

# I/O Data Prefetching based on Sequential Stream Recognition

## Abstract

This paper presents a cache prefetching algorithm for mid-size and large storage systems. A storage system sees physical block addresses of the submitted I/O requests but has no knowledge of the mapping between these physical blocks and their corresponding files. Since I/O requests for various files arrive in an interleaved fashion at a storage system, it is difficult for the storage controller to determine if some of these file accesses are sequential. The cache prefetching algorithm addresses this problem by keeping track of past I/O request addresses and identifying sequential file accesses when the address of an incoming I/O request adjoins that of a past I/O request address. Only identified sequential streams are prefetched into the storage system cache. Simulations results show that the prefetching algorithm improves storage system performance by increasing the read-ahead hit rate of an array cache while reducing the read-ahead traffic generated to the disks.

**Index Terms:** RAID, caches, prefetching, I/O workload, read-ahead hit rate, I/O performance evaluation, disk array.

## 1 Introduction

Storage systems must satisfy the capacity, availability, and retrieval requirements of a large number of data-intensive applications. In order to satisfy these requirements, most large and mid-size storage systems contain one or more disk arrays. In addition to multiple disks, disk arrays contain large array caches and sophisticated array controllers. Since array caches are an order of magnitude faster than the disks, the performance of a storage system can be improved significantly if I/O requests can be serviced directly from the cache. The presence of large array caches combined with sophisticated array controllers should imply that caching would have a significant impact on improving the performance of disk arrays. However, some recent studies suggest that array caches are not being effectively used [14, 15]. In this paper, we identify some of the reasons for inefficient use of disk array caches and then present an algorithm that uses this knowledge to improve the efficiency of disk array caches.

Caches, at all levels of the memory hierarchy, are much smaller and much faster than the next lower level storage device. A *cache hit* occurs if data that are needed are found in the cache, else a *cache miss* occurs and data must be accessed from the next lower level device. The reason that caches work is that data accesses typically follow the *locality-of-reference* principle. There are two types of locality: *temporal locality* refers to the observation that recently accessed data are likely to be accessed in the near

future; and *spatial locality* refers to the observation that data that are stored sequentially are likely to be accessed sequentially in time. To exploit temporal locality, a request that is referenced in the past should be kept in the cache long enough that a re-referencing of this request results in a *re-reference cache hit*. To exploit spatial locality, data close to currently accessed data should be prefetched into the cache from the next lower level device so that a sequential access results in a *read-ahead cache hit*.

The performance of a cache depends on the locality-of-reference of its workload. If the I/O workload does not exhibit locality-of-reference, then the cache size and caching policies can do little to improve the hit rate of the cache. The I/O workload is dependent on the file system workload and the performance of file system caches. The file system workload that results in file system cache misses are submitted to the disk array. If the file system caches are much smaller than storage caches, then re-referenced data that result in file system cache misses could result in disk array cache hits. However, current storage systems are typically shared by several computers as a result of which the disk array cache size is comparable to the file system caches above it. Hence, re-reference file system cache misses typically result in re-reference storage cache misses [15]. The I/O workload would also exhibit lower spatial locality than file system workloads since file system caches typically prefetch sequential stream data, resulting in random stream data requests being submitted more frequently to the storage system than sequential stream data requests. Thus, the I/O workload exhibits lower locality-of-reference than file system workloads.

Disk array caching algorithms that do not consider the impact of file system caches on the locality-of-reference of the I/O workload are inefficient [15]. A re-reference disk array algorithm that uses file system caches to improve the overall re-reference hit rate is presented by Wong and Wilkes [15]. In this paper, we present a disk array caching algorithm that improves the read-ahead hit rate of disk array caches. The *Sequential Stream Recognition (SSR)* algorithm developed here improves the read-ahead hit rate by identifying sequential I/O streams and only prefetching sequential stream data. A sequential I/O stream is defined to be one that accesses data from a file sequentially. A random I/O stream is defined to be one that accesses data from a file randomly. Identifying sequential I/O streams is difficult since the disk array controller receives no information regarding the correspondence between I/O requests and their corresponding files. Moreover, even if the I/O workload consists of several completely sequential streams, the disk array controller sees a random I/O workload since requests from the various sequential streams interleave. For example, Figure 1 shows three sequential streams accessing a disk array. The requests from these three streams interleave so that the disk array controller sees just one random stream of requests. The SSR algorithm allows the disk array controller to identify sequential I/O streams from the interleaved I/O workload, and this information is used to prefetch requests from sequential streams, thereby, allowing the caching algorithm to make the best use of the available array cache space. A simulation study shows that the algorithm improves the performance of a disk array by improving the read-ahead hit rate of the disk array without generating unnecessary traffic at the disks.

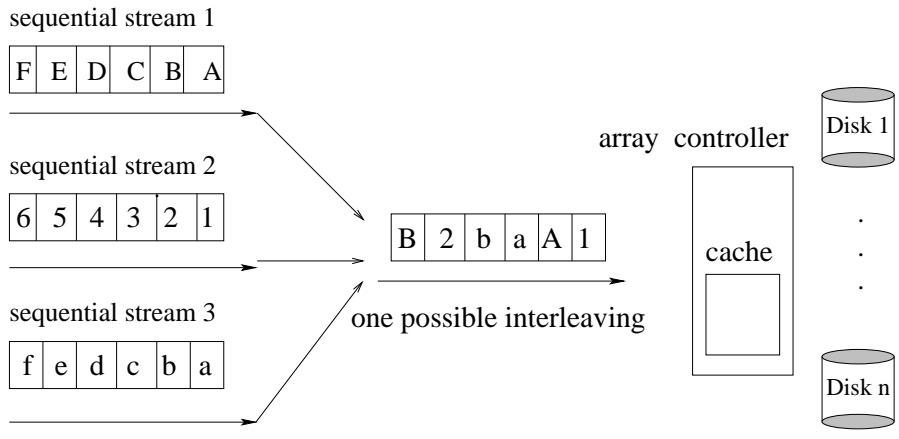


Figure 1: Interleaved sequential I/O streams

The rest of the the paper is organized as follows: Section 2 presents a survey of related papers on prefetching in operating systems and storage systems. Section 3 explains the SSR algorithm. Section 4 presents a comparison study of the SSR algorithm against a standard baseline prefetching algorithm. Section 5 presents the conclusion.

## 2 Related Work

Several papers deal with informed prefetching of files by the operating system. The papers differ in the techniques they use to determine prefetching information for a file. Smith [10] presents a probabilistic model to identify sequential data accesses in database systems. The prefetch size is determined by analyzing past database accesses and tracking the length of sequential runs of data accesses. Tait and Duchamp [12] determine prefetch size for file data by using a graph-based technique to record file access patterns. Griffioen and Appleton [3] present a probability graph technique to record file access patterns, and this information is used to determine whether a file should be prefetched when it is opened. Kroeger and Long [6] present a data compression technique to keep track of file system access patterns. Patterson, Gibson and Satyanarayanan [7] present an informed prefetching technique that depends on hints supplied by application developers.

The above papers deal with prefetching at the file system level. Storage systems typically do not have file system information and do not know the mapping between logical and physical blocks. Thus, storage prefetching decisions have to be based solely on physical block address information, a much harder problem. As a result, most papers on caching in storage systems analyze cache design and caching policies. An early paper by Smith [9] considers several design aspects of disk caching, namely, the capacity, cache segment size, location, and replacement policies for disk caches. Karedla, Love and Wherry [5] extended this work by studying the effects of different caching strategies and cache replacement algorithms. Neither of these papers focus on prefetching by the disk array controller.

Several papers develop performance models of storage systems that incorporate the effect of prefetching. Shriver, Merchant and Wilkes [8] present an analytic model of disks that incorporates read-ahead caches. Uysal, Alvarez and Merchant [13] present an analytic model of disk arrays under asynchronous workloads that incorporates array caches with read-ahead. Varki, Merchant, Xu, and Qiu [14] present an analytic model of disk arrays under synchronous workloads that isolates the effect of prefetching in disk arrays.

Compared to papers on prefetching at the file system level, there are fewer papers on prefetching at the storage system level. Valery [11] shows experimentally that disk level prefetching will perform as well as main memory prefetching if requests are queued at the disk controllers and if the storage system has knowledge of the requests to be prefetched. Kallahalla and Varman [4] present a prefetching and caching algorithm for multiple disk storage systems where the storage system has knowledge of the data to be prefetched. Enrique and Ricardo [2] present a prefetching technique that uses file system information to map between logical and physical blocks. This information is used to prefetch physically contiguous blocks that are also logically contiguous.

The above papers on storage system prefetching assume that the storage system has knowledge of mapping between logical and physical blocks and of the data to be prefetched. Most storage systems do not have any knowledge of the applications accessing them. The contribution of this paper is that the SSR algorithm presented here uses only information available at the storage system level to make decisions on data to be prefetched. The SSR algorithm uses physical block addresses of I/O requests to recognize sequential streams and prefetches only recognized sequential streams.

### 3 Sequential Stream Recognition Algorithm

The performance of a disk array can be improved if data that are needed in the near future are prefetched into the array cache. Unfortunately, a disk array controller has no information about the applications accessing storage data, and consequently, the controller can only predict future access by analyzing I/O data requests and recognizing sequential data streams. However, recognizing sequential data streams is a difficult task since I/O requests from various sequential streams interleave, so that the disk array controller sees a single random stream of I/O requests (refer to Figure 1). Here, we present the SSR algorithm that allows disk array controllers to recognize I/O sequential streams. A disk array controller can then use this information to prefetch only sequential stream data, thus improving the hit rate of the array cache and improving the overall performance of the disk array.

The algorithm divides the array cache into three logical caches: (a) a *tracking* cache to keep track of past request addresses in order to recognize sequential streams; (b) a *prefetch* cache to store prefetched data from recognized sequential streams; and (c) a *buffer* cache to store I/O request data from non-

recognized sequential streams and from random streams. When a request arrives at the disk array controller, the prefetch cache is searched first. If there is a hit in the prefetch cache then it is presumed that this request is part of a recognized sequential stream and in addition to transmitting the hit request to the host, the SSR algorithm generates a prefetch request for adjoining data. The amount of data prefetched depends on the prefetch size and the RAID organization of the disk array. For example, suppose a disk array is configured as a RAID 1/0 array with 8 disks where disks 2, 4, 6, 8 are mirrors of disks 1, 3, 5, 7 and suppose the prefetch size is equal to the request size. If a sequential request accesses data from disks 1 and 3, then adjoining data from disks 5 and 7 are also accessed and this prefetched data are loaded into the prefetch cache. Hence, upon a prefetch cache hit, the disk array controller not only transmits the requested data to the host but also generates disk I/Os for prefetch data.

If there is a miss in the prefetch cache, then it is presumed that this request is either from a random stream or from an unrecognized sequential stream. In this case, the tracking cache is searched for past disk I/O request addresses that are contiguous to this incoming request. The RAID configuration is required to determine if the incoming request is contiguous to past requests. For example, consider a disk array with the configuration given above. Suppose a request to disk 3 arrives, where the request address coincides with the start of a stripe unit. The SSR algorithm searches the tracking cache for a request address on disk 1 (or its mirror, disk 2) on the same stripe (row). If there is a hit in the tracking cache, this request is considered to be from a sequential stream and data for this stream are prefetched into the prefetch cache. When there is a miss in the tracking cache, this request is considered to be from a random stream (or the first request from an unrecognized sequential stream) and the request's data are fetched into the buffer cache. The address of this request is inserted into the tracking cache.

Keeping track of past disk I/O accesses could be expensive in terms of cache space. In the case of sequential concurrent streams, at least one request from each stream must be tracked. Once a request is recognized as a sequential stream request, this request stream can be removed from the tracking system. For example, if there are 3 sequential I/O streams accessing the disk array at a time, the first request from each stream would be stored in the tracking system. When the second request from a sequential stream arrives at the disk array, there would be a hit in the tracking system, thereby, allowing this request stream to be removed from the tracking system. In the case of random streams, all requests must be tracked since it is difficult to determine whether these requests are from a sequential stream with a low arrival rate or from a random stream. Hence, requests from random streams can fill up the sequential stream tracking cache quickly. The size of the tracking cache must be large so that requests from sequential streams reside in the cache until there is a hit. Setting aside limited disk array cache space for a large tracking cache would reduce the size of the prefetch cache, thus lowering the prefetch hit rate. However, a key observation regarding the tracking mechanism is that current request addresses have to be compared to past request addresses. Thus, the tracking cache stores only request addresses and not request data, so

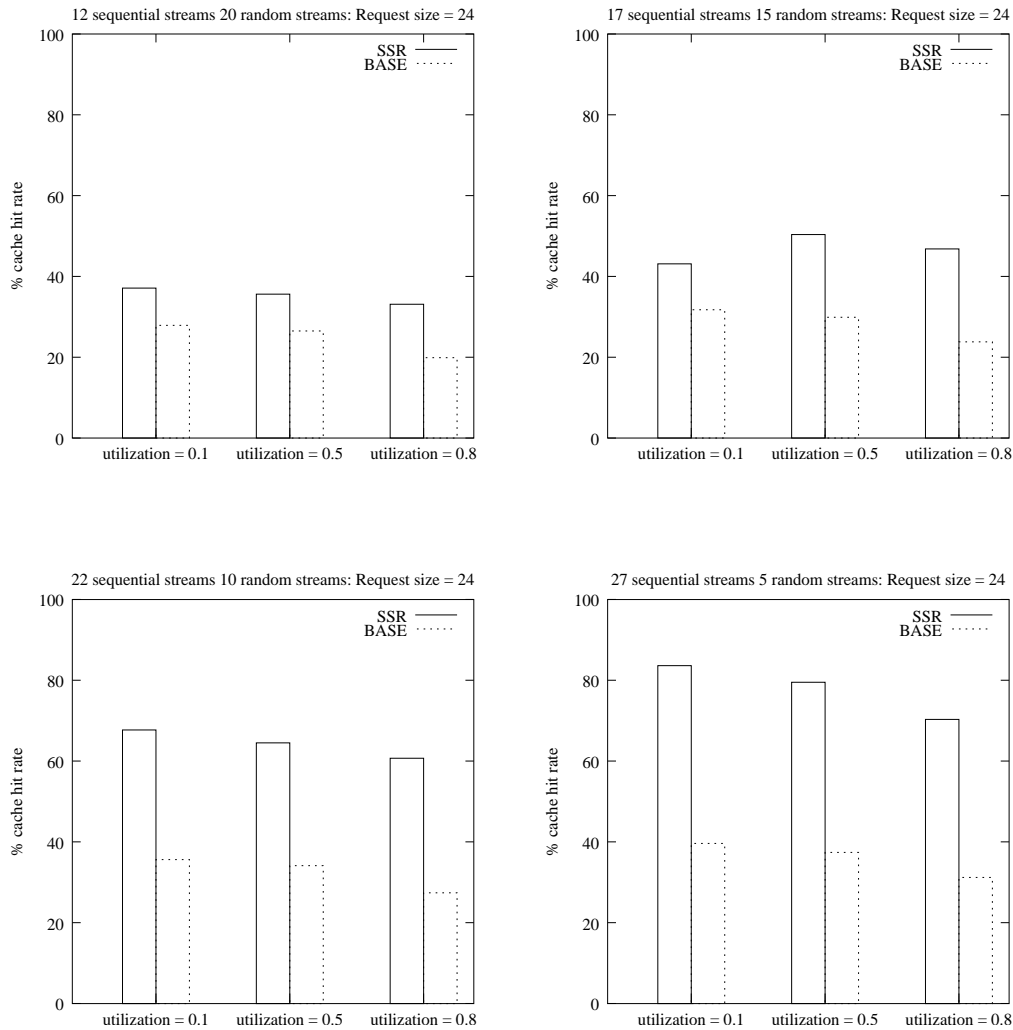


Figure 2: Disk array cache hit rates

this cache is really a virtual cache that does not take up the limited disk array cache space.

The sizes of the tracking cache, prefetch cache, and the buffer cache depends on the size of the disk array cache and its workload. Since the tracking cache is a virtual cache that does not use up any of the array cache space, its size is theoretically unlimited and depends on the designers' choice and the speed of the lookup function. The prefetch cache and the buffer cache sizes are determined by the size of the array cache. The prefetch cache is implemented using cache replacement schemes such as Least Recently Used (LRU), while the tracking cache is implemented using a simple cache replacement scheme such as First In First Out (FIFO).

## 4 Experimental Analysis of SSR

We implemented the SSR algorithm in the disksim [1] storage simulator. Disksim is a validated storage system simulator that simulates storage subsystem components like device drivers, buses, caches,

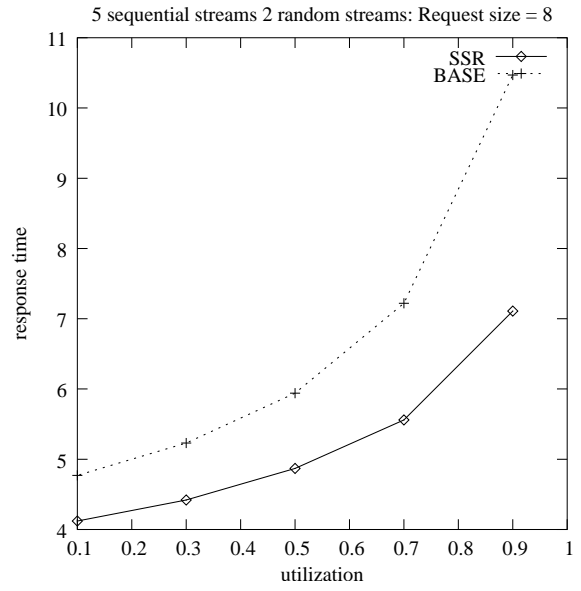
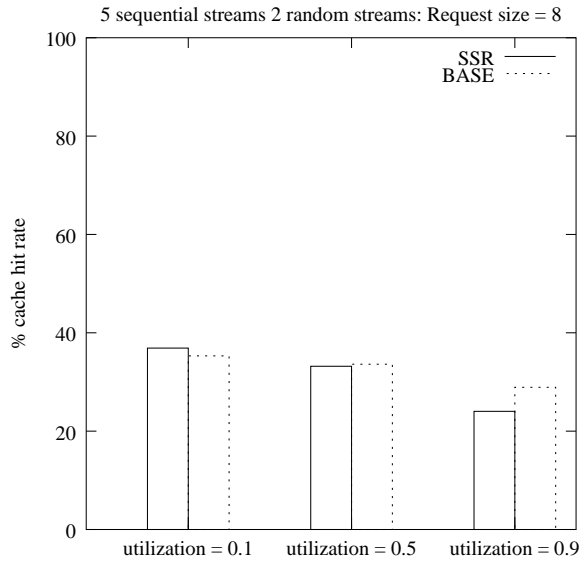


Figure 3: Hit rate versus disk array performance

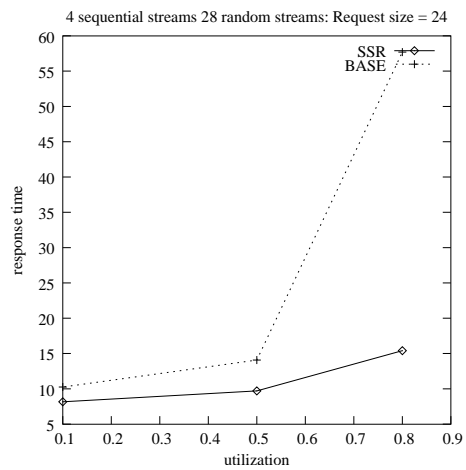
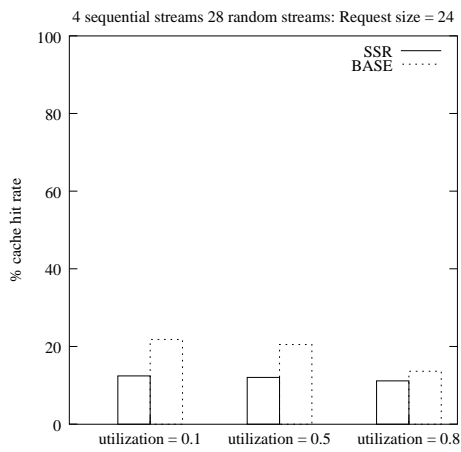
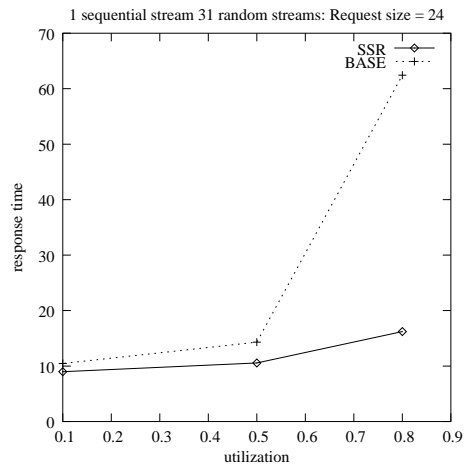
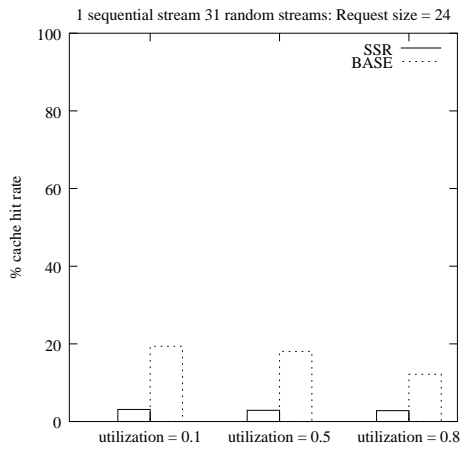


Figure 4: Impact of read-ahead disk traffic

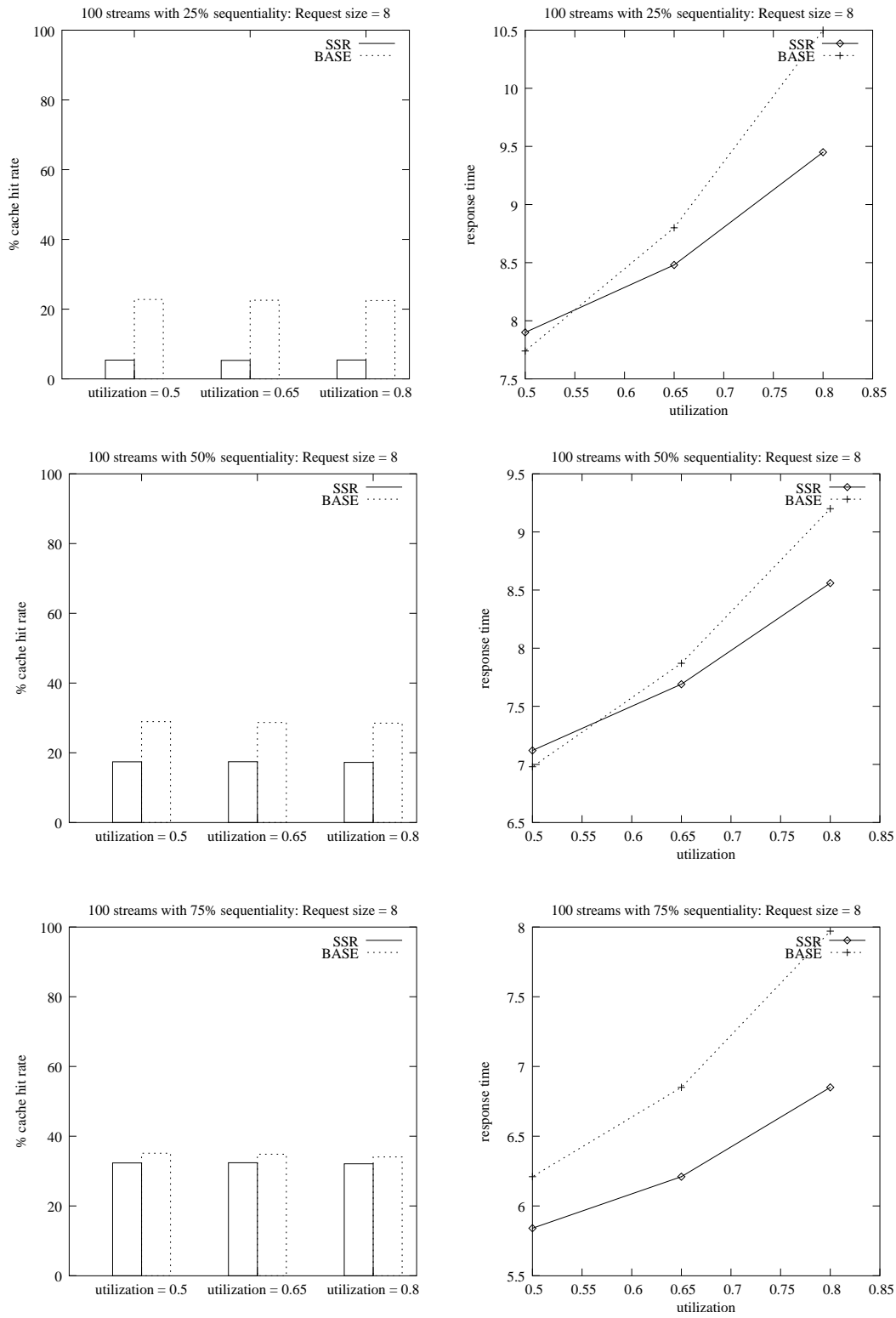


Figure 5: Performance of SSR vs BASE with partly sequential streams

Disksim parameter	Value
array controller cache size	8192 blocks
array controller cache line	16 blocks
array controller cache replacement policy	LRU
array controller cache transfer rate	49 MB/sec
tracking cache size	32768 virtual blocks
RAID organization	RAID 0
stripe unit size	16 blocks
number of disks	4
disk type	cheetah9LP
disk capacity	17783240 blocks
mean disk read seek time	5.4 msec
maximum disk read seek time	10.63 msec
disk revolutions per minute	10045 rpm
disk transfer rate	4.9 MB/sec

Table 1: Storage simulator setup

controllers, and disk drives. Table 1 presents the disksim parameters of significance to our study.

In order to analyze the performance of the SSR algorithm, we compare it against the performance of a disk array caching algorithm that prefetches every request, regardless of whether there is a hit or miss in the array cache. We refer to this baseline prefetching algorithm as the BASE algorithm. The SSR algorithm prefetches only recognized sequential streams while the BASE algorithm prefetches every stream. An identical disksim hardware platform is used when comparing the performance of the SSR algorithm against the performance of the BASE algorithm. That is, the size of the array cache, the cache line size, the prefetching size, the number of disks, the RAID organization, etc. are similar regardless of whether the SSR algorithm or the BASE algorithm is used. Disksim generated synthetic workloads are used to analyze the performance of the SSR algorithm. The workload consists of a mix of random and sequential data streams. Each simulation was run until the 95% confidence interval for each metric was less than 2% of the point value.

Figure 2 plots hit rates of the disk array cache for various numbers of sequential and random streams. In each case, SSR hit rates are better than that of the BASE algorithm which prefetches every request. Figure 3 plots both hit rates and response times for the disk array when there are 2 random streams and 5 sequential streams. In this case, the BASE algorithm, which prefetches every request has a higher hit rate than the SSR algorithm when the disk array utilization is high. Figure 4 plots both hit rates and response times for the disk array when there are few sequential streams and several random streams. In this case, the BASE algorithm has a higher hit rate than the SSR algorithm at all plotted disk array utilizations. The BASE algorithm prefetches each stream, and some of this prefetched data may result in random-reference hits.

While the hit rates for the BASE algorithm are higher, the disk array response times under the SSR algorithm are much lower than the disk array response times under the BASE algorithm. The reason for the superior response time performance of the disk array with the SSR caching algorithm is that lesser disk traffic is generated by the SSR algorithm than by the BASE algorithm. At high utilizations, the performance benefit from the higher cache hit rate does not offset the performance degradation by the increased traffic at the disks due to blind prefetching of every stream. Hence, these graphs highlight the cost incurred by algorithms that do not make smart prefetching decisions.

The above simulations consider I/O data streams that are completely sequential or completely random. Since the SSR algorithm only prefetches sequential streams, these simulation results show the performance when all prefetch decisions made by SSR are correct. If I/O streams are only partially sequential, with sequential runs interspersed by random data, then the SSR algorithm would prefetch data that may not result in read-ahead hits. Figure 5 compares the performance of SSR and BASE algorithms when the degree of sequentiality is varied in the I/O data streams. At low degrees of sequentiality, the hit rate of BASE is better since BASE's hit rates incorporate the effect of both random-reference and read-ahead hits. At higher degrees of sequentiality, the hit rates of BASE and SSR are similar since they start making similar prefetch decisions. The response times of the disk array with SSR are generally lower than the response times of the disk array with BASE when the disk array utilization is more than 50%. The higher BASE's response time values represent the cost incurred due to increased disk traffic generated by incorrect prefetch decisions.

## 5 Conclusion

The SSR algorithm is a smart cache prefetching algorithm designed for disk array systems. The prefetching algorithm identifies sequential streams by tracking past request addresses and only prefetches recognized sequential streams. Sequential I/O streams are recognized by using only the physical block addresses and the RAID architecture of the storage system. Thus, a key feature of the SSR algorithm is that it does not use any file system information to determine prefetch data. Simulation studies show that the SSR algorithm improves the read-ahead hit rate of a disk array system without generating unnecessary traffic to the disks.

In this study, the performance of the SSR algorithm is analyzed using synthetic workloads. In future work, we plan to analyze the performance of the SSR algorithm using real workloads. We plan to experimentally and analytically analyze the performance of disk arrays under the SSR algorithm to determine parameter values such as the optimal sizes of the tracking cache, the prefetch cache, and the buffer cache, and the optimal prefetch size. By comparing the performance of SSR against the BASE algorithm and other baseline algorithms (such as an algorithm that only prefetches when the disk array utilization is

low), we plan to analyze the the impact of prefetching/no-prefetching and the cost of wrong prefetch decisions.

## References

- [1] John S. Bucy and G. R. Ganger. The disksim simulation environment version 3.0 reference manual. Technical Report CMU-CS-03-102, Carnegie Mellon University, School of Computer Science, January 2003.
- [2] E. Carrera and R. Bianchini. Improving disk throughput in data-intensive servers. Technical Report DCS-TR-500, Rutgers University, September 2002.
- [3] Jim Griffioen and Randy Appleton. Reducing file system latency using a predictive approach. In *USENIX Summer*, pages 197–207, 1994.
- [4] Mahesh Kallahalla and Peter J. Varman. Optimal prefetching and caching for parallel i/o sytems. In *SPAA '01: Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 219–228, New York, NY, USA, 2001. ACM Press.
- [5] Ramakrishna Karedla, J. Spencer Love, and Bradley G. Wherry. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, 1994.
- [6] Thomas M. Kroeger and Darrell D. E. Long. Predicting file-system actions from prior events. In *USENIX Annual Technical Conference*, pages 319–328, 1996.
- [7] R. Hugo Patterson, Garth A. Gibson, and M. Satyanarayanan. A status report on research in transparent informed prefetching. *SIGOPS Oper. Syst. Rev.*, 27(2):21–34, 1993.
- [8] Elizabeth Shriver, Arif Merchant, and John Wilkes. An analytic behavior model for disk drives with read-ahead caches and request reordering. In *SIGMETRICS '98/PERFORMANCE '98: Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 182–191. ACM Press, 1998.
- [9] Alan J. Smith. Disk cachemiss ratio analysis and design considerations. *ACM Trans. Comput. Syst.*, 3(3):161–203, 1985.
- [10] Alan Jay Smith. Sequentiality and prefetching in database systems. *ACM Trans. Database Syst.*, 3(3):223–247, 1978.
- [11] Valery Soloviev. Prefetching in segmented disk cache for multi-disk systems. In *IOPADS '96: Proceedings of the fourth workshop on I/O in parallel and distributed systems*, pages 69–82. ACM Press, 1996.
- [12] C. D. Tait and D. Duchamp. Detection and exploitation of file working sets. In *Proceedings of the 11th International Conference on Distributed Computing Systems (ICDCS)*, pages 2–9, Washington, DC, 1991. IEEE Computer Society.
- [13] M. Uysal, G. Alvarez, and A. Merchant. A modular, analytical model for modern disk arrays. In *IEEE MASCOTS*, pages 183–193, Cincinnati, OH, August 2001.
- [14] Elizabeth Varki, Arif Merchant, Jianzhang Xu, and Xiaozhou Qiu. Issues and challenges in the performance analysis of real disk arrays. *IEEE Transactions on Parallel and Distributed Systems*, 15(6):559–574, 2004.
- [15] Theodore M. Wong and John Wilkes. My cache or yours? making storage more exclusive. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 161–175, Berkeley, CA, USA, 2002. USENIX Association.