Victim Disk First: An Asymmetric Cache to Boost the Performance of Disk Arrays under Faulty Conditions

Shenggang Wan, Qiang Cao, Jianzhong Huang,*
Siyi Li, Xin Li, Shenghui Zhan, Li Yu, Changsheng Xie
Huazhong University of Science & Technology
Wuhan, China 430074
**Corresponding Author: Caoqiang@hust.edu.cn*

Xubin He
Electrical & Computer Engineering
Virginia Commonwealth University
Richmond, VA 23284, USA
xhe2@vcu.edu

Abstract

The buffer cache plays an essential role in smoothing the gap between the upper-level computational components and the lower-level storage devices. A good buffer cache management scheme should be beneficial to not only the computational components, but also to the storage components by reducing disk I/Os. Existing cache replacement algorithms are well optimized for disks in normal mode, but inefficient under faulty scenarios, such as a parity-based disk array with faulty disk(s).

To address this issue, we propose a novel asymmetric buffer cache replacement strategy, named Victim (or faulty) Disk(s) First (VDF) cache, to improve the reliability and performance of a storage system consisting of a buffer cache and disk arrays. The basic idea is to give higher priority to cache the blocks on the faulty disks when the disk array fails, thus reducing the I/Os directed to the faulty disks.

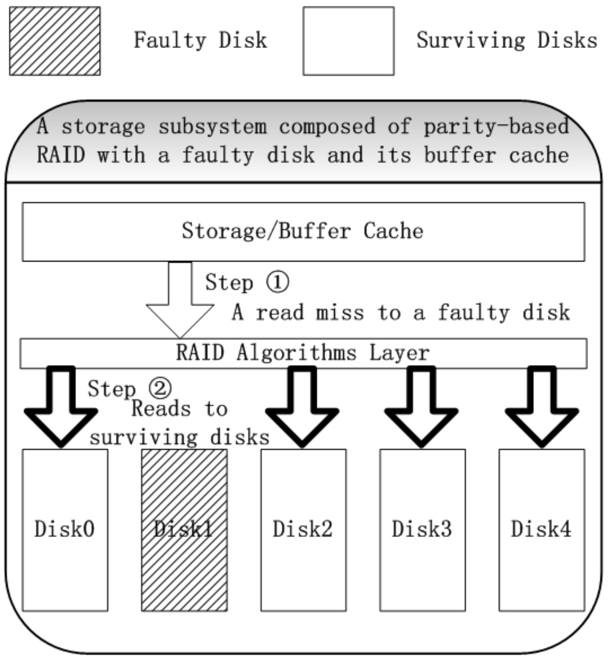
To verify the effectiveness of the VDF cache, we have integrated VDF into two popular cache algorithms LFU and LRU, named VDF-LFU and VDF-LRU, respectively. We have conducted extensive simulations as well as a prototype implementation. The simulation results show that VDF-LFU can reduce disk I/Os to surviving disks by up to 42.3% and VDF-LRU can reduce those by up to 36.2%. Our measurement results also show that VDF-LFU can speed up the online recovery by up to 46.3% under a spare-rebuilding mode with online reconstruction, or improve the maximum system service rate by up to 47.7% under a degraded mode without a reconstruction workload. Similarly, VDF-LRU can speed up the online recovery by up to 34.6%, or improve the system service rate by up to 28.4%.

1 Introduction

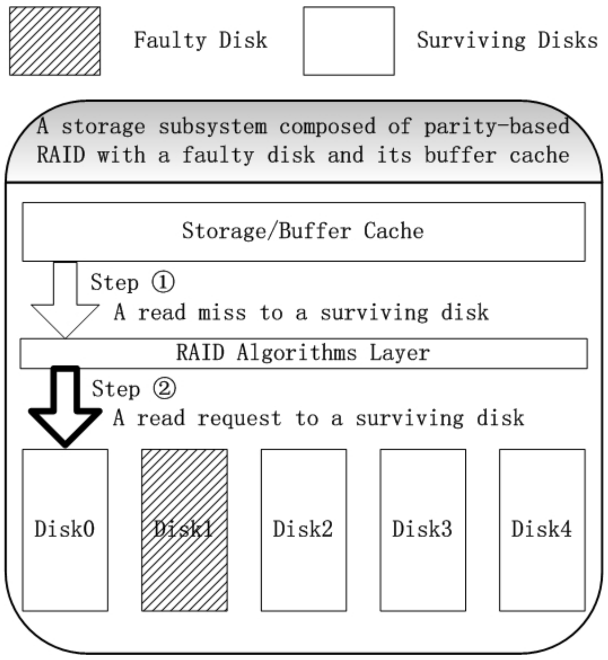
To reduce the number of I/O requests to the low level storage device, such as disk arrays, a cache is widely used and many cache algorithms exist to hide the long disk latencies. These cache algorithms work well for disk arrays under normal fault-free mode. However, when some disks in a disk array fail, the RAID may still work under this faulty scenario, either in a *spare-rebuilding mode* with online reconstruction or in a *degraded mode* without online reconstruction. The cost of a miss to faulty disks might be dramatically different compared to the cost of a miss to surviving disks. Existing cache algorithms cannot capture this difference because they treat the underlying (faulty or surviving) disks the same.

We take an example as shown in Figure 1, which illustrates two different cache miss situations in a storage subsystem composed of a parity-based RAID with one faulty disk in degraded mode. As shown in Figure 1(a), the missed data resides in the faulty disk. The RAID controller accesses the surviving disks to fetch all data and parity in the same stripe to regenerate the lost data. Therefore, to service one cache miss, several read requests are needed depending on the RAID organization. However, if the missed data is in a surviving disk as shown in Figure 1(b), only one read request to the corresponding surviving disk is generated. Similar situations are observed in spare-rebuilding mode. A simple analysis shows that in a RAID5 system consisting of n disks, when a disk fails, the cost to fetch data from a faulty disk might be $n-1$ times higher than the cost to access data from a surviving disk. This extra disk I/O activity will in turn reduce the effective array bandwidth available for reconstruction or user access.

When a disk array starts online reconstruction, it uses up regular bandwidth. Compared to offline reconstruction, during the process of online reconstruction, the user workflow interferes with the reconstruction workflow. As a result, the online reconstruction duration grows significantly compared to offline reconstruction. Wu et al. [1] point out that, in a heavy user workflow, the duration of online reconstruction would grow as much as 70 times as that of the offline reconstruction. In this



(a) A read miss to a faulty disk might result in several additional read requests to the surviving disks.


Faulty Disk
Surviving Disks
A storage subsystem composed of parity-based RAID with a faulty disk and its buffer cache
Storage/Buffer Cache
Step ①
A read miss to a surviving disk
RAID Algorithms Layer
Step ②
A read request to a surviving disk
Disk0
Disk1
Disk2
Disk3
Disk4

(b) A read miss to a surviving disk would result in only one request to the corresponding surviving disk.

Figure 1: Two typical cache-miss situations in a storage subsystem composed of a parity-based RAID in a degraded mode.

case, more requests to the surviving disks, caused by user requests, reduce the available reconstruction bandwidth and lengthen the reconstruction duration, which reduces the reliability of the storage system.

On the other hand, in a degraded mode without a reconstruction workflow, a miss to faulty disks would cause all the surviving data in the same parity chain (stripe in RAID-5) to be read and add additional workflow to surviving disks. With a decreasing serviceability and an increasing user workflow caused by misses to faulty disks, the storage subsystem might be overloaded under a heavy user workflow.

Therefore, in parity-based disk arrays under faulty conditions, a miss to faulty disks is much more expensive than a miss to surviving disks. Based on this observation, we propose an asymmetric buffer cache replacement strategy, named Victim (or faulty) Disk(s) First cache, or VDF for short, to improve the performance of storage subsystem composed of a parity-based disk array and its buffer cache. The basic idea is to design a cache scheme to treat the faulty disks more favorably, or give higher priority to cache the data associated with the faulty disks. The goal of this scheme is to reduce the cache miss directed to the faulty disk, and thus to reduce the I/O requests to the surviving disks overall. Reduced disk I/O caused by the user workflow will (1) improve the performance of the disk array, and (2) allow more bandwidth for online reconstruction which in turn speeds up the recovery, and thus improves the reliability. We make the following four contributions in this paper:

1. We proposed a new metric, *Requests Generation Ratio* or RGR, to capture the disk I/O activities of user workflows on the surviving disks when a storage system is under faulty conditions. This would directly influence the maximum bandwidth for reconstruction in a spare-rebuilding mode and the bandwidth available to user workflows in a degraded mode.

2. We developed a novel cache-replacement scheme, VDF, by giving higher priority to cache the data associated with the faulty disks, to minimize the RGR. VDF is flexible and could be integrated into existing cache algorithms such as LRU and LFU.

3. We conducted extensive simulations to verify the effectiveness of VDF under different workloads. The simulation results show that VDF-LRU can reduce overall disk I/Os to surviving disks by up to 36.2% and VDF-LFU can reduce those by up to 42.3%.

4. We implemented VDF in the Linux software RAID system. As a result, VDF-LFU can speed up the online recovery by up to 46.3% under spare-rebuilding

mode, or improve the maximum system service rate by up to 47.7% under degraded mode. Similarly, VDF-LRU can speed up the online recovery by up to 34.6%, or improve the system service rate by up to 28.4%.

The rest of the paper is organized as follows: Section 2 gives a brief overview of the background information and related work. In Section 3, we describe our new metric, RGR, and the design of VDF. A case study of VDF cache is given in Section 4; we describe integrating VDF into two typical cache-replacement algorithms, LRU and LFU, based on RAID-5. We provide our simulation results of VDF in Section 5, and prototyping and measurement results in Section 6. We conclude our paper and describe future work in Section 7.

2 Background and Related Work

In this section we briefly overview some background materials and related work.

2.1 Optimizations of Disk Arrays under Faulty Conditions

Redundant **A**rrays of **I**ndependent **D**isks **RAID** [2] are popular solutions to provide high performance and reliability for today's storage systems. Depending on its organizations, RAID could prevent data loss incurred by disk failures and even offer online services under faulty conditions. With a faulty disk, these RAIDs would work in a spare-rebuilding mode to support online reconstruction, or in a degraded mode without reconstruction.

RAID can offer continuous online services even in faulty mode. However, the recovery workload and user request can interfere with each other, and lead to longer recovery times. Many solutions are proposed to solve this problem, such as optimizations of data/parity/spare layout [3–7], reconstruction workload [8–12], and user workload [1, 13, 14].

Menon et al. present a method to distribute spares to all disks, which would not only reduce the lost data per disk but also parallelize the reconstruction [4]. Holland et al. [3] propose a trade-off between RAID-1 (mirror) and RAID-5, named parity declustering, to balance the storage efficiency and the recovery performance. Xin et al. use a RUSH-like hash algorithm to evenly distribute data, parity, and spares among the nodes in a distributed environment [5].

The track-based recovery (TBR) [8] algorithm provides a trade-off between block-based recovery and cylinder-based recovery, and balances the user response time and the recovery duration. However, TBR requires much more buffer space compared to block-based recovery. The pipelined recovery (PR) scheme [9] addresses this problem, and significantly reduces the buffer requirements close to that of the block-based recovery algorithms. The disk-oriented recovery (DOR) algorithm [10] rebuilds the array at the disk-level instead of the stripe-level. With this approach, DOR could absorb the bandwidth of the array as much as possible. The popularity-based recovery (PRO) algorithm [11, 12], builds upon the DOR algorithm, further improving the recovery performance by utilizing the spatial locality of user requests.

Two techniques named *redirection of reads* and *piggybacking of writes* [13] are proposed to reduce the user workflow by employing the reconstructed spare disk to absorb parts of the requests to the faulty disk. However, they need to maintain a bitmap in the dedicated cache in the RAID device to record the reconstruction status; as the increasing of disk size, a fine-granularity bitmap would consume too much memory, and increase synchronization costs. For example, a bitmap with granularity of 4KB for a 2TB disk would require 64MB of memory, which limits the use of piggybacking of writes. During the reconstruction, at most a coarse-granularity bitmap could be used only to redirect reads. MICRO [14] achieves improved recovery performance by writing back the in-memory surviving data of the faulty disks into a spare disk first and using a file popularity table to find the hotspot. MICRO treats all the blocks in the cache equally, which is similar to the general cache-replacement algorithms and has the same limitations. WorkOut [1], an array-cache-array method, offloads the write requests and popular read requests to another disk array. As a result, WorkOut speeds up the recovery process and improves the user response time. However, WorkOut requires another disk array to help with the reconstruction and need maintain an addressing translation map, which might be much larger than a fine-granularity bitmap, in the dedicated cache on the RAID device. This suffers from the same problem as redirection of reads and piggybacking of writes.

2.2 Buffer Cache Replacement Algorithms

RAID-based storage systems usually work together with the buffer cache. To improve the efficiency of the buffer cache, researchers have proposed many cache-replacement algorithms, such as LRU [15], LFU, FBR [16], LRU-k [17, 18], 2Q [19], LRFU [20, 21], MQ [22, 23], LIRS [24, 25], ARC [26], DULO [27], DISKSEEN [28] and more. Each cache-replacement algorithm weigh the cached blocks with a different method, such as access interval, access frequency and so on, then decide which cached blocks to evict.

Table 1: Variables and Definitions

Symbols	**Definition**
C	Total number of blocks in the buffer cache
T	Total number of data blocks in a disk array
B_i	Data block i
p_i	Access probability of each block B_i
MP_i	Miss penalty of each block B_i
BW	Total serviceability of all surviving disks in terms of I/O bandwidth
BW_U	I/O bandwidth available to user workload, or service rate of the system from the user's point of view
BW_R	I/O bandwidth for an online reconstruction workload
RGR	The ratio of the # of requested blocks to surviving disks and the # of requested blocks to buffer cache
Q	Total amount of data from surviving disks to reconstruct faulty disks

The LRU (Least-Recently-Used) algorithm is one of the most popular and effective policies for buffer cache management. When a block needs to be inserted into the cache, the candidate to be evicted is the block which is least recently used. That is to say, the weight of the cached blocks in LRU is its last access timestamp. The block with the smallest last access timestamp is evicted. The LFU (Least-Frequently-Used) algorithm replaces the least frequently used block. In other words, the weight of the cached blocks in LFU is its number of accesses. The block with the smallest number of accesses is evicted. Other algorithms, such as LRU-k, 2Q, LRFU, MQ, LIRS, and ARC, integrate LRU and LFU algorithm together and demonstrate good performance under various scenarios. DULO and DISKSEEN consider both temporal and spatial locality when a block needs to be replaced.

However, the above cache-replacement algorithms work well when the RAID system is under normal operating mode. When some disks in the RAID system fail, it runs under faulty condition, but the buffer cache layer is not aware of the underlying failures in RAID and thus the existing cache algorithms do not work well as explained in Section 1. This motivates us to propose VDF: a cache scheme to treat the faulty disks more favorably, or give a higher priority to cache the data associated with faulty disks. The goal is to reduce the cache misses directed to the faulty disk and thus to reduce the I/O requests to the surviving disks overall. As our VDF only increases the weight of blocks in the faulty disks, theoretically it could work with the above-mentioned general cache-replacement algorithms.

3 Design of VDF

In this section, we propose a new metric to describe the cache efficiency of disk I/O activities. We show how to use it to evaluate disk arrays under faulty conditions, and then we describe our VDF scheme. Before our discussion, we summarize the symbols in Table 1.

3.1 RGR: A New Metric to Evaluate Cache Performance with Various Miss Penalty

Traditional cache-replacement algorithms are essentially evaluations on access probability of cached blocks, based on the assumption that the penalty of each miss at the same level is the same. However, in parity-based RAID with faulty disk(s), the penalty of a miss to the lost data in the faulty disks might be much more expensive than that of a miss to surviving data. Therefore, from the aspect of a RAID device, the buffer cache performance should not be simply evaluated by the traditional metrics such as Hit Ratio or Miss Ratio, particularly when the RAID is under faulty conditions. To address this issue, we propose a new metric called *Requests Generation Ratio* or RGR. This is the ratio of the number of requested blocks to the surviving disks and the number of the requested blocks to buffer cache, to evaluate the cache performance from the view point of a faulty RAID device. RGR represents the disk activities to service an I/O request to the buffer cache. Ideally, if all I/O requests are serviced by the buffer cache, RGR will be 0 (no disk I/Os are generated). For missed I/O requests, RGR will be different depending on the penalty to each underlying disk. For example, in Figure 1(a) the RGR of a miss to the faulty disk is 4 because 4 disk I/Os are generated to service the missed I/O request, and in Figure 1(b), the RGR of a miss to surviving disks is 1.

To calculate the RGR, we assume a parity-based RAID of T data blocks with a buffer cache of C blocks. The access probability of a block B_i is p_i, where $0 \leq i \leq T-1$, with a miss penalty of MP_i in terms of the total requested blocks to surviving disks caused by a miss. From the viewpoint of a certain workload, p_i actually represents the ratio of the number of request on B_i and the number of total block requests. If block B_i is not referenced in this workload, p_i should be 0. As we have mentioned above, different cache algorithms evaluate p_i with different approaches in runtime environments. If a block is serviced by the cache, the corresponding miss penalty $MP_i = 0$. Therefore, the RGR of the next block request can be described by the following Equation 1.

$$RGR = \sum_{i=0}^{T-1} (p_i \times MP_i) \tag{1}$$

3.2 Using RGR to Evaluate the Cache Efficiency in Faulty Mode

Consider a system composed of a buffer cache and a RAID in faulty mode which service a certain user workload. We have the following symbols. First, the total serviceability of all surviving disks is BW in terms of I/O bandwidth. Second, the unfiltered user workload would take BW_U bandwidth, which is the service rate of the system from a user's perspective. The average RGR of the buffer cache is RGR. Therefore, the filtered user workload should take about $BW_U \times RGR$ bandwidth. Third, all the remaining bandwidth BW_R of all surviving disks could be utilized for reconstruction. Lastly, the total amount of surviving data for reconstruction is Q. Equation 2 describes the relationships among BW, BW_U, RGR, and BW_R.

$$BW = BW_U \times RGR + BW_R \tag{2}$$

We first consider the spare-rebuilding mode. The surviving disks would suffer from more requests as explained in Section 1. It means that the I/O bandwidth available for reconstruction on the surviving disks would be less than the I/O bandwidth for reconstruction on the spare disk. The total amount of requested data for reconstruction on each disk (including the surviving disks and the spare disks) is the same. Therefore, to the online recovery process, the I/O bandwidth for reconstruction on the surviving disks is the bottleneck. The reconstruction duration RD could be described with Equation 3.

$$RD = \frac{Q}{BW - BW_U \times RGR} \tag{3}$$

From Equation 3, we can find that, if Q, BW, and BW_U are fixed, with the decreasing RGR, the reconstruction duration RD (and thus MTTR) decreases. Therefore, to minimize the MTTR, we should minimize the RGR.

We next consider the degraded mode without reconstruction. Each surviving disk would suffer from the extra requests caused by the access to faulty disks. The filtered user workload should not exceed the total serviceability of all surviving disks. In another words, the maximum unfiltered user workload BW_U should not exceed $\frac{BW}{RGR}$. Therefore, we should minimize the RGR to maximize the system serviceability, which is described with a maximum BW_U.

From the above discussion, we notice that compared to the traditional metrics on cache evaluation, such as miss ratio, RGR is a useful metric to demonstrate two important indicators of a faulty disk array more clearly and directly. One is the reconstruction time which is directly related to MTTR and affects the system reliability. The other is the throughput that indicates the performance of the storage system.

3.3 VDF Cache

Based on the above analysis, we propose our VDF cache aiming at reducing the RGR for parity-based RAID under faulty conditions, either to enhance the system reliability by speeding up the reconstruction process in spare-rebuilding mode, or to improve the system performance by increasing the system serviceability in degraded mode without reconstruction workloads. As it operates at the buffer cache level, VDF is practical and does not suffer from the same problems of the small dedicated cache in a RAID controller.

Cache-replacement algorithms are essentially evaluations on access probability of cached blocks. Once a miss occurs, typically a block should be evicted from cache, and the missed block would be loaded to the free space. General replacement algorithms evict the block with the smallest access probability to reduce the total access probability of the remaining blocks out of buffer cache. However, to minimize the RGR, the eviction of a block should not only be determined by the access probability but also by the miss penalty. In our VDF cache, we adopt the same evaluation approach of access probability p_i for each cached block as the general cache. Furthermore, the miss penalty MP_i of each block is evaluated with the requested blocks to the surviving disks of this block; the block with the minimum product of $p_i \times MP_i$ is evicted from the cache to minimize the RGR.

4 A Case Study of VDF

To verify the effectiveness of VDF, we apply VDF in a RAID-5 system of n disks where one disk fails. We focus on read operations for two reasons: first, in many applications, users are typically sensitive to read latency, particularly in a disk array under faulty conditions; second, in many storage systems, independent non-volatile memory is deployed as a write cache to enhance the reliability and this cache uses a dedicated write cache algorithm.

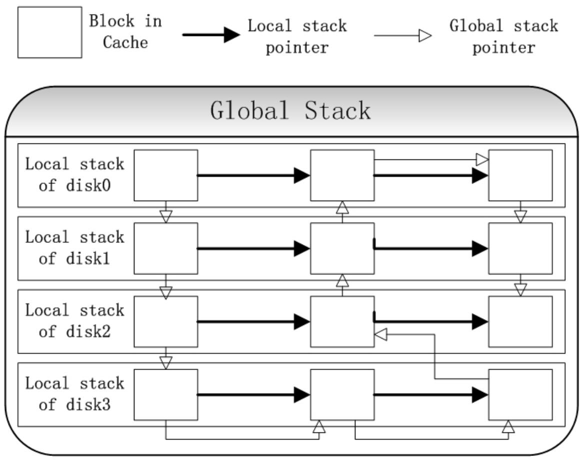


Figure 2: VDF implementation with two types of stacks.

4.1 Integrating VDF into LRU and LFU

Although VDF cache can cooperate with caches at other levels by adjusting the miss penalty of blocks, for demonstration purposes we just consider a one-level buffer cache above the disk array. Therefore, the miss penalty of blocks on the faulty disk would be $n - 1$ in our following discussion, which means one cache miss to the faulty disk will result in $n - 1$ I/O requests to the surviving disks. To integrate VDF with any cache algorithm, the access probability should be evaluated with a quantitative approach. Different cache-replacement algorithms evaluate the access probability of blocks using different approaches. Most existing cache management algorithms can be categorized into LRU-like and LFU-like algorithms. In LRU-like algorithms, the weight of blocks is often evaluated by the access time interval. As it is costly to record the real access timestamp, a simple alternative is to record the access sequence number, and use the reciprocal of the interval access sequence number as the access probability. This approach is widely used in many LRU-like algorithms. In LFU-like algorithms, the weight of blocks is majorly evaluated by the access frequency. Thus, to integrate VDF into these LFU-like algorithms, it needs only to keep the original evaluation approach. Furthermore, different from the access sequence number, the access frequency of two blocks might be the same. Therefore, in VDF based LFU-like algorithms, the access sequence number is also employed for choosing which block to evict with the same access frequency. In VDF cache the access probability of a block would not be the absolute but the relative value, because both the reciprocal of interval of access sequence number and the access frequency are actually the relative values.

The conversion from the original cache algorithms to the VDF-based algorithms should be smooth, because VDF takes effect in faulty mode. In other words, the buffer cache should be managed with the original algorithms in fault-free mode, and the VDF policy becomes effective when disk failures occur. Thus, a smooth runtime conversion between the original algorithm and the VDF-based algorithm is needed, which is quite different from the general cache algorithms. Therefore, in VDF-based algorithm, we employ two types of stacks to achieve the smooth runtime conversion: one is the global stack (GS) which is similar to the stack in a general algorithm such as global LRU stack, and the other is the local stack (LS) holding the blocks on the same disk in cache. All blocks should be in two types of stacks concurrently as shown in Figure 2. When the system works in fault-free mode, it evicts the block with the smallest weight at the bottom of the GS stack. Once a disk array drops to a faulty mode, it evicts the block with the smallest weight at the bottom of each LS stack instead of evicting the block at the bottom of the GS stack.

Algorithm 1: VDF-LRU for RAID-5 with n disks

```
Input: The request stream $x_1, x_2, x_3, ..., x_i, ...$
VDF_LRU_Replace($x_i$){
/*For every $i \geq 1$ and any $x_i$, one and only one of the following cases
must occur.*/
if $x_i$ is in $LS_k$, $0 \leq k < n$ then
    /*A cache hit has occurred.*/
    Update $TS$ of $x_i$, by $TS = GTS$;
    Move $x_i$ to the heads of $LS_k$ and $GS$.
else
    /*A cache miss has occurred.*/
    if Cache is full then
        foreach block at the bottom of $LS_j$, $0 \leq j < n$ do
            if $LS_j$ is a corresponding stack to a faulty disk then
                Its weight $W$=$GTS - TS$;
            else
                Its weight $W$=$(GTS - TS) * (n - 1)$;
        Delete the block with maximum $W$ to obtain a free block;
    else
        /*Cache is not full.*/
        Get a free block.
    Load $x_i$ to the free block.
    Update $TS$ of $x_i$, by $TS = GTS$;
    Add $x_i$ to the heads of $GS$ and the corresponding $LS$.
Update $GTS$, by $GTS = GTS + 1$;
}
```

4.2 Detailed Description of VDF-LRU and VDF-LFU

Detailed descriptions of VDF-LRU and VDF-LFU for n-disk RAID-5 are given in Algorithm 1 and Algorithm 2, respectively, using the variables summarized in Table 2.

5 Simulation Results and Analysis

To evaluate the effectiveness of VDF, we conducted simulations under three typical workloads: SPC-1-web, LM-TBE, and DTRS.

Table 2: Variants in VDF and Explanation

Variants	Explanation
x	A block request to buffer cache
LS	The local stack holding the blocks on one certain disk in the buffer cache
GS	The global stack holding all the blocks on all the disks in the buffer cache
n	The total number of disks including the faulty disk and surviving disks
TS	The timestamp of a block: records the access sequence number
F	The access frequency of a block
GTS	The global timestamp: it is equal to the timestamp of currently accessed block
W	The weight of a block

Algorithm 2: VDF-LFU for RAID5 of n disks

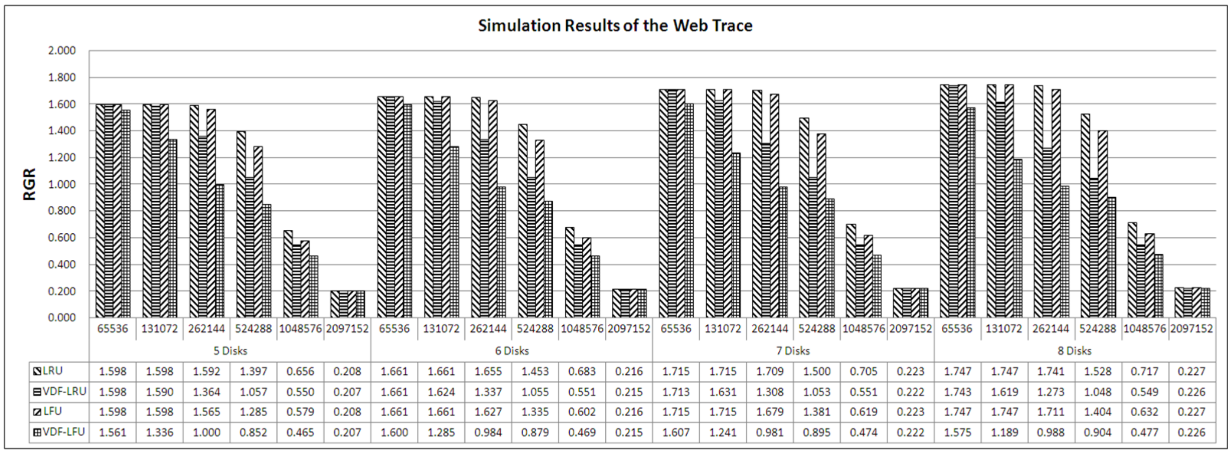
```
Input: The request stream $x_1, x_2, x_3, ..., x_i, ...$
VDF_LFU_Replace($x_i$){
/*For every $i \geq 1$ and any $x_i$, one and only one of the following cases
must occur.*/
if $x_i$ is in $LS_k$,$0 \leq k < n$ then
    /*A cache hit has occurred.*/
    Update $F$ and $TS$ of $x_i$, by $F = F + 1$;
    Move $x_i$ to right place of $LS_k$ and $GS$ according to $F$ and $TS$.
else
    /*A cache miss has occurred.*/
    if Cache is full then
        foreach block at the bottom of $LS_j$, $0 \leq j < n$ do
            if $LS_j$ is a corresponding stack to a faulty disk then
                Its weight $W$=$F * (n - 1)$;
            else
                Its weight $W$=$F$;
        Delete the block with minimum $W$ and $GTS - TS$ to obtain
        a free block;
    else
        /*Cache is not full.*/
        Get a free block.
    Load $x_i$ to the free block.
    Initialize the frequency $F$ and $TS$ of $x_i$, by $F = 1$ and
    $TS = GTS$;
    Move $x_i$ to right place of $LS_k$ and $GS$ according to $F$ and $TS$.
Update $GTS$, by $GTS = GTS + 1$;
}
```

SPC-1-web, a trace used in the SPC-1 benchmark suites, was collected in a search engine, which is widely used in the evaluation of storage systems [1,11,14]. LM-TBE and DTRS are provided by Microsoft Corporation collected in 2008. The LM-TBE trace was collected in back-end servers supporting a front-end Live Maps application. The DTRS trace was collected in a file server accessed by more than 3000 users to download various daily builds of Microsoft Visual Studio. Both traces were taken in a period of 24 hours and broken into pieces with 1-hour intervals [29]. We choose only the piece with most intensive I/O activities. For fairness and simplicity, we consider only the read operations and all block sizes are 4KB. We report RGR of LRU, LFU, VDF-LFU, and VDF-LRU under these workloads as shown in Figures 3, 4, and 5, respectively.

Our simulator, named VDF-Sim, is written in C and the source code is approximately 3000 lines. It slices/splits the trace records into block requests as the input. Data blocks in a stack or blocks with the same hash values are linked via double circular lists. For a certain block in our simulator, we record its logical offset as the unique ID since the disk array is transparent to the upper level systems such as a file system. According to the data/parity distribution of the disk array and the logical offset of a block, it is easy to identify on which disk the block resides. The arriving timestamps of the requests are also recorded to evaluate p_i as we mentioned in Section 4 and to generate the misses trace used in the prototype discussed in the next section.

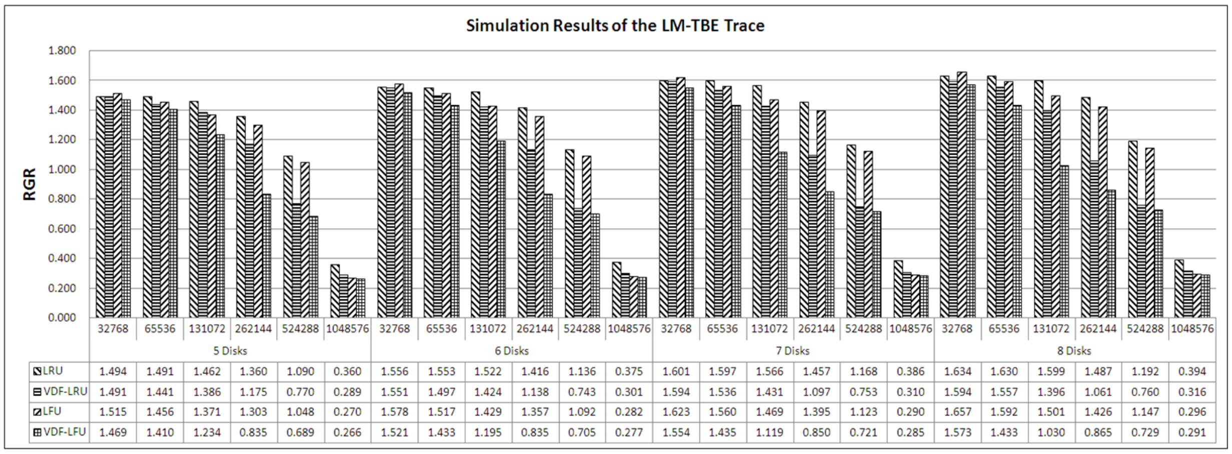
The results show that, compared to the original LRU and LFU algorithms, VDF optimized algorithms achieved better performance consistently by reducing the RGR. Compared to LRU, VDF-LRU reduces the RGR by up to 31.4%, 36.2%, and 22.7% under SPC-1-web, LM-TBE, and DTRS traces, respectively. Compared to LFU, VDF-LFU reduces the RGR by up to 42.3%, 39.4%, and 24.4%, respectively.

We find that the efficiency of VDF grows with the increased number of disks under the same number of cache-resident blocks in most cases. The efficiency of VDF is more significant with a moderate number of cache-resident blocks than that with a too small or too large number of cache-resident blocks. This can be explained as follows. The cache-resident blocks of a faulty disk in the original algorithm would occupy $1/n$ cache space with total n disks. With the fixed cache-resident blocks and the increased n, the number of cache-resident blocks of the faulty disk would be smaller. From the aspect of cache management, the impact of the marginal utility of blocks on hit ratio tends to decrease with the increased cache size. For example, adding $P1$ blocks to a cache with $P2$ blocks might improve the hit ratio with a larger gain compared to adding $P1$ blocks to cache with $P3$ blocks when $P2 < P3$. Thus, the marginal utility of blocks would be more obvious with more disks and thus the efficiency of VDF grows accordingly. However, if the number of cache-resident blocks is too small, it is hard to find hot blocks even with an extended period due



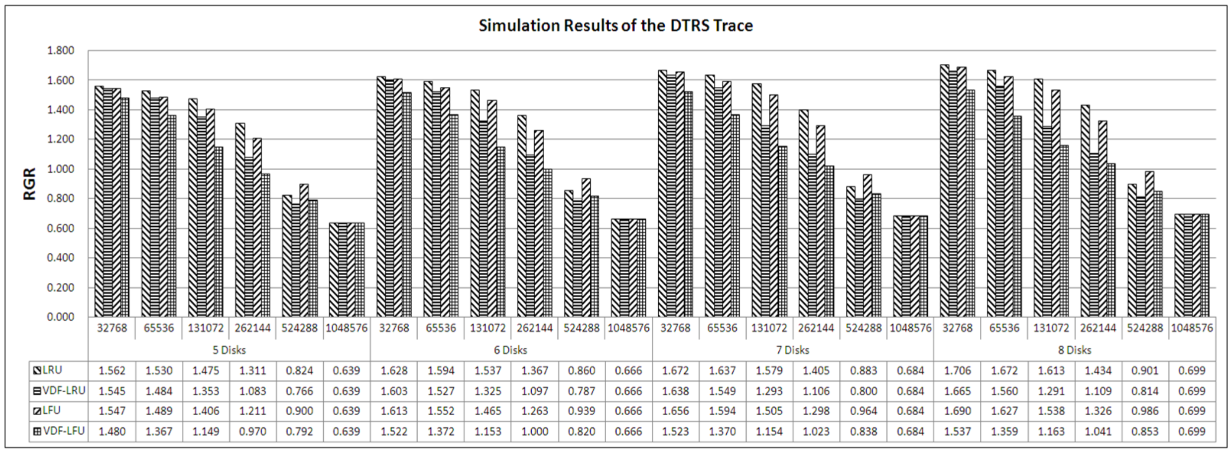
	65536	131072	262144	524288	1048576	2097152	65536	131072	262144	524288	1048576	2097152	65536	131072	262144	524288	1048576	2097152	65536	131072	262144	524288	1048576	2097152
	5 Disks						6 Disks						7 Disks						8 Disks					
▧LRU	1.598	1.598	1.592	1.397	0.656	0.208	1.661	1.661	1.655	1.453	0.683	0.216	1.715	1.715	1.709	1.500	0.705	0.223	1.747	1.747	1.741	1.528	0.717	0.227
⊟VDF-LRU	1.598	1.590	1.364	1.057	0.550	0.207	1.661	1.624	1.337	1.055	0.551	0.215	1.713	1.631	1.308	1.053	0.551	0.222	1.743	1.619	1.273	1.048	0.549	0.226
▨LFU	1.598	1.598	1.565	1.285	0.579	0.208	1.661	1.661	1.627	1.335	0.602	0.216	1.715	1.715	1.679	1.381	0.619	0.223	1.747	1.747	1.711	1.404	0.632	0.227
⊞VDF-LFU	1.561	1.336	1.000	0.852	0.465	0.207	1.600	1.285	0.984	0.879	0.469	0.215	1.607	1.241	0.981	0.895	0.474	0.222	1.575	1.189	0.988	0.904	0.477	0.226

Figure 3: Simulation results under the SPC-1-web trace. The number of disks ranges from 5 to 8, and number of cache blocks varies from 64K to 2M with the block size of 4KB.



	32768	65536	131072	262144	524288	1048576	32768	65536	131072	262144	524288	1048576	32768	65536	131072	262144	524288	1048576	32768	65536	131072	262144	524288	1048576
	5 Disks						6 Disks						7 Disks						8 Disks					
▧LRU	1.494	1.491	1.462	1.360	1.090	0.360	1.556	1.553	1.522	1.416	1.136	0.375	1.601	1.597	1.566	1.457	1.168	0.386	1.634	1.630	1.599	1.487	1.192	0.394
⊟VDF-LRU	1.491	1.441	1.386	1.175	0.770	0.289	1.551	1.497	1.424	1.138	0.743	0.301	1.594	1.536	1.431	1.097	0.753	0.310	1.594	1.557	1.396	1.061	0.760	0.316
▨LFU	1.515	1.456	1.371	1.303	1.048	0.270	1.578	1.517	1.429	1.357	1.092	0.282	1.623	1.560	1.469	1.395	1.123	0.290	1.657	1.592	1.501	1.426	1.147	0.296
⊞VDF-LFU	1.469	1.410	1.234	0.835	0.689	0.266	1.521	1.433	1.195	0.835	0.705	0.277	1.554	1.435	1.119	0.850	0.721	0.285	1.573	1.433	1.030	0.865	0.729	0.291

Figure 4: Simulation results under the LM-TBE trace with various numbers of disks and cache blocks. The block size is 4KB.



	32768	65536	131072	262144	524288	1048576	32768	65536	131072	262144	524288	1048576	32768	65536	131072	262144	524288	1048576	32768	65536	131072	262144	524288	1048576
	5 Disks						6 Disks						7 Disks						8 Disks					
▧LRU	1.562	1.530	1.475	1.311	0.824	0.639	1.628	1.594	1.537	1.367	0.860	0.666	1.672	1.637	1.579	1.405	0.883	0.684	1.706	1.672	1.613	1.434	0.901	0.699
⊟VDF-LRU	1.545	1.484	1.353	1.083	0.766	0.639	1.603	1.527	1.325	1.097	0.787	0.666	1.638	1.549	1.293	1.106	0.800	0.684	1.665	1.560	1.291	1.109	0.814	0.699
▨LFU	1.547	1.489	1.406	1.211	0.900	0.639	1.613	1.552	1.465	1.263	0.939	0.666	1.656	1.594	1.505	1.298	0.964	0.684	1.690	1.627	1.538	1.326	0.986	0.699
⊞VDF-LFU	1.480	1.367	1.149	0.970	0.792	0.639	1.522	1.372	1.153	1.000	0.820	0.666	1.523	1.370	1.154	1.023	0.838	0.684	1.537	1.359	1.163	1.041	0.853	0.699

Figure 5: Simulation results under the DTRS trace with various numbers of disks and cache blocks. The block size is 4KB.

to the large access interval. On the other hand, if the number of cache-resident blocks is too large, most of the requested blocks from the faulty disks would be cached, and the marginal utility of blocks becomes insufficient.

We also find that the VDF strategy becomes more efficient with LFU than LRU under the three workloads. One possible reason is that the temporal locality of these traces is weak as they are server-end traces and already filtered by upper level caches. Thus, the *Stack Depth Distribution* property of these traces is weak. As a result, the *Independent Reference Model* property of these traces would be relatively improved. Although we improve the weight of the blocks on faulty disk to $n-1$ times in both VDF-LRU and VDF-LFU, the efficiency is not the same. Mostly, the caching duration of blocks from a victim disk in VDF-LFU is longer than that in VDF-LRU, especially when the total number of cache-resident blocks is not large. Therefore, VDF-LFU works better than VDF-LRU in these traces in most cases.

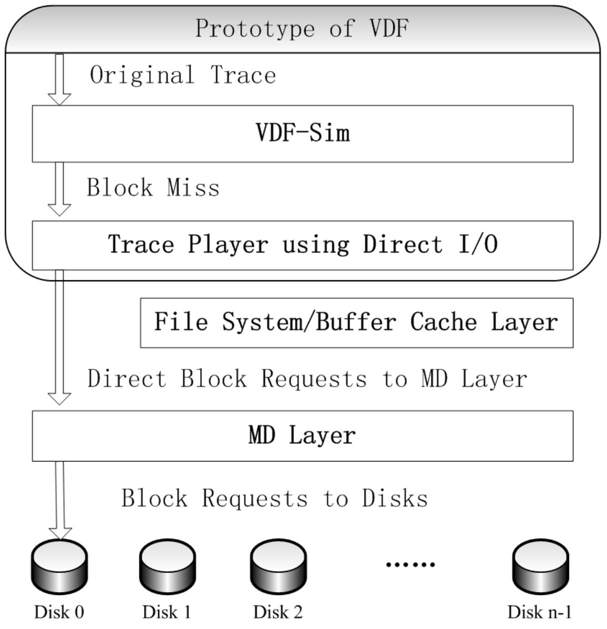


Figure 6: Architecture of VDF prototype.

6 Prototyping of VDF

To further evaluate VDF, we implemented a prototype of VDF in a software RAID system in Linux known as *MD*. In this section, we present our measurement results, including the efficiency of online recovery duration in full-bandwidth reconstruction mode and system service rate under the degraded mode without reconstruction.

6.1 Evaluation Methodology

Measurements on real world systems are welcome in research of computer systems. However, implementation in a real system is a lengthy process and always complex and challenging. Here, we use a straightforward and accurate measurement approach to evaluate the efficiency of VDF. The architecture of our prototype is shown in Figure 6. First, we collect the cache miss information during our simulation in Section 5, which includes not only the block ID but also the real access timestamps. Then, we treat the RAID as a file device, and use an application in user mode to play the traces we have collected from our simulations which is similar to RAID-meter [1, 11]. However, the difference is that our application uses direct I/O (available in Linux 2.6 and up) instead of buffered I/O to avoid the requests being recached by the file system buffer cache. All missed I/O requests sent to the MD layer directly. Thus our simulation and the application join together to exploit the buffer cache and replacement algorithm. The trace player is also written in C and the source code is approximately 500 lines.

In our experiment, we evaluated the effectiveness of VDF, including the online reconstruction duration in full-bandwidth reconstruction mode, and the system serviceability in the degraded mode without reconstruction. For online reconstruction in full-bandwidth reconstruction mode, an open-loop measurement approach is adopted, where all filtered traces are played according to their timestamps as recorded in the original trace file. For degraded mode, a closed-loop measurement approach is adopted, where all filtered traces are played only according to their original sequence one by one and without any interval, to find the system serviceability.

6.2 Experimental Environment

In our experiment, we employ a SuperMicro storage server with two Intel(R) Xeon(R) X5560 @2.67GHz (six cores) CPUs, 12GB DDR3 main memory. All disks are Western Digital WD10EALS Caviar Blue SATA2, which are connected by an Adaptec 31605 SAS/SATA RAID controller with a 256MB dedicated cache. We disabled the RAID function of the controller and only used the direct I/O mode to connect each disk. The operating system is Linux Fedora 12 x86_64 with the kernel version of 2.6.32.

In Linux, there is a software implementation of RAID called Multiple Devices *MD*, which is popular in verification of RAID optimization scheme [1, 11]. To facilitate the analysis and verification of VDF cache, we also used MD as our experimental platform. We used the default settings of MD: the chunk size is 64KB, the number of stripe-heads is 256 and the data layout is left-symmetric. In our open-loop testing, the minimum

Table 3: Experimental Results of an Open-loop Testing Using 5 to 8 Disks

Disks	**Blocks**	**LRU (s)**	**VDF-LRU (s)**	**Improvement**	**LFU (s)**	**VDF-LFU (s)**	**Improvement**
5 disks	131072	2662	2543	4.5%	2710	1929	28.8%
	262144	2958	1935	34.6%	2851	1531	46.3%
	524288	1845	1407	23.7%	1786	1310	26.7%
6 disks	131072	1176	1147	2.5%	1175	964	18.0%
	262144	1234	943	23.6%	1226	921	24.9%
	524288	1027	818	20.4%	1005	806	19.8%
7 disks	131072	730	685	6.2%	733	652	11.1%
	262144	758	659	13.1%	761	657	13.7%
	524288	691	599	13.3%	687	598	13.0%
8 disks	131072	504	485	3.8%	509	485	4.7%
	262144	558	501	10.2%	560	501	10.5%
	524288	527	483	8.4%	526	479	8.9%

reconstruction bandwidth is set to 100MBps to utilize all remaining bandwidth for reconstruction besides the bandwidth taken by user workloads.

As VDF targets the storage server consisting of disk arrays which run under faulty conditions, we chose the server-end trace SPC-1-web as our experimental material, which spans a 60GB dataset. The workload is filtered by 131,072 to 524,288 blocks in our simulation to generate the experimental inputs. In the open-loop measurement, we test VDF with 5 to 8 disks. The results are reported in terms of reconstruction speed. The improvements of VDF over the original LRU and LFU are calculated. In the close-loop measurement, we used a multi-threaded application to play the filtered trace to measure the service rate. We also evaluated the impact of thread number to service rate, in addition to the impact of the number of blocks and the number of disks. The results are reported as system service rate improvement by VDF cache compared to original LFU and LRU. We ran each test three times and report the average. The results are very stable and consistent as the difference among all three rounds was very small (less than 5%).

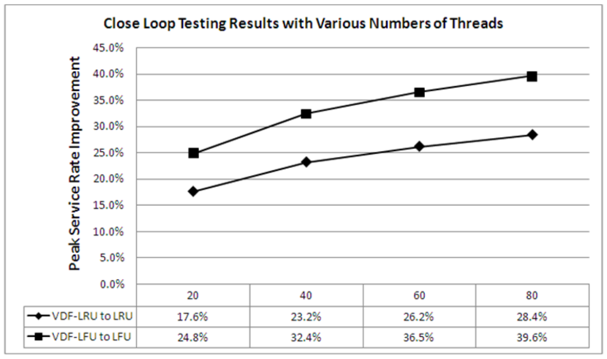
6.3 Open-loop Measurement Results and Analysis

Table 3 describes the results under an open-loop testing using the SPC-1-web trace, where the number of disks ranges from 5 to 8 and the number of blocks ranges from 131,072 to 524,288. The experimental results of the open-loop testing show that compared to the original LRU and LFU algorithms, the VDF-optimized algorithm speeds up the online reconstruction process. The speedup of VDF-LFU is up to 46.3% compared to LFU. VDF-LRU speeds up the online reconstruction by up to 34.6% compared to LRU.

With the same number of cache-resident blocks, the experimental results show that the improvement of reconstruction durations of VDF-LFU to LFU decreases with the increased number of disks. A similar trend is also observed for the improvement of VDF-LRU over LRU, except for a smaller improvement of RGR using VDF when the number of blocks is 131,072. These trends are in contrast to our previous simulation results where the efficiency of VDF tends to be more sufficient with increased number of disks. From Equation 3, the improvement of reconstruction durations of VDF-LFU to LFU, which is presented by $Imprv$, could be described with Equation 4. $BW_U \times RGR_{VDF}$ is always less than BW otherwise the system would be overloaded, so $\frac{BW}{RGR_{VDF}}$ is larger than BW_U. VDF works in most cases where $\frac{RGR_{ORI}}{RGR_{VDF}}$ is larger than 1, thus $\frac{BW_U * RGR_{ORI}}{RGR_{VDF}}$ is larger than BW_U. Therefore, the change rate of improvement according with the number of total disks should only be determined by the changing rates of $BW_U \times RGR_{ORI}$ and BW. Obviously, BW linearly grows with the total number of disks. From the simulation result, we can find that the changing rate of $BW_U \times RGR_{ORI}$ is slower than that of BW with the number of disks from 5 to 8. Therefore, in most of the above cases, the improvement of reconstruction durations of VDF cache to original cache decreases with the increased number of disks.

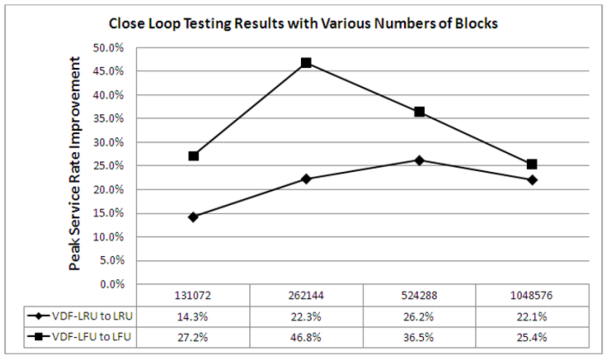
$$RD_{Imprv} = \frac{\frac{BW_U * RGR_{ORI}}{RGR_{VDF}} - BW_U}{\frac{BW}{RGR_{VDF}} - BW_U} \quad (4)$$

With the same number of disks, we notice that the reconstruction duration of the trace filtered by 131,072 blocks is less than the reconstruction duration of trace filtered by 262,144 blocks in many cases. On one hand, as we use a number of blocks to warm up the cache, this part of the miss information is not recorded in our filtered trace file. The first 131,072 block misses in the



	20	40	60	80
VDF-LRU to LRU	17.6%	23.2%	26.2%	28.4%
VDF-LFU to LFU	24.8%	32.4%	36.5%	39.6%

Figure 7: Service rate improvement of VDF in degraded mode of a RAID-5 of 8 disks. The number of blocks is 524,288 and the number of threads ranges from 20 to 80.



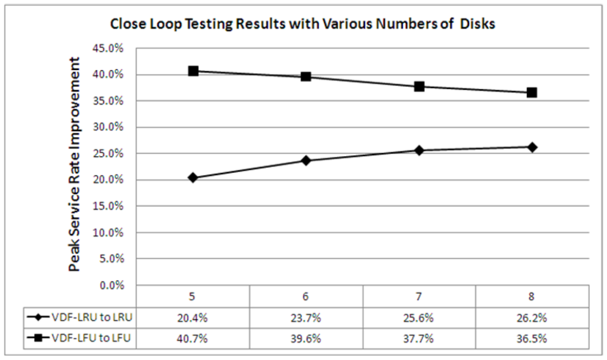
	131072	262144	524288	1048576
VDF-LRU to LRU	14.3%	22.3%	26.2%	22.1%
VDF-LFU to LFU	27.2%	46.8%	36.5%	25.4%

Figure 8: Service rate improvement of VDF in degraded mode of a RAID-5 of 8 disks. The number of blocks ranges from 131,072 to 524,288 and the number of threads is 60.

trace filtered by 262,144 blocks has a lower average arrival rate than the remaining part. On the other hand, when the number of blocks in the cache is 131,072 or 262,144, the cache is too small to find the hot blocks, which implies fewer hits in those cases and the RGRs are similar. Therefore, the reconstruction duration of the trace filtered by 131,072 blocks might be less than that of the trace filtered by 262,144 blocks in those cases.

6.4 Close-loop Measurement Results and Analysis

Figures 7, 8, and 9 present the close-loop testing results under different scenarios with various numbers of threads, disks, and data blocks. The results are reported as service rate improvement, which is inversely proportional to the *Play Duration* (**PD**) of a whole filtered trace. For example, the corresponding service rate improvement of VDF-LRU to LRU should be calculated by $\frac{PD_{LRU}-PD_{VDF-LRU}}{PD_{VDF-LRU}}$.



	5	6	7	8
VDF-LRU to LRU	20.4%	23.7%	25.6%	26.2%
VDF-LFU to LFU	40.7%	39.6%	37.7%	36.5%

Figure 9: Service rate improvement of VDF in degraded mode of a RAID-5 of 5 to 8 disks. The number of blocks is 524,288 and the number of threads is 60.

From the experimental results, we find that VDF is effective in improving the system service rate. Compared to LFU, VDF-LFU improves the system service rate up to 46.8% with 60 threads under 8 disks and 262,144 blocks. Compared to LRU, VDF-LRU improves the system service rate by up to 28.4% with 80 threads under 8 disks and 524,288 blocks.

With the increasing number of I/O threads, the service rate improvement increases accordingly and gets close to the theoretical value calculated by the simulation results. Although the whole trace would be evenly distributed on all disks due to the round-robin addressing in RAID, the incoming user requests might not be evenly distributed on all the disks during a short period. Therefore, with a larger number of threads, which implies a longer scheduling window, the distribution of incoming user requests is more balanced and the user service rate is closer to the maximum system service rate.

With the same number of blocks and a fixed number of threads, the service rate improvement of VDF-LRU to LRU is consistent with the trend of the simulation result. However, to our surprise, the results of VDF-LFU to LFU were just opposite with the trend of the simulation result. As per our analysis, this was primarily due to two reasons. First, from the simulation result, the relative RGR reduction of VDF-LFU to LFU with 524,288 blocks is in a small area from 33.7% to 35.6%. Second, the number of concurrent threads is fixed, which means that the number of threads per disk would increase with the decreased total number of disks. Thus, based on the above analysis, when the total number of disks is small, the improvement is closer to the theoretical value. Therefore, under close to theoretical peak service rates and more I/O threads per disk, the trend of service rate improvement of VDF-LFU to LFU is very possibly opposite with the trend of the simulation result. As a result, with the same number of disks and a fixed number of threads, which means a fixed number of I/O threads per disk, the service rate improvement is quite consistent with the trend of the simulation results.

6.5 Further Discussion

Several more issues deserve further discussion. The first issue is the implementation cost of VDF. As we mentioned in Section 4, to make the smooth conversion between the original cache algorithms and the VDF-based algorithms, two types of stacks should be employed to implement VDF cache. This adds both spatial and temporal overhead. The spatial overhead include the extra information in each block head such as the timestamp and the extra stack pointer of the local stack. Compared to the buffer cache size, this overhead is very small. The temporal overhead is the computation of the weight of block at the bottom of each LS stack. Due to the high computation ability of today's CPU, this should not influence the overall system performance.

Second, can we integrate VDF into other optimizations in faulty mode? As the VDF cache essentially reduces the user requests to the surviving disks, it can be integrated with other optimizations in faulty mode, such as optimization on data/parity/spare layout and reconstruction workloads. The approach of redirection of reads utilizes the reconstructed data in a spare disk to serve part of the reads to the faulty disk. Thus the miss penalty of these reconstructed data block is zero in terms of extra requests to the surviving disks. There still exist hot data with large miss penalty on faulty disks. Therefore, VDF can still help.

Third, could RGR be suitable to describe the status of write operation? From its definition, RGR is determined by MP_i and p_i. The calculation of p_i in write operations is similar to read operations. However, the calculation of MP_i in write operations is quite different from read operations, as they might be done with two approaches based on the parity distribution in RAID with faulty disk(s). One is the Read-Modify-Write, and the other is Parity-Reconstruction-Write. Here, we take an example of short writes on an n-disk RAID-5 with one faulty disk to demonstrate the MP_i calculation for write operations. Once a short write is sent to the surviving disks and the corresponding parity is not on the faulty disk, the Read-Modify-Write should be performed which results in two reads and two writes on the surviving disks, and thus the MP_i is 4. Otherwise, the Parity-Reconstruction-Write should be performed which results in $n-1$ reads and one write on the surviving disks, and thus the MP_i is n.

7 Conclusions and Future Work

In this paper, we present an asymmetric buffer cache replacement strategy, named Victim (or faulty) Disk(s) First (VDF) cache, to improve the reliability and performance of a RAID-based storage system, particularly under faulty conditions. The basic idea of VDF is to treat the faulty disks more favorably, or give a higher priority to cache the data associated with the faulty disks. The benefit of this scheme is to reduce number of the cache miss directed to the faulty disk, and thus to reduce the I/O requests to the surviving disks overall. Less disk I/O activity caused by the user workflow will (1) improve the performance of the disk array, and (2) allow more bandwidth for online reconstruction which in turn speeds up the recovery, and thus improves the reliability. Our results based on both simulation and prototyping implementation has demonstrated the effectiveness of VDF in terms of reduced disk I/O activities and a faster recovery.

To further understand VDF, we have the following plans as our future work. First, we plan to build VDF into more general cache algorithms such as CLOCK [30] and ARC [26]. Second, we are working to implement VDF in the kernel level and thus to directly run real benchmarks to conduct more extensive measurements. Third, in addition to RAID-5, we will investigate the scheme to apply VDF to other RAID levels such as RAID-4 and RAID-6.

Acknowledgments

We are very grateful to our shepherd Erez Zadok and anonymous reviewers for their helpful comments. This research is sponsored in part by the National Basic Research 973 Program of China under Grant No. 2011CB302303, the National Natural Science Foundation of China under Grant No. 60933002, the National 863 Program of China under Grant No. 2009AA01A402, and the Innovative Foundation of Wuhan National Laboratory for Optoelectronics. The author He's work is supported by the U.S. National Science Foundation (NSF) under Grant Nos. CCF-1102605, CCF-1102624, and CNS-1102629. 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 funding agencies.

References

[1] S. Wu, H. Jiang, D. Feng, L. Tian, and Bo Mao. WorkOut: I/O workload outsourcing for boosting RAID reconstruction performance. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies*, San Francisco, USA, February 2009.

[2] D. A. Patterson, G. A. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the*

1988 ACM SIGMOD international conference, Chicago, Illinois, USA, June 1988.

[3] M. Holland and G. A. Gibson. Parity declustering for continuous operation in redundant disk arrays. In *Proceedings of the 5th Conference on Architectural Support for Programming Languages and Operating Systems*, pages 23–35, Boston, Massachusetts, USA, October 1992.

[4] J. Menon and D. Mattson. Distributed sparing in disk arrays. In *Proceedings of the 37th international conference on COMPCON*, pages 410–421, San Francisco, California, USA, Feb 1992.

[5] Q. Xin, P. L. Miller, and T. J. E. Schwarz. Evaluation of distributed recovery in large-scale storage systems. In *Proceedings of 13th IEEE International Symposium on High Performance Distributed Computing*, pages 172–181, June 2004.

[6] G. K.M, X. Li, and J. J. Wylie. Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs. In *IEEE 26th Symposium on Mass Storage Systems and Technologies*, Incline Village, NV, May 2010.

[7] S. Wan, Q. Cao, C. S. Xie, B. Eckart, and X. He. Code-M: A non-MDS erasure code scheme to support fast recovery from up to two-disk failures in storage systems. In *IEEE/IFIP International Conference on Dependable Systems and Networks*, Chicago, IL, USA, June 2010.

[8] R. Y. Hou, J. Meno, and Y. N. Patt. Balancing I/O response time and disk rebuild time in a RAID5 disk array. In *Proceeding of the Twenty-Sixth Hawaii International Conference on System Sciences*, pages 70–79, Jan 1993.

[9] J. Y.B. Lee and J. C.S. Lui. Automatic recovery from disk failure in continuous-media servers. *The Computer Journal*, 13(5):499–515, May 2002.

[10] M. Holland, G. A. Gibson, and D. P. Siewiorek. Fast, on-line failure recovery in redundant disk arrays. In *The Twenty-Third International Symposium on Fault-Tolerant Computing*, pages 422–431, Toulouse , France, Jun 1993.

[11] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, Z. Wang, and Z. Song. PRO: A popularity-based multi-threaded reconstruction optimization for RAID-structured storage systems. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, pages 277–290, February 2007.

[12] L. Tian, H. Jiang, and D. Feng. Implementation and evaluation of a popularity-based reconstruction optimization algorithm in availability-oriented disk arrays. In *24th IEEE Conference on Mass Storage Systems and Technologies*, pages 233–238, San Diego, CA, USA, Sep 2007.

[13] R. R. Muntz and J. C. S. Lui. Performance analysis of disk arrays under failure. In *Proceedings of the 16th International Conference on Very Large Databases*, pages 162–173, 1990.

[14] T. Xie and H. Wang. MICRO: A multilevel caching-based reconstruction optimization for mobile storage systems. *IEEE Transactions on Computers*, 57(10):1386–1398, Oct 2008.

[15] A. Dan and D. Towsley. An approximate analysis of the LRU and FIFO buffer replacement schemes. In *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 143–152, Boulder, Colorado, USA, May 1990.

[16] J. T. Robinson and M. V. Devarakonda. Data cache management using frequency-based replacement. In *Proceedings ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 134–142, Boulder, Colorado, USA, May 1990.

[17] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the ACM SIGMOD international Conference on Management of data*, pages 297–306, Washington, D.C., USA, May 1993.

[18] E. J. O'Neil, P. E. O'Neil, and G. Weikum. An optimality proof of the LRU-K page replacement algorithm. *Journal of ACM*, 46(1):92–112, Jan 1999.

[19] T. Johnson and D. Shasha. 2Q: A low overhead high performance buffer management replacement algorithm. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 439–450, Santiago de Chile, Chile, USA, September 1994.

[20] D. Lee, J. Choi, J. H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 134–143, Atlanta, Georgia, USA, May 1994.

[21] D. Lee, J. Choi, J. H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transactions on Computers*, 50(12):1352–1361, Dec 2001.

[22] Y. Zhou, J. F. Philbin, and K. Li. The multi-queue replacement algorithm for second level buffer caches. In *Proceedings 2001 Annual USENIX Technical Conference*, pages 91–104, Boston, Massachusetts, USA, June 2001.

[23] J. F. Philbin Y. Zhou and K. Li. Second-level buffer cache management. *IEEE Transactions on Parallel and Distributed Systems*, 15(6):505–519, June 2004.

[24] S. Jiang and X. Zhang. LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. In *Proceedings of the 2002 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 31–42, Marina Del Rey, California, USA, June 2002.

[25] S. Jiang and X. Zhang. Making LRU friendly to weak locality workloads: a novel replacement algorithm to improve buffer cache performance. *IEEE Transactions on Computers*, 54(8):939–952, Aug 2005.

[26] N. Megiddo and D. S. Modha. ARC: A self-tuning, low overhead replacement cache. In *Proceedings of 2nd USENIX Conference on File and Storage Technologies*, pages 115–130, San Francisco, CA, USA, Mar 2003.

[27] S. Jiang, X. Ding, F. Chen, E. Tan, and X. Zhang. DULO: An effective buffer cache management scheme to exploit both temporal and spatial localities. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies*, pages 14–16, San Francisco, CA, USA, December 2005.

[28] X. Ding, S. Jiang, F. Chen, K. Davis, and X. Zhang. DiskSeen: Exploiting disk layout and access history to enhance I/O prefetch. In *Proceedings of the USENIX Annual Technical Conference*, Santa Clara, CA, USA, Jun 2007.

[29] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda. Characterization of storage workload traces from production windows servers. In *IEEE International Symposium on Workload Characterization*, Seattle, WA, USA, Sept 2008.

[30] F. J. Corbato. A paging experiment with the Multics system. *In MIT Project MAC Report MAC-M-384*, May 1968.