clients=8.

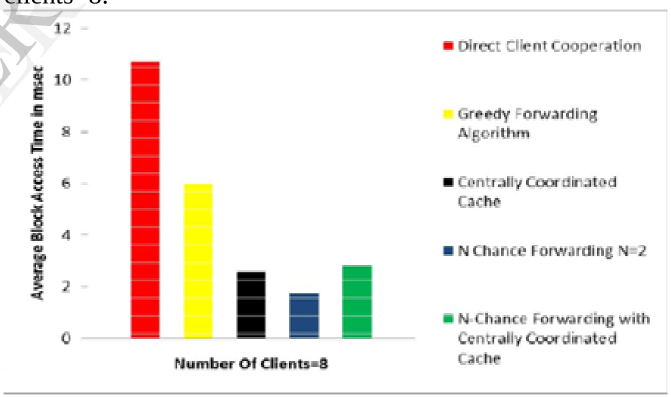

Fig 5: Graph of Average block access time for 5 algorithms

6.2 Number of Blocks in the disk is greater than available cache size

The simulation parameters for this test case are as follows:

Number Of Files	3000
Blocks Per File	5
Total Blocks	15000
Total File Reads	4000
Client Cache Size	750 blocks
Server Cache Size	2000 blocks
Local Memory Access Time	0.25 msec
Server Memory Access Time	1.05 msec
Remote Client Memory Access Time	1.25 msec
Disk Access time	15.85 msec

Table 8: Simulation parameters for test case 2.

In this test case we increase number of blocks in the disk. So now the number of blocks in disk is greater than all the available cache size even when the number of clients is equal to 20. So in this case there will be disk access in all the algorithms even for large number of clients.

The results for the above simulation parameters for five algorithms are as follows:

1. Direct Client Cooperation

Number of Clients	Disk Access (Number Of blocks)	Local Cache Access (Number of Blocks)	Average Access Time (msec)
4	16823	1991	13.41938
8	16753	1988	13.3677
12	16345	2354	13.05114
16	16563	2088	13.2231
20	16501	2110	13.17634

Table 9: Reading for Direct Client CooperationCache

2. Greedy Forwarding Algorithm

Number Of Clients	Disk Access (Number Of Blocks)	Local Cache Access (Number Of Blocks)	Remote Client(Number Of blocks)	Average access time (msec)
4	14880	1592	1739	12.01491
8	12037	1195	4788	9.95746
12	9953	1073	6786	8.44016
16	8678	1064	7858	7.50774
20	7337	1118	9299	6.52765

Table 10: Reading for greedy forwarding algorithm

3. Centrally Coordinated Cache Algorithm

Number Of Clients	Disk Access (Number Of Blocks)	Global Cache Access (Number Of Blocks)	Local Cache Access (Number Of Blocks)	Average access time (msec)
4	13994	3309	180	11.43145
8	10822	6368	270	9.11116
12	7408	9857	183	6.62317
16	4403	12840	205	4.42842
20	1184	16594	141	2.08646

Table 11: Reading for Centrally cordinated Cache Algorithm

4. N-Chance Forwarding Algorithm

Number Of Clients	**Disk Access (Number Of blocks)**	**Local Cache Access (Number Of Blocks)**	**Remote Client Access (Number Of Blocks)**	**Average access time(msec)**
4	12235	988	4678	10.11116
8	6273	964	10490	5.75836
12	1982	1071	14645	2.62029
16	1140	1061	15441	2.00557
20	661	1040	15893	1.65647

Table 12: Reading for N-Chance Forwarding Algorithm

5. N-Chance Forwarding with Centrally Coordinated Cache Algorithm

Number Of Clients	**Disk Access (Number Of blocks)**	**Global Cache Access (Number Of Blocks)**	**Local Cache Access (Number Of Blocks)**	**Remote Client Access (Number Of Blocks)**	**Average access time(msec)**
4	12197	2031	478	3030	10.10727
8	6675	3331	584	7039	6.06984
12	2994	4873	584	9136	3.38229
16	1351	6142	544	9511	2.18451
20	689	7248	500	9194	1.70428

Table 13: Readings for N-Chance Forwarding with Centrally Coordinated Cache Algorithm

The graph in figure 6 shows the number of blocks accessed from the disk when the number of clients is increased from 4 to 20. Total number of blocks read= 20000.

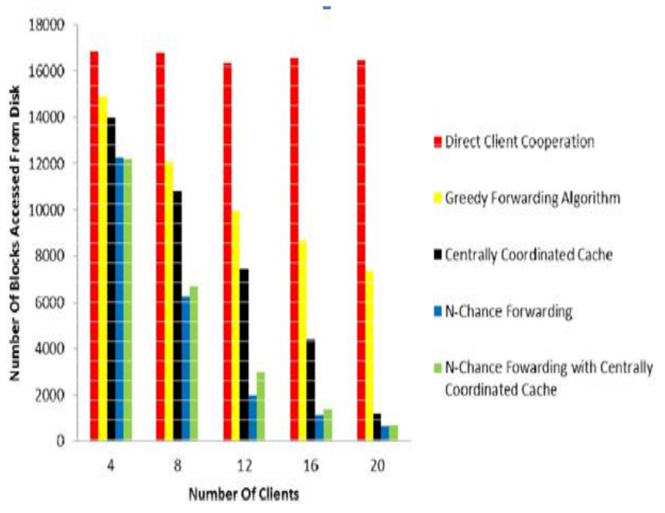

Fig 6: Graph of Number of Clients vs. Number of blocks accessed from disk for 5 algorithms

As we can see in figure 6 that when the number of blocks on the disk is very large there is disk access in all algorithms even when the number of clients is equal to 20. Also the disk access in direct client cooperation is highest and remains constant independent of number of clients. The number of disk access in greedy forwarding algorithm decreases with increase in number of clients but not as much as other three algorithms. In contrast to the first test case, the number of disk access in Centrally Coordinated Cache algorithm is greater than N-Chance Forwarding algorithm and N-Chance Forwarding with Centrally Coordinated Cache algorithm. This is because even when 80% of client's cache is maintained globally, the total cache size is less than number of blocks on the disk. So not all blocks are cached and thus disk access occurs even when the number of clients is equal to 20. The N-Chance Forwarding with Centrally Coordinated Cache and N-Chance Forwarding perform almost equally.

The graph in figure 7 shows the average access time to read a block for all algorithms when the numbers of clients is increased.

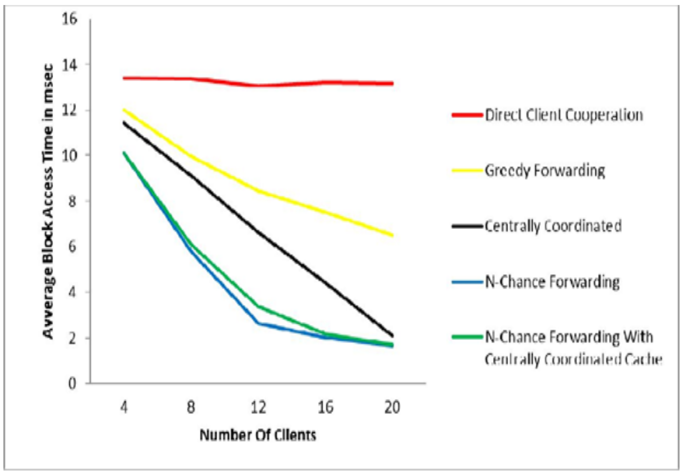

Fig7: Graph of Number of Clients vs. Average Block Access time for 5 algorithms

From graph in figure 7 it can be seen that the average access time for Direct Client Cooperation is very high and remains constant independent of the number of clients. The average block access time for Greedy Forwarding algorithm decreases with increase in clients. The average block access time of Centrally Coordinated Cache decreases with increase in clients but it is higher than the block access time in test case 1. This is because of disk accesses which occur because all the blocks of the disk are not cached. The average block access time for N-chance Forwarding algorithm and N-Chance Forwarding with

Centrally Coordinated Cache is the same and much less compared to the other three algorithms.

The graph in figure 8 shows the distribution of how the 20000 blocks are read in the five algorithms. The distribution shows the number of blocks read from disk, server cache, global cache, client local cache and remote client's cache.

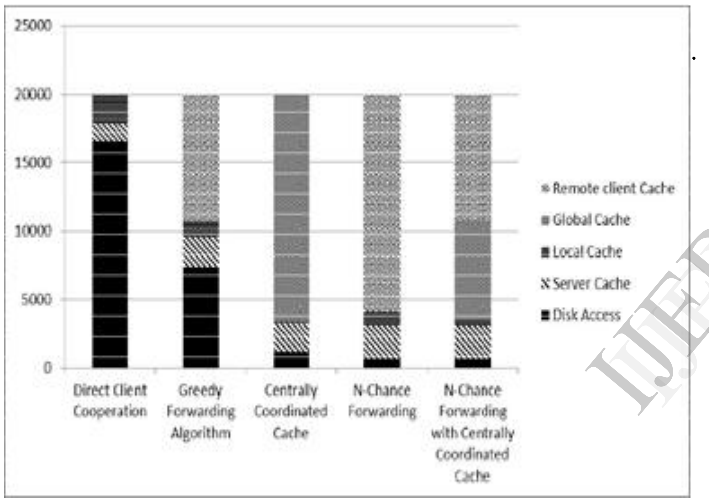

Fig 8: Graph of distribution of location of blocks for the 5 algorithms

The graph in figure 5 shows the average access time to read a block for all the 5 algorithms when number of clients=8.

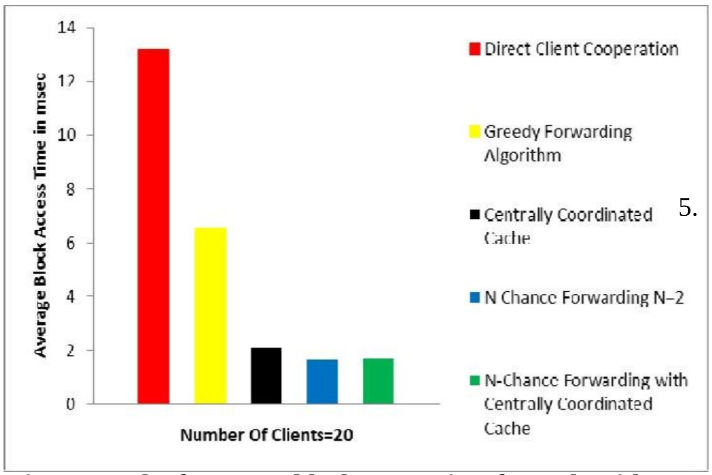

Fig 9: Graph of Average block access time for 5 algorithms

VII. Analysis of Results

The following analysis can be made based on the results of the test cases above:

1. In both the test cases the Direct Client Cooperation algorithm has the worst performance compared to the other algorithms. This is because the clients are unaware of other clients' cache content. So disk access occurs even if the block is present in another client's cache.

2. The performance of greedy forwarding algorithm is much better than Direct Client Cooperation algorithm in both test cases because of coordination of clients' cache content. The client requests a block for remote client if it has that block rather than server performing a disk access. But still the performance of greedy forwarding is not as good as the following three algorithms because of presence of multiple copies of block among clients.

The Centrally Coordinated Cache Algorithm outperforms all the other algorithms in test case 1. This is because when the number of clients in the system is greater than 8, the global cache formed by the 80% of all clients' cache caches all the blocks present in the system. Thus there is no need of performing disk access as all the blocks can be found in local cache, server cache or global cache.

But in test case 2, Centrally Coordinated Cache Algorithm does not perform as well as N-Chance Forwarding Algorithm or N-Chance Forwarding with Centrally Coordinated Cache Algorithm. This happens because of the large number of blocks on the disk. So even with more than 20 clients in the system all the blocks in the disk are not cached by Centrally Coordinated Cache because of which disk access occurs.

4. N-Chance Forwarding algorithm performs the best under both the test cases. This is because N-Chance forwarding algorithm prefers singlets. By doing this N-Chance Algorithm tries hardest to have at least one copy of every block in the system. Also in N-Chance Algorithm the number of local cache hits is high as clients use the entire local cache. Because of the high global and local hit ratio the average access time of N-Chance Algorithm is the least among all the algorithms under both the test cases.

5. N-Chance Forwarding with Centrally Coordinated Cache algorithm performs as well as the N-chance forwarding algorithm and better than the Centrally Coordinated Cache algorithm. It performs better than Centrally Coordinated Cache algorithm because in Centrally Coordinated Cache algorithm the number of local cache hits is much less as it uses only 20% of clients' cache as local cache where as in N-Chance Forwarding with Centrally Coordinated Cache the local cache uses 50% of clients' cache. Also in N-Chance Forwarding with

Centrally Coordinated Cache because of preference of singlets and 50% of clients' cache as global cache the global hit ratio is high.

VIII. Conclusions

In Distributed File Systems, the high number of disk access can largely affect the performance of the system. Based on the results of the performance of the various Cooperative Caching algorithms under the test cases, it can be concluded that Cooperative Caching algorithms can improve the read performance by reducing the amount of disk access. Cooperative Caching algorithms take advantage of remote clients' cache memory and thus avoiding disk access. Though simple in implementation and without many changes in System Architecture the Cooperative Caching algorithms can provide very good read performance improvement. Greedy forwarding algorithm is very simple in terms of implementation and can provide some read performance improvement. Centrally Coordinated Cache algorithm can provide very good read performance improvement when the number of files in system is small but it imposes load on the server. N-Chance Forwarding algorithm and N-Chance Forwarding with Centrally Coordinated Cache can provide significant improvement in the overall performance of the system. This project implementation shows that Greedy Forwarding Algorithm, Centrally Coordinated Cache Algorithm, N-Chance Forwarding Algorithm and N-Chance Forwarding Algorithm reduce the number of disk access, reducing the average block access time which increases the overall performance of the system.

Acknowledgment

I would like to thank Prof. Hans-Peter Bischof for his guidance and help throughout implementation of my Master's Capstone Project. I would also like to thank Prof. James Heliotis for his valuable inputs for improving my Master's Project Report. I am grateful to Prof. Alan Kaminsky for his Computer Science Course Library which helped me for implementation of my Master's Project.

References

[1] M. Dahlin, R. Wang, T. E. Anderson, and D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. OSDI'94, 1994

[2] Michael D. Dahlin, Clifford J. Mather, Randolph Y. Wang, Thomas E. Anderson, David A. Patterson. Quantitative Analysis of Cache Policies fir Scalable Network File Systems. Proc. 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems. May 1994.

[3] Benjamin Reed and Darrell D. E. Long. 1996. Analysis of caching algorithms for distributed file systems. SIGOPS Oper. Syst. Rev. 30, 3 July 1996.

[4] Sanjay Ghemawat, Howard Gobioff, Shun Tak Leung. The Google File System. 19th ACM Symposium on Operating System Principles. October 2003.

[5] Siddhartha Annapureddy, Michael J. Freedman, David Mazi`ere. Shark: Scaling File Servers via Cooperative Caching. In Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation - Volume 2 (NSDI'05)

[6] Randolph Y Wang, Thomas E Anderson. xFS: A Wide Area Mass Storage File System. Whitepaper.

[7] http://www.tesisenxarxa.net/TESIS_UPC/AVAILABLE /TDX-0721109- 153311//TTCR2de2.pdf