2011 USENIX Annual Technical Conference, Portland, OR, USA

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, Xubin He

Contents

1 Motivation and Background
2 RGR: A new cache metric
3 VDF
4 Evaluation
5 Conclusions

Background

- Some important targets of modern storage systems
  - ***Performance** : Throughput or Response Time*
  - ***Reliability** : MTTDL Mean Time To Data Loss*
  - *Others : such as spatial utilization, scalability, manageability, power and so on*

RAID

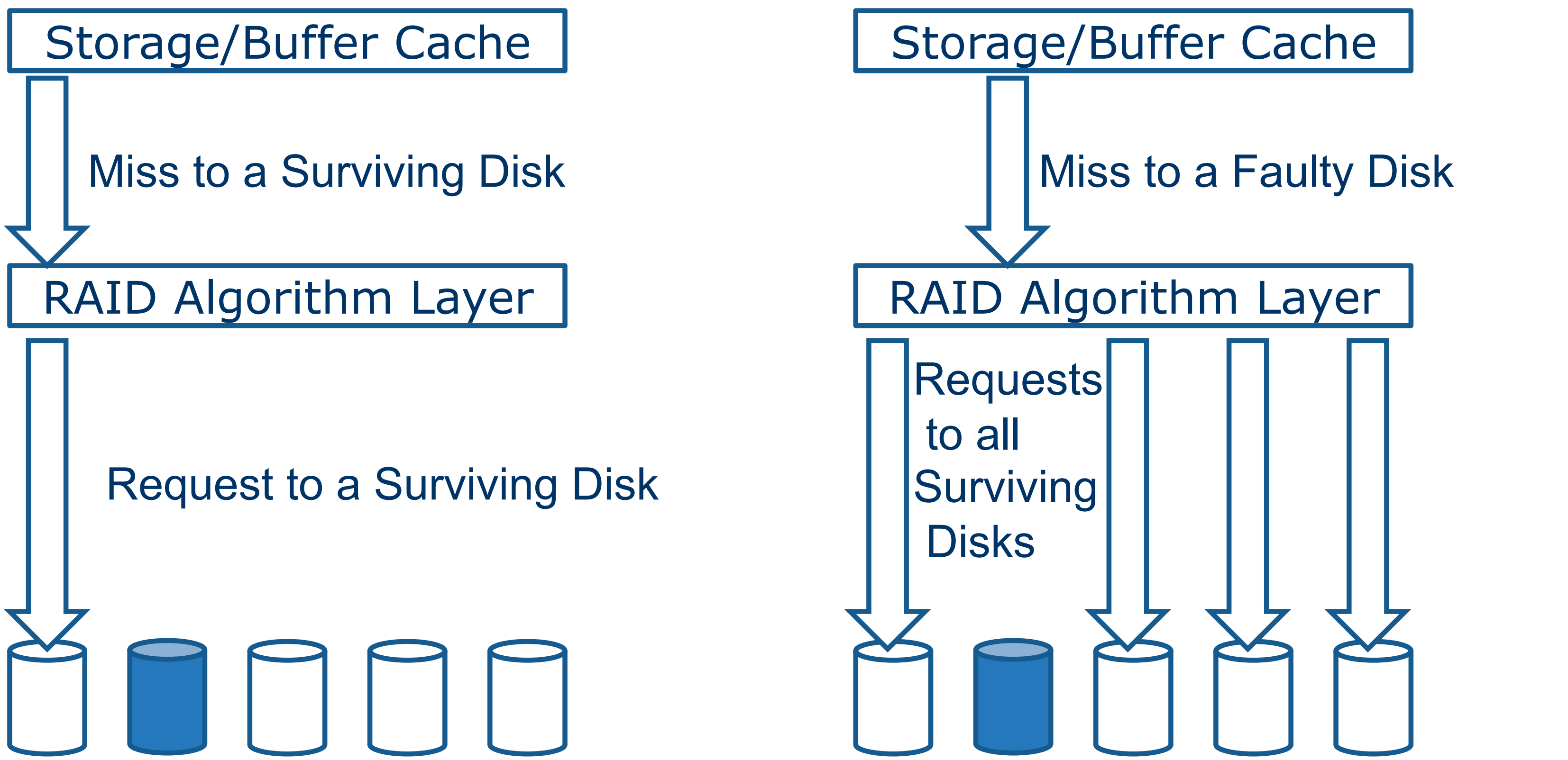
- *Redundant Array of Inexpensive Disks*
- Classifications :
  - No redundancy RAID, e.g. RAID-0
  - Mirror-based RAID, e.g. RAID-1, RAID-10
  - Parity-based RAID, e.g. RAID-4, RAID-5, RAID-6, provides high performance, reliability with high spatial utilization

Weakness of Parity-based RAID

- *High decoding cost*
  - *To CPU, it is recognized as computational complexity*
  - *To the storage device?*
    - *Extra Reconstruction I/O for User Requests*
    - *Extra Reconstruction I/O for System Recovery*

Example

- Different I/O cost of user requests



Existing Solutions

Cache

- Take disk or disk array as Cache: redirection of reads, piggy-backing of writes, WorkOut
- Memory level cache: MICRO

Others

- Data layout
- Scheduling

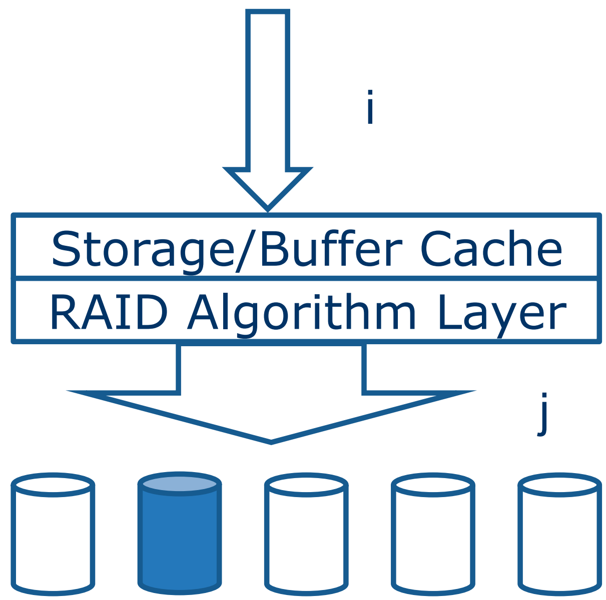
VDF: A Solution in Memory Level

- Observation : Different costs between access to the faulty data and surviving data as shown in the example.

- Our solution : Victim Disk(s) First (VDF): Give higher priority to cache the blocks from the faulty disks when a disk array fails, thus reducing the I/Os directed to the faulty disks and reducing the extra reconstruction I/O for user requests.

RGR: A new cache metric

- ***Miss Ratio***, the old metric to cache, limited in description of faulty condition.
- ***Requests Generation Ratio (RGR)***:
  - The ratio of the number of the requested blocks to the **surviving disks** and the number of the requested blocks to **buffer cache,** j/i in figure.
  - It takes into account different miss situations.
  - It can be used to describe the performance and the reliability in faulty condition, quantitatively and directly.



Formal Description of RGR

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

- ***T*** : Total number of data blocks in a disk array.
- p_i : Access probability of each block.
- MP_i : Miss penalty of each block.

RGR and Performance

$$BW = BW_U \times RGR + BW_R$$

- ***BW*: Total serviceability of all surviving disks in terms of I/O bandwidth.**
- ***BW_U*: I/O bandwidth available to user workload, **throughput.****
- ***BW_R*: I/O bandwidth for a reconstruction workload.**

RGR and Reliabilty

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

VDF and RGR

- Essentially, VDF is to replace the block with minimum (p_i*MP_i) rather than only the p_i, compared to the traditional cache algorithms.

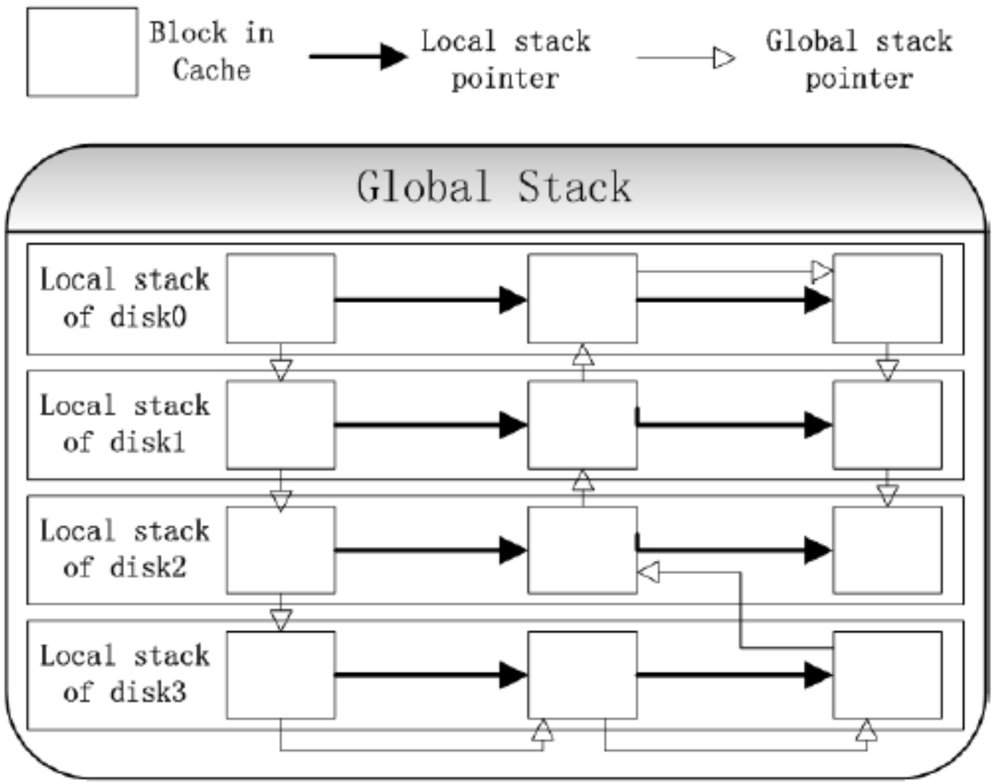
- In many cases, the MP_i of blocks from faulty disks is larger than MP_i of blocks from surviving disks and tend to be kept in cache more probably, so we named this scheme as Victim Disk First.

Case Study

- Only read requests are considered here
  - Usually, users are more sensitive to read latency.
  - Non-volatile memory is deployed as a write cache.
- RAID-5 is evaluated: one of the most popular parity-based RAID structures.
  - MP_i of blocks from surviving disk is 1.
  - MP_i of blocks from faulty disk is n-1, n is the number of disks.
- Two popular cache algorithms: LRU and LFU
  - LRU: Reciprocal of the interval access sequence number is used as p_i of each block in cache, relatively.
  - LFU: Access frequency is chosen here.

Normal mode/faulty mode

- VDF cache only takes effect in faulty condition.



- Two types of stacks are employed to make a smooth conversion.
- Global stack takes charge in fault-free condition.
- Local stacks take charge in faulty condition.

VDF-LRU

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 ≥ 1 and any x_i, one and only one of the following cases
must occur.*/
if x_i is in LS_k, 0 ≤ 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 ≤ 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;
}
```

- 1/(GTS-TS) is p_i
- 1 or (n-1) is MP_i based on the miss conditions
- Choose the max ((GTS-TS)*MP_i) to evict

VDF-LFU

Algorithm 2: VDF-LFU for RAID5 of n disks

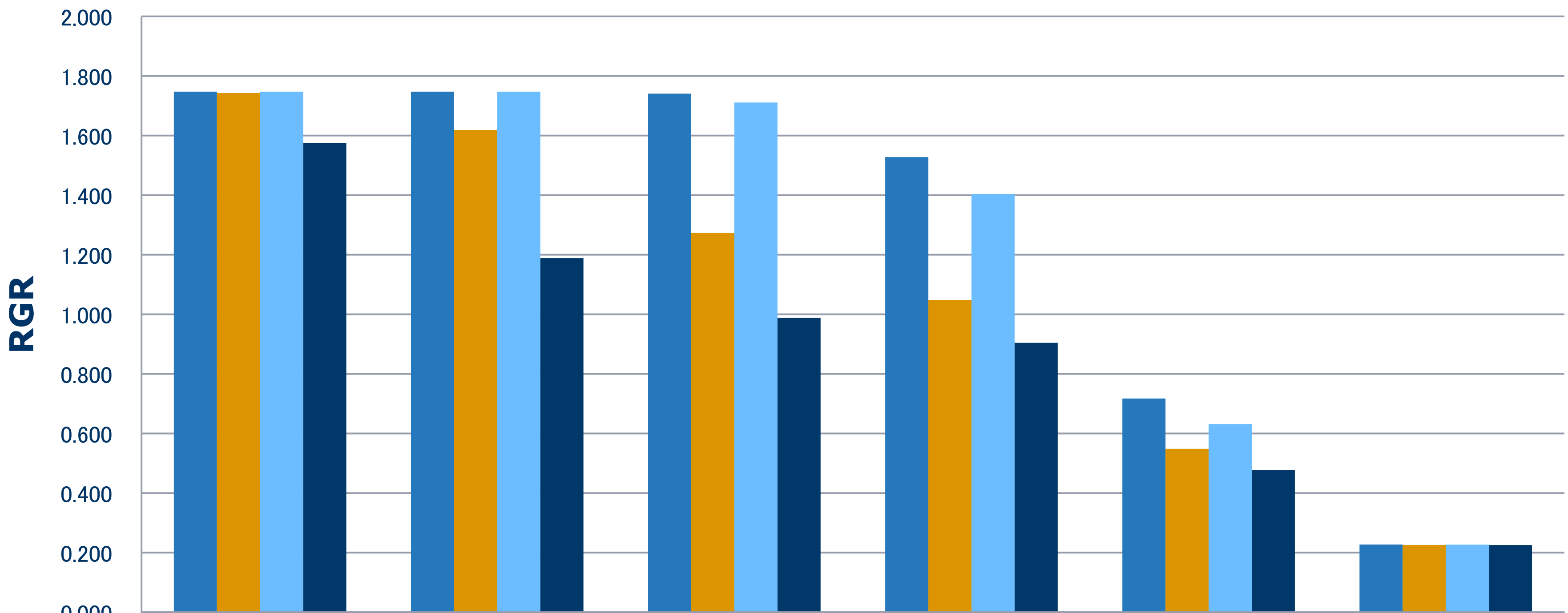
```
Input: The request stream x_1, x_2, x_3, ..., x_i, ...
VDF_LFU_Replace(x_i){
/*For every i >= 1 and any x_i, one and only one of the following cases
must occur.*/
if x_i is in LS_k, 0 <= k < n then
    /*A cache hit has occurred.*/
    Update F and TS of x_i, by F = F + 1;
    Move x_i to right place 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 <= 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;
}
```

- F is p_i
- 1 or (n-1) is MP_i based on the miss conditions
- Choose the min ($F*MP_i$) to evict

Simulation

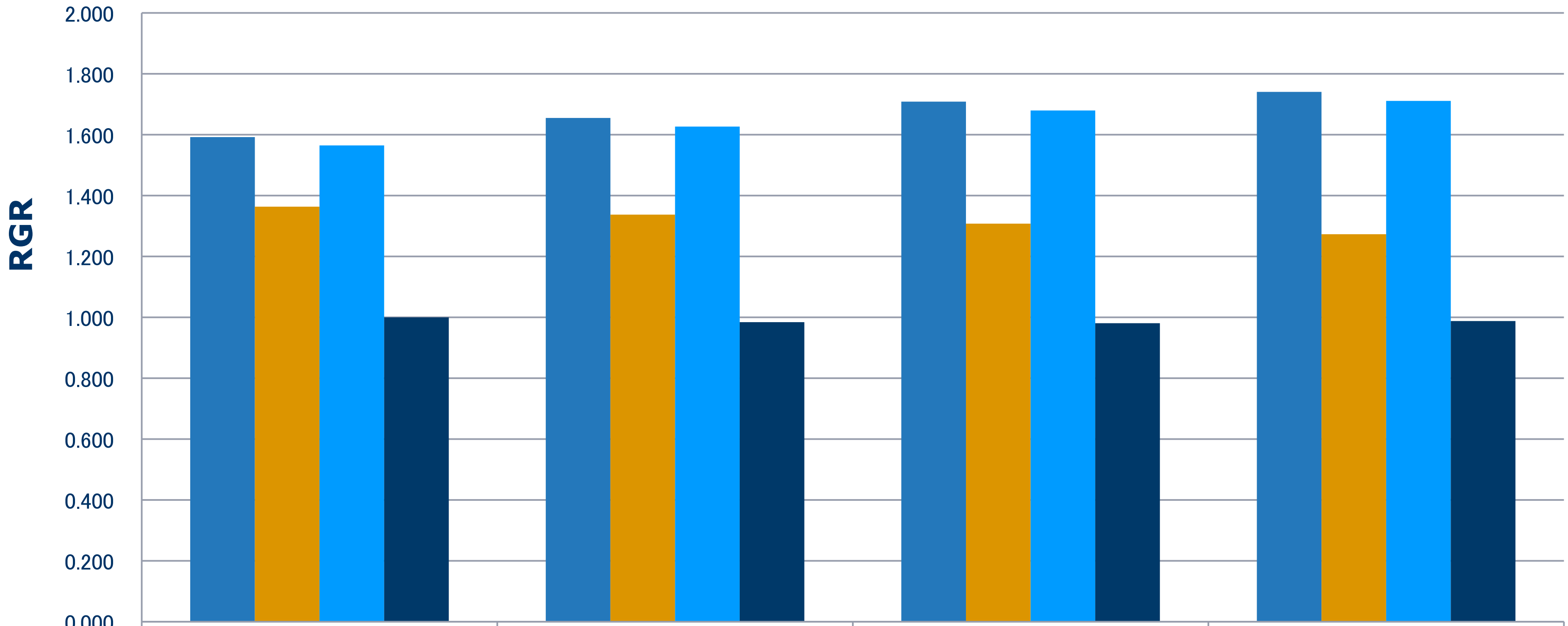
- **Targets:** Effect of VDF on reducing RGR
- **Traces:**
  - SPC-1-web: (Storage Performance Council)
  - LM-TBE， DTRS: (Microsoft)
- **Simulator:** VDF-Sim (about 3000 lines in C)

SPC Web Results of 8 disks



	65536	131072	262144	524288	1048576	2097152
LRU	1.747	1.747	1.741	1.528	0.717	0.227
VDF-LRU	1.743	1.619	1.273	1.048	0.549	0.226
LFU	1.747	1.747	1.711	1.404	0.632	0.227
VDF-LFU	1.575	1.189	0.988	0.904	0.477	0.226

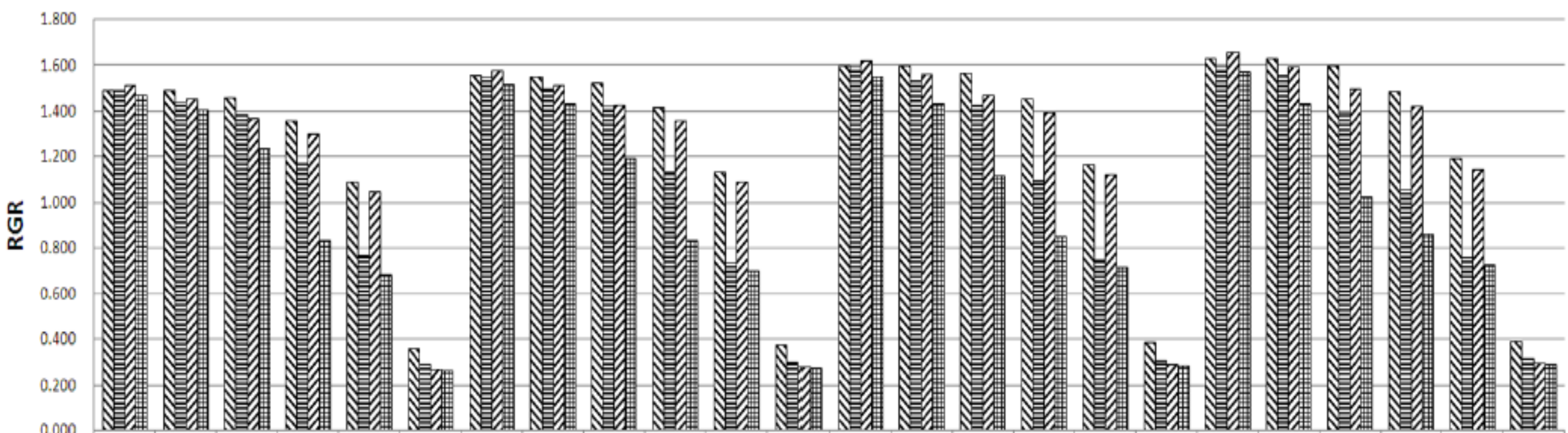
SPC Web Results of 262144 blocks



	5 Disks	6 Disks	7 Disks	8 Disks
LRU	1.592	1.655	1.709	1.741
VDF-LRU	1.364	1.337	1.308	1.273
LFU	1.565	1.627	1.679	1.711
VDF-LFU	1.000	0.984	0.981	0.988

LM-TBE Results

Simulation Results of the LM-TBE Trace

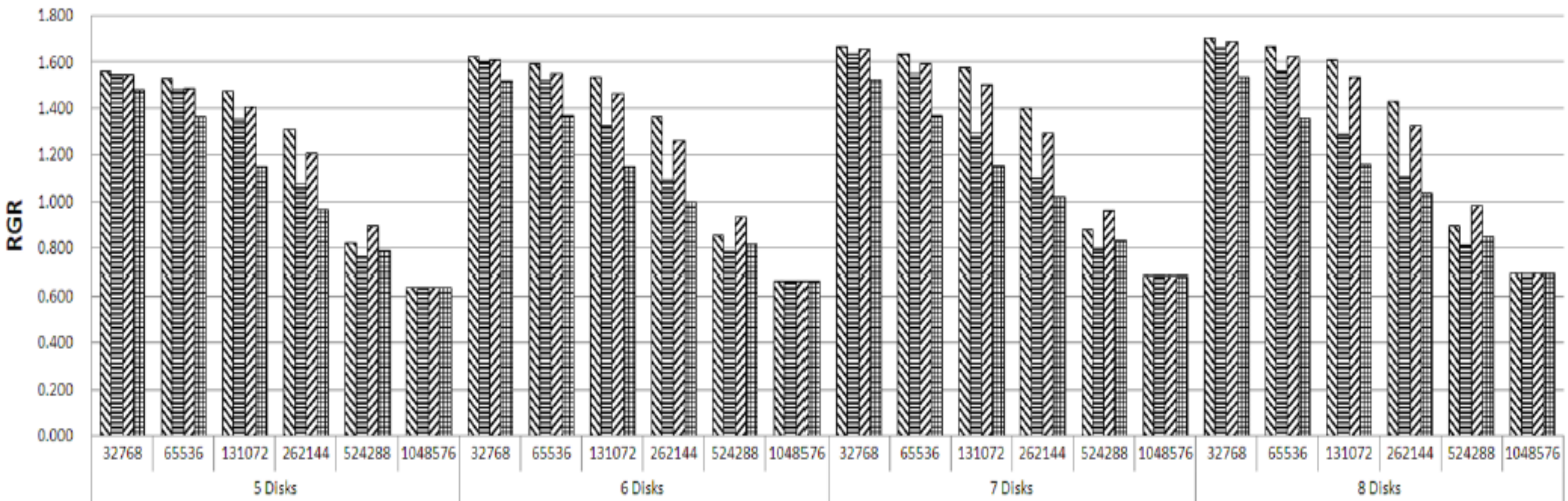


	5 Disks						6 Disks					
	32768	65536	131072	262144	524288	1048576	32768	65536	131072	262144	524288	1048576
LRU	1.494	1.491	1.462	1.360	1.090	0.360	1.556	1.553	1.522	1.416	1.136	0.375
VDF-LRU	1.491	1.441	1.386	1.175	0.770	0.289	1.551	1.497	1.424	1.138	0.743	0.301
LFU	1.515	1.456	1.371	1.303	1.048	0.270	1.578	1.517	1.429	1.357	1.092	0.282
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

	7 Disks						8 Disks					
	32768	65536	131072	262144	524288	1048576	32768	65536	131072	262144	524288	1048576
LRU	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	1.594	1.536	1.431	1.097	0.753	0.310	1.594	1.557	1.396	1.061	0.760	0.316
LFU	1.623	1.560	1.469	1.395	1.123	0.290	1.657	1.592	1.501	1.426	1.147	0.296
VDF-LFU	1.554	1.435	1.119	0.850	0.721	0.285	1.573	1.433	1.030	0.865	0.729	0.291

DTRS Results

Simulation Results of the DTRS Trace



	5 Disks						6 Disks						7 Disks						8 Disks					
	32768	65536	131072	262144	524288	1048576	32768	65536	131072	262144	524288	1048576	32768	65536	131072	262144	524288	1048576	32768	65536	131072	262144	524288	1048576
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

Prototype

- **Targets:** Effect of VDF on improving the **throughput** and shortening the reconstruction duration (**MTTR**).
- **Trace:** SPC-1-web

Architecture

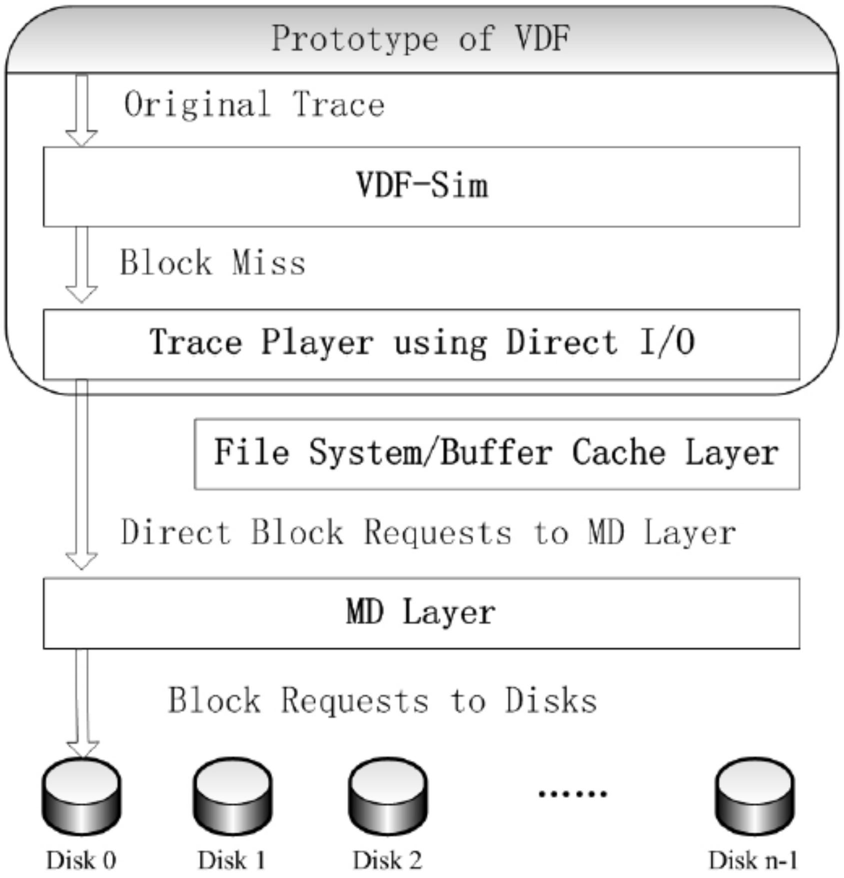


Figure 6: Architecture of VDF prototype.

- Collected block miss information from VDF-Sim, with the real time-stamp (micro second level).
- Play the collected requests to the MD device, using direct I/O to bypass file system cache.

Methodology

Open-loop testing

- Requests are re-played according to their timestamps (fixed BW_u).
- To find the effect on reconstruction duration (MTTR).

Close-loop testing

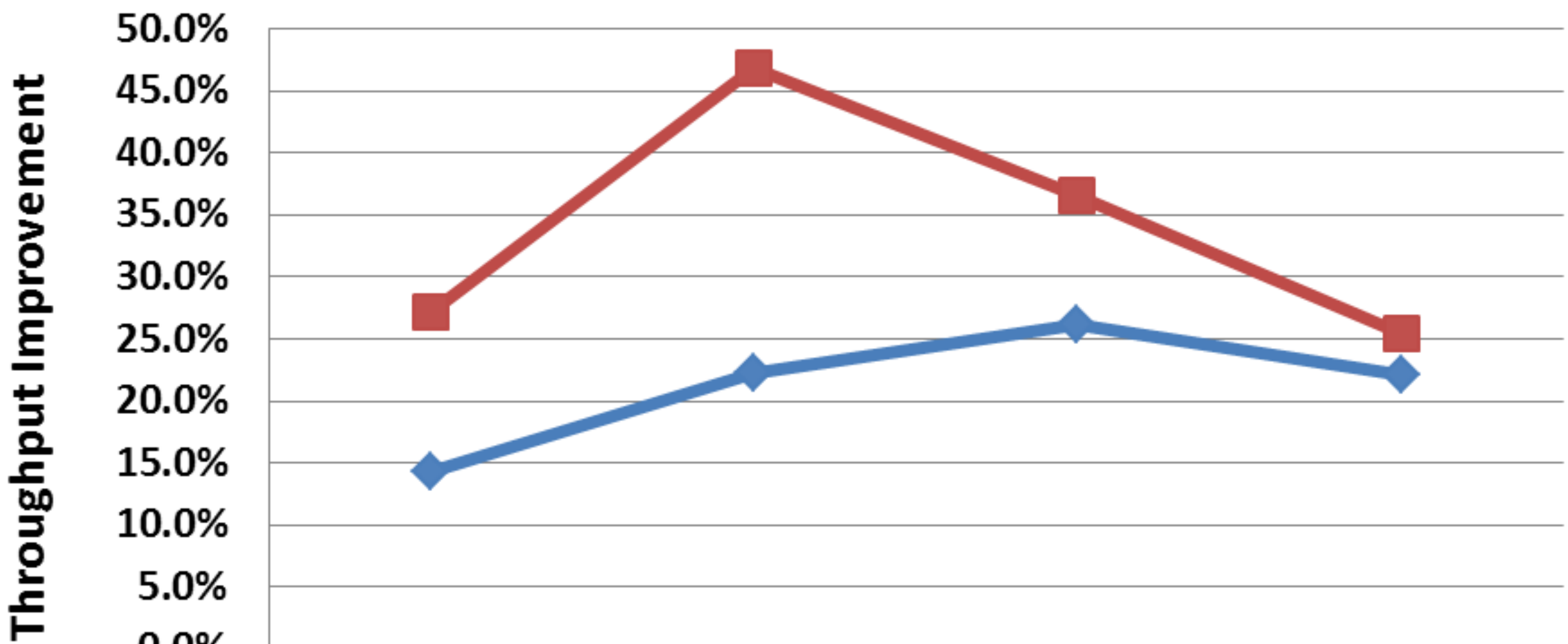
- Requests are re-played one by one.
- To find the effect on throughput.

Open-loop Testing Results

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%

- **60GB dataset: 15GB to reconstruct with 5 disks, 12GB with 6 disks…**

Results by Changing Numbers of Blocks



	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%

Future Work

- Integrate VDF into more general cache algorithms.
- Apply VDF to other RAID levels such as RAID-6 to evaluate the impact of VDF on concurrent failures.

Conclusions

- 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.
- Our results based on both simulation and prototyping implementation have demonstrated the effectiveness of VDF in terms of reduced disk I/O activities and a faster recovery.

Acknowledgments

- Our thorough shepherd, Erez Zadok, for his very detailed comments and helpful suggestions.
- The anonymous reviewers for their invaluable comments.
- National Basic Research 973 Program of China, the National Natural Science Foundation of China, the National 863 Program of China, and the Innovative Foundation of Wuhan National Laboratory for Optoelectronics.
- U.S. National Science Foundation (NSF).

Thank You !

Questions?