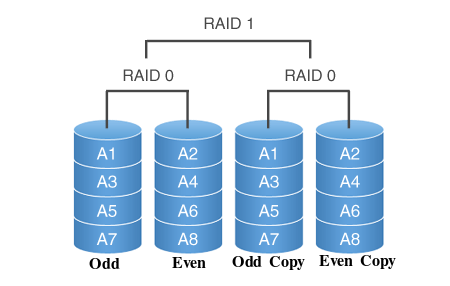
**RAID 01 with Copy-on-Write Snapshot Simulation & Performance Analysis with different workloads**

Project Report

Abhinav Gupta (120050029) Aman Gour (120050030)

**Introduction**

RAID 01 is a type of hybrid raid, combining striping from RAID 1 and mirroring from RAID 0. RAID 01 involves a number of striped disks and these striped disks are mirrored onto similar sets of disks.



Copy-on-Write is a snapshot optimization technique where the data is backed up only when it is modified. Initially, there is no extra data stored. As and when a page is written, the old page is copied and marked as a snapshot.

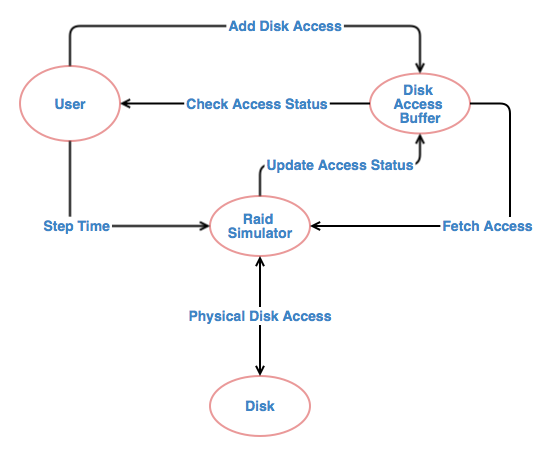
**Design**

**RAID**

For the required RAID 01, we will assume 4 drives (two striped drives mirror onto another two drives) and the stripe size to be of 1 page, i.e. adjacent pages in a file are in different drives. We will assume all files to start in the same disk and negligible access time for the file header.

**Simulation**

For simulating RAID we will maintain a *disk access buffer*. Any page that is not in the Main Memory will be added to the disk access buffer. The raid simulator will then fetch the access instructions from the disk access buffer and complete them from the physical disk. At every time step we will perform maximum possible data operations. The stepping of time is controlled by the user. The total time step count at the end of the workload will represent the time taken to complete that workload. At every time step we will initiate/progress operations based on disk drives not in use. Any disk accesses in the buffer may be reordered to optimize the access. At any instant, the user can check the status of his accesses.



**PROCESS DIAGRAM**

**Snapshot**

To simulate the time taken, we will simply add another disk access to the RAID buffer when a page is written. For the actual snapshot, we will create a separate backup file (with a different header), which contains only the modified pages. To access the snapshot, first the page will be looked up in the backup and if not available, it will be accessed from the actual file.

**Performance Analysis**

We wrote a script to generate a random workload based on two parameters.

**Fraction of Reads**: Fraction of read transactions in the workload

**Fraction Immediate**: Fraction of read requests that need to be served immediately, that is, the RAID simulation should step until that request is complete.

We also varied the*buffer size* of the PF Layer. We measured simulation times for these workloads both with and without RAID (but with optimized access ordering).

|  |  |  |  |
| --- | --- | --- | --- |
| **Fraction of Reads: 0.7**  **Fraction Immediate: 0.4** | **With RAID 01** | **Without RAID01** | **Improvement** |
| Buffer Size: 20 | 838 | 1560 | 46% |
| Buffer Size: 40 | 620 | 1183 | 48% |
| Buffer Size: 100 | 267 | 561 | 52% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Fraction of Reads: 0.4**  **Fraction Immediate: 0.4** | **With RAID 01** | **Without RAID01** | **Improvement** |
| Buffer Size: 20 | 1136 | 1965 | 42% |
| Buffer Size: 40 | 839 | 1487 | 43% |
| Buffer Size: 100 | 359 | 641 | 44% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Fraction of Reads: 0.7**  **Fraction Immediate: 0.8** | **With RAID 01** | **Without RAID01** | **Improvement** |
| Buffer Size: 20 | 1145 | 1668 | 31% |
| Buffer Size: 40 | 829 | 1294 | 36% |
| Buffer Size: 100 | 366 | 592 | 38% |

**Implementation**

We modified the existing PF Layer code to incorporate our elements and added a new RAID simulator. We also provided interfaces to access the snapshots.

**pf.c**

The pf.c file maintains a list of PFRAID\_buf\_ele elements which are the accesses made by the user. For read requests, whenever a page is not present in the main memory it is fetched from the disk and it’s access added to the above list and also passed to the RAID simulator. Each access maintains a flag with the status of the access (completed or pending). For write requests, first, the original data is copied to the snapshot file (and a corresponding write access passed to the RAID simulator) and the actual write access is passed to the RAID simulator. the list of PFRAID\_buf\_ele elements does not contain write requests as writes are only done when the page is being removed from the buffer.

**raid.c**

The raid.c file contains the RAID simulator. It keeps it’s own list of RAID\_buf\_ele elements in which elements are inserted by the read and write functions of pf.c. At every time step, the simulator picks the optimal access to process for each disk depending on the last file accessed by the disk, i.e. pages of the same file are accessed consecutively on the same disk to minimise seek time. We consider a seek time of 2 units and a transfer time of 1 unit. Once an access is complete, it is removed from the list and it’s status updated in the PFRAID\_buf\_ele list in pf.c

**Snapshots**

Snapshots can be taken by calling a function on the file to be snapped. The snapshot creates a new file named snapshot\_<original\_file\_name>. The first page of this snapshot file contains the number of pages in the snapped file and then the position of each page. A position of 0 implies the page is yet to be edited and hence is available in the original file. A position greater than 0 implies the page is at that page position in the snapshot file. Snapshots can be taken only once.

**Conclusion**

**Impact of *Fraction of Reads***

The improvement is higher if there are more number of read requests. This is because write requests involve writing to two disks in RAID 01. Hence, 4 read requests can be processed concurrently but only 2 write requests can be processed concurrently.

**Impact of *Fraction Immediate***

The improvement is higher when a lot of disk reads are not required immediately. This gives the RAID implementation more room to schedule requests optimally. Also, the RAID buffer tends to remain non-empty and hence disk-idle time is reduced.

**Impact of *Buffer Size***

The improvement is higher with increasing buffer size. This is because the reduction in number of disk accesses is not proportional to the performance improvement. This reduction in number is similar for both RAID and non-RAID and hence the percentage improvement increases.

**Verdict**

All in all, RAID 01 shows significant improvements over a non-RAID implementation for a typical workload.

**References**

<http://www.tutorialspoint.com/cprogramming/>

<http://en.wikipedia.org/wiki/Nested_RAID_levels>

<http://en.wikipedia.org/wiki/Copy-on-write>