Clustering and Prefetching techniques for Network-based Storage Systems

Dhawal N. Thakker
Networking Research Group
Middlesex University
London, NW4 4BT
d.thakker@mdx.ac.uk

Glenford E. Mapp
Networking Research Group
Middlesex University
London, NW4 4BT
g.mapp@mdx.ac.uk

Abstract

The usage of network-based applications are increasing, as the network speed is increasing. Also, the access to streaming applications, e.g BBC I-Player, Youtube etc, over the networks are increasing, as the processor speed and the amount of memory available is increasing. These applications access data sequentially over the network. However, the rate at which streaming applications access data is much faster than the rate at which the blocks can be fetched from the network storage due to the availability of the network bandwidth. Therefore, there is a need to analyse the prefetching and clustering techniques for the network-based storage system. In addition to sequential access, the system also need to satisfy random access.

In this paper, we analysed the number of blocks that need to be prefetch for the streaming applications and the number of block request that could be clustered in a network buffer, so that the streaming applications can run without jittering and demand misses can be satisfied in reasonable time.

Dans ce document, nous avons analysé le nombre de blocs qui doivent łtre prefetch pour le streaming d'applications et le nombre de bloc de demande qui pourraient łtre regroupés dans un réseau de tampon, de sorte que le streaming d'applications peuvent fonctionner sans jittering rate et de la demande peut łtre satisfaite en délai raisonnable

1. Introduction

Due to the increase of CPU speed and the amount of memory available, there is an increase in the use of the multimedia (streaming applications) and database applications in the working environment. These applications access files sequentially from the storage device (e.g disk) and need to be served at constantly high data rates while they are executing, e.g. High Definition video (HD) data rate is 5 MB/Sec, MPEG-4 data rate is 2.5 MB/Sec etc. Also, due to the increase in network speed, most of these applications are accessed over the network rather than using a local disk storage.

As the future access pattern is known for applications mentioned above, prefetching could be used. Prefetching enables the file system to bring blocks of data before they are requested. This allows applications to run without waiting for the blocks to be fetched from the storage device i.e. without stalling, thus reducing the latency experienced by the running application. For example, Ext2 and Ext3 file systems used in Linux perform static prefetching whereby a maximum depth of 128KB is used to prefetch when sequential access is detected.

Prefetching can only work, if it is economical. That is, if more blocks are fetched than requested, the time to fetch additional and requested blocks should be comparable to the time it would take to fetch only the requested blocks. The additional time incurred by prefetching will result in a latency on waiting requests which might need to be promptly serviced.

Clustering should be used to minimise the latency experienced. It has the ability to fetch multiple blocks simultaneously thereby reducing the latency when compared to fetching them one block at a time. However, the number of blocks clustered in a request and the reduction in latency depends on the storage mechanism being used. This can be exemplified using the disk hardware where the major component of latency is the "seek time", the time to locate the correct track. Once the correct track is located all the data on the same track can be obtained without seek time. This means that a disk can only cluster if multiple requests are laid on the same track. Clustering on a disk can obtain up to 23 MBps compared to 200KBps [1], when blocks are fetched one at a time.

This observation demonstrates that clustering could provide high data rates, but it is storage dependent. Also it could be exploited by prefetching, as it allows the system to

[1] Assuming 5 millisecond revolution time (latency) to fetch one

prefetch blocks economically [2].

The elevator algorithm, RAID system and caching can be used in addition to prefetching and clustering techniques. These techniques also minimise the latency and can obtain high data rates for the disk systems. The elevator algorithm rearranges the requests in a way such that the disk will experience minimal amount of seek time. This is done by arranging requests in the increasing track numbers, minimising the movement of the head. RAID systems can satisfy multiple requests in parallel and obtains high data rates.

In most operating systems, recent accesses to the disk are stored in a memory cache such as the UNIX buffer cache. Therefore, when blocks are needed by an application, the cache is initially searched in order to satisfy a request. If the block is found in the cache, no latency is experienced by the running application [3], if it is not found it generates demand misses which must be satisfied promptly.

Using selective or all of the above techniques discussed, file systems can fetch blocks for streaming applications (for sequential accesses) and demand misses from the disk storage system, such that the overall latency experienced by the running applications is minimised.

However, due to the increase in speed of the network and their availability (e.g. 1 Gigabit networks are readily available and the availability of 10 Gigabit network speed is not very far in the future), most of the multimedia and database applications such as BBC I-Player, Youtube, Systems Applications and Products (SAP) etc, run over the network. Some of the discussed techniques to obtain high data rates and minimise latency are not relevant, as there is no seek time involved in the network storage. Using similar techniques like a RAID system to satisfy multiple requests in parallel are not readily available for the network storage. Caching techniques in the network storage system could remain similar to the disk-based systems, as the accessed data could be cached regardless of being fetched from the disk or network storage. In addition, prefetching and clustering techniques could also be used in network storage. As pointed out earlier, the clustering effect depends on the storage mechanism, therefore it needs to be explored.

2. Motivation

In order to explore the network characteristics the Network Memory Server (NMS) was developed [3, 7]. The NMS server stores all the data of the clients in the memory of the server and therefore it will not involve the cost of fetching data from the disk. Hence, the latency experienced in fetching a block of data will be dominated by the characteristics of the network.

[2]For disk, the statement will be true, only if prefetched blocks are laid on the same track

[3]Obviously, it will take time to search a block in the cache

Analysis of the network characteristics

NMS is composed of the Latency cost (L) and the constant cost (C). The latency cost is the overhead time (going through the stack up and down on client and server side) and transmission time (sending network buffer to and fro between client and server). It varies depending on the network load. The constant cost [1] is the time taken to search for the block and copy it into the network buffer on the server, and to copy the block from the network buffer into memory on the client. These two variables summate to the time $T_{net}(y)$ which is the time taken to fetch y number of blocks requested in a network buffer, as demonstrated by the formula below:

$$T_{net}(y) = L + Cy \tag{1}$$

It has been observed from experiments that the time to read one block from NMS takes 200 μsec in which 170 $\mu sec(L)$ is the Latency and 30 $\mu sec(C)$ is the Constant i.e. 85% overhead to give data rate of 5MB/Sec. By this system, if 5 blocks are requested at a time then the transfer rate rises to 16MB/sec with a latency of 320μsec. Moreover, if 10 blocks are requested at a time then the transfer rate obtained will be 33MB/Sec with a latency of 470μsec. This analysis and experimental results are shown in the Figure 1. They indicate that there is a huge clustering effect which can provide high data rates that could be exploited by prefetching. The above experiments were done when there was no other traffic on the network. However, the data rates obtained through clustering could provide sustainable transfer rates for varying network loads.

The key issue becomes whether network file systems can take advantage of the network characteristics to improve performance using clustering and prefetching over the network. The most popular implementation of a network file system is the Network Filing System (NFS) developed by Sun Microsystems. The NFS is a file server in which the client communicates with the server using file commands. It reads and writes data one file block at a time and it does not implement any prefetching or clustering techniques. Hence, NFS is unable to make use of high network bandwidth that is available to obtain high data rate.

So the research question to be addressed is: "Can a file system be develop using clustering and prefetching techniques over the network which can be ideal for the working environment, where it can allow streaming applications (once they are started) to run with no stalling while satisfying demand requests in reasonable time?"

[1]Constant cost will vary depending on the number of request waiting to get served on the server, it is referred as constant because we assume that there is no queue at the server-end

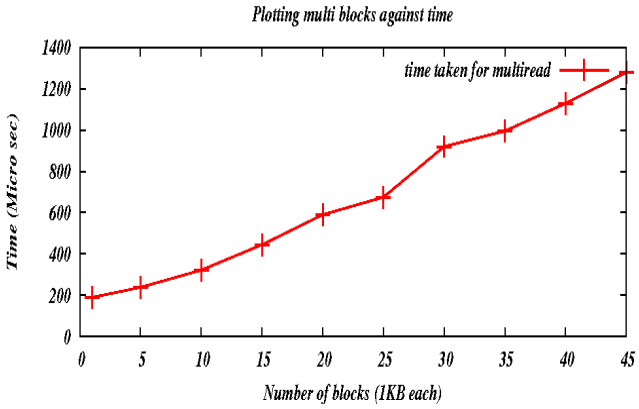

Figure 1. Multiple block Latency

3. Related Work

The important research work done in the area of interest are listed below:

- Pei Cao et al.(1995) [2] proposed four rules that optimal integrated strategies for prefetching and caching must satisfy. However, the prefetching work was more theoretical and the implemented prefetching strategy was static prefetching.

- Papathanasiou and Scott(2005) [10] argued that technological trends and emerging system design goals have dramatically reduced the potential costs and increased the potential benefits of highly aggressive prefetching policies. The authors proposed that memory management need to be redesigned to embrace such policies. This work motivated us to explore memory management.

  These authors also came up with the efficient prefetching and caching techniques [9] to maximise power-down opportunities (without performance loss) by creating an access pattern characterized by intense bursts of activity separated by long idle times.

- Li et al. work(2007) [6] used the knowledge of I/O switch time, to decide how much to prefetch, to improve performance of the sequential access.

- Patterson et al. work(1995) [11] proposed the notion of prefetch horizon i.e when to initiate prefetching for the known reference, to use cache effectively and to minimise the execution time of the applications.

  However, Li et al. and Patterson et al. work was disk centric and both of these research did not consider the time taken to consume blocks by applications. We will explore similar ideas for the network based storage.

- Rochberg and Gibson(1997) [12] extended the work of Patterson et al. by implementing the Patterson et al. framework over the network, but due to its limitation on clustering similar to NFS, the Rochberg and Gibson work did not perform as well as disk.

The distributed file systems such as the Google File System (GFS) [4], the OceanStore [5], the Serverless Network File System (xFS) [1], the Sprite network operating system [8] implemented caching techniques to improve response time. They also used huge block sizes, to utilise the available network bandwidth. Their work did not looked at clustering requests or to prefetch multiple blocks of different files at a time.

This research will build upon these efforts by looking at the network characteristics. To develop an algorithm, using prefetching and clustering techniques. This will guarantee the quality of service for the applications running over the network.

4. Proposed Work

This section analyses the design to address the research question. In first section, it analysis how to treat streaming applications. In second section, it talks about spare-time. In third section, it analysis how to treat demand misses. Lastly, it talks about developing an analytical model, which will be used to explore proposed work boundaries and performance.

4.1. Streaming Access

The streaming applications access blocks sequentially at a constant rate. Knowing that streaming applications access blocks constantly over the period of time, we will first derive an equation to analyse how many blocks should be prefetched so that streaming applications can run without any jitter once they are started. Streaming applications will not start until enough resources i.e network and prefetch buffers, are available, we refer to this wait time as start stall-time.

Let T_{cpu} be the time to consume a block for a streaming access, then the rate at which it will consume y blocks will be $T_{process}(y)$ i.e.

$$T_{process}(y) = T_{cpu} * y \tag{2}$$

From our experiments, we have observed that the time to fetch y blocks over the network is equal to $T_{net}(y)$ which is

$$T_{net}(y) = L + Cy \tag{3}$$

where L is the Latency i.e. the time to set up connection between the client and server. The L can vary depending on the network load and the bandwidth available. However, it

can be assumed to be a constant for a given network. The y is the number of block request in a network buffer and C is the time taken to get one block.
Now, for an application to run without any delay or jitter, the time taken to fetch blocks should be less than or equal to the time taken to consume them i.e.

$$\begin{aligned} T_{net}(y) &\leq T_{process}(y) \\ L + Cy &\leq T_{cpu} * y \\ L &\leq (T_{cpu} - C) * y \\ L/(T_{cpu} - C) &\leq y \end{aligned} \quad (4)$$

The equation shows that the number of blocks prefetched for streaming applications should be equal to or greater than $L/(T_{cpu} - C)$. And it should be fetched at an interval of $T_{process} * (y)$, to allow them to run without stalling[3].

Now if $L/(T_{cpu} - C) = y$, only double buffering is needed to satisfy the request for streaming accesses as shown in the Figure 2, that is to use only two sets of buffers for prefetching and to initiate next prefetching as soon as the first buffer is available (at an interval of $(T_{process} * y)$) and this will not cause any jitter/delay in streaming applications, as required blocks will be prefetched just before they are needed, this is similar to the notion of Prefetch Horizon which was discussed in Informed Prefetching and Caching.

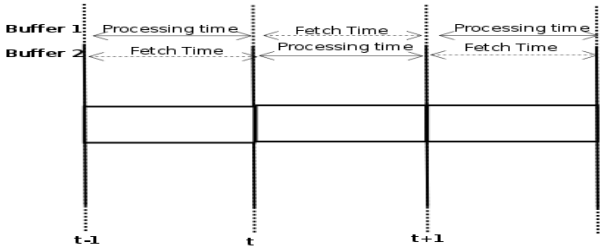

Figure 2. Double Buffering in steady state: Processing time is the time taken to consume block and fetch time is the time taken to fetch the block from the NMS. There is no spare-time as the time to fetch blocks is equal to processing blocks.

Note having more than two network buffers will be a waste of memory for the above condition, as the time taken to fetch a network buffer is equal to the time taken to process blocks. The condition will also not give us any slack to satisfy any demand misses or to allow new streaming applications to join, as asking more blocks than y will increase the fetch time and will incur stall/jitter in the running applications and will not obtain the quality of service needed.

In Informed Prefetching and Caching there was no need to prefetch beyond the prefetch horizon, as the work used parallel disk, this allowed it to satisfy demand misses as they occurred, therefore it made sense to not prefetch beyond the prefetch horizon.

[3]It will experience start stall time to prefetch first y blocks

However in our case, networks cannot take benefit of parallelism. Hence they are constraint. Therefore prefetching beyond the prefetch horizon would make sense whenever it is possible, to allow streaming applications to run while the demand misses are being satisfied.

The only way to minimise the constraint in network is to make sure that time taken to process prefetched blocks should be greater than the time taken to fetch prefetch blocks. This will leave spare time as show in Figure 3, but having y too much greater than $L/(T_{cpu} - C)$ will increase the start stall-time. Therefore, y value should be in such a way that it leaves enough spare-time and it minimises the start stall-time.

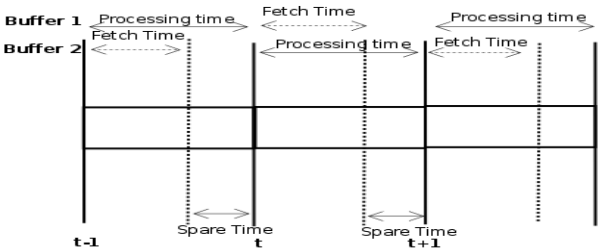

Figure 3. Double Buffering in steady state: Processing time is the time taken to consume block and fetch time is the time taken to fetch the block from the NMS. Spare time is the time difference between the processing and fetch time, which can allow to do more than double buffering if needed.

In short, the Equation 4 guarantees the quality of service for streaming applications. There will be no overload on the network and buffering system, as it only prefetches the required number of blocks in a cycle that are need for streaming applications to keep them running. Therefore, if a streaming application is cancelled in between by the user then atmost it will only bring in the P number of uneccessary blocks, as from next cycle, when a stream is discontinued the number of P blocks fetched will be reduced. However, we might also need to increase prefetching, to satisfy increasing number of demand misses or to allow new streaming applications to join.

4.2. Spare-time

The spare-time is very important to provide the required level of quality of service. Spare time can satisfy demand misses and allow new streaming applications to join the ongoing prefetching without incurring any jittering in the running applications.

Spare-time can be used in following way:

- To fetch additional blocks with ongoing prefetch, for demand misses or for joining new streaming applications. The number of additional blocks $(N_{AB}(p + d, j))$ (Number of Additional Blocks with Prefetching, can be Demand blocks (d) or for Joining new stream

(j)) that can be fetched with ongoing prefetching (p) without affecting prefetching for streaming applications is equal to:

$$N_{AB}(p+d,j) = SP/C \tag{5}$$

where C is the constant cost for fetching a block. SP is the spare-time between prefetches. The greater the spare-time more additional blocks could be fetch without penalising streaming applications prefetch. $N_{AB}(p+d,j)$ is the maximum number of blocks that could be clustered with ongoing prefetch cycle without affecting the running streaming applications.

The Equation 5 shows the maximum number of block requests that can be fetched with the ongoing prefetch cycle using spare-time. This spare-time can also be increased by doing more aggressive prefetching i.e prefetching more than required number of blocks for the running streaming applications as shown in the Figure 4, but this is only possible, if we have spare-time at the first place i.e. if there is no spare time then we cannot increase spare time.

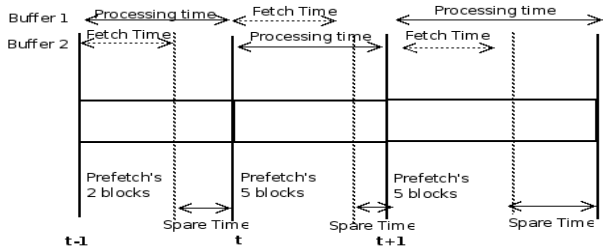

Figure 4. Prefetching Beyond Prefetch Horizon: Doing more prefetching increases the fetch time for the cycle t and increase the processing time for cycle $t+1$. Hence increases the spare-time at the cycle $t+1$.

For example, let $N_{AB}(p+d,j)$ equal to 6 and the ongoing prefetching requires 2 block at an constant interval of ($T_{process} * y$). Now as shown in Figure 4, in cycle $t-1$, it will prefetch 2 blocks. If we prefetch 5 blocks instead of 2 blocks for streaming applications, as in cycle t, then the fetch time will increase for the cycle t. However, fetching more blocks at cycle t will increase the processing time for the cycle $t+1$, as extra 3 blocks are available to consume at cycle $t+1$.

In this example, if $T_{CPU} = 400$ and $C = 30$, then the increase in the spare-time $= 3(400-30) = 1110\mu$ sec. The analysis showed that spare-time can be used to do more prefetching i.e buffering, therefore the Equation 5 notation will change to:

$$N_{AB}(p+d,j,b) = SP/C \tag{6}$$

where b refers that additional blocks could be for buffering i.e more prefetching.

However, the question is how much spare-time is required for the system to provide the required quality of service. The approach to the raised question will be discussed later.

4.3. Demand Access

These accesses are generated by applications without giving any prior notice and will need to be satisfied as soon as possible.

Clustering a demand miss with ongoing fetches will add additional 30 μsec to the fetch operation. To bring in additional d demand misses with ongoing prefetch p will be $T_{demand}(d+p)$:

$$\begin{aligned} T_{demand}(p+d) &= L + C_p + C_d \\ T_{demand}(p+d) &= L + C_{p+d} \end{aligned} \tag{7}$$

where $C_p(C*p)$ is the time to prefetch p number of prefetch blocks. The $C_d(C*d)$ is the time to bring in additional blocks with on going prefetch. The number of demand misses (C_d) fetched should be less than or equal to $N_{AB}(p+d,j,b)$.

Note that number of requests clustered into a network buffer increase the time to fetch the network buffer, therefore adding more block requests to the network buffer will add more delay to the demand requests. This shows that there is a need to analyse how many requests should be clustered, so that demand misses are not penalised by clustering too many requests.

Let T_{disk} represent the average time taken to satisfy the demand requests. $T_{wait}(1)$, is the waiting time of a demand request which is received first in the demand queue. Now as long as we can guarantee that the time taken to satisfy each demand request over the network is less than or equal to the average time taken to satisfy a demand request in the disk, the quality of service will be better than or equal to the disk. From this we have,

$$T_{demand}(p+d) + T_{wait}(1) \leq T_{disk} \tag{8}$$

The Equation 8 shows that in fetching the network buffer, the sum of the time taken to fetch the first received demand request ($T_{demand}(d)$) and the waiting time ($T_{wait}(1)$) of that request should be less than or equal to T_{disk}.

Substituting Equation 7 in Equation 8 for clustering demand misses.

$$\begin{aligned} L + C_{p+d} + T_{wait}(1) &\leq T_{disk} \\ & and \\ C_d &\leq N_{AB}(p+d,j,b) \end{aligned} \tag{9}$$

The Equation 4 and Equation 9, will guarantee the quality of service for demand requests and for running streaming applications respectively. The analysis can also give us

an idea of how much to cluster when dealing with demand requests. This means that we cluster until the Equation 8 is satisfied.

4.4. Putting it all together

In the section 4.1 and section 4.3, we obtained an equation for the streaming applications and demand accesses individually i.e without considering each other.

Now, we will obtain an equation so that both of these requests can be satisfied simultaneously without penalising each other. As shown in the Figure 4, as long as the time to consume prefetched blocks which were fetched at cycle $t-1$ is greater than the time to fetch blocks at cycle t, streaming applications can run without jittering. If $P_{(t-1)}$ is the number of blocks fetch at cycle $t-1$ then,

$$\begin{aligned} P_{(t-1)} * T_{CPU} &\geq T_{demand}(p+d)_t \\ P_{(t-1)} * T_{CPU} &\geq (L + C_{p+d})_t \end{aligned} \tag{10}$$

where $T_{demand}(p+d)_t$ is the time taken to fetch, p prefetch and d demand blocks at cycle t. Also from the Equation 9, we should have $(L + C_{p+d})_t + (T_{wait}(1))_t \leq T_{disk}$ and $(C_d)_t \leq (N_{AB}(p+d, j, b))_t$.

The Equation 10 should be satisfied at any point of time in the system, to guarantee the needed quality of service. However, the number of blocks that need to be prefetched for the streaming applications under varying rate of demand misses and network bandwidth still need to be analyse.

4.5. Towards an Analytical Model

To analyse how much spare-time is required at different rate of demand misses and at varying network load, so that quality of service is guaranteed, the work will develop an analytical model. This will analyse boundaries of the over all proposed system i.e how many ongoing streaming applications can be executed and the rate at which demand misses can be satisfied simultaneously, given the network bandwidth. Beyond the analysed boundaries, the proposed work will not be able guarantee the quality of service. Currently, the development of a mathematical model will be explored using queuing theory.

5. Conclusion

In our study we investigated prefetching and clustering techniques for streaming applications and demand access which run over the network. It showed that the streaming applications can run without stalling once it is started. Demand access can be satisfied with on-going prefetching for the streaming applications.

However, this design still needs to be explored using mathematical modeling, to analyse the boundaries and stability of the proposed design.

References

[1] T. Anderson, M. Dahlin, J. Neefe, D. Pat-terson, D. Roselli, and R. Wang. Serverless network file systems. In *In Proceedings of the 15th Symposium on Operating System Principles. ACM*, pages 109–126, Copper Mountain Resort, Colorado, December 1995.

[2] P. Cao, E. W. Felten, A. R. Karlin, and K. Li. A study of integrated prefetching and caching strategies. In *SIGMETRICS '95/PERFORMANCE '95: Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 188–197, New York, NY, USA, 1995. ACM Press.

[3] O. Gemikonakli, G. Mapp, D. Thakker, and E. Ever. Modelling and performability analysis of network memory servers. In *ANSS '06: Proceedings of the 39th annual Symposium on Simulation*, pages 127–134, Washington, DC, USA, 2006. IEEE Computer Society.

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA, 2003. ACM Press.

[5] J. Kubiatowicz, D. Bindel, Y. Chen, S. C. P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 190–201, New York, NY, USA, 2000. ACM Press.

[6] C. Li, K. Shen, and A. E. Papathanasiou. Competitive prefetching for concurrent sequential i/o. *SIGOPS Oper. Syst. Rev.*, 41(3):189–202, 2007.

[7] G. Mapp, D. Thakker, and D. Silcott. The design of a storage architecture for mobile heterogeneous devices. *icns*, 0:41, 2007.

[8] M. N. Nelson, B. B. Welch, and J. K. Ousterhout. Caching in the sprite network file system. *ACM Trans. Comput. Syst.*, 6(1):134–154, 1988.

[9] A. E. Papathanasiou and M. L. Scott. Energy efficient prefetching and caching. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 22–22, Berkeley, CA, USA, 2004. USENIX Association.

[10] A. E. Papathanasiou and M. L. Scott. Aggressive prefetching: an idea whose time has come. In *HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems*, pages 6–6, Berkeley, CA, USA, 2005. USENIX Association.

[11] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 79–95, New York, NY, USA, 1995. ACM.

[12] D. Rochberg and G. Gibson. Prefetching over a network: early experience with ctip. *SIGMETRICS Perform. Eval. Rev.*, 25(3):29–36, 1997.