Enhancing Shared RAID Performance Through online Profiling

Jiguang Wan[1*], Jibin Wang[2†], Yan Liu[3‡], Qing Yang[4§], Jianzong Wang[5†] and Changsheng Xie[6*]
[*]Wuhan National Laboratory for Optoelectronics,430074, China
[†]School of Computer Science, Huazhong University of Science and Technology, 430074, China
[‡]Department of Computer Science and Technology, Hua Qiao University, China
[§]Dept. of ECE, University of Rhode Island, Kingston, RI, 02881, USA
[1]jgwan@mail.hust.edu.cn, {[2]wangjibin, [3]yliuhust and [5]hanker.wong}@gmail.com, [4]qyang@ele.uri.edu
Corresponding Author: [6]cs_xie@mail.hust.edu.cn

***Abstract*—Enterprise storage systems are generally shared by multiple servers in a SAN environment. Our experiments as well as industry reports have shown that disk arrays show poor performance when multiple servers share one RAID due to resource contention as well as frequent disk head movements. We have studied IO performance characteristics of several shared storage settings of practical business operations. To avoid the IO contention, we propose a new dynamic data relocation technique on shared RAID storages, referred to as DROP, Dynamic data Relocation to Optimize Performance. DROP allocates/manages a group of cache data areas and relocates/drops the portion of hot data at a predefined sub array that is a physical partition on the top of the entire shared array. By analyzing profiling data to make each cache area owned by one server, we are able to determine optimal data relocation and partition of disks in the RAID to maximize large sequential block accesses on individual disks and at the same time maximize parallel accesses across disks in the array. As a result, DROP minimizes disk head movements in the array at run time giving rise to high IO performance. A prototype DROP has been implemented as a software module at the storage target controller. Extensive experiments have been carried out using real world IO workloads to evaluate the performance of the DROP implementation. Experimental results have shown that DROP improves shared IO performance greatly. The performance improvements in terms of average IO response time range from 20% to a factor 2.5 at no additional hardware cost.**

I. Introduction

Storage consolidation has been widely adopted by IT organizations for high efficiency, good scalability, easy management, and low cost. In such a consolidated storage environment, many servers typically share a storage pool through either a storage area network (SAN) or network attached storage (NAS). It is not uncommon to find, in a IT department of small to medium size enterprises, a high performance or a mid range disk array that is shared by many servers that perform different applications and data services. The performance of the shared disk array is essential to the effectiveness of the data services of the enterprises.

Many IO optimization techniques exist at OS level or application level [1]–[3] by either increasing sequential data accesses on individual disks or maximizing parallelism in RAID. These techniques work well on storages for single servers. When the RAID is shared by multiple servers that run different applications and different file systems, the IO throughput drops sharply as the number of servers sharing the storage increases. Such a rapid decrease in IO performance is the direct result of intermixes of IOs from heterogeneous servers. Although IOs are optimized from each server point of view by means of large block accesses or parallelism, the mixed IOs from different servers as viewed from the shared disk array causes a great amount of disk head movements. As a result, the overall IO throughput becomes very poor and gets worse as the number of servers increases.

A straightforward solution to this problem is to statically allocate a fixed number of disks from the array to each server so that the RAID is contention free. Each server only accesses its own allocated disks and the IOs are optimized for the applications and the OS running on each server. For example, if a shared disk array has N disks shared by P servers, each server can be allocated N/P disks dedicated to provide disk storage to the server. If the relative IO requirements of servers are known a priori, disks can be allocated to servers proportionally to their disk storage requirements. While this solution eliminates the problem of disk contention and excessive disk head movements, it results in poor load balancing and goes against the original motivation of storage consolidation. Furthermore, it results in excessive waste of disk storage space if all servers are to be provided with high IO throughput and fault tolerances that are expected from a disk array.

Based on the analysis of real world IO access characteristics, we propose a new dynamic data relocation technique that allows balanced storage sharing of a RAID among heterogeneous servers. When a LUN (logic unit number) is created for a server, the entire disk array is available to the server for storage allocation to maximize data parallelism and fault tolerance, leading to true storage consolidation and sharing. To minimize the frequent seek operations caused by intermixes of the IO streams from different servers, we perform dynamic data relocation to optimize performance (DROP) of IOs from different servers. DROP relocates hot data to a special partition of the disk array in such a way that large block accesses on any individual disk and the parallelism of data accesses across multiple disks are maximized. The special partition forms different RAID levels to match the IO access patterns

of servers sharing the storage for optimal IO response time. The partition works in a similar way as a lower level storage cache with a capacity several orders of magnitude larger than a RAM cache. The high-speed access of this cache is achieved by means of large sequential block accesses on individual disks and high parallelism across an array of disks.

A prototype DROP has been implemented at block level as a storage target. The prototype consists of profiling module, dynamic data relocation module, algorithm module, RAID and interface module. All these modules are implemented in the standard Linux iSCSI target [9]. Multiple servers with iSCSI initiators are connected to the iSCSI storage target with the DROP prototype. Real world traces are used to drive the storage target. Experimental results have shown that DROP improves IO response time greatly. The average improvement in terms of IO response time ranges from 20% to a factor 2.5 without any additional hardware. Furthermore, DROP is the least intrusive to servers because it is implemented in a storage target at block level without any need to change servers' OS or applications.

This paper makes the following contributions:

1) A new dynamic data relocation technique from a large disk array to a small cache disk array that is a partition of the large disk array to avoid the IO contention;
2) A prototype implementation of the new data relocation technique, DROP, at block level in the Linux iSCSI target;
3) Extensive experiments and performance evaluations using real world IO traces and standard benchmarks to show that DROP improves IO response time greatly with no additional hardware cost.

The rest of the paper is organized as follows. Next section presents the architecture and algorithm of DROP followed by the implementation details. Section 3 describes our evaluation methodology and workload characteristics. We discuss our experimental results in Section 4. Related work is discussed in Section 5. Section 6 concludes the paper.

II. DROP DESIGN

A. DROP layout

The main idea of DROP is dynamically relocating active data in the shared disk array based on on-line profiling so that IO performance can be optimized. The shared disk array is partitioned into three different areas. The main area is used to store data belonging to all servers sharing the storage. Another area is used as a storage cache that stores active data like a second level cache with orders of magnitude larger capacity. The third area stores necessary metadata to implement the dynamic data relocation. Consider an example disk array with 8 disks shared by three different servers as shown in Figure 1. At the bottom of this array called common storage area, three LUNs are allocated to the three servers as their respective data storage. On top of the LUN area, 6GB space is reserved on each of the 8 disks as a partition for cache purpose. At the top of the figure, 1GB partition from each disk is used to store

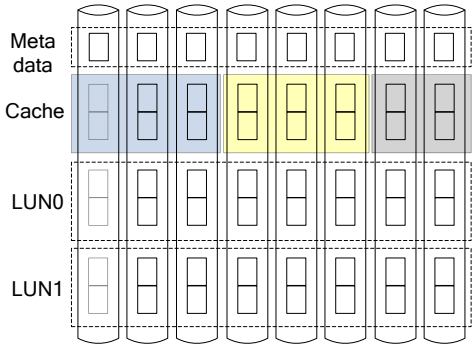

Fig. 1. Data layout on a DROP array.

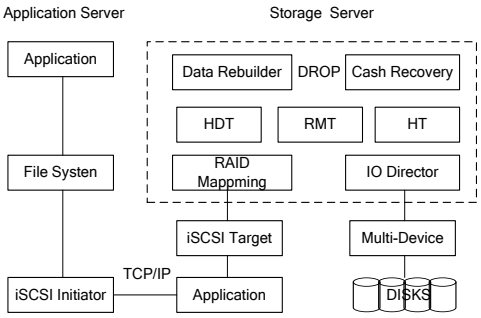

Fig. 2. Software structure of DROP design.

the necessary metadata for data relocation including hotspot data information for each server, RAID mapping table, and Hash table from each server. The configuration of the cache area (gray color area) is done dynamically at system idle time depending on the IO activities and profiling analysis of each server. Such configuration includes the number of disks and what RAID level should be used for each server. Based on our analysis of profiling data, we dynamically form sub-RAID [10] in the cache area for each server, which can be deployed as RAID5 for reliability. Every sub-RAID is owned by one server. Data allocation in the cache area is done in a way to maximize large sequential disk access on any individual disk and to maximize parallel IOs across disks in each sub-RAID.

By partitioning the shared area into three disjoint areas and dynamically relocating data, DROP provides several advantageous features. First of all, active data of different servers are mapped to different sub-RAID in the cache area so that contention among different servers is minimized and so is the number of disk head movements. Secondly, hot data of each server are physically allocated close to each other in its cache area to reduce disk head movements. In addition, dynamic relocation based on profiling data makes it possible to balance the IO workload. Once relocation is done, the cache mapping table is fixed for a long period of time until next relocation. The mapping table is stored in metadata area of the disks as persistent data to allow crash recovery.

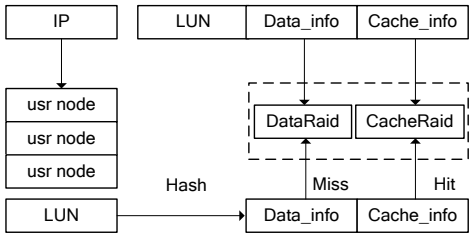

Fig. 3. Structure and relationship of RMT and HT.

B. DROP structure

The overall structure of DROP is shown in Figure 2 as a software module embedded in the standard iSCSI target. It consists of three tables (HDT: Hotspot Data Table, RMT: RAID Mapping Table and HT: Hash Table) to manage metadata and maintain the data consistence.

Metadata Structures: The cache area in the DROP system consists of many sub-RAIDs as shown in Figure 1. Data in this cache area cannot be accessed directly using requests' LBAs. The hotspot data will be compiled statistically in terms of access time during a period and recorded in the HDT. Two major metadata structures for mapping LBAs to physical addresses in the cache: RMT and HT. The structure and the relationship between the two mapping tables are shown in Figure 3. The server can locate the LUN storage resources according the users' IP information, including DataRaid and CacheRaid_from RMT when an authorized user requests the service. Then, the read/write IOs uses the block address (block_id) to calculate the HASH value and handle the IO by indexing HT. The reliability of metadata depends on RAID level used

Data Rebuilder:

This function is acting as data importor (write back the dirty data) and plays the role of data prefetching. The data importing thread will start when the storage system is idle. This thread can import dirty data writing back into the corresponding position of main data areas and update the data mapping relationship in cache areas. The parallel processing of data importing can improve efficiency due to the multi-stream transferring from cache areas to memoires. After gathering the access time of blocks and recording into HDT, data rebuilder relocates hot data to the cache area. Before such migration, data rebuilder first writes back all dirty data in the cache area to their respective locations in the LUNs of the main area as shown at the bottom of Figure 1. As soon as the hot data are relocated to the cache area, the corresponding RMT and HT are written in the metadata area of the disk array. Every time such relocation is initiated, the data rebuilder reorganizes sub-RAID in the cache area and allocates data according to the profiling analysis. Some servers may have larger sub-RAID than others depending on the history of access profile. We decide the number of sub-RAID disks of the server with less workloads prior to the one with more workloads, and the last server (with the heaviest workloads) is allocated with the left disks after all other servers are considered. The exact number of disks allocated to a server as a sub-RAID is based on the following formulas:

$$\begin{cases} No\_of\_disk = Max(2, p \times Total\_No\_disks) \\ p = \dfrac{No\_IO\_server}{Total\_IO} \times \alpha + \dfrac{Data\_size\_server}{Total\_data\_size} \times (1-\alpha) \end{cases}$$

where the number of disks (No_of_disk) used by relocating process mainly depends on the load balance parameter p, which is driven by two parameters, the number of IO(No_IO_server) and data size (Data_size_server) from the server. And α is a tunable weight coefficient with default being 0.5 that means the equal importance of them. Total_IO and Total_data_size are all variables depending on hotspot data statistical interval. After all sub-RAIDs are formed, hot data are stored in the sub-Raid sequentially based on their LBAs. If the total hot data size exceeds the capacity of the cache, data are selected based on their temperature values from high to low.

RAID Mapping:

I/O requests will divide the data streams into the corresponding sub-RAID after the processing of RAID mapping modules. The source of IO requests can be analyzed from the iSCSI connection by checking the IP address to obtain block data address of the servers' data areas and cache areas.

IO Director:

IO director is responsible to process IO requests. IET (iscsitarget-0.4.15) processes IOs based on Linux memory pages (4KB). IET divides all IO requests from servers into 4KB pages and IO director looks for them in the corresponding RMT. If a block is found in the RMT, then it is a cache hit and the corresponding cache block is returned. Otherwise, the request is passed to the regular LUNs at the bottom of the disk array. Lastly, if the HASH hit the Cache, access the sub-RAID according to the CacheRaid and Cache_LBA, or else by DataRaid and Data_LBA to obtain the exact position of the block in the sub-RAID.

We use the existing RAID software of Linux, MD (Multiple Device), for sub-RAID implementation in the cache area dynamically. MD can be used to implement the bottom part of shared RAID in the LUN area of DROP as described in Figure 1.

III. Evaluation methodology

This section presents our experimental setting and methodology that we use to study quantitatively the performance of the DROP prototype as compared to traditional RAID systems.

A. Experimental settings

Our experimental setting consists of a storage server that forms a shared RAID and 5 application servers interconnected using the Cisco 3750 with Gb Ethernet. The shared RAID consists of eight 500 GB SATA drives with 60791 cylinders each controlled by the storage server to form a RAID5 or other RAID levels. The details of the storage server and the disks are listed in Table I. The hardware configurations of the

TABLE I
HARDWARE DETAILS OF THE STORAGE SERVER FOR THE TRACE REPLAY TESTBED

OS	Fedora Core 8.0
Disks	1 Seagate ST3160023AS, 160GB, 7200RPM. 8 Seagate ST3500320AS, 500GB, 7200RPM.
mainboard	SUPER X7DVL-I
CPU	Intel(R) Xeon(R) CPU 5110 1.60GHz
NIC	Intel PRO/1000
memory	1G DDR2
HBA	Highpoint 2220

TABLE II
HARDWARE DETAILS OF APPLICATION SERVERS FOR THE TRACE REPLAY TESTBED

OS	Fedora Core 8.0
Disks	Seagate ST3160023AS, 160GB7200RPM.
mainboard	GA-945GCMX-S2
CPU	Intel(R) Celeron(R) CPU 2.80GHz
NIC	Reltek 8169
memory	512MB DDR2

application servers are shown in Table II. Our DROP prototype is installed in the iSCSI target running on the storage server.

B. Workloads

The GIS traces were collected from real world storage users' environment that has a storage shared by three servers: a database server, a multimedia server, and a web server. They store mainly geographical information (GIS) of an area similar to a large city.

On the application servers, we replay the collected IO traces using btreplay program of the blktrace tool in Linux. As results of the replay, IO requests are generated to the RAID in the form of iSCSI requests. The shared area using RAID5 level acting as an iSCSI target and is allocated to the servers with different LUNs. 1.5TB LUN is allocated to multimedia server and 1TB LUN is allocated to other servers. Our traces were collected over 5-7 days. To speed up our experiments, we have eliminated the interval time between two subsequent IO requests by using option -no-stalls when running btreplay.

C. Baseline systems

We evaluate the performance of DROP in comparison with two baseline systems. The first baseline system (RAID) is the traditional shared RAID with no data relocation and disk caching mechanism used. In this case, data are stored in the RAID based on the upper layer file systems of the servers. The second baseline system (CACHE) uses data relocation and disk caching mechanism similar to prior research reported in [8]. The implementation is referred to as CACHE that reserves 3 areas with 16GB across disks in the array as 3 caches for the three servers, respectively. Hot and correlated IO data of each server are stored in their respective cache area. The cached data are ordered based on their LBAs to ensure sequential and contiguous accesses of hot data in the cache. The major difference between our DROP and the CACHE is relocating hot data of different servers to different exclusive sub-RAID areas to minimize interferences among servers' IO requests and to maximize parallelism of IO operations in the cache area.

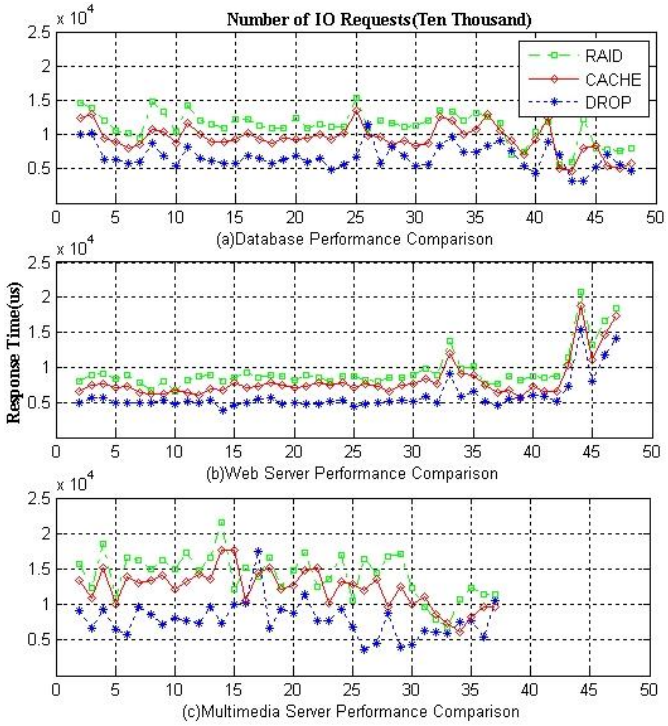

Fig. 4. Average IO response times of the GIS.

IV. NUMERICAL RESULTS AND DISCUSSIONS

Using our prototype implementation and the experimental settings, we measured the average IO response time for each server. Figure 4 shows the average IO response times of the three servers over all IO requests. We noticed low contention and low response time experienced by the DB server as shown in the Figure 4(a). The same results are demonstrated by the Web Server and Multimedia server in Figure 4(b), Figure 4(c) respectively. Our DROP software relocates data to the sub-RAID in the cache area. As a result of this relocation, the service performance improved greatly. We observed that CACHE improves shared RAID performance by 20%, 19%, and 17% for DB server, Web server, and multimedia server, respectively. With DROP, much greater performance improvements were observed. For the three servers, DROP improves the shared RAID performance by 74%, 63%, and 96%, respectively. This greater performance improvement of DROP compared to the CACHE can be attributed to the dynamic and separate allocations of physical sub-RAID to different servers. The result of such relocation is reduction of interferences of IOs from different servers sharing the RAID.

V. RELATED WORK

Extensive research has been reported in the literature on IO access characteristics [4]–[7] that show spatial and temporal locality. Researchers have found that different file systems

show different IO access characteristics [15], [16]. However, one common characteristic is that active working set is usually a very small portion of the entire data set. Making use of this IO characteristics, many caching, prefetching, and data allocation techniques have been proposed [7], [8], [17], [18]. Their results are very promising indicating the effectiveness of exploiting data locality and repetitive IO accesses. DROP differs from these existing research works in its consideration of intermixed IOs from heterogeneous servers sharing a disk array and its optimization techniques on sub-RAIDs in the disk cache area.

One good example is BORG [8] that optimizes disk IO performance by means of a cache partition on a disk. Correlated hot data are placed in the cache area contiguously to allow fast and sequential accesses and minimize disk head movements. While both BORG and DROP try to minimize seek operations on disks, DROP focuses on minimizing interferences among IOs from separate and independent servers in addition to maximizing sequential disk accesses. DROP dynamically allocates sub-RAIDs in the cache area to different servers to allow more parallel IOs and to minimize seek operations.

Distributed file system (DFS) allows many clients to share storage [19]–[22]. Some DFS systems allocate data based on IO characteristics such as ROMIO [23] that uses application hint and file access pattern to improve individual and parallel IO performance, and web [24] that replicate read only data.

While both DFS and DROP deal with shared storages, there are many substantial differences. 1) Data allocation of DFS includes metadata/data allocation, caching, replication as well as file migration policies. DROP's data relocation focuses on migrating block level data within a shared RAID with no file system semantics. 2) DFS improve file system performance by replicating and migrating data closer to client to minimize data transfer distance and time upon IO requests, avoid hotspot, and balance load among clients. DROP achieves high performance by minimizing disk head movements within a disk array upon IO requests.

There is a rich set of literature on buffer cache, secondary level cache, and prefetching techniques for disk storages. A good summary and comparison of these cache designs can be found in [25], [25], [27], [28]. DROP is similar to these cache designs in terms of exploiting locality of data accesses but differs greatly from them in terms of placing and relocating data blocks within one shared disk array as opposed to managing RAM caches. DCD [29] improves small writes performance by means of a log disk on top of data disk. DROP, on the other hand, tries to improve both reads and writes IOs base on dynamic profiling information in shared RAID environments.

VI. Conclusion

This paper presented a new performance optimization technique for RAID storage that is shared by multiple servers. In order to understand the IO characteristics of shared RAID in real world, we carried out onsite studies of three storage user scenarios where a small set of different application servers shares a disk array as a consolidated SAN storage. To avoid the IO contention, we came up with a new dynamic data relocation technique referred to as DROP (dynamic relocation to optimize performance). DROP forms sub-RAIDs on top of traditional RAID acting as the cache data areas and relocates the data to make the cache area serve each server individually in order to minimize disk head movements caused by intermix of disk requests from different servers. Through such dynamic data relocation, DROP achieves load balance among disks in the array, high speed disk accesses, and true storage sharing and consolidation. A prototype DROP has been built on Linux OS as a storage target. Extensive experiments have shown that DROP improves disk IO performance greatly over a variety of IO workloads and sharing scenarios.

Acknowledgment

This Project is supported by the National Basic Research Program (973) of China (No. 2011CB302303), the National Natural Science Foundation of China (No. 60933002), the Natural Science Foundation of Hubei province (NO. 2010CDB01605), the HUST Fund under Grant (Nos.2011QN053 and 2011QN032), the Fundamental Research Funds for the Central Universities. Qing Yang's research is supported in part by National Science Foundation under Grants CCF-0811333, CPS- 0931820, CCF-1017177. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The authors are grateful to anonymous referees for their comments that improve the quality of the paper.

References

[1] B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling Algorithms for Modern Disk Drives. In SIGMETRICS: 241-251,1994.

[2] D. M. Jacobson, and J. Wilkes. Disk Scheduling algorithms based on rotational position. Hewlett-Packard Laboratories, Technical report HPL-CSP-91-7, 1991.

[3] J. Schindler, J. L. Griffin, C. R. Lumb, and G. R. Ganger. Track-aligned Extents: Matching Access Patterns to Disk Drive Characteristics. Conference on File and Storage Technologies, pages 259-274, USENIX Association, 2002.

[4] C. Ruemmler, and J. Wilkes. UNIX disk access patterns. In Proc. of the Winter USENIX Conference, 1993.

[5] M. Sivan-Zimet. A Comparison of Access Pattern (92-99). UCSC CS290S Project, 2000.

[6] M. Gmez, and V. Santonja. Characterizing Temporal Locality in I/O Workload. In Proc. of the International Symposium on Performance Evaluation of Computer and Telecommunication systems, 2002.

[7] W. W. Hsu, A. J. Smith, and H. C. Young. The automatic improvement of locality in storage systems. ACM Transactions on Computer Systems, 23(4): 424-473,2005.

[8] Medha Bhadkamkar, Jorge Guerra, Luis Useche, Sam Burnett, Jason Liptak, Raju Rangaswami, and Vagelis Hristidis. BORG: Block-reORGanization for Self-optimizing Storage Systems. In Proc. of USENIX FAST 2009.

[9] iSCSI Enterprise Target. http://sourceforge.net/projects/iscsitarget/.

[10] J.G. Wan, G.B. Wang, Qing Yang and Changsheng Xie. S2-RAID: A New RAID Architecture for Fast Data Recovery. In Proc. of 26th IEEE Symposium on Massive Storage Systems and Technologies (MSST2010). Nevada, USA, 2010.

[11] J. Katcher. PostMark: A new file system bench-mark. Net-work Appliance, Tech. Rep. 3022, 1997.

[12] Transaction Processing Performance Council . TPC Bench-markTM C Standard Specification. http://www.tpc.org/tpcc, 2005.

[13] J. Piernas, T. Cortes, and J. M. Garcia. TPCC- UVA: A free, open-source implementation of the TPC-C Benchmark. http://www.infor.uva.es/ diego/tpcc-uva.html, 2005.

[14] H. W. Cain, R. Rajwar, M. Marden, and M. H. Lipasti. An Architectural Evaluation of Java TPC-W. In Proc. of HPCA 2001, Nuevo Leone, Mexico, 2001.

[15] D. Roselli, J. R. Lorch, and T. E. Anderson. A Comparison of File System Workloads. In Proc. of the 2000 USENIX Annual Technical Conference, pages 41-54, San Diego, CA, 2000.

[16] D. Ellard, J. Ledlie, P. Malkani, and M. Seltzer. Passive NFS Tracing of Email and Research Workloads. In Proc. of the 2nd USENIX Conference on File and Storage Technologies (FAST'03), pages 203-216, San Francisco, CA, March 2003.

[17] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou. C-Miner: Mining Block Correlations in Storage Systems. In Proc. of the 3rd USENIX Conference on File and Storage pages 173-186, San Francisco, CA, USA, 2004.

[18] B. Salmon, E. Thereska, C. A. N. Soules, and G. R. Ganger. A two-tiered software architecture for automated tuning of disk layouts. In First Workshop on Algorithms and Archietectures for Self-Managing Systems, 2003.

[19] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: an Architecture for Global-scale Persistent Storage. ACM SIGOPS Operating Systems Review, 35(5): 190-201,2000.

[20] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg. IBM Storage Tank-a Heterogeneous Scalable SAN File System. IBM Systems Journal, 42(2): 250-267,2003.

[21] P. J. Braam. The Lustre Storage Architecture. http://www.lustre.org/document.html, Cluster File Systems, INC., Aug., 2004.

[22] F. Schmuck, and R. Haskin. GPFS: a Shared-disk File System for Large Computing Clusters. In Proc. of 1st Conference on File and Storage Technologies (FAST), pages 231-244, 2002.

[23] R. Thakur, E. Lusk, and W. Gropp. ROMIO: A High-Performance, Portable MPI-IO Implementation. 2002.

[24] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In Proc. of INFOCOM, 2001.

[25] J. Song, X. Ding, F. Chen, E. Tan, and X. Zhang. DULO: an Effective Buffer Cache management Scheme to exploit both Temporal and Spatial Locality. In Proc. of the 4th Conference on File and Storage Technologies(FAST'05), 2005.

[26] B. S. Gill, M. Ko, B. Debnath, and W. Belluomini. STOW: A Spatially and Temporally Optimized Write Cache Algorithm. USENIX 2009 Annual Technical Conference, 2009.

[27] Y. Y. Zhou, J. Philbin, and K. Li. The Multi-queue Replacement Algorithm for Second Level Buffer Caches. In Proc. of the USENIX Annual Technical Conference, pages 91-104, June 2001.

[28] T. M. Wong, and J. Wilkes. My cache or yours? Making storage more exclusive. In USENIX Annual Technical Conference (USENIX 2002), pages 161-175, 2002.

[29] Y.Hu and Q.Yang. DCD-Disk Caching Disk: A New Approach for Boosting I/O Performance. Proc. of the International Symposium on Computer Architecture, 1995.