Caching for NASD

Chen Zhou Wanli Yang
{chenzhou, wanli}@cs.wisc.edu

Department of Computer Science
University of Wisconsin-Madison
Madison, WI 53706

Abstract

NASD is a totally new storage system architecture, and puts some new requirements on data caching. The goal of this paper is to evaluate caching schemes on NASD. Basically caching could be done on server side, on client side or both sides, but only server side is NASD-aware, thus it is our focus. Trace-driven simulation is used to evaluate how efficiently different schemes work and how they interact with each other (server side and client side), and synthetic workload is used in simulation. First we presented a simple access model and several caching schemes for NASD, especially the cooperative 1 Chance caching scheme and its variations. Then trace-driven simulation is used to compare their performances on server side, and the results show the cooperative schemes always perform better. After that, both server side and client side caching are taken into consideration, and their interactions are discussed. The best caching configuration in our model turns out to be 1 Chance on server side and non-cooperative on client side. Some sensitivity experiments of our model are also carried out.

Keywords: NASD, Cooperative caching scheme, 1 Chance scheme.

1. Introduction

Network-Attached Secure Disk drives (NASD) represents a revolutionary change in disk-drive and file-system technologies, eliminating the file server as the bottleneck from network and distributed file systems [Gibson99]. The bottleneck arises because a single server separates the storage network and the client network and store-and-forwards data, and also functions as concurrency control and maintains metadata consistency. In NASD, the disk management functions are embedded into distributed device, and clients are allowed to contact storage objects directly and all data and most control information travels across the network only once. Thus both high data throughput and scalability could be achieved in NASD.

Caching is a common technique for improving the performance of distributed file systems. Basically caching could be done on either client side or server side or both. Client side caching filters application I/O requests to avoid network traffic and server access, and server side caching filters server access to reduce disk accesses. Cooperative caching is a technique that seeks to improve network file system performance via coordinating cache content on different clients [Dahlin94]. So far most cooperative caching schemes are used on client side, and there seems no need to use cooperative caching schemes on server side. It is understandable since caching could be easily handled by a centralized server on server side. However, NASD makes things different, and there is no centralized server any more (although there could be a manager which takes care of security and metadata stuffs). Basically those cooperative caching schemes could also be used on server side of NASD, however, some modifications are necessary according to the NASD unique characteristic.

In the next section, we are going to present some caching schemes for NASD, which is quite similar to the existing client side schemes. In the third section, we will introduce the method we used to evaluate those schemes--the conventional trace-driven simulation. After that, some of the results will be given, based on which we draw our conclusions.

2. Caching schemes

First we are going to introduce a simple logical behavior model. It is simple enough so we can keep out unrelated information and concentrate on the caching schemes. It is also quite similar to real NASD system structure such that we believe the results based on this model still make sense in real NASD systems.

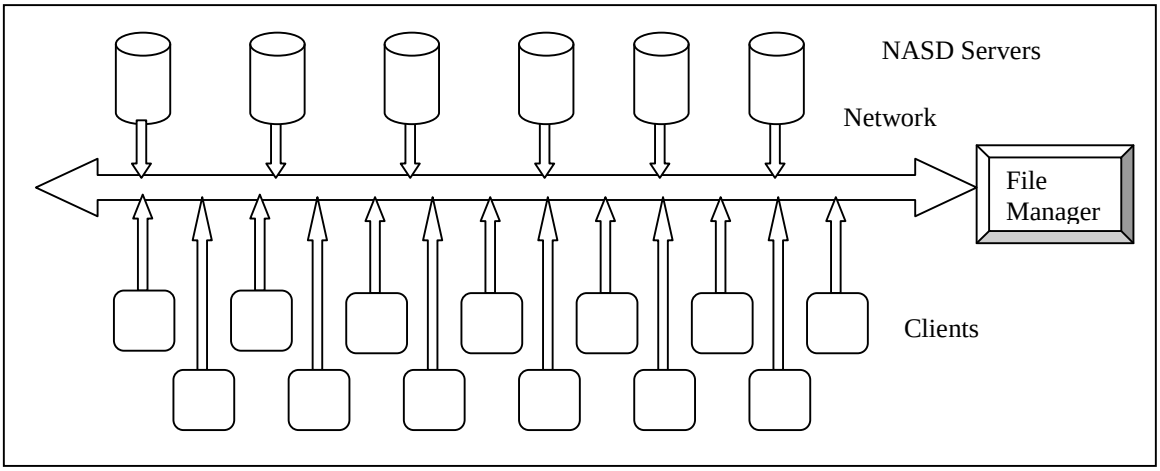

Fig. 1: System model

Basically, we assume the above behavior model. There could be many NASD servers, and even more client workstations, and a File Manager. All these are connected via a network (usually a LAN), and we assume an uniform single hop network latency between any two of the entities by default. Each of the entities has some amount of cache except the Manager.

Before the file access to servers, clients need to contact the Manager to get the capability and related information. What Manager does is to check the access rights, and if valid, return the device id/address of the specific NASD server to the client. For a single file and a certain kind of access, clients contact the Manager only once, and all subsequent file access requests could be directly forwarded to the servers. The Manager keeps some information like the location of each file, and the access rights of each file, etc. There is not much workload on the Manager, so the Manager could support quite a large number of clients and servers. We also assume that the Manager could snoop on the network to collect information about workloads on each server. This information may be useful hint to balance the workload among servers.

Then based on this model, we are going to present some caching schemes. The following taxonomy will be used intensively in this paper.

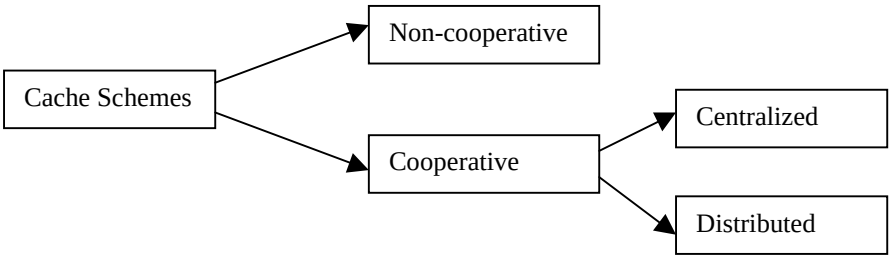

Fig. 2: Taxonomy of caching schemes

As mentioned before, caching could be done on either server side or client side or both. Since only server side scheme is NASD-aware and our goal is to evaluate caching schemes for NASD, server side will be our focus. Client side schemes are relatively independent of server side ones, although quite similar. The reason we cover them here is that we want to know how client side caching can interact with server side caching and how different combinations of client side and server side schemes work. We will introduce them separately.

Server side

Different from client side cache, server side cache usually contains only one copy of each data block. Different clients that want to access the same data block will definitely go to the same NASD server and share the data copy. We assume no consistency problem here. Actually the consistency problem is more likely to occur on client side, and maintaining the consistency will only degrade the performance a little [Nelson88]. It is neither necessary nor efficient to make cache replications on server side. However, it is possible that the files on a special NASD server are accessed too frequently and it is adequate to move some files from the server to other servers to balance the workload among the servers. The discussion of this situation is beyond this paper.

For comparison, we consider the following schemes. Except the baseline, all schemes are cooperative schemes.

- non-cooperative (baseline)

Each NASD server has its local cache, but not cooperative yet. Each server manages its cache using LRU. This is the baseline scheme.

- Global

Each NASD server has its local cache, and there is also a global overflow cache server. The total size of caches of all servers is the same for comparison reason. LRU is used both in each NASD server and the global overflow cache server. Once the local cache is full for one NASD server, the victims replaced using LRU will be forward to the global overflow cache server. When the global overflow cache is full, its victims replaced will be dropped. The performance of this scheme should be similar to another scheme that statically partitions each NASD server's cache into local part and global part. By default, the size of global overflow cache is 75% of total cache size.

- 1-Chance

This scheme is similar to N-Chance algorithm [Dahlin94]. Each NASD server has its own cache. Once the cache is full for one NASD server, the victim replaced using LRU could be forwarded to some other server or simply dropped. N-Chance algorithm try to favor the last copy of data block ("singlet") by forwarding it N times, while duplication could be dropped. There is extra overhead to maintain the "singlet" information, either in a centralized or distributed way. However, at NASD server side every data block is the last copy, thus it makes no sense to favor one over the other. Furthermore, forwarding

a block many times is more likely to cause some “thrashing” or “circular forwarding” situation which is badly disliked. 1-Chance scheme we presented here allows each data block to be forwarded only once, and only the data block originates from that NASD server can be forwarded to another server. Otherwise, it is dropped. The receiver always accepts the data blocks in our scheme. We also tried some other methods in which the receiver could reject a block if it thinks it is not worthwhile keeping the incoming block and kick out another block. But no big difference is observed, and we ignore those methods here. Every server maintains a reference list that trace the data blocks that have been forwarded out. There is another question left: what’s the destination of forwarding? Any other server could be the choice. Based on different policies to choose the destination of forwarding, we have some variations of 1-Chance scheme:

1) Via Manager

This variation allows the NASD server that needs forwarding to contact the Manager to know about the least busy server as the destination of forwarding. We assume the manager could snoop on the network to get the workload information of each NASD server. This is our default 1-Chance scheme, which is centralized.

2) Random forwarding

This variation allows the NASD server that needs forwarding to randomly choose one peer as the destination of forwarding. This is a distributed variation.

3) Use hints

This variation allows the server to choose the destination of forwarding based on hints. The hints should be easy to get locally. The hints we are using is the sum of the number of blocks forwarded to a server and the number of blocks came from that server, which could be a reasonable indication of how much workload is on that server. This is also a distributed variation.

4) Better

This variation is better in the sense that it assumes every NASD server could snoop on the network traffic and knows what it wants to know about other servers, as Manager does in the default variation. No Manager contact is required when forwarding, thus the performance should be definitely better than the default variation. However, it is not necessarily better than other variations or schemes.

5) Return on hit

This is actually not a separate variation, but an enhancement to any other 1-Chance variations. When combined with other variations, “return on hit” returns those blocks being remotely hit to their original servers. The basic idea here is try to reduce the network traffic for the subsequent accesses to the remote block that has been hit by satisfying those accesses in the original server. This method should has almost the same effect on all 1-Chance variations, thus we just combined it with our default variation (via Manager) to see the difference. For convenience, we still call it the Return variation in this paper.

Client side

Client side caching is relatively independent of server side caching. Basically existing client side caching schemes, such as N-Chance, could be applied directly to the client side in our model (However, N-Chance only takes care of reads which has no consistency problem). Usually there could be duplicated cache blocks on the client side, since different users could access the same file on the same NASD server and cache the same data block. Since we focuses on NASD caching, we concern the server side caching more than the client side. We try to simplify the client side caching without

hurting the interaction between server side caching and client side caching. Thus we assume that each client has a distinct working set, which has no intersection with other clients. On one hand this assumption guarantees that no replica exists in client side cache, which means we can apply our server side schemes almost directly to client side, thus simplifies our simulation. Another good thing is that no consistency problem will exist even we consider writes. On the other hand, this assumption actually presses the cache resource more since no share is possible, and thus cache memory is less efficiently used than it should be. Anyway, these issues are of little importance in the comparisons of relative performance of different schemes.

4. Methodology

The method we use to evaluate the schemes is the conventional trace driven simulation. One big problem we have met is how to get the right trace for NASD. NASD is so new an architecture that it seems that there is no existing file access trace for NASD. Furthermore, most traces are high-level traces for generality, which means the traces record system calls and not the detailed operation to specific physical devices. Thus only a centralized logic view is presented in the trace even if it comes from a distributed network file system. Low-level traces should be more useful to us. So far, it seems we have to generate the trace by ourselves.

One advantage of synthetic workload is that we could generate any kinds of traces on our will based on some source traces. We set up a configuration for each synthetic trace and generate the trace according to the configuration. Configuration for trace is basically a distribution of workload. Another big problem is that it is too hard to figure out which configuration is closest to the real workload. Therefore we have to try many different configurations. However, we set up the trace configurations in a way that we can distribute the same mount of workload differently, from extremely uneven to almost even. We think the real workload could between those cases.

Source Trace	Function	Access Pattern
Dinero	Trace-driven simulator	Sequential, multiple times
Cscope1	Name searching	Database file, sequential, multiple times
Cscope2	Text searching in large kernel source	Multiple large files, sequential, multiple times
Cscope3	Text searching in small kernel source	Multiple small files, sequential, multiple times
Glimpse	searching for key words among a collection of text files	Both large and small file, sequential, multiple times
Ld	link-editing the kernel	Random, different size
Postgres1	Indexed join	Random
Postgres2	Indexed and non-indexed join	Random, sequential
XdataSlice	3-D volume rendering	Random access, regular stride
Sort	UNIX external sorting utility	Intensive read write, sequential, multiple times

Table 1: Source Traces

The source traces comes from ten applications [Cao94]. Table 1 gives a brief summary of the ten source traces. Each source trace accesses only one device. We assume each source trace correspond to

the requests of one client in our model. By default we have 20 clients (we tried each source trace twice), and what we do is try to distribute the workload among the servers (by default 8 servers) differently. The configuration files we used in synthesizing traces basically describe how different source traces map to different NASD servers. A simple lottery algorithm is also used when merging the traces [Waldspurger94] to pick the next trace record from different source traces. The reason we use lottery here is that we want each source trace to span the whole period of simulation, not just in a short period. Initial "tickets" are assigned to each source trace according to the size of the source trace in the configuration file.

Most source traces have a default data block size 8KB, which we followed in our simulation by using 8KB as the cache block size. Table 2 gives some parameters used in simulation, which are quite similar to the parameters used in [Dahlin94] and [Sarkar96]. Refratio is the ratio of # entries of reference list, which is used to trace forwarded blocks, to # entries of local cache. Refratio is used only in 1-Chance scheme and its variations as an indication of size of remote cache that is accessible. # hops between server and client is used to calculate the network time for the communication between client and server.

Parameter	Default
# servers	8
# clients	20
Server cache size	8K blks
Client cache size	2K blks
Disk I/O time	Seek 14000 us, trans. 1000 us/blk
Mem. access time	250 us/blk
Network time	Lat. 200 us, trans. 400 us/blk
Refratio	6
#hops between server and client	1

Table 2: Simulation parameters

During the simulation, we collected following information:
- # cache accesses
- # local hits
- # remote hits
- # I/O accesses
- memory access, disk access, network time

Our major performance measurements are the average response time per block and the block miss rate. In most cases, we found that the two measurements are parallel to each other since the disk time dominates the response time. Therefore we only present the response time in the next section.

Another implementation issue is the efficiency of simulation. The synthetic trace is quite large and each data block access will introduce one or more cache lookups. Cache blocks are usually managed via a LRU linked list and one lookup will cause a scan of the whole list, which turns out to be time-consuming. We introduced a hash table and combined it with the LRU list. The implementation is a little tricky, while the result is very encouraging. Before using hash table, it usually takes 5-6 hours to get the result of one case, now it takes only 3 minutes, which makes it possible for us to explore a larger test space.

5. Results

The test space in our experiment is quite large. Lots of parameters could vary: server side caching scheme, client side scheme, variation of 1 Chance schemes, local cache size, remote accessible cache size, the number of hops between servers and clients, and the configuration of trace. We just simulated part of the combinations, but still got almost 600 cases. How to deal with this amount of results is also an interesting problem. Since we don't know which trace configuration is closest to real workload, it makes no sense to average the results of different configurations. However, we are glad to see how different distributions effect the results. It may be a good idea to leave them alone.

First we are going to focus on server side schemes and see how different scheme works (assume no client side scheme this time). The number of server is fixed and the amount of workload is the same in most cases. Figure 3 shows different server side schemes on different workload distributions. The X axis corresponds to different trace configurations. The larger is the number, the more evenly is the workload distributed among the servers. As we can see from this graph, generally the cooperative caching schemes (Global and 1-Chance (Manager)) perform better than the non-cooperative scheme, and they seem to be less sensitive to the change of the distribution of workload. 1-Chance scheme shows a little bit better performance over the Global scheme. When the workload is more evenly distributed among the servers, the cooperative schemes achieve less performance gain over the non-cooperative scheme. In some extreme cases, when workload is uniformly distributed (identical workload on every servers, not shown in this graph), the non-cooperative scheme could even perform better than cooperative schemes, since it causes less network traffic.

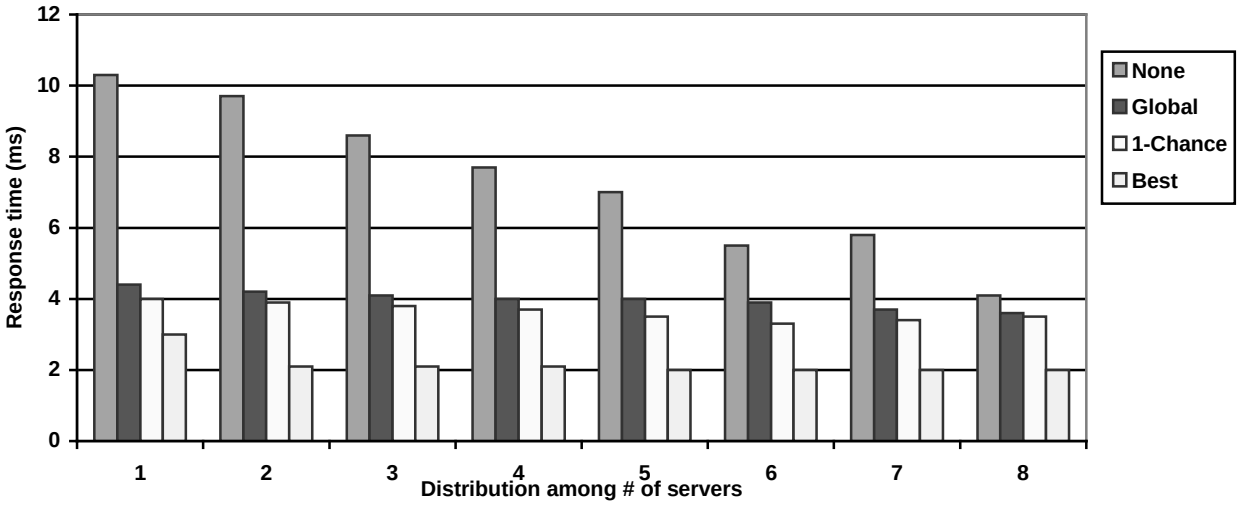

Fig. 3: Server side schemes vs. workload distributions

Figure 4 shows the performance of the variations of 1-Chances schemes. Except the default variation "Manager" and "Return" variation, all other variations are distributed caching schemes. Detailed introduction of these variations is given in the previous section. It seems no variation performs significant better than the other variations. The "Better" variation always performs a little bit better than the "Manager" variation since basically they have the same behavior but "Better" needs no Manager contact and thus have less network traffic time. In most cases, "Return on hit" enhancement achieves some performance gain over the default variation, and we believe that this is a useful enhancement to cut down network traffic. All the variations are distinct from others, and some

perform better in some cases. However, it is not the goal of this paper to explore the nuances between these variations. We conclude that these variations have comparable performance. In the sense of scalability, we prefer the distributed 1-Chance schemes much more than the centralized scheme, since no manager is needed during cooperative caching. However, in the following experiments, we only concern the relative performance, and it makes little difference to choose which 1-Chance variation to use. We still use "Manager" as our default 1-Chance variation.

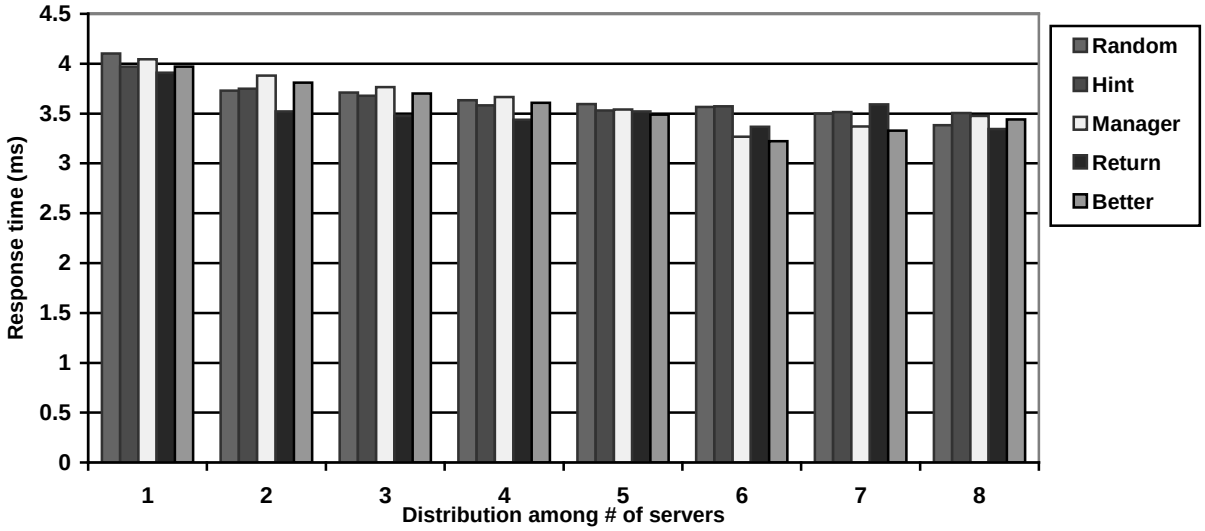

Fig. 4: Variations of 1-Chance scheme

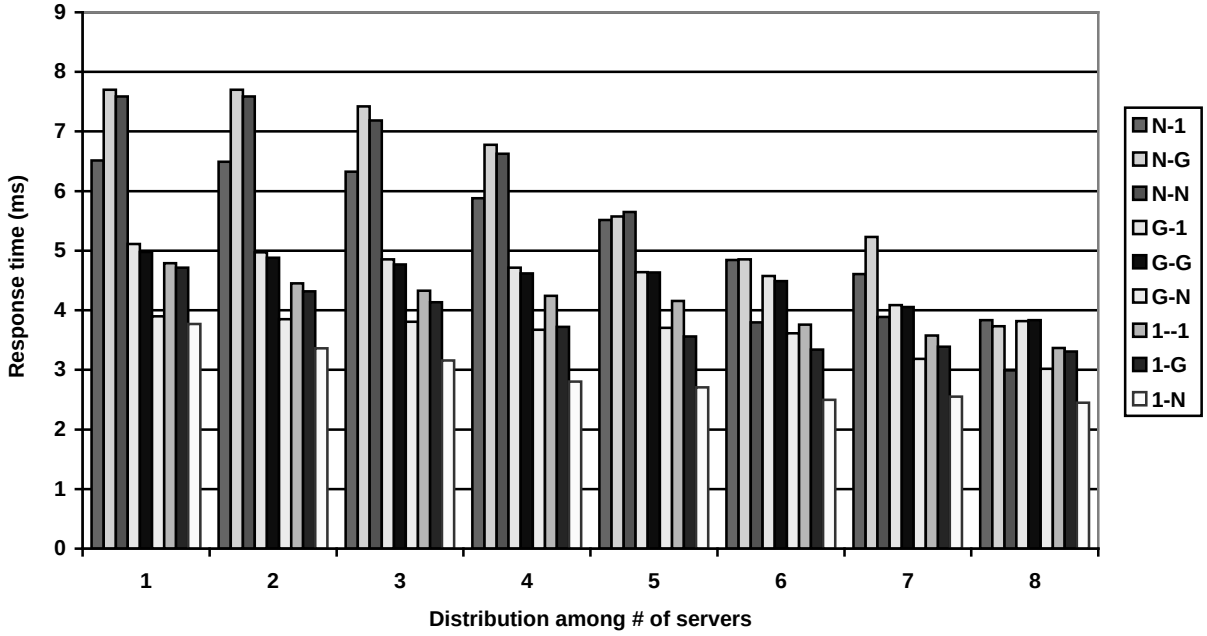

Fig. 5: Combinations of server side and client side

Now we are going to consider both server side and client side caching and see how interaction happens. As mentioned before, our server side schemes could also be applied to the client side. Taking

both non-cooperative and cooperative schemes (the Global and 1-Chance schemes) into consideration, we could get 9 different combinations in total. We tried them all and results are shown in Figure 5. The symbol “N”, “G” and “1” represent the non-cooperative scheme, the Global scheme and the 1-Chance scheme respectively. “N-1” means server side uses non-cooperative scheme and client side uses the 1-Chance scheme.

First, the best combination turns out to be 1-Chance scheme on server side, and non-cooperative scheme on client side. Second, whichever scheme is used on client side, cooperative schemes always perform better than the non-cooperative scheme on server side. Third, in many cases, cooperative schemes perform even worse than the non-cooperative scheme on client side. These results are not so straightforward, since we thought the cooperative schemes usually should perform better than the non-cooperative one on client side. We checked the overall miss rate, and the cooperative schemes on client side do show lower miss rates than the non-cooperative schemes. The cooperative caching scheme, say 1-Chance, has two effects on client side comparing with the non-cooperative scheme: reduce the miss rate and increase the miss penalty. It increases the miss penalty since it involves more network traffic than the non-cooperative scheme when local cache miss occurs. Which effect dominates determines which scheme will perform better. Intuitively, since by default we assume an uniform network latency (1 hop) between server and server, between server and client, and between client and client, the cost to hit a data block on a server’s cache is the same as to hit a data block on another client’s cache. Because usually a server has much larger a cache than a client, if local cache misses, one client gets more chance to hit the data block on a server cache than hit on another client, and the cost is the same. We are going to verify our analysis in two ways. First, we will increase the cache size of the client and see what happens. Second we will increase the number of hops between the client and server. Back to the discussion of the effect of the cooperative scheme on client side, the first method will actually reduce the client side cache miss rate, thus reduce the effect of miss penalty. The second method will actually minimize the effect of miss penalty increase caused by the cooperative scheme since other kinds of latency will dominate the miss penalty.

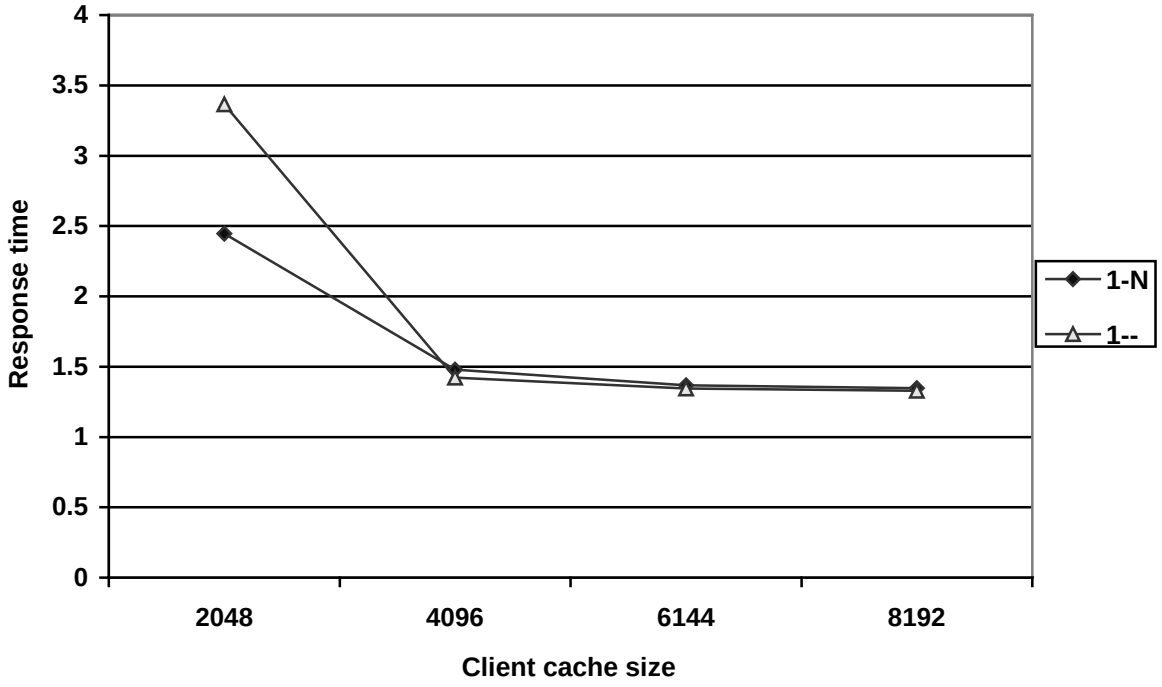

Fig. 6: Sensitivity to the client cache size

To make our analysis simple and clear, we pick only one case in which 1-Chance is used on server side and either non-cooperative or 1-Chance is used on client side ("1-N" or "1-1"), and trace configuration 8 is used. Figure 6 shows what happens when the client cache size is increased. When the cache size is 2048 blocks (default), the non-cooperative scheme is better than 1-Chance. When the client cache size is larger than 4096 blocks, 1-Chance scheme performs better than the non-cooperative scheme on client side. The reason of the flattening in the graph is that the client cache size is almost large enough to fulfill the requirement of workload of clients.

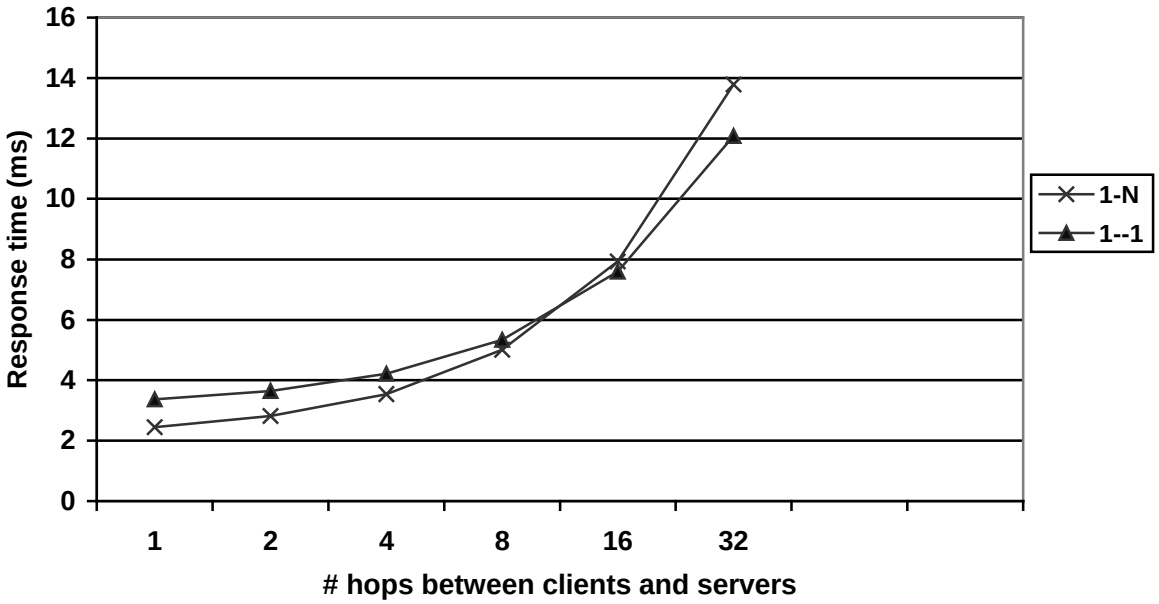

Fig. 7: Sensitivity to # hops between clients and servers

Figure 7 shows what happens when the number of hops between clients and servers changed. When # hops = 1, the non-cooperative scheme works better than the cooperative schemes on client side. As the increase of # hops, the response time of "1-N" increases faster than the other one, and finally the "1-N" exceeds "1-1", which means the non-cooperative scheme becomes worse than the 1-Chance scheme on client side. When calculating the network time, we only count in the network latency and transfer time and assume the network is absolutely reliable and congestion free. If the queuing time and other delays are taken into consideration, "1-N" could exceed "1-1" at a much smaller number of hops.

6. Conclusions

Based on the results we got through the simulation, we have reached the following conclusions for the caching on NASD:

When only caching on server side, generally the cooperative schemes perform better than the non-cooperative scheme, and they are less sensitive to the change of workload distribution among servers. Especially, when the workload is more unevenly distributed, the cooperative schemes have more performance gain over the non-cooperative scheme.

Several variations of 1-Chance scheme presented in this paper have comparable performances. When scalability is concerned or when network become the bottleneck, the distributed schemes ("Random forwarding", "Use hints") are highly preferred. "Return on hit" seems to be an useful enhancement to the 1-Chance scheme and could be combined with any variation.

As for the interaction between server side caching and client side caching, the scheme combination that works best in our default model seems to be 1-Chance scheme on server side and non-cooperative scheme on client side. Whichever scheme is used on client side, cooperative schemes always do better than the non-cooperative scheme on server side. The evaluation of client side schemes is much more complicated, and it is sensitive to not only the scheme used on server side, but also the cache size, network latency, and other parameters in our model.

We have given some introduction to the variations of 1-Chance scheme, and our future work should focus on the detailed difference of variations and the scalability analysis. Other cases of the system model should also be considered, such as clients being far away from each other, servers being far away from each other (in this paper we only discussed the case that all servers and clients are close to each other, and the case that servers are close to each other, clients are close to each other but servers are far away from clients). Furthermore, other system models that may have different access behaviors are also interesting research topics. Realistic workload should be used in the future simulation.

7. Acknowlegements

We would like to thank Remzi H. Arpaci-Dusseau from University of Wisconsin for giving us so much useful advice and helping us understand the problems and keep making progresses.

References

[Cao94] P. Cao, E. W. Felten, and K. Li. Implementation and performance of Application-Controlled File Caching. In proceedings of First USENIX Symposium on Operating Systems Design and Implmentation, P165-178, Nov. 1994.

[Dahlin94] M. D. Dahlin, R. Y. Wang, T. E. Anderson, D. A. Patterson. Cooperative Caching: Using Remote Client Memory to Improve File System Performance. In proceedings of the 1st Symposium on Operating System Design and Implementation, P267-280, Nov. 1994.

[Gibson97]G. A. Gibson, D. F. Nagle, K. Amiri, Fay W. Chang, Eugene M. Feinberg, H. Gobioff, C. Lee, etc. File Server Scaling with Network-Attached Secure Disks. In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (Sigmetrics '97), Jun. 1997.

[Gibson99] G. A. Gibson, D. F. Nagle, W. Courtright II, N. Lanza, P. Mazaitis, M Unangst, J. Zelenka. NASD Scalable Storage Systems.

[Nelson88] M. N. Nelson, B. B. Welch, and J. K. Ousterhout. Caching in the Sprite Network File System. ACM Transactions of Computer Systems, 6(1):134-154, Feb. 1988

[Sandberg85] R. Sandberg, d. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and Implementation of the Sun Network File System. In Proceedings of the Summer 1985 Usenix Conference, P119-130, June 1985.

[Sarkar96] P. Sarkar, J. Hartman. Efficient Cooperative Caching using Hints. In proceedings of 2nd Symposium on Operating Systems Design and Implementation. Oct. 1996.

[Waldspurger94] C. A. Waldspurger, W. E. Weihl. Lottery Scheduling: Flexible Proportional-Share Resource Managerment. In proceedings of the First Symposium on Operating Systems Design and Implementation, p1-11, Nov. 1994.