Random Striping for News on Demand Servers

Juan Alemany and Jayram S. Thathachar

Technical Report UW-CSE-97-02-02
February, 1997

Department of Computer Science and Engineering
University of Washington
Box 352350
Seattle, WA 98195

Random Striping for News on Demand Servers

Juan Alemany and Jayram S. Thathachar
{juanito,jayram}@cs.washington.edu
Department of Computer Science and Engineering
University of Washington
Box 352350
Seattle, Washington 98195–2350

Abstract

We address the problem of assigning blocks of video streams to disks in News on Demand video servers. Specifically, given the common assumption that video streams are striped across the disks, we consider how to map successive blocks to disks to maximize performance.

Our work begins by noting that the commonly proposed round-robin assignment, in which successive blocks of every stream are mapped to successive disks, provides the guarantee that once a video stream starts it will never miss a deadline, but that the latency to start a stream may be high due to the "convoy effect."

We propose and analyze an alternative scheme, the random mapping of successive blocks to disks. Using both analytic and simulation techniques, we show that random mapping and replication significantly reduce the latency to start a new stream, and the number of missed deadlines is negligible. Further, we prove that with enough copies, no request sequence will miss any deadlines, for loads that are a significant fraction of the maximum supportable by the hardware.

1 Introduction

News on Demand servers are expected to provide thousands of users with immediate access to hundreds of video clips with the latest breaking news. These servers will store a huge number of short clips with a constantly changing distribution concentrated only on the latest or more sensational news.

Using the same implementation strategies of Movies on Demand servers to implement News on Demand servers results in high latency. Most video server prototypes use *round robin striping* to provide high throughput and real time guarantees under arbitrary request distributions. However, a combination of a skewed access pattern and short clips can cause a significant fraction of requests to experience relatively high latency. Storing multiple copies of each clip in the server alleviates this problem but does not eliminate it altogether.

We present random striping, an alternative organization for news servers that guarantees minimal response time at the expense of strong real time guarantees. This scheme replicates each video a small number of times and randomly assigns pieces of each copy to all disks. Given the random nature of the assignment, this scheme introduces the possibility that the server will be unable to deliver all video frames to some set of requests in a timely fashion. If this happens, some viewers will experience a discontinuity in video playback, also known as a *hiccup.*

In this paper we provide experimental evidence that random striping is a better strategy than round robin striping for News on Demand servers. Our experiments show that random striping provides minimal latency and does not suffer from hiccups even under loads close to the full server bandwidth. In contrast, our experiments show that round robin striping at similar loads produces high latency for a significant fraction of requests.

We present theoretical evidence that confirms that random striping does not suffer from hiccups even under high loads. We prove that with enough copies, for *any* request distribution, and for any fixed fraction of the maximum server bandwidth, random striping produces *no* hiccups. We provide conservative lower bounds on the number of copies necessary to make sure that there are no hiccups. For comparison, we show the estimates of the maximum hiccup–free load predicted by our result, and in a simple experiment, we show a much more practical estimate of the hiccup–free load supported by our scheme.

The paper is organized as follows: Section 2 introduces the basic facts about video servers, striping and replication. Section 3 introduces random striping and explains why it guarantees low latency but may suffer hiccups. Section 4 describes round robin striping and explains why requests might suffer high latency. Section 5 contains an experimental evaluation of random striping and round robin striping for a News on Demand application. Section 6 presents our theoretical result guaranteeing no hiccups for any distribution of requests; the proof of this result is presented in the appendix. The last two sections, 7 and 7.1, contain respectively, a summary of related work and a conclusion.

2 Video Servers, Striping and Replication

This section briefly describes the basic operation of disk based video servers, and introduces the basics of striping and replication.

A video server consists of a collection of disks connected to a group of viewers by a network. Video clips are digitally compressed and divided into *blocks* of equal size before they are stored on disks. The server operates in a series of *rounds* of equal length: in each round, the server retrieves the next block of data for each playing clip and the network delivers it to the appropriate viewer. We assume that a block is stored contiguously on one disk. In order to provide a smooth flow of images, a block must contain enough video data to last each viewer for the duration of a round and the server must be able to retrieve the next blocks for all viewers before the end of the round. If the server fails to retrieve a block before the end of the round, the viewer will perceive a discontinuity which we call a *hiccup.*

Striping is a technique for spreading out data on multiple disks to balance load and improve latency and throughput. A striped server partitions each video into small consecutive blocks which are distributed among multiple disks. Spreading blocks of a video clip over several disks allows multiple disks to serve each request, balancing load. The key parameters affecting performance in striping are the size of each block and the number of disks that a clip is striped over. Smaller blocks result in lower latency and allow better load balancing when clips are very short. Larger blocks result in higher latency but utilize disks more efficiently, improving throughput. Striping for video servers is discussed in [BMG$^+$94], [Che94] and [BBD$^+$96].

Replication is a technique for improving latency and throughput. A replicated server stores multiple copies of each video clip in different sets of disks. This improves latency because a new request for a clip is more likely to find a disk with free bandwidth and a copy of the first block of the requested clip. In a News on Demand server, replication is essential for throughput because clips are so short that striping may not span all disks in the server. Replication in combination with striping is discussed in [Che94].

3 Random Striping

Random striping is a scheme that combines replication and striping to randomly lay out video data on disks. In this section we describe this layout and present an algorithm that assigns requests to disks in a way that obeys the bandwidth constraints of each disk. The following table defines the parameters that characterize the disks, clips, degree of replication and the maximum load on the system.

d	number of replicas of each clip.
F	number of blocks that must be stored in the system.
S	number of block requests in a round.
N_c	maximum number of blocks that can be stored on each disk.
S_c	maximum number of blocks served by a disk in a round.

3.1 The Data Layout

We make d copies of each block and randomly assign all the copies to $\frac{dF}{N_c}$ disks as follows: we number the copies (there are totally dF of them) and the "block slots" available in all disks (each disk stores N_c blocks so there are $\frac{dF}{N_c} \times N_c = dF$ block slots available) and then choose a one-to-one assignment of copies to slots at random. Note that consecutive blocks of any single clip are placed completely independent of each other.

We represent the above layout as a bipartite multi-graph $G = (A, B, E)$, where A is the set of F blocks, B is the set of $(dF)/N_c$ disks, and E denotes the set of edges between A and B. Each block in A is connected to exactly those disks in B where its d copies are stored. Note that in this graph the vertices in A have degree d whereas all vertices in B have degree N_c. G is a multi-graph because two copies of the same block may be assigned to the same disk.

3.2 Assigning Requests to Disks

We now describe how the server assigns requests for blocks to the appropriate disks in each round. Finding an assignment of requests to disks is not trivial, because each block request should be assigned to a disk containing one of its copies and no disk should be assigned more requests than it can serve in a round(S_c).

Our algorithm for assigning requests to disks treats all requests identically without distinction between new requests and requests which have already started. At each round, each of the S users requests a block, which results in a set $X \subseteq A$ of at most S blocks[1] that the server must supply. We define a bipartite subgraph $H = (X, Y, E')$ associated with X. The set Y consists of $\frac{dF}{N_c} \cdot S_c$ vertices obtained

[1]It is fewer than S blocks if two users arrived in the same round and requested the same clip.

by replicating each vertex in B exactly S_c times. As before, a vertex x of X is connected to a vertex y of Y if and only if one of the copies of the block x is stored in the disk associated with y.

At each round, we compute the graph H and run the bipartite matching algorithm on this graph to maximally match the vertices in X. This assigns each matched vertex in X to a *unique* vertex in Y. Since each disk is represented by S_c vertices in Y this matching assigns no more than S_c requests to each disk. If we match every vertex in X, then we have an assignment of all the requests to disks such that at most S_c blocks are assigned to each disk, so it will be possible to serve all blocks requests in this round without hiccups. Hiccups only occur when H does not have a matching that includes all vertices in X. In this case, we satisfy those requests that are matched and all the viewers with unmatched requests skip their blocks.

3.3 Properties of Random Striping

Random striping ensures that all requests are satisfied with minimal latency but exposes requests to the possibility of hiccups. Requests are satisfied immediately because when a request arrives, the server will include a block request for it that will get matched in the next round. On the other hand, there are no *a priori* guarantees that all blocks will be matched in each round, so a viewer may suffer hiccups whenever the block it requested does not get matched.

4 Round Robin Striping

Most movie server prototypes use a variant of round robin striping to lay out video data on disks and serve requests. This section describes a version of robin striping and shows how data is laid out, and how the server assigns requests to disks.

4.1 Description of Round Robin Striping

First, we describe how round robin lays out data on disks. We describe *wide* striping, where each clip is striped across all disks in the server. Suppose that we make d copies of each clip. Let us order the disks arbitrarily and number them consecutively. The first block of each copy of any clip is placed at an arbitrarily chosen disk (*e.g.* at random) and the remaining blocks of that copy, in order, are placed in consecutive disks (with wrap-around if necessary).

We assign requests for the blocks to disks as follows: when a new request arrives, the server notifies the least loaded disk which contains the first block of the new request that it should serve that block at the start of the next round. For each clip already playing, once a disk has served some block of that clip in a round, the next disk (with wrap-around if necessary) will serve the next block (if any) in the next round. Whenever a disk receives more requests for blocks than it can serve, it gives priority to clips which are already playing and delays serving the first block to a new request until a subsequent round. A new request may queue up for multiple rounds at the disk which has been assigned to it until the disk has free bandwidth to serve it. Therefore, random robin exposes requests to latency.

4.2 Properties of Round Robin Striping

Prioritizing old requests guarantees that viewers never suffer hiccups. This is because all the requests move from disk to disk in the same direction so, at the beginning of each round, each disk will be able to serve all the old requests which were served by the previous disk in the previous round.

In round robin striping, requests are susceptible to latency when the system is heavily loaded. Under heavy load, each disk is likely to be serving many requests during a round. When a new request arrives, it is likely to be delayed by the old requests that are passing through that disk in that round. Furthermore, when the new request starts, it will join the tail of the convoy of requests that initially held it up, worsening the situation for the next new request. This effect is exacerbated when a few clips are very popular, and when storing short clips because a lot of requests enter the system at each round, and all these requests descend on a small set of disks.

5 Experimental Evaluation of Striping Strategies for News on Demand

In this section we compare both striping strategies for a news on demand server. We describe our simulations, show our results, and discuss the relative merits of round robin and random striping.

5.1 News on Demand Workload

We simulated a 50 disk server with 1, 2, 3, and 4 copies of 100 clips. We assumed that each disk can serve 5 seconds of video in a one second round. These values correspond to the transfer rate of a current disk drive and a display rate of 6.0 Mbs. Each clip lasts 30 seconds, and the request pattern obeys a Zipf [Knu73] distribution. In both striping schemes, we divided each clip into blocks of one second of video. Thus $S_c = 5$, $F = 3000$, d varies, and N_c is sufficiently large.

We chose these parameters (number of clips, disks and requests) to reduce simulation time. Simulating one round with random striping requires that we compute a matching on a bipartite graph which takes about half a second in a SGI workstation so simulating a long run of the server requires in the order of an hour of computing time.

5.2 Experimental Methodology

We subjected the server to varying loads; for each load we fixed the number of initial requests in the system and replaced each request that finished with a new request. To prevent requests from moving in lockstep through the system our simulation waits a random interval between 1 and 5 rounds before a new request starts. Therefore for each load level, the maximum and average load differ slightly. The first two rows of Figure 1 show the maximum and average loads we placed on the server for both striping schemes. We allowed the simulations to warm up for 600 rounds, and then we took our measurements for the next 600 requests.

For a server using random striping, our goal was to evaluate the number of hiccups as we increased the load for varying number of copies of each clip. We counted hiccups by flagging all the unmatched vertices. For simplicity, we assumed that when a request suffers a hiccup it skips the current block and requests the next block in the next round.

For a server using round robin striping we measured the response time of all requests. For each load, we measured the maximum response time experienced by 90 and 95 percent of the requests.

5.3 Performance of Random Striping

In this section, we present our simulation results for random striping. Figure 1 shows the result of our experiments: with as few as two copies the number of hiccups is zero for most loads and negligible at maximum load. The first two rows of the figure show the increasing load on the server. The first row shows each value of the load while the second row shows the average number of requests for that load. For each d (the number of copies in the server) the figure shows the number of hiccups, the number of blocks that were requested, the number of requests served and the number of requests that suffered a hiccup.

Maximum Number of Requests	125	150	175	200	225	250
Average Number of Requests	118	141	165	188	212	236
One Copy						
Blocks Requested	175709	210953	246026	281292	316337	351423
Hiccups (Blocks Not Served)	5051	10190	18398	28562	42557	58290
Total Requests	5867	7031	8211	9388	10548	11724
Requests w. Hiccups	2130	3324	4403	5562	6717	7844
Two Copies						
Blocks Requested	175820	210900	246052	281251	316263	351343
Hiccups (Blocks Not Served)	0	0	0	0	0	9
Total Requests	5862	7038	8213	9385	10555	11719
Requests w. Hiccups	0	0	0	0	0	9
Three Copies						
Blocks Requested	175835	210800	246137	281143	316185	351356
Hiccups (Blocks Not Served)	0	0	0	0	0	0
Total Requests	5868	7034	8222	9381	10553	11726
Requests w. Hiccups	0	0	0	0	0	0

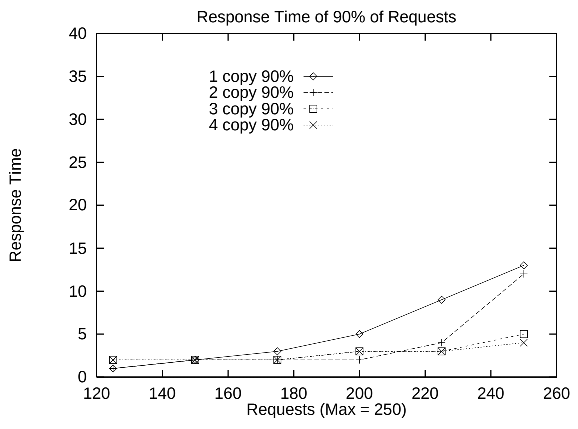
Figure 1: This figure shows the number of blocks served, the number of hiccups, the number of requests processed by the server, and the number of requests affected by hiccups. All requests experience a response time of at most one round. The figure shows that random striping with a single copy suffers a large number of hiccups with with at least two copies the number of hiccups becomes negligible.

From Figure 1 we see that with a single copy the server suffers an enormous fraction of hiccups at any load. However, once we place at least 2 copies the matching algorithm succeeds in assigning virtually all block requests to disks and the simulation registers almost no hiccups.

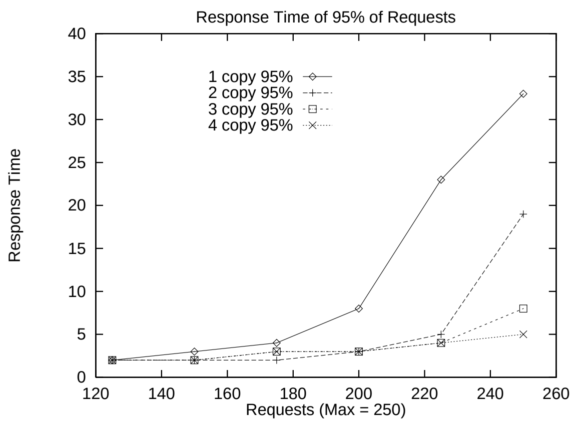
5.4 Performance of Round Robin Striping

In this section we present the result of our experimental evaluation of round robin striping.

Figure 2 shows that for the 90 and 95 percentile of requests, striping 1 or 2 copies yields relatively large response times while striping 3 or 4 copies results in relatively small response times. Figure 2



(a)



(b)

Figure 2: This figure shows the maximum response time number of 90 and 95 percent of requests for a server with 1,2,3 and 4 copies of each clip. Figure (a) shows response time for 90 percent and (b) shows the response time for 95 percent of requests. These graphs show that latency is significant at high load and that replication helps reduce latency. The labels on the horizontal axis correspond to the maximum load values. The corresponding average load is in the second row of Figure 1

show that striping a single copy of each clip results in large response times for the 90 and 95 percentile of the requests. Adding one extra copy improves the response times somewhat but response times do not drop below 5 seconds even when the server uses 3 or 4 copies of each clip.

6 Theoretical Conditions for Hiccup-Free Random Striping

In this section we show that random striping with a sufficiently large number of copies can satisfy any load arbitrarily close to the maximum server bandwidth ($(dF/N_c) \cdot S_c$) without any hiccups. Our main result on random striping is the following: let F, N_c, S_c and S be inputs, and let d be the smallest number of copies that satisfies the following inequality:

$$\left(\frac{N_c S}{d S_c F}\right)^{d-\frac{1}{S_c}} \left(e^{\frac{1}{S_c}+1}\right) \left(\frac{F}{S}\right) < \frac{1}{4} \tag{1}$$

Then, if we randomly stripe with d copies, with probability $1 - O(1/F)$, the resulting layout produces no hiccups for *any* set of S requests. In other words, this random layout has the property that *all* sets of S block requests can assigned to disks such that no disk serves no more than S_c requests. This ensures that the matching algorithm always succeeds. The proof of this result appears in the appendix, where we consider a more general problem that has potential uses in other applications.

Observe that in the expression in the left hand side of Equation 1, if we choose d to be large enough, then the only significant term that determines the value of the expression is $\frac{N_c S}{d S_c F}$. Rewriting this expression as $\frac{S}{(dF/N_c)\cdot S_c}$, we can see that for any S arbitrarily close to the maximum server bandwidth ($(dF/N_c) \cdot S_c$), we can d large enough to satisfy Equation 1.

It is important to understand the role of randomness in this result. If we assign d copies of each block to disks at random, we have a very high probability of getting a layout that will *never* produce any hiccups. We make no assumption about the distribution of requests; the probability in this result is about finding good layouts. Our result shows that good layouts are plentiful, and once we find one, it will be able to satisfy all sets of requests.

Once the layout is chosen, it never changes. Although we can not efficiently verify that our layout is good (this is co-NP hard [BKV$^+$81]), we will know it is not good if a set of requests cannot be assigned to disks without creating a bottleneck. If this happens we can choose another random layout. While these re–assignment operations are expensive, they only happen with very low probability ($O(1/F)$).

6.1 Evaluation of Theoretical Estimates

In this section we evaluate our theoretical result by calculating the maximum load that is guaranteed to be free of hiccups using the same parameters as our previous simulation on a News on Demand server. Then we argue that the values we obtained are too conservative because of our theoretical analysis.

We computed the maximum load that we support using the parameters (d, F, S_c and N_c) of our round robin simulation and calculating the maximum S that satisfies Equation 1. Figure 3 shows the maximum load as a function of the number of copies for a 50 disk server with 100 clips of 30 seconds each. The number in the second column is the maximum load that the server can support with *no hiccups*, *minimal latency* (i.e. at most one round), under *any distribution* of requests.

Copies	Maximum Theoretical Load
2	8
3	56
4	96
5	123
6	143
7	157
8	168
9	177

Figure 3: This figure shows the maximum load guaranteed to be hiccup free by the theoretical result. These requests will have 1 round latency and no hiccups under any distribution of requests. The server has bandwidth for 250 requests. We believe these values are disappointingly low due to the conservative analysis.

Given that the maximum load the server can support is 250, the numbers in Figure 3 are surprisingly low. We believe this is the result of our conservative analysis. We expect that a different proof technique or a tighter analysis can show that the server can support a higher load without hiccups.

To validate our intuition that these values were a result of excessively conservative analysis we conducted the following simple experiment. We simulated a server with 5000 blocks replicated d times on $50 \times d$ disks. Our goal was to evaluate how many blocks we could consistently retrieve in each round without suffering any hiccups. Each disk can serve 5 blocks. We derived the theoretical maximum load using Equation 1 for values of d ranging from 2 to 9. We also computed the maximum number of blocks retrieved with the following experiment: for each value of d we randomly assigned $5000 \times d$ copies to $50 \times d$ disks. For each value of d we estimated the maximum hiccup free load as follows: we chose a starting value of S and retrieved S blocks chosen uniformly at random. Whenever the match failed to retrieve all S blocks we decreased S to the number of blocks it matched. When S did not change after 500 rounds matching S items, we declared it a maximum load. We show these values of S in the graph in Figure 4.

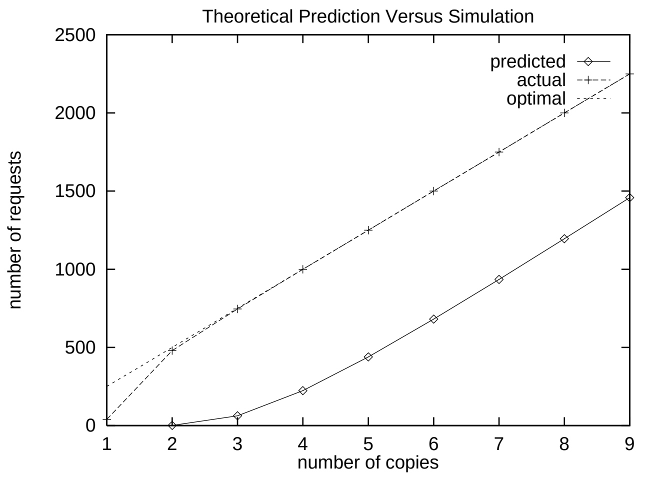


Figure 4: This graph shows the optimal load (labeled "optimal"), the maximum load estimated by our experiment (labeled "actual"), and the maximum hiccup free load predicted by our result (labeled "predicted"). The vertical axis shows the maximum number of blocks that can be drawn without hiccups. The horizontal axis shows d, the number of replicas of each block (the number of disks increases linearly with d in this example).

The graph shows that for two or more copies, the estimate of maximum load is close to the maximum server bandwidth and much larger than the load predicted by the result.

7 Related Work

Video servers have received a lot of attention in the last few years and several studies have touched on the performance of round robin striping for video servers ([BMG$^+$94], [ORS96], [TDM$^+$95],[Che94], [CPK95], [BBD$^+$96]).

Several authors have identified and proposed schemes to address high latency in round robin striping. Chervenak *et al.* ([Che94],[CPK95]) studied round robin striping on disks with a Zipf workload and found that for large disks arrays a significant fraction of requests experienced high latency. They showed that striping 2 copies of each movie reduced latency but that replication in proportion to movie popularity was essential to almost eliminate latency. Their solution depends on *a priori* knowledge of an unchanging request distribution. Ghandeharizadeh *et al.* [GK95b] developed an analytical model to estimate the expected startup latency of a request as a function of load and server configuration. They propose replication and request migration to reduce latency. Bolosky *et al.* [BBD$^+$96] describe an implementation of a server using round robin striping and mention that latency in striped servers is analogous to insertion in a open hash table with sequential chaining. They propose using bounds on hashing to determine the maximum load a server can handle before exceeding a threshold average latency.

We are not aware of any available studies of random striping. In fact, we are not aware of previous work using bipartite matching to assign requests to servers. Tewari *et al.* mention in [TMD$^+$95] and [TDM$^+$95] an IBM Research report that compares the performance of random striping to round robin striping. Tewari informed us that the report is not publicly available. From the brief discussion of that report in [TMD$^+$95] it does not appear that it considered random striping with replication. Matching is only necessary if there are multiple copies of each block.

Several studies ([DKS95]) consider other techniques that can be used in addition to striping to reduce latency. Most of these techniques are dynamic, such as replicating popular items to adapt to high demand, moving requests to lightly loaded devices during playback, and caching popular items in memory.

Several studies ([Mou96], [BBD$^+$96]) have stated that striping makes video servers more susceptible to disk failures and have considered replication for fault tolerance. We have not studied the fault tolerant properties of random striping.

7.1 Summary and Conclusion

Most disk–based movie servers use round robin striping because it offers high throughput and strong real time guarantees. Unfortunately, round robin striping can produce high latency for an application like News on Demand where clips are short and most requests access only a few very popular items. Although striping *with* replication alleviates this problem, it does not eliminate it altogether.

Random striping is a less known striping technique that guarantees minimal latency. However, because of the random nature of the assignment of data to disks, it is possible that the server will suffer hiccups.

Our main practical contribution is an experimental evaluation of random striping and round robin striping for a News on Demand application. We show that, under heavy load, round robin striping produces high latency for a significant fraction of all requests. In contrast, we show that random striping not only provides minimal latency but suffers no hiccups under a wide variety of loads. At the highest load we simulated, random striping with two copies suffers a negligible number of hiccups. With three copies, random striping suffers no hiccups even at our maximum load.

Our main theoretical contribution is a result that gives conditions for configuring a random striped server which guarantee no hiccups under any request distribution. Our conditions require at least two copies of each clip, and the number of requests that can be served with strong guarantees increases with the number of copies.

Unfortunately, our theoretical analysis is too conservative, and its estimates for when a server never has hiccups are too low (although a fixed fraction of the full server bandwidth). We believe this to be the result of our analysis rather than an intrinsic property of the scheme. To test this hypothesis we ran a little experiment to test the maximum load it could serve for 500 rounds without a hiccup. We found that the load was very close to the maximum load for 2 copies of each block and extremely close for 3 or more copies.

We believe random striping with a few copies of each clip is a simple layout strategy that offers high throughput and minimal latency for all request distributions. Our theoretical result shows that it suffers no hiccups when lightly loaded and our experiments suggests it suffers no hiccups at all but the highest load.

References

[BMG$^+$94] Berson S., Muntz R., Ghandeharizadeh S., Xiangyu Ju, "Staggered Striping in multimedia information systems" SIGMOD Record, Vol. 23, Number 2, pp. 79–90 June 1994.

[Mou96] Antoine Mourad, "Doubly–Striped Disk Mirroring: Reliable Storage for Video Servers" *Multimedia Tools and Applications* 2, 273-297 (1996), Kluwer Academic Publishers.

[TDM$^+$95] Renu Tewari, Dan Dias, Rajat Mukherjee, and Harrick Vin, "High Availability in Clustered Video Servers" *IBM Research Report RC-20108* , June 1995.

[TMD$^+$95] Renu Tewari, Rajat Mukherjee, Dan Dias, and Harrick Vin "Design and Performance Trade-offs in Clustered Multimedia Servers" *Proceedings of IEEE-ICMCS*,Tokyo, June 1996.

[ORS96] Ozden B., Rastogi R., and Silberschatz A., "Disk striping in video server environments", *Proceedings of the International Conference on Multimedia Computing and Systems*, IEEE Comput. Soc. Press. Los Alamitos, CA, USA. 1996.

[BKV$^+$81] M. Blum, R.M. Karp, O. Vornberger, C.H. Papdimitriou, and M. Yannakakis "The Complexity of Testing Whether a Graph is a Superconcentrator" *Information Processing Letters*, 13:164-167, 1981.

[BBD$^+$96] William J. Bolosky, Joseph S. Barrera, III, Richard P. Draves, Robert P. Fitzgerald, Garth A. Gibson, Michael B. Jones, Steven P. Levi, Nathan P. Myhrvold, Richard F. Rashid. "The Tiger Video Fileserver", Proceedings of the Sixth International Workshop on Network and Operating System Support for Digital Audio and Video. IEEE Computer Society, Zushi, Japan, April, 1996.

[Che94] Ann L. Chervenak, "Tertiary Storage: An Evaluation of New Applications" PhD Thesis, University of California at Berkeley Technical Report UDB/CSD 94/847, December, 1994.

[CPK95] Ann L. Chervenak, David A. Patterson, and Randy H. Katz, "Storage Systems for Movies–on–Demand Video Servers" *Proceedings of the Fourteenth IEEE Symposium on Mass Storage Systems*, Monterey, CA, USA. pp. 246-56. IEEE Comput. Soc. Tech. Committee on Mass Storage Syst. 11-14 Sept. 1995.

[DKS95] Asit Dan, Martin Kienzle, Dinkar Sitaram, "A dynamic policy of segment replication for load–balancing in video–on–demand servers" *Multimedia Systems*, Vol.3, Number 3. pp. 93-103. July 1995

[GK95a] Shahram Ghandeharizadeh, Seon Ho Kim, "Striping in multi-disk video servers." Proceedings of the SPIE - The International Society for Optical Engineering. vol.2604. pp. 88-102. 1996. Conference on High-Density Data Recording and Retrieval Technologies, Philadelphia, PA, USA. SPIE. 23-24 Oct. 1995.

[GK95b] Shahram Ghandeharizadeh and Seon Ho Kim "An Analysis of Striping in Scalable Multi–Disk Video Servers", University of Southern California Technical Report 95-623, 1995. http://www.usc.edu/dept/cs/technical_reports.html

[Knu73] Donald E. Knuth, "The Art of Computer Programming Vol. III" Addison Wesley Publishing Company, 1973

8 Appendix

Consider the following static model: we are given a collection of disks each of which can store N_c blocks and can serve any of $S_c < N_c$ blocks that it stores.[2] Suppose that there are a total of F blocks and $S = \alpha F$ users, for some α, who can each request anyone of the F blocks. We assume that there are no more than k requests to any single block. (For the News on Demand problem, $k = 1$ since it is sufficient to just serve a single copy of each block even though there may be multiple requests to that block.)

By way of motivation, consider the problem of obtaining a general layout scheme that uses replication and has the required property that all sets of S requests can be satisfied immediately. We can obtain some crude estimates on the minimum number of disks C that is required by any such scheme. Since F/N_c is the minimum number of disks required to store at least one copy of each block and S/S_c is the minimum number of disks required to serve S distinct blocks,overcome the bandwidth constraint, it is easy to see that $C \geq \max\{S/S_c, F/N_c\}$. If we restrict the accesses so all blocks requested are distinct ($k = 1$ in our workload model) we can solve this problem with F/S_c disks. In this case the solution is to store only S_c items in each disk. Another solution is to make S/S_c copies of each block, for a total of FS/N_cS_c disks, and assign them to disks such that no two copies of a block are in the same disk. Thus, we have $C \leq \min\{F/S_c, FS/N_cS_c\}$. The drawback in the first scheme is that wastes $N_c - S_c$ spaces in each disk. The problem with the second is that the number of copies grows linearly with the user population.

To minimize the cost of the solution, we want to make as few replicas as possible. Unfortunately, for arbitrary F, S, k, S_c and N_c it is not easy to determine which is the minimum number of copies necessary. Even worse, even if we guess the necessary number of copies, it is not clear which assignment of copies to disks has the property that any set of requests can be immediately satisfied.

The main problem addressed in this section is the following: given F, S, k, S_c, N_c, what is the smallest value of d such that if we replicate each block d times and distribute all the resulting blocks among dF/N_c disks at random,[3] then *any* set of at most S blocks can be accessed *immediately*. By "immediately" we mean that for any set of requests, each request is assigned to a disk in such a way that no disk is assigned more than S_c requests (so that no request has to wait for another request to complete).

The algorithm for assigning requests to disks is a slight modification to the algorithm used in Section 3. The graph G is the same as before, but in the graph H, we add as many vertices for each block as the number of times it appears in the request set. Formally, for a (multi)set of at most S requests where each block does not appear more than k times, construct the bipartite graph $H = (X, Y, E')$ as follows:

[2]A block may be served many times as long as the total number of blocks that is served including duplicates is at most S_c.

[3]This random placement can be achieved by numbering the copies of each block (for a total of dF copies) and numbering the "block slots" available in all disks (each disk stores N_c blocks so there are $\frac{dF}{N_c} \times N_c = dF$ block slots available) and then choosing a one-to-one assignment of copies to slots at random.

- For each block represented by some node $v \in L$ that appears j times in the request set, for some $j \leq k$, add j new vertices $v_1, \ldots, v_j$ to X.
- For each disk represented by some node $u \in R$, add S_c new vertices $u_1, \ldots, u_{S_c}$ to Y.
- As before a vertex x of X is connected to a vertex y of Y if and only if the block associated with x is stored in the disk associated with y.

Now construct a matching on this graph that matches every vertex in X. This matching gives an assignment of blocks to disks such that each disk serves no more than S_c blocks.

We will prove below that the matching always exists for the parameters stated above with an appropriate choice of S. This value of S is given by the largest value that satisfies the inequality of Equation 2, which we state below:

$$\left(\frac{N_c \beta S}{dF}\right)^{d-\beta k} \left(e^{\beta k+1}\right) \left(\frac{kF}{S}\right) < \frac{1}{4} \tag{2}$$

Note that for $k = 1$, this corresponds to the inequality in Equation 1 Consider the "bad" event that for some set of at most S requests there is no matching of the set X', $|X'| \leq S$, associated with these set of requests in the request graph H. By Hall's theorem, this implies that for some set $X' \subseteq X$, the size of the neighborhood set $N(X') \subseteq Y$ of X in H must be less than than $|X|$.

Let $U \subseteq L$ be the set of nodes (blocks) in G associated with the nodes in X' and let $\hat{V} \subseteq R$ be the set of nodes (disks) associated with the nodes in $N(X')$. Observe that $N(U) = \hat{V}$ in G, $|X'|/k \leq |U| \leq S$ and $|\hat{V}| = \beta|N(X')|$, where $\beta = 1/S_c$. Since $|N(X')| < |X'|$, we can infer that

$$|\hat{V}| < \frac{1}{S_c}|X'| \leq \beta \cdot \min\{k|U|, S\}.$$

Given such a $\hat{V}$, we can always obtain a V by adding dummy nodes to $\hat{V}$, so that $|V| = \beta \cdot \min\{k|U|, S\}$ and $N(U) \subseteq V$.

Summarizing the observations above, the probability that the layout is bad is bounded by the probability that there is some $U \subseteq L$ and some $V \subseteq R$, where $|U| \leq S$ and $|V| = \beta \min\{k|U|, S\}$, such that $N(U)$ is contained in V. We will bound this latter quantity by estimating for an arbitrary U and V, the probability that $N(U) \subseteq V$ and then summing up over all choices of U and V satisfying these constraints.

Set $|U| = u$ and $|V| = v$. For constructing G, we chose a random 1–1 assignment that connected d edges from each vertex in L to the c "block slots" of each vertex in R. The probability of $N(U) \subseteq V$ is the number of assignments that map all du edges originating from U to the $N_c v$ "edge slots" of all vertices in V, divided by the total number of assignments. This is $\binom{N_c v}{du}(du)!(dF - du)!$ divided by $(dF)!$, which can be simplified to $\binom{N_c v}{du} / \binom{dF}{du}$.

Summing up over all choices of the set U and the set V, we have that the probability of a bad layout is bounded by

$$\sum_{1 \leq u \leq S, v = \beta \cdot \min\{ku, S\}} \binom{F}{u} \binom{dF/N_c}{v} \frac{\binom{N_c v}{du}}{\binom{dF}{du}}.$$

In order to show this probability is small, we will simplify and bound the above expression. We use the standard inequalities for binomial coefficients: $\binom{n}{r} \leq (\frac{ne}{r})^r$ and for $m < n$, $\binom{m}{r} / \binom{n}{r} \leq (\frac{m}{n})^r$. Applying this in the equation above, we have,

$$\sum_{1 \leq u \leq S, v=\beta \cdot \min\{ku, S\}} \left(\frac{Fe}{u}\right)^u \left(\frac{dFe}{N_c v}\right)^v \left(\frac{N_c v}{dF}\right)^{du}. \tag{3}$$

We will split this sum into two main parts: for the first part, $1 \leq u \leq S/k$ which implies that $v = \beta ku$; for the second part $S/k < u \leq S$ whence $v = \beta S$. We will now show that each part can be bounded to a small quantity.

Case 1 **(**$1 \leq u \leq S/k$**):**

Substituting $v = \beta ku$ in Equation 3, we obtain

$$\left(\frac{Fe}{u}\right)^u \left(\frac{dFe}{N_c \beta ku}\right)^{\beta ku} \left(\frac{N_c \beta ku}{dF}\right)^{du} = \left[\left(\frac{u}{F}\right)^{d-\beta k-1} \left(e^{\beta k+1}\right) \left(\frac{N_c \beta k}{d}\right)^{d-\beta k}\right]^u. \tag{4}$$

We will now use the constraint on d as given by Equation 2 to simplify the above expression. First, we re-arrange Equation 2 to derive the following inequality:

$$\left(\frac{N_c \beta k}{d}\right)^{d-\beta k} < \frac{1}{4}\left(e^{-\beta k-1}\right)\left(\frac{kF}{S}\right)^{d-\beta k-1} \tag{5}$$

If we substitute this inequality in Equation 4 we get

$$\left[\left(\frac{u}{F}\right)^{d-\beta k-1} \left(e^{\beta k+1}\right) \frac{1}{4} \left(e^{-\beta k-1}\right) \left(\frac{kF}{S}\right)^{d-\beta k-1}\right]^u = \left[\left(\frac{ku}{S}\right)^{d-\beta k-1} \frac{1}{4}\right]^u \tag{6}$$

Now, we can split the sum into three parts:

$$\left(\frac{k}{S}\right)^{d-\beta k-1} \frac{1}{4} + \sum_{u=2}^{\log F} \left[\left(\frac{ku}{S}\right)^{d-\beta k-1} \frac{1}{4}\right]^u + \sum_{u=\log F}^{S/k} \left[\left(\frac{ku}{S}\right)^{d-\beta k-1} \frac{1}{4}\right]^u \tag{7}$$

Recall that $d \geq 3$ and $S_c \geq 1$ means that $d - \beta - 1 \geq 1$. The first term of the sum is obviously $O(1/F)$. The second term is $O((\log F)(\frac{\log F}{F})^2)$ which is $O(1/F)$. The third term is $O(F \cdot (\frac{1}{4^{\log F}}) = O(F\frac{1}{F^2})$ which is $O(1/F)$.

Case 2 **(**$S/k < u \leq S$**)**

Here we substitute $v = \beta S$ in Equation 3 to obtain

$$\left(\frac{Fe}{u}\right)^u \left(\frac{dFe}{N_c \beta S}\right)^{\beta S} \left(\frac{N_c \beta S}{dF}\right)^{du} = \left(\frac{F}{u}\right)^u \left(e^{\beta S+u}\right) \left(\frac{N_c \beta S}{dF}\right)^{du-\beta S}$$

$$\leq \left(\left(\frac{N_c\beta S}{dF}\right)^{d-\beta k} \left(e^{\beta k+1}\right) \left(\frac{kF}{S}\right) \right)^u \left(\frac{N_c\beta S}{dF}\right)^{\beta k\left(u-\frac{S}{k}\right)}$$

$$\left[\text{Because } \left(\frac{F}{u}\right)^u \leq \left(\frac{kF}{S}\right)^u \text{ and } e^{\beta S} \leq e^{\beta k u}\right]$$

$$\leq \left(\frac{1}{4}\right)^u$$

Therefore the sum is at most $\sum_{u=\frac{S}{k}}^{S} \left(\frac{1}{4}\right)^u$ which is $O(1/F)$.

In conclusion, the probability the graph violates the conditions of the theorem is $O(1/F)$ and the probability that a randomly chosen graph satisfies the theorem is $1 - O(1/F)$.