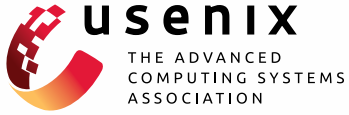


CRAID: Online RAID Upgrades Using Dynamic Hot Data Reorganization

Alberto Miranda, ***Barcelona Supercomputing Center (BSC-CNS);*** **Toni Cortes,**
Barcelona Supercomputing Center (BSC-CNS) and Technical University of Catalonia (UPC)

https://www.usenix.org/conference/fast14/technical-sessions/presentation/miranda

This paper is included in the Proceedings of the
12th USENIX Conference on File and Storage Technologies (FAST '14).

February 17–20, 2014 • Santa Clara, CA USA

ISBN 978-1-931971-08-9

Open access to the Proceedings of the
12th USENIX Conference on File and Storage
Technologies (FAST '14)
is sponsored by

CRAID: Online RAID Upgrades Using Dynamic Hot Data Reorganization

A. Miranda[§], T. Cortes[§‡]
[§]*Barcelona Supercomputing Center (BSC–CNS)* [‡]*Technical University of Catalonia (UPC)*

Abstract

Current algorithms used to upgrade RAID arrays typically require large amounts of data to be migrated, even those that move only the minimum amount of data required to keep a balanced data load. This paper presents CRAID, a self-optimizing RAID array that performs an online block reorganization of frequently used, long-term accessed data in order to reduce this migration even further. To achieve this objective, CRAID tracks frequently used, long-term data blocks and copies them to a dedicated partition spread across all the disks in the array. When new disks are added, CRAID only needs to extend this process to the new devices to redistribute this partition, thus greatly reducing the overhead of the upgrade process. In addition, the reorganized access patterns within this partition improve the array's performance, amortizing the copy overhead and allowing CRAID to offer a performance competitive with traditional RAIDs.

We describe CRAID's motivation and design and we evaluate it by replaying seven real-world workloads including a file server, a web server and a user share. Our experiments show that CRAID can successfully detect hot data variations and begin using new disks as soon as they are added to the array. Also, the usage of a dedicated partition improves the sequentiality of relevant data access, which amortizes the cost of reorganizations. Finally, we prove that a full-HDD CRAID array with a small distributed partition ($<$1.28% per disk) can compete in performance with an ideally restriped RAID-5 and a hybrid RAID-5 with a small SSD cache.

1 Introduction

Storage architectures based on Redundant Arrays of Independent Disks (RAID) [36, 10] are a popular choice to provide reliable, high performance storage at an acceptable economic and spatial cost. Due to the ever-increasing demand of storage capabilities, however, applications often require larger storage capacity or higher performance, which is normally achieved by adding new devices to the existing RAID volume. Nevertheless, several challenges arise when upgrading RAID arrays in this manner:

1. To regain uniformity in the data distribution, certain blocks must be moved to the new disks. Traditional approaches that try to preserve the round-robin order [15, 7, 49] end up redistributing large amounts of data between old and new disks, regardless of the number of new and old disks.
2. Alternative methods that migrate a minimum amount of data, can have problems to keep a uniform data distribution after several upgrade operations (like the Semi-RR algorithm [13]) or limit the array's performance (GSR [47]).
3. Existing RAID solutions with redundancy mechanisms, like RAID-5 and RAID-6, have the additional overhead of recalculating and updating the associated parities, as well as the necessary metadata updates associated to stripe migration.
4. RAID solutions are widely used in online services where clients and applications need to access data constantly. In these services, the downtime cost can be extremely high [35], and thus any strategy to upgrade RAID arrays should be able to interleave its job with normal I/O operations.

To address the challenges above, in this paper we propose a novel approach called CRAID, whose purpose is to minimize the overhead of the upgrade process by redistributing only "relevant data" in real-time. To do that, CRAID tracks data that is currently being used by clients and reorganizes it in a specific partition. This partition allows the volume to maintain the performance and distribution uniformity of the data that is actually being used by clients and, at the same time, significantly reduce the amount of data that must be migrated to new devices.

Our proposal is based on the notion that providing good levels of performance and load balance for the current working set suffices to preserve the QoS[1] of the RAID array. This notion is born from the following observations about long-term access patterns in storage: (i) data in a storage system displays a *non-uniform access frequency distribution*: when considering coarse-granularity time spans, "frequently accessed" data is usually a small fraction of the total data; (ii) this active data set exhibits *long-term temporal locality* and is *stable*, with small amounts of data losing or gaining importance gradually; (iii) even

[1] In this paper, the term QoS refers to the performance and load distribution levels offered by the RAID array.

Trace	Year	Workload	Reads (GB)		Writes (GB)		R/W ratio	Total accessed data (GB)	Accesses to Top 20% data
			Total	Unique	Total	Unique			
cello99	1999	research	73.73	10.52	129.91	10.92	0.62	203.65	65.77%
deasna	2002	research/email	672.4	23.32	231.57	45.45	2.54	903.97	86.88%
home02	2001	NFS share	269.29	9.07	66.35	4.49	3.94	335.64	61.36%
webresearch	2009	web server	–	–	3.37	0.51	–	3.37	51.33%
webusers	2009	web server	1.16	0.45	6.85	0.50	0.09	8.01	56.17%
wdev	2007	test server	2.76	0.2	8.77	0.42	0.21	11.54	72.44%
proj	2007	file server	2152.74	1238.86	367.05	168.88	7.33	2519.79	57.64%

Table 1: Summary statistics of 1-week long traces from seven different systems.

within the active data set, usage is heavily skewed, with "really popular" data receiving over 90% accesses [29].

These observations are largely intuitive and similar to the findings on short-term access patterns of other researchers [14, 20, 38, 2, 37, 42, 41, 5]. To our knowledge, however, there have not been any attempts to apply this information to the upgrade process of RAID arrays.

This paper makes the following contributions: we prove that using a large cache-like partition that uses all storage devices can be better than using dedicated devices due to the improved parallelism, in some cases even when the dedicated devices are faster. Additionally, we demonstrate that information about hot data can be used to reduce the overhead of rebalancing a storage system.

The paper is organized as follows: (i) we study the characteristics of several I/O workloads and show how the findings motivate CRAID (§2), (ii) we present the design of an online block reorganization system that adapts to changes in the I/O working set (§3), (iii) we evaluate several well-known cache management algorithms and their effectiveness in capturing long-term access patterns (§4), and (iv) we simulate CRAID under several real-system workloads to evaluate its merits and weaknesses (§5).

2 Characteristics of I/O Workloads

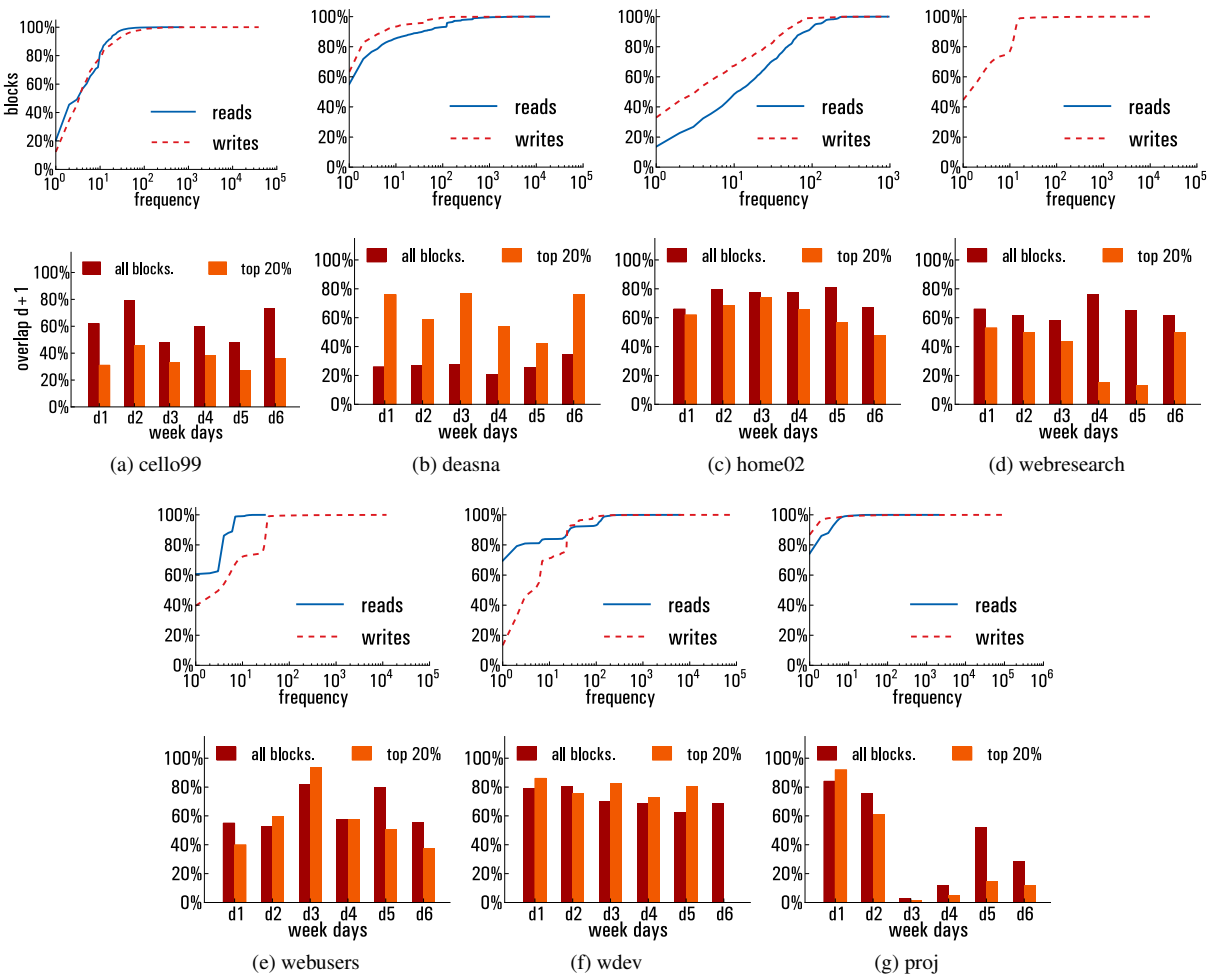
In this section we investigate the characteristics of several I/O workloads, focusing on those properties that directly motivate CRAID. In order for CRAID to be successful, the cost of reorganizing data must be lower than the potential gain obtained from the improved distribution, or it would not make sense to reorganize this data. Thus, we need to prove that long-term working sets exist and that they account for a large fraction of I/O. To do that, we analyzed a collection of well-known traces taken from several real systems. To increase the scope of our analysis, we use traces representing many different workloads and collected at different points in time over the last 13 years. Even if some of these traces are rather old, they can be helpful to establish a historical perspective on long-term hot data. Table 1 summarizes key statistics for one week of these traces, which we describe in detail below:

- The *cello99* traces are a set of well-known block-level traces used in many storage-related studies [22, 34, 46, 51]. Collected at HP Labs in 1999, they include one year of I/O workloads from a research cluster.
- The *deasna* traces [12] were taken from the NFS system at Harvard's Department of Engineering and Applied Sciences over the course of six weeks, in mid-fall 2002. Workload is a mix of research and email.
- The *home02* traces [12] were collected in 2001 from one of fourteen disk arrays in the Harvard CAMPUS NFS system. This system served over 10,000 school and administration accounts and consisted of three NFS servers connected to fourteen 53GB disk arrays. The traces collect six weeks worth of I/O operations.
- The *MSRC* traces [31] are block-level traces of storage volumes collected over one week at Microsoft Research Cambridge data center in 2007. The traces collected I/O requests on 36 volumes in 13 servers (179 disks). We use the *wdev* and *proj* servers, a test web server (4 volumes) and a server of project files (5 volumes), as they contain the most requests.
- The *SRCMap* traces are block-level traces collected by the Systems Research Laboratory (SyLab) at Florida International University in 2009 [41]. The traces were collected for three weeks at three production systems with several workloads. We use the *webresearch* and *webusers* workloads as they include the most requests. The first was an Apache web server managing FIU research projects, and the second a web server hosting faculty, staff, and graduate student web sites.

Our analysis of the traces shows that the following observations are consistent across all traces and, thus, validate the theoretical applicability of CRAID.

Obs. 1 *Data usage is highly skewed with a small percentage of blocks being heavily accessed.*

Fig. 1 (top row), shows the CDF for block access frequency for each workload. All traces show that the distribution of access frequencies is highly skewed: for read



(a) cello99 (b) deasna (c) home02 (d) webresearch

(e) webusers (f) wdev (g) proj

Figure 1: Block-frequency and working-set overlap for 1-week traces from seven different systems. The top row plots depict the CDF of block accesses for different frequencies: a point (f, p_1) on the block percentage curve indicates that p_1% of total blocks were accessed at most f times. Bottom row plots depict changes in the daily working-sets of the workloads: a bar (d, p_2) indicates that days d and $d+1$ had p_2% blocks in common. This is shown for all blocks and for the 20% blocks receiving more accesses.

requests ≈76–98% blocks are accessed 50 times or less, while for write requests this value rises to ≈89–98%. On the other hand, a small fraction of blocks (≈0.05–0.7%) is very heavily accessed in all cases (read or write requests).

This skew can also be seen in Table 1: the top 20% most frequently accessed blocks account for a large fraction (≈51–83%) of all accesses, which are similar results to those reported in previous studies [14, 24, 5, 41, 29].

Obs. 2 *Working-sets remain stable over long durations.*

Based on the first observation, we hypothesize that data usage exhibits long-term temporal locality. By long-term, we refer to a locality of hours or days, rather than seconds or minutes which is more typical of memory accesses. It is fairly common for a developer to work on a limited number of projects or for a user to access only a fraction of his data (like personal pictures or videos) over a few days or weeks. Even in servers, the popularity of the data accessed may result in long-term temporal locality. For instance, a very popular video uploaded to the web will receive bursts of accesses for several weeks or months.

Fig. 1 (bottom row), depicts the changes in the daily working-sets for each of the workloads. Each bar represents the percentage of unique blocks that are accessed both in day d and $d+1$. Most workloads show a significant overlap (≈55%–80%) between the blocks accessed in immediately successive days, and we also observe that there is a substantial overlap even when considering the top 20% most accessed blocks. Trace *deasna* is particularly interesting because it shows low values of overlap (≈20%–35%) when considering all accesses, which increases to ≈55%–80% for the top 20% blocks. This means that the working-set for this particular workload is more diverse but still contains a significant amount of heavily reused blocks. Based on the observations above, it seems reasonable that exploiting long-term temporal

locality and non-uniform access distribution can deal performance benefits. CRAID's goal is to use these to amortize the cost of data rebalancing during RAID upgrades.

3 CRAID Overview

The goal behind CRAID is to reduce the amount of data that needs to be migrated in reconfigurations while providing QoS levels similar to those of traditional RAID.

CRAID claims a small portion of each device and uses it to create a *cache partition* (P_C) that will be used to place copies of heavily accessed data blocks. The aim of this partition is to separate data that is currently important for clients from data that is rarely (if ever) used. Data not currently being accessed is kept in an *archive partition* (P_A) that uses the remainder of the disks. Notice that this partition can be managed by any data allocation strategy, but it is important that the archive can grow gracefully and any archived data is accessed with acceptable performance.

Effectively optimizing the layout of heavily used blocks within a small partition is beneficial for several reasons:

(i) It is possible to create a large cache by using a small fraction of all available disks, which allows important data to be cache-resident for longer periods.

(ii) A disk-based cache is a persistent cache: any optimized layout continues to be valid as long as it is warranted by access semantics, even if it is necessary to shutdown or reconfigure the storage system.

(iii) The size of the partition can be easily configured by an administrator or an automatic process to better suit storage demands.

(iv) Clustering frequently accessed data together offers the opportunity to improve access patterns: data accesses that were originally scattered can be sequentialized if the layout is appropriate. This also helps reduce seek times and rotational delays in all disks since "hot" blocks are placed close to each other.

(v) Whenever new devices are added, current strategies need to redistribute large amounts of data to be able to use them effectively and also to maintain QoS levels (e.g. performance or load balance). A disk-based cache offers a unique possibility to maintain QoS by redistributing only most accessed data. This should reduce the cost of the upgrade process significantly.

(vi) Extending the partition over all devices has three advantages over using dedicated devices. First, it maximizes the potential parallelism offered by the storage system. Second, it is much more likely to saturate a reduced set of dedicated devices than a large array. Third, benefits can be gained with the existing set of devices, without having to acquire more.

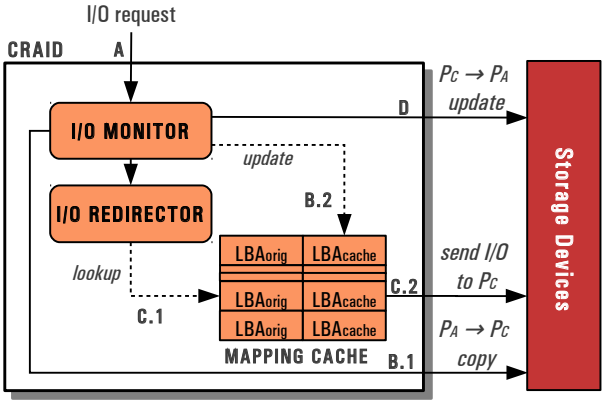


Figure 2: CRAID's I/O control flow.

Fig. 2 shows the control flow supported by CRAID's architecture: when an I/O request enters the system (A), it is captured by CRAID's *I/O monitor* which determines if the accessed data must be considered "active". If so, data blocks are copied to the caching partition if they are not already in it (B.1) and an appropriate mapping $\langle LBA_{original}, LBA_{cache} \rangle$ is stored in the *mapping cache* (B.2). From this point on, an *I/O redirector* will redirect all future accesses to $LBA_{original}$ to the caching partition (C.1 and C.2). This continues until the I/O monitor decides that data is no longer active and removes the entry from the mapping cache. Any update to the contents of the data is then written back to P_A (D). This flow means that the upgrade process begins immediately when a new disk is added to CRAID (which forces P_C to grow), and is interleaved with the array's normal I/O operation. This permits CRAID to use the new disks from the moment they are added to the array.

4 Detailed Design

This section elaborates on CRAID's design details by discussing its individual components mentioned in §3: the *I/O monitor*, the *I/O redirector* and the *mapping cache*.

4.1 I/O Monitor

The *I/O monitor* is responsible for analyzing I/O requests to identify the working set and schedule the appropriate operations to copy data between partitions. The I/O monitor uses a conservative definition of working set that includes the latest k distinct blocks that have been *more active*, where k is P_C's current capacity.

When a request forces an eviction in P_C, the I/O monitor checks if the cached copy is dirty and, if so, schedules the corresponding I/O operations to update the original data. Otherwise, the data is replaced by the newly cached block. Currently, the I/O monitor supports the following simple policies in order to make replacement decisions:

- *Least Recently Used* (LRU) uses recency of access to decide if a block has to be replaced.

- *Least Frequently Used with Dynamic Aging* (LFUDA) uses popularity of access and replaces the block with the smallest key $K_i = (C_i * F_i) + L$, where C_i is the retrieval cost, F_i is a frequency count and L is a running age factor that starts at 0 and is updated for each replaced block [3].
- *Greedy-Dual-Size with Frequency* (GDSF) includes the size of the original request, S_i, and replaces the block with minimum $K_i = (C_i * F_i)/S_i + L$ [21, 9, 3].
- *Adaptive Replacement Cache* (ARC) [28] balances between recency and frequency in an online and self-tuning fashion. ARC adapts to changes in the workload by tracking *ghost hits* (recently evicted entries) and replaces either the LRU or LFU block depending on recent history.
- *Weighted LRU* (WLRU_w) is a simple extension of the LRU algorithm that tries to find the least recently used block that is also *clean* (i.e. not dirty). In order to avoid lengthy $O(k)$ traversals it uses a parameter $w \in \mathbb{R}$ to limit the number of blocks that will be evaluated to $k * w$. If no suitable candidate is found in $k * w$ steps, the LRU block is replaced.

We evaluate the effectiveness of these basic strategies to accurately predict the workload in §5.1. We implemented these basic strategies instead of more complex ones because these algorithms are typically extremely efficient and consume few resources, which makes them suitable to be included in a RAID controller. Furthermore, their prediction rates are usually quite high. Exploring more sophisticated strategies and/or data mining approaches to model complex data interrelations is left for the future.

The I/O monitor is also in charge of rebalancing P_C. When new devices are added, the I/O monitor invalidates all the blocks contained in P_C (writing back to P_A the copies that need updating) and starts filling it with the current working set when blocks are requested. This conservative approach allows us to create long sequential chains of potentially related blocks, which improves the sequentiality and parallelism of the data in P_C. Note that since P_C always holds 'hot blocks', the rebalancing is never completely finished unless the working set remains stable for a long time. Nevertheless, as we show in §5, the cost of this 'on-line' monitoring and rebalancing is amortized by the performance obtained.

4.2 Mapping Cache

The *mapping cache* is an in-memory data structure used to translate block addresses in the P_A to their corresponding copies in P_C. The structure stores, for each block copied to P_C, the block's LBA in P_A, the corresponding LBA in P_C and a flag indicating if the cached copy has been modified.

Our current implementation uses a tree-based binary structure to handle mappings, which ensures that the total time complexity for a lookup operation is given by $O(log\, k)$. Concerning memory, for every block in P_C, CRAID stores 4 bytes for each LBA and 1 dirty bit, plus 8 additional bytes for the structure pointer. Assuming that all k blocks are occupied, that the configured block size is 4KB and P_C size of S GB, the worst case memory requirement is $2 \times S$ MB for LBAs, $S/2^5$ for the dirty information, and $4 \times S$ MB for the tree pointers. Thus, in the worst case, CRAID requires memory of 0.58% the size of the cache partition, or ≈5.9MB per GB, an acceptable requirement for a RAID controller.

Notice that the destruction of the mapping cache can lead to data loss since block updates are performed in place in the cache partition. Failure resilience of the mapping cache is provided by maintaining a persistent log of which cached data blocks have been modified and their translations. This ensures that these blocks, whose cached copies were not identical to the original data, can be successfully recovered. Blocks that were not dirty in P_C don't need to be recovered and are invalidated.

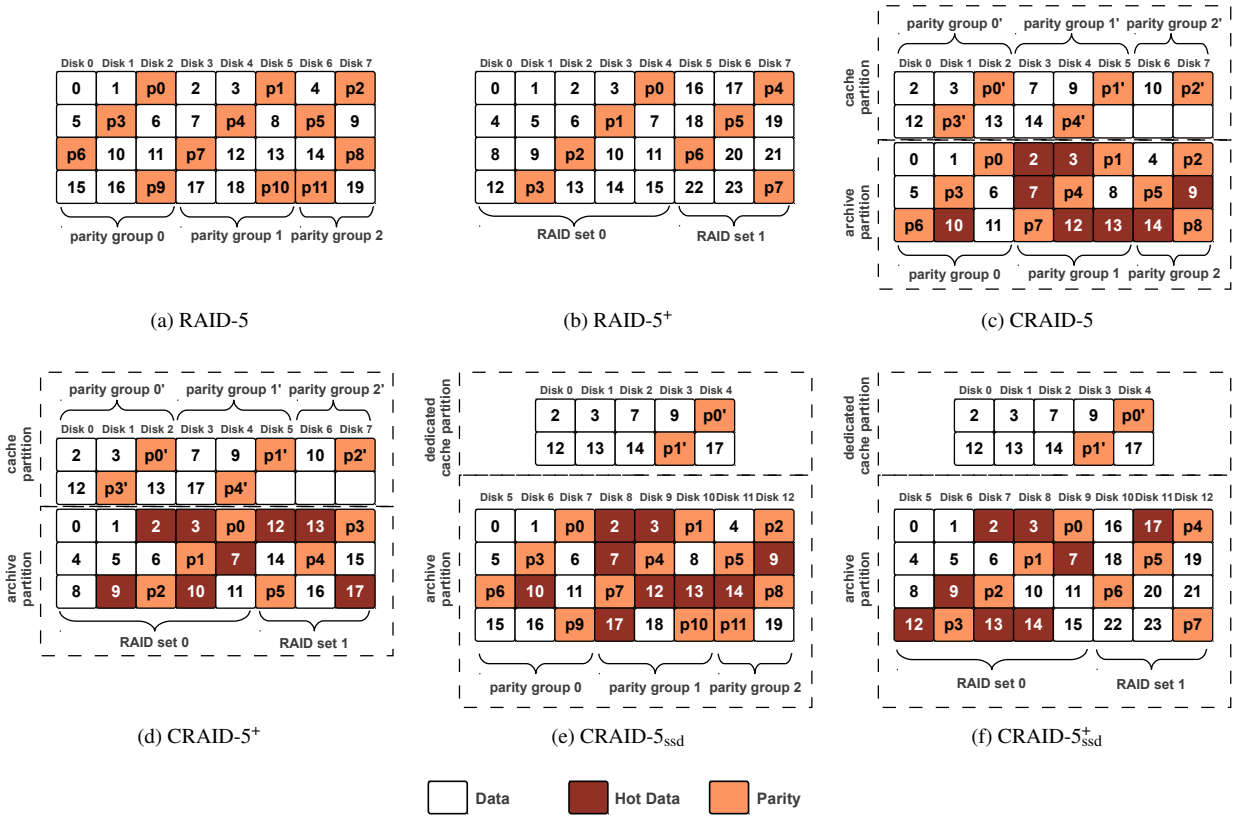
4.3 I/O Redirector

The *I/O redirector* is responsible for intercepting all read and write requests sent to the CRAID volume and redirect them to the appropriate partition. For each request, it first checks the mapping cache for an existing cached copy. If none is found, the request is served from P_A. Otherwise, the request is dispatched to the appropriate location in P_C. Multi-block I/Os are split as required.

5 Evaluation

In this section we evaluate CRAID's behavior using a storage simulator. We seek to answer the following questions: (1) How well does CRAID capture working sets? (2) How does CRAID impact performance? (3) How sensitive is load balance to CRAID's I/O redirection? To answer these questions, we evaluate CRAID under realistic workloads, using detailed simulations where we replay the real-time storage traces described in §2. Since some of these traces include data collected over several weeks or months, which makes them intractable for fine-grained simulations, we simulate an entire continuous week (168 hours) chosen at random from each dataset. Note that in this paper, we only describe the evaluations of several CRAID variants that use RAID-5 in P_C. For brevity's sake, we do not include similar results with RAID-0 [4].

Simulation system. The simulator consists of a workload generator and a simulated storage subsystem composed of



(a) RAID-5 (b) RAID-5$^+$ (c) CRAID-5

(d) CRAID-5$^+$ (e) CRAID-5$_{ssd}$ (f) CRAID-5$^+_{ssd}$

Figure 3: Overview of the different allocation policies evaluated.

an array controller and appropriate storage components. For each request recorded in the traces, the workload generator issues a corresponding I/O request at the appropriate time and sends it down to the array controller.

The array controller's main component is the *I/O processor* which encompasses the functions of both the I/O monitor and the I/O redirector. According to the incoming I/O address, it checks the *mapping cache* and forwards it to the caching partition's segment of the appropriate disk. The workload generator, the mapping cache and the I/O processor are implemented in C++, while the different storage components are implemented in DiskSim. DiskSim [8] is an accurate and thoroughly validated disk system simulator developed in the Carnegie Mellon University, which has been used extensively in research projects to study storage architectures [1, 32, 50, 25].

All experiments use a simulated testbed consisting of Seagate Cheetah 15,000 RPM disks [39], each with a capacity of 146GB and 16MB of cache. This is the latest (validated) disk model available to Disksim. Though somewhat old, we decided to use these disks in order to use the detailed simulation model offered by Disksim, rather than a less detailed one. Besides, since our analysis is a comparative one, the disks' performance should benefit or harm all strategies equally. For the simulations involving SSDs, we use Microsoft Research's idealized SSD model [1]. Since the capacity and number of disks in the original traced systems differs from our testbed, we determine the datasets for each trace via static analysis. These datasets are mapped onto the simulated disks uniformly so that all disks have the same access probability.

Trace	LRU	LFUDA	GDSF	ARC	WLRU$_{0.5}$
cello99	**65.23**	65.23	48.75	**65.66**	65.22
deasna	89.63	**89.90**	67.24	89.65	**89.73**
home02	**93.91**	93.86	77.93	**93.92**	93.90
webresearch	81.14	78.92	54.41	**82.38**	**82.14**
webusers	80.40	78.72	60.49	**81.01**	**81.40**
wdev	91.04	**91.88**	32.78	**91.06**	91.02
proj	75.55	75.73	25.43	**75.58**	**75.65**

Table 2: Hit ratio (%) for each cache partition management algorithm. Best and second best shown in bold.

Strategies evaluated. All experiments evaluate the six following allocation policies, an overview of which is shown in Fig. 3:

- **RAID-5:** A RAID-5 configuration that uses all disks available. Stripes are as long as possible but are divided into *parity groups* to improve the robustness and recov-

Trace	LRU	LFUDA	GDSF	ARC	$WLRU_{0.5}$
cello99	34.76	34.76	51.24	**34.31**	**33.76**
deasna	10.36	**10.09**	32.74	10.34	**10.34**
home02	6.08	6.13	22.06	**6.07**	**6.08**
webresearch	18.84	21.06	45.58	**17.60**	**18.83**
webusers	19.58	21.26	39.50	**18.98**	**19.28**
wdev	8.88	**8.04**	67.13	8.85	**8.58**
proj	24.42	**24.24**	74.55	**24.39**	24.72

Table 3: Replacement ratio (%) for each cache partition management algorithm. Best and second best in bold.

erability of the array (Fig. 3a). This policy will help establish a comparison baseline as it provides maximum parallelism and ideal workload distribution. Notice, however, that expanding such an array in real life can be prohibitively expensive.

- **RAID-5⁺:** A RAID-5 configuration that has been expanded and restriped several times. Each expansion phase adds 30% additional disks [27] that constitute a new independent RAID-5. Thus the system can be considered a collection of independent RAID-5 arrays (or *sets*), each with its own stripe size, that have been added to expand the storage capacity (see Fig. 3b). This serves as a comparison baseline to a realistic system upgraded many times.
- **CRAID-5** and **CRAID-5⁺:** CRAID configurations that use RAID-5 for the caching partition. CRAID-5 also uses RAID-5 for the archive partition while CRAID-5⁺ uses RAID-5⁺. The first one serves to evaluate the performance impact of using CRAID on an ideally restriped RAID-5 and the effects on performance of data transfers from/to the cache. With the second one, we evaluate the benefits of using CRAID in a storage system that has grown several times, with a P_A that grows by aggregation.
- **CRAID-5_{ssd}** and **CRAID-5^{+}_{ssd}:** CRAID configurations analogous to CRAID-5 and CRAID-5⁺ but using a fixed number of SSDs for the cache partition. This allows us to evaluate the advantages, if any, of using disk-based CRAID against using dedicated SSDs, which is a common solution offered by storage vendors.

We simulate RAID-5 and RAID-5⁺ in their ideal state, i.e., when the dataset has been completely restriped. The reason is that since CRAID is permanently in an "expansion" phase and sacrifices a small amount of capacity, in order to be useful its performance should be closer to an optimum RAID-5 array, rather than one being restriped. All the arrays simulated use 50 disks, a number chosen based on the datasets of the traces examined, except those for CRAID-5_{ssd} and CRAID-5^{+}_{ssd} that include 5 additional SSDs (10%) for the dedicated cache. RAID-5 uses a parity group size of 10 disks both as a stand-alone

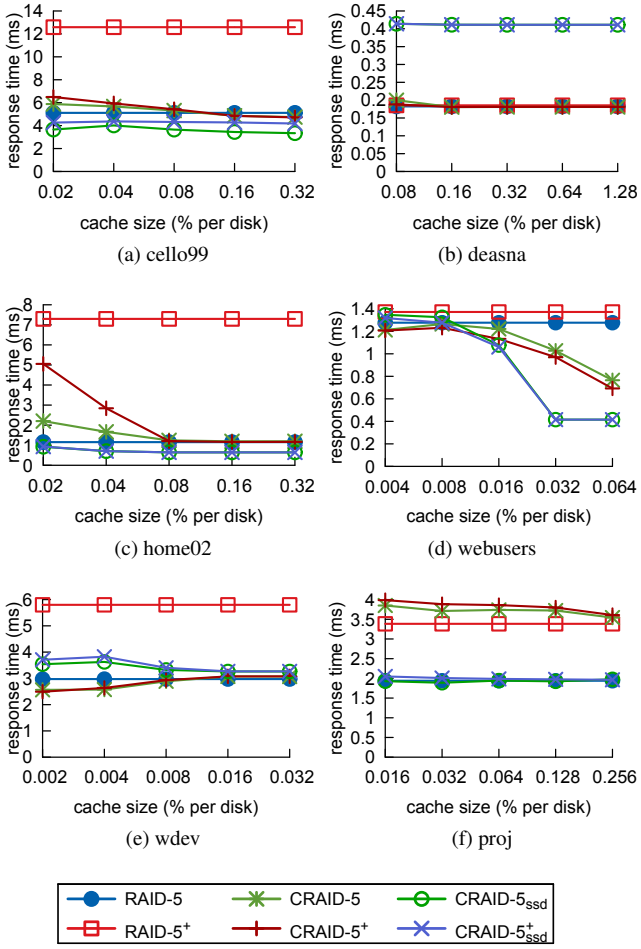


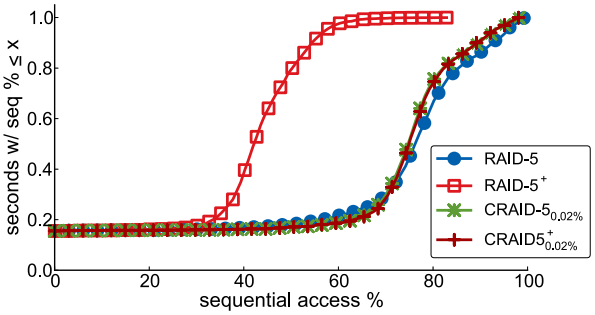
Figure 4: Comparison of I/O response time (read requests).

allocation policy and as a part of a CRAID configuration. Similarly, RAID-5⁺ begins with 10 disks and adds a new array of 3, 4, 5, 7, 9 and 12 disks (+30%) in each expansion step until the 50-disk mark is reached. The stripe unit for all policies is 128KB based on Chen's and Lee's work [11]. In all experiments, the cache partition begins in a cold state.

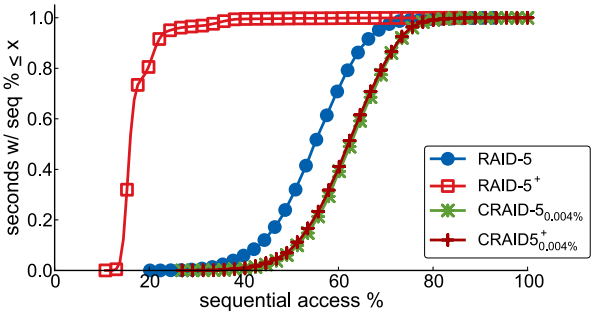
5.1 Cache Partition Management

Here we evaluate the effectiveness to capture the working set of the different cache algorithms supported by the I/O monitor (refer to §4.1). In this experiment we are concerned with the ideal results of the prediction algorithms to select the best one for CRAID. Thus, we use a simplified disk model that resolves each I/O instantly, and allows us to measure the properties of each algorithm with no interferences. The remaining experiments use the more realistic disk model.

Tables 2 and 3 show, respectively, the hit and replacement ratio delivered by each algorithm using a P_C size of 0.1% the weekly working set. We observe that, for each trace, all algorithms except one show similar hit and re-



(a) cello99



(b) webusers

Figure 5: Sequential access distribution (CDF) for the cello99 *and* webusers *traces. Sequentiality percentages captured each second. Other traces show similar results.*

placement ratios with the ARC algorithm showing the best results in both evaluations. The only exception is the GDSF algorithm, which shows significantly worse results due to the addition of the request size as a metric which does not seem very useful in this kind of scenario.

For CRAID strategies based on RAID-5, however, evictions of clean blocks are preferred as long as the effectiveness of the algorithm is not compromised. This is because evicting a dirty block forces CRAID to update the original blocks and its parity in the P_A, which requires 4 additional I/Os (2 reads and 2 writes). In this regard, the WLRU strategy is more suitable since it helps reduce the number of I/O operations needed to keep consistency: if the data block replaced has not been modified, there's no need to copy it back to P_A. Thus, in the following experiments we configure the I/O monitor with the $\text{WLRU}_{0.5}$ algorithm since it shows hit and replacement ratios similar to ARC, and reduces the amount of dirty evictions.

5.2 Response Time

In this section we evaluate the performance impact of using CRAID. For each allocation policy and configuration, we measure the response time of each read and write request occurred during the simulations. Figs. 4 and 6 show the response time measurements[2] of each CRAID variant, compared to the RAID-5 and RAID-5$^+$ baselines.

Note that each strategy was simulated with different cache partition sizes in order to estimate the influence of this parameter on performance. In the results shown in this section, the cache partition is successively doubled until no evictions have to be performed. This represents the best case for CRAID since data movement between the partitions is reduced to a minimum.

Read requests. The results for read requests are shown in Fig. 4. First, we observe that requests take notably longer to complete in RAID-5$^+$ than in RAID-5 in all cases. This is to be expected since the longer stripes in RAID-5 increase its potential parallelism and provide a more effective workload distribution.

Second, in most traces, hybrid strategies CRAID-5 and CRAID-5$^+$ offer performance comparable to that of an ideal RAID-5, or even better for certain cache sizes (e.g. *webusers* trace, Fig. 4d). The explanation lies in the fact that CRAID's cache partition is able to better exploit the spatial locality available in commonly used data: colocating hot data in a small area of each disk helps reduce seek times when compared to the same data being randomly spread over the entire disk, and also increases the sequentiality of access patterns. This can be seen in Fig. 5, that shows the probability distribution (CDF) of the *sequential access percentage* for the *cello99* and *webusers* traces (computed as $\frac{\#Seq\_Access}{\#Accesses}$ and aggregated per second of simulation). Here we see that access sequentiality in CRAID-5 and CRAID-5$^+$ is similar to that of RAID-5 and significantly better than that of RAID-5$^+$. This helps reduce the response time per request and contributes to the overall performance of the array.

Nevertheless, CRAID's effectiveness depends on how well hot data is predicted. Despite the good results shown in §5.1, Fig. 4f shows that performance results for the *proj* trace are not as good as in the other traces. Table 4 shows that CRAID's best hit ratio for the *proj* trace is lower than in other traces (e.g. 85.25% vs. 99.51% in *home02*) and that its eviction count is higher. These two factors contribute to more data being transferred to the cache partition and explain the drop in performance.

Most interestingly, the performance and sequentiality provided by CRAID-5$^+$ is similar to that of CRAID-5, even though it uses a RAID-5$^+$ strategy for the archive partition. This proves that the cache partition is absorbing most of the I/O, and the array behaves like an ideal RAID-5, regardless of the strategy used for stale data.

Third, increasing the size of the cache partition improves read response times in all CRAID-5 variants. This is to be expected since a larger cache partition increases the probability of a cache hit and also decreases the number of evictions, which greatly improves the effectiveness of the strategy. In most traces, however, once a certain partition

[2] 95% confidence interval.

Trace	Best hit ratio		Worst eviction ratio	
	reads	writes	reads	writes
cello99	97.85%	98.88%	21.28%	9.53%
deasna	99.53%	97.80%	0.92%	3.17%
home02	99.51%	99.53%	3.32%	2.59%
webresearch	-	98.76%	-	7.66%
webusers	94.95%	99.33%	16.65%	6.56%
wdev	98.62%	99.40%	1.90%	10.76%
proj	85.25%	88.45%	21.97%	9.13%

Table 4: Best hit ratio and worst eviction ratio (all simulations).

Strategy	Mean		99^{th} pctile		Max	
	Ioq	*Cdev*	*Ioq*	*Cdev*	*Ioq*	*Cdev*
CRAID-5$^+$	2.11	8.65	20	44	381	50
CRAID-5$^+_{ssd}$	4.74	6.49	45	23	427	40

Table 5: Comparison of CRAID's SSD-dedicated vs. full-HDD approach. Ioq: ioqueue size, Cdev: concurrent devices. Trace: wdev*, P_C size: 0.002%. Other traces show similar results.*

size S_M is reached, response times stop improving (e.g. *deasna* with $S_M = 0.16\%$ or *home02* with $S_M = 0.08\%$, Figs. 4b and 4c, respectively). Examination of these traces shows that CRAID is able to achieve a near maximum hit ratio with a partition of size S_M, and increasing it further provides barely noticeable benefits.

Finally, we see that the performance with dedicated SSDs is better than using a distributed partition for most traces. This is to be expected since SSDs are significantly faster than HDDs, and requests can be completed fast enough to avoid saturating the devices. Note, however, that for some P_C sizes, full-HDD CRAID is able to offer similar performance levels (Figs. 4a, 4b, 4d, and 4e), and, given the difference in $/GB between SSDs and HDDs, it might be an appropriate option when it is not possible to add 10% SSDs to the storage architecture. Additionally, a full-SSD RAID should also benefit from the improved parallelism offered by an optimized P_C.

Write requests. The results for write requests are shown in Fig. 6. Similarly to read requests, we observe that write requests are significantly slower in RAID-5$^+$ than in RAID-5, for all traces. Most importantly, the hybrid strategies CRAID-5 and CRAID-5$^+$ perform better than traditional RAID-5 in all traces except *webusers*, where performance is slightly below that of RAID-5.

These improved response times can be explained by two reasons. First, since write requests are always served from the cache partition (except in the case of an eviction), response times benefit greatly from the improved spatial locality and sequentiality provided by the cache partition.[3] Second, the smaller the P_C fragment for each disk

[3]Obviously, as long as the prediction of the working set is accurate.

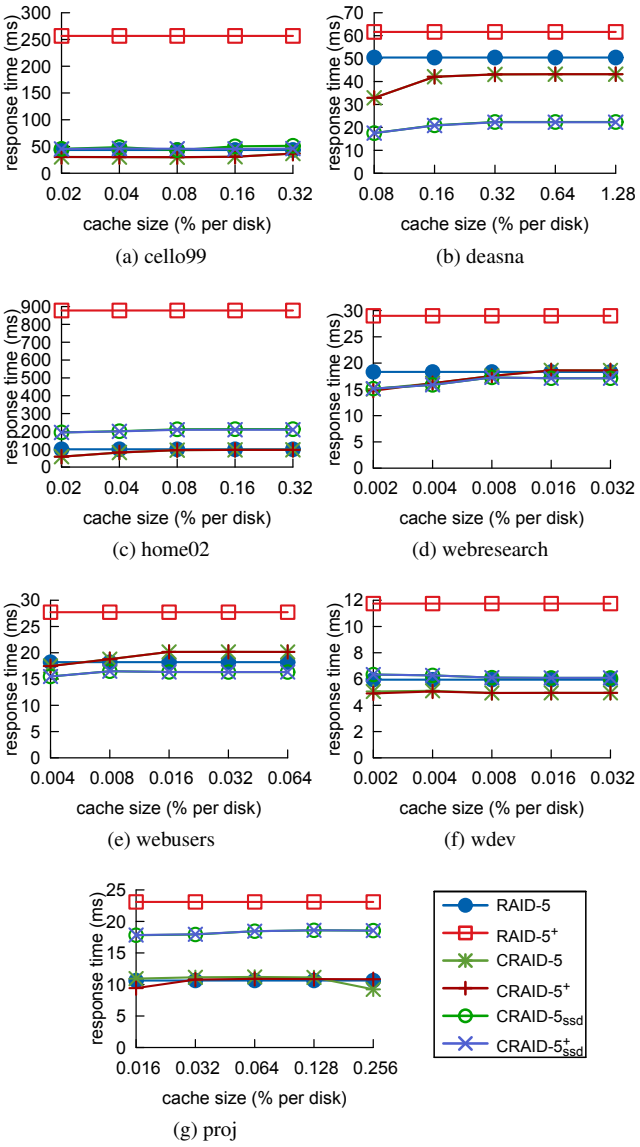


Figure 6: Comparison of I/O response time (write requests).

is, the more likely it is that accesses to this fragment benefit from the disk's internal cache. This explains why response times in Fig. 6 increase slightly for larger partition sizes: a smaller P_C means more evictions in CRAID, but it also means a smaller fragment for each disk and a more effective use of its internal cache. The effect of this internal cache is highly beneficial, to the point that it amortizes the additional work produced by extra evictions.

On the other hand, SSD-based strategies CRAID-5$_{ssd}$ and CRAID-5$^+_{ssd}$ show significantly worse response times than their full-HDD counterparts in some traces (see Figs. 6a, 6c, 6f, or 6g). Examination of these traces reveals that the I/O queues in the dedicated SSDs have significantly more pending requests than those in full-HDD CRAID. Also, the number of concurrently active disks during the simulation is lower (see Table 5). In ad-

Trace	CRAID-5 P_C		CRAID-5$^+$ P_C	
	best c_v	**worst** c_v	**best** c_v	**worst** c_v
cello99	0.02%	0.32%	0.02%	0.32%
deasna	0.08%	1.28%	0.08%	1.28%
home02	0.02%	0.32%	0.02%	0.32%
webresearch	0.002%	0.032%	0.002%	0.032%
webusers	0.004%	0.064%	0.004%	0.064%
wdev	0.002%	0.032%	0.002%	0.032%
proj	0.016%	0.256%	0.016%	0.256%

Table 6: Influence of P_C size on workload distribution.

dition, we discovered that Disksim's SSD model does not simulate a read/write cache. Thus, the lower number of pending requests coupled with the HDD cache benefit explained above, makes full-HDD CRAID faster for write requests in some traces.

5.3 Workload Distribution

In this experiment we evaluate CRAID's ability to maintain a uniform workload distribution. For each second of simulation we measure the I/O load in MB received by each disk and we compute the *coefficient of variation* as a metric to evaluate the uniformity of its distribution. The coefficient of variation (c_v) expresses the standard deviation as a percentage of the average ($\frac{\sigma}{\mu}$), and can be interpreted as how the actual workload deviates from an ideal distribution.[4] We perform this experiment for all strategies described and uses the same P_C sizes of §5.2.

Impact of CRAID. Figs. 7a and 7b show CDFs of c_v per % of samples (seconds) for the *deasna* and *wdev* traces, respectively. Notice that for CRAID strategies we show both the *best* and *worst* curves obtained (Table 6 shows the correspondence with actual P_C sizes) and we compare them with the results for RAID-5 and RAID-5$^+$.
We observe that there is a significant difference between the workload distribution provided by RAID-5 and that of RAID-5$^+$, which is to be expected since the "segmented" nature of RAID-5$^+$ naturally hinders a uniform workload distribution. Most interestingly, all CRAID strategies demonstrate a workload distribution very similar to (and sometimes better than) RAID-5. More importantly, this benefit appears in even those CRAID configurations that use RAID-5$^+$ for the archive partition, despite its poor performance and uneven distribution. This proves that the cache partition is successful in absorbing most I/O, and that it behaves close to an ideal RAID-5 despite the cost of additional data transfers.

Influence of the cache partition size. Though barely noticeable, an unexpected result is that, in all traces, the workload distribution degrades as the cache partition grows (see Table 6). Examination of the traces shows that a larger cache partition slightly increases the probability that certain subsets of disks are more used than others due to the different layout of data blocks. This is reasonable since our current prototype doesn't perform direct actions to enforce a certain workload distribution, but rather relies on the strategy used for the cache partition. Improving CRAID to employ workload-aware layouts is one of the subjects of our future investigation.

Workload with dedicated SSDs. The curves shown in Figs. 7a and 7b show a worse workload distribution for CRAID-5_{ssd} and CRAID-5^+_{ssd} when compared to the full-HDD strategies. This is to be expected since the dedicated SSDs absorb much of the I/O workload and end up degrading the global workload of the system. Note that this does not necessarily mean that the workload directed to the dedicated disks is unbalanced, but rather that the other devices are underutilized. This proves that a spread partition has a higher chance of producing a balanced workload, and can compete in performance, than a dedicated one, even if the devices used for the latter are faster.

6 Discussion and Future Work

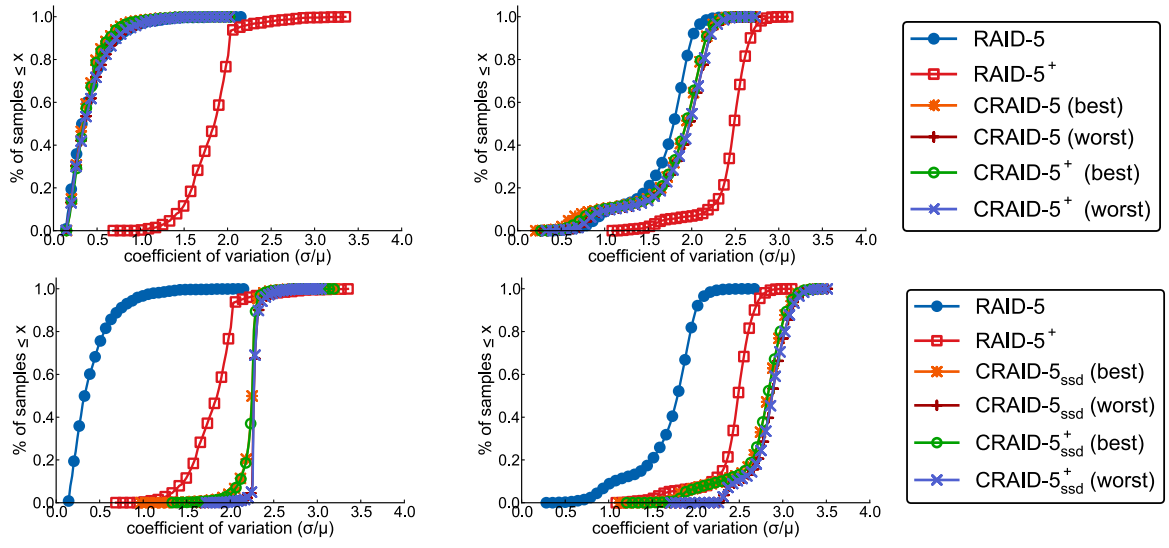
While our experiences with CRAID have been positive in RAID-0 and RAID-5 storage, we believe that they can also be applied to RAID-6 or more general erasure codes, since the overall principle still applies: rebalancing hot data should require less work than producing an ideal distribution. The main caveat of our solution, however, is the cost of additional parity computations and I/O operations for dirty blocks, which directly increases with the number of parity blocks required. Whether this cost can be leveraged by the performance benefits obtained, will be explored in a fully-fledged prototype.

It should also be possible to extend the proposed solution beyond RAID arrays, adapting the techniques to distributed or tiered storage. Specifically, we believe the monitoring of interesting data could be adapted to work with pseudo-randomized data distributions like CRUSH [43] or Random Slicing [30] in order to reduce data migration during upgrades. What to do with blocks that stop being interesting is a promising line of research.

Additionally, while the current CRAID prototype has served to verify that it is possible to amortize the cost of a RAID upgrade by using knowledge about hot data blocks, it uses simple algorithms for prediction and expansion. We envision several ways to improve the current prototype that can serve as subjects of future research.

Smarter prediction. The current version of CRAID does not take into account the relations between blocks in order to copy them to the caching partition, but rather relies on the fact that blocks accessed consecutively in a short

[4] The smaller c_v is, the more uniform the data distribution.



(a) deasna (b) wdev

Figure 7: CRAID workload distribution: full-HDD (top) vs SSD-dedicated (bottom). Figures show CDFs of c_v per % of samples (seconds) for traces deasna *and* wdev. *Other traces show similar results.*

period of time tend to be related. More sophisticated techniques to detect block correlations could improve CRAID significantly, allowing the I/O monitor to migrate data to P_C before it is actually needed.

Smarter rebalancing. The current invalidation of the entire P_C when new disks are added is overkill. Though it benefits the parallelism of the data distribution and new disks can be used immediately, the current strategy was devised to test if our hypothesis held in the simplest case, without complex algorithms. Since working sets should not change drastically, CRAID could benefit greatly from strategies to rebalance the small amount of data in P_C more intelligently, like those in §7.2.

Improved data layout. Similarly, currently CRAID does not make any effort to allocate related blocks close to each other. Alternate layout strategies more focused on preserving semantic relations between blocks might yield great benefits. For instance, it might be interesting to evaluate the effect of copying entire stripes to the cache partition as a way to preserve spatial locality. Besides, this could help reduce the number of parity computations, thus reducing the background I/O present in the array.

7 Related Work

We examine the literature by organizing it into data layout optimization techniques and RAID upgrade strategies.

7.1 Data Layout Optimization

Early works on optimized data layouts by Wong [45], Vongsathorn *et al.* [42] and Ruemmler and Wilkes [37] argued that placing frequently accessed data in the center of the disk served to minimize the expected head movement. Specifically, the latter proved that the best results in I/O performance came from infrequent shuffling (weekly) with small (block/track) granularity. Akyurek and Salem also showed the importance of reorganization at the block level, and the advantages of copying over shuffling [2].

Hu *et al.* [48, 33] proposed an architecture called Disk Caching Disk (DCD), where an additional disk (or partition) is used as a cache to convert small random writes into large log appends, thus improving overall I/O performance. Similarly to DCD, iCache [16] adds a log-disk along with a piece of NVRAM to create a two-level cache hierarchy for iSCSI requests, coalescing small requests into large ones before writing data. HP's AutoRAID [44], on the other hand, extends traditional RAID by partitioning storage in a mirrored zone and a RAID-5 zone. Writes are initially made to the mirrored zone and later migrated in large chunks to RAID-5, thus reducing the space overhead of redundancy information and increasing parallel bandwidth for subsequent reads of active data.

Li *et al.* proposed C-Miner [26], which used data mining techniques to model the correlations between different block I/O requests. Hidrobo and Cortes [18] accurately model disk behavior and compute placement alternatives to estimate the benefits of each distribution. Similar techniques could be used in CRAID to infer complex access patterns and reorganize hot data more effectively.

ALIS [20] and, more recently, BORG [5], reorganize frequently accessed blocks (and block sequences) so that they are placed sequentially on a dedicated disk area. Contrary to CRAID, neither explores multi-disk systems.

7.2 RAID Upgrade Strategies

There are several deterministic approaches to improve the extensibility of RAID-5. HP's AutoRAID allows an online capacity expansion without data migration, by which newly created RAID volumes use all disks and previously created ones use only the original disks.

Conventional approaches redistribute data and preserve the round-robin order. Gonzalez and Cortes proposed a Gradual Assimilation (GA) algorithm [15] to control the overhead of expanding a RAID-5 system, but it has a large redistribution cost since all parities still need to be modified after data migration. US patent #6000010 presents a method to scale RAID-5 volumes that eliminates the need to rewrite data and parity blocks to the original disks [23]. This, however, may lead to an uneven distribution of parity blocks and penalize write requests.

MDM [17] reduces data movement by exchanging some blocks between the original and new disks. It also eliminates parity modification costs since all parity blocks are maintained, but it is unable to increase (only keep) the storage efficiency by adding new disks. FastScale [50] minimizes data migration by moving only data blocks between old and new disks. It also optimizes the migration process by accessing physically sequential data with a single I/O request and by minimizing the number of metadata writes. At the moment, however, it cannot be used in RAID-5. More recently, GSR [47] divides data on the original array into two sections and moves the second one onto the new disks keeping the layout of most stripes. Its main limitation is performance: after upgrades, accesses to the first section are served by original disks, and accesses to the second are served only by newer disks.

Due to the development of object-based storage, randomized RAID is becoming more popular, since it seems to have better scalability. The cut-and-paste strategy proposed by Brinkmann *et al.* [6] uses a randomized function to place data across disks. When a disk is added to disks, it cuts off ranges of data $[1/(n+1), 1/n]$ from the original n disks, and pastes them to the new disk. Also based on a random hash function, Seo and Zimmermann [40] proposed finding a sequence of disks additions that minimized the data migration cost. On the other hand, the algorithm proposed in SCADDAR [13] moves a data block only if the destination disk is one of the newly added disks. This reduces migration significantly, but produces an unbalanced distribution after several expansions.

RUSH [19] and CRUSH [43] are the first methods with dedicated support for replication, and offer a probabilistically optimal data distribution with minimal migration. Their main drawback is that they require new capacity to be added in chunks and the number of disks in a chunk must be enough to hold a complete redundancy group. More recently, Miranda *et al.*'s Random Slicing [30] used a small table with information on insertion and removal operations to reduce the required randomness and deliver a uniform load distribution with minimal migration.

These randomized strategies are designed for object-based storage systems, and focus only on how blocks are mapped to disks, ignoring the inner data layout of each individual disk. In this regard, CRAID manages blocks rather than objects and is thus more similar to deterministic (and extensible) RAID algorithms. To our knowledge, however, it is the first strategy that uses information about data blocks to reduce the overhead of the upgrade process.

8 Conclusions

In this paper, we propose and evaluate CRAID, a self-optimizing RAID architecture that automatically reorganizes frequently used data in a dedicated *caching partition*. CRAID is designed to accelerate the upgrade process of traditional RAID architectures by limiting it to this partition, which contains the data that is currently important and on which certain QoS levels must be kept.

We analyze CRAID using seven real-world traces of different workloads and collected at several times in the last decade. Our analysis shows that CRAID is highly successful in predicting the data workload and its variations. Further, if an appropriate data distribution is used for the cache partition, CRAID optimizes the performance of read and write traffic due to the increased locality and sequentiality of frequently accessed data. Specifically, we show that it is possible to achieve a QoS competitive with an ideal RAID-5 or RAID+SSD array, by creating a small RAID-5 partition of at most 1.28% the available storage, regardless of the layout outside the partition.

In summary, we believe that CRAID is a novel approach to building RAID architectures that can offer reduced expansion times and I/O performance improvements. In addition, its ability to combine several layouts can serve as a starting point to design newer allocation strategies more conscious about data semantics.

Acknowledgments

We wish to thank anonymous reviewers and our shepherd C.S. Lui for their comments and suggestions for improvement. Special thanks go to André Brinkmann, María S. Pérez and BSC's SSRG team for insightful feedback that improved initial drafts significantly. This work was partially supported by the Spanish and Catalan Governments (grants SEV-2011-00067, TIN2012-34557, 2009-SGR-980), and EU's FP7/2007–2013 (grant RI-283493).

References

[1] AGRAWAL, N., PRABHAKARAN, V., WOBBER, T., DAVIS, J., MANASSE, M., AND PANIGRAHY, R. Design tradeoffs for SSD performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference* (2008), pp. 57–70.

[2] AKYÜREK, S., AND SALEM, K. Adaptive block rearrangement. *ACM Transactions on Computer Systems (TOCS) 13*, 2 (1995), 89–121.

[3] ARLITT, M., CHERKASOVA, L., DILLEY, J., FRIEDRICH, R., AND JIN, T. Evaluating content management techniques for web proxy caches. *ACM SIGMETRICS Performance Evaluation Review 27*, 4 (2000), 3–11.

[4] ARTIAGA, E., AND MIRANDA, A. PRACE-2IP Deliverable D12.4. Performance Optimized Lustre. *INFRA-2011-2.3.5 – Second Implementation Phase of the European High Performance Computing (HPC) service PRACE* (2012).

[5] BHADKAMKAR, M., GUERRA, J., USECHE, L., BURNETT, S., LIPTAK, J., RANGASWAMI, R., AND HRISTIDIS, V. BORG: block-reORGanization for self-optimizing storage systems. In *Proccedings of the 7th conference on File and storage technologies* (2009), USENIX Association, pp. 183–196.

[6] BRINKMANN, A., SALZWEDEL, K., AND SCHEIDELER, C. Efficient, Distributed Data Placement Strategies for Storage Area Networks. In *Proceedings of the* 12^{th} *ACM Symposium on Parallel Algorithms and Architectures (SPAA)* (2000), pp. 119–128.

[7] BROWN, N. Online RAID-5 resizing. drivers/md/raid5. c in the source code of Linux Kernel 2.6. 18, 2006.

[8] BUCY, J., SCHINDLER, J., SCHLOSSER, S., AND GANGER, G. The DiskSim Simulation Environment Version 4.0 Reference Manual (CMU-PDL-08-101). *Parallel Data Laboratory* (2008), 26.

[9] CAO, P., AND IRANI, S. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems* (1997), vol. 193.

[10] CHEN, P., LEE, E., GIBSON, G., KATZ, R., AND PATTERSON, D. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys (CSUR) 26*, 2 (1994), 145–185.

[11] CHEN, P. M., AND LEE, E. K. *Striping in a RAID level 5 disk array*, vol. 23. ACM, 1995.

[12] ELLARD, D., LEDLIE, J., MALKANI, P., AND SELTZER, M. Passive NFS tracing of email and research workloads. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies* (2003), USENIX Association, pp. 203–216.

[13] GOEL, A., SHAHABI, C., YAO, S., AND ZIMMERMANN, R. SCADDAR: An efficient randomized technique to reorganize continuous media blocks. In *Data Engineering, 2002. Proceedings. 18th International Conference on* (2002), IEEE, pp. 473–482.

[14] GÓMEZ, M., AND SANTONJA, V. Characterizing temporal locality in I/O workload. In *Proc. of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems* (2002).

[15] GONZALEZ, J., AND CORTES, T. Increasing the capacity of RAID5 by online gradual assimilation. In *Proceedings of the international workshop on Storage network architecture and parallel I/Os* (2004), ACM, pp. 17–24.

[16] HE, X., YANG, Q., AND ZHANG, M. A caching strategy to improve iSCSI performance. In *Local Computer Networks, 2002. Proceedings. LCN 2002. 27th Annual IEEE Conference on* (2002), IEEE, pp. 278–285.

[17] HETZLER, S. R., ET AL. Data storage array scaling method and system with minimal data movement. US Patent 8,239,622.

[18] HIDROBO, F., AND CORTES, T. Autonomic storage system based on automatic learning. In *High Performance Computing-HiPC 2004*. Springer, 2005, pp. 399–409.

[19] HONICKY, R., AND MILLER, E. L. Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International* (2004), IEEE, p. 96.

[20] HSU, W., SMITH, A., AND YOUNG, H. The automatic improvement of locality in storage systems. *ACM Transactions on Computer Systems (TOCS) 23*, 4 (2005), 424–473.

[21] JIN, S., AND BESTAVROS, A. GreedyDual* Web caching algorithm: exploiting the two sources of temporal locality in Web request streams. *Computer Communications 24*, 2 (2001), 174–183.

[22] LEE, S., AND BAHN, H. Data allocation in MEMS-based mobile storage devices. *Consumer Electronics, IEEE Transactions on 52*, 2 (2006), 472–476.

[23] LEGG, C. Method of increasing the storage capacity of a level five RAID disk array by adding, in a single step, a new parity block and N–1 new data blocks which respectively reside in a new columns, where N is at least two, Dec. 7 1999. US Patent 6,000,010.

[24] LEUNG, A., PASUPATHY, S., GOODSON, G., AND MILLER, E. Measurement and analysis of large-scale network file system workloads. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference* (2008), pp. 213–226.

[25] LI, D., AND WANG, J. EERAID: energy efficient redundant and inexpensive disk array. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop* (2004), ACM, p. 29.

[26] LI, Z., CHEN, Z., SRINIVASAN, S., AND ZHOU, Y. C-miner: Mining block correlations in storage systems. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies* (2004), vol. 186, USENIX Association.

[27] LYMAN, P. How much information? 2003. *http://www.sims.berkeley.edu/research/projects/how-much-info-2003/* (2003).

[28] MEGIDDO, N., AND MODHA, D. ARC: A self-tuning, low overhead replacement cache. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies* (2003), pp. 115–130.

[29] MIRANDA, A., AND CORTES, T. Analyzing Long-Term Access Locality to Find Ways to Improve Distributed Storage Systems. In *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on* (2012), IEEE, pp. 544–553.

[30] MIRANDA, A., EFFERT, S., KANG, Y., MILLER, E. L., BRINKMANN, A., AND CORTES, T. Reliable and randomized data distribution strategies for large scale storage systems. In *High Performance Computing (HiPC), 2011 18th International Conference on* (2011), IEEE, pp. 1–10.

[31] NARAYANAN, D., DONNELLY, A., AND ROWSTRON, A. Write off-loading: Practical power management for enterprise storage. *ACM Transactions on Storage (TOS) 4*, 3 (2008), 10.

[32] NARAYANAN, D., THERESKA, E., DONNELLY, A., ELNIKETY, S., AND ROWSTRON, A. Migrating server storage to SSDs: analysis of tradeoffs. In *Proceedings of the 4th ACM European conference on Computer systems* (2009), ACM, pp. 145–158.

[33] NIGHTINGALE, T., HU, Y., AND YANG, Q. The design and implementation of DCD device driver for UNIX. In *Proceedings of the 1999 USENIX Technical Conference* (1999), pp. 295–308.

[34] PARK, J., CHUN, H., BAHN, H., AND KOH, K. G-MST: A dynamic group-based scheduling algorithm for MEMS-based mobile storage devices. *Consumer Electronics, IEEE Transactions on 55*, 2 (2009), 570–575.

[35] PATTERSON, D., ET AL. A simple way to estimate the cost of downtime. In *Proc. 16th Systems Administration Conf.— LISA* (2002), pp. 185–8.

[36] PATTERSON, D., GIBSON, G., AND KATZ, R. *A case for redundant arrays of inexpensive disks (RAID)*, vol. 17. ACM, 1988.

[37] RUEMMLER, C., AND WILKES, J. Disk shuffling. Tech. rep., Technical Report HPL-91-156, Hewlett Packard Laboratories, 1991.

[38] RUEMMLER, C., AND WILKES, J. UNIX disk access patterns. In *Proceedings of the Winter 1993 USENIX Technical Conference* (1993), pp. 405–420.

[39] Seagate Cheetah 15K.5 FC product manual. `http://www.seagate.com/staticfiles/support/disc/manuals/enterprise/cheetah/15K.5/FC/100384772f.pdf` Last retrieved Sept. 9, 2013.

[40] SEO, B., AND ZIMMERMANN, R. Efficient disk replacement and data migration algorithms for large disk subsystems. *ACM Transactions on Storage (TOS) 1*, 3 (2005), 316–345.

[41] VERMA, A., KOLLER, R., USECHE, L., AND RANGASWAMI, R. SRCMap: energy proportional storage using dynamic consolidation. In *Proceedings of the 8th USENIX conference on File and storage technologies* (2010), USENIX Association, pp. 20–20.

[42] VONGSATHORN, P., AND CARSON, S. A system for adaptive disk rearrangement. *Software: Practice and Experience 20*, 3 (1990), 225–242.

[43] WEIL, S. A., BRANDT, S. A., MILLER, E. L., AND MALTZAHN, C. Crush: Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing* (2006), ACM, p. 122.

[44] WILKES, J., GOLDING, R., STAELIN, C., AND SULLIVAN, T. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems (TOCS) 14*, 1 (1996), 108–136.

[45] WONG, C. Minimizing expected head movement in one-dimensional and two-dimensional mass storage systems. *ACM Computing Surveys (CSUR) 12*, 2 (1980), 167–178.

[46] WONG, T., GANGER, G., WILKES, J., ET AL. *My Cache Or Yours?: Making Storage More Exclusive*. School of Computer Science, Carnegie Mellon University, 2000.

[47] WU, C., AND HE, X. Gsr: A global stripe-based redistribution approach to accelerate raid-5 scaling. In *Parallel Processing (ICPP), 2012 41st International Conference on* (2012), IEEE, pp. 460–469.

[48] YANG, Q., AND HU, Y. DCD—disk caching disk: A new approach for boosting I/O performance. In *Computer Architecture, 1996 23rd Annual International Symposium on* (1996), IEEE, pp. 169–169.

[49] ZHANG, G., SHU, J., XUE, W., AND ZHENG, W. SLAS: An efficient approach to scaling round-robin striped volumes. *ACM Transactions on Storage (TOS) 3*, 1 (2007), 3.

[50] ZHENG, W., AND ZHANG, G. FastScale: accelerate RAID scaling by minimizing data migration. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST)* (2011).

[51] ZHU, Q., CHEN, Z., TAN, L., ZHOU, Y., KEETON, K., AND WILKES, J. Hibernator: helping disk arrays sleep through the winter. In *ACM SIGOPS Operating Systems Review* (2005), vol. 39, ACM, pp. 177–190.