**RAID 01 with Copy-on-Write Snapshot Simulation**

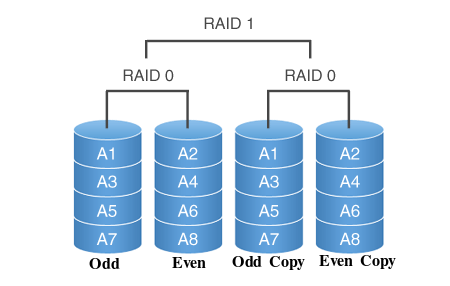
**Design Document**

**Abhinav Gupta (120050029) Aman Gour (120050030)**

**RAID 01**

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.

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. Therefore, a maximum of 4 pages of a file can be accessed concurrently.



**RAID 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.

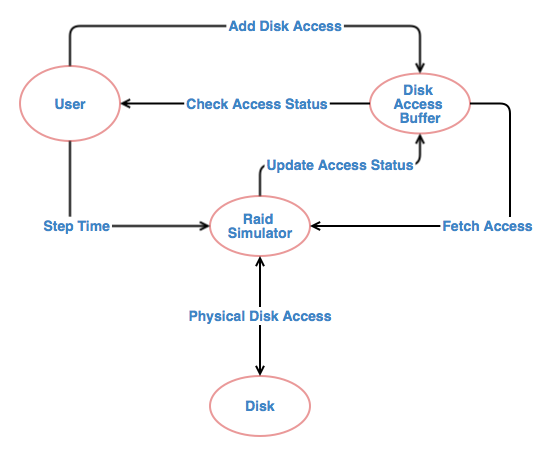
**Time Steps**

The key idea in the project is that of time steps. 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.

This idea of time steps will incorporate only the disk access time. It assumes that all CPU operations and Main Memory accesses are much faster than the disk access and can be ignored.

**User Program**

The user will have to explicitly call the RAID simulator’s step time function for it to take any step towards the physical disk access. 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. Therefore, if the user wants to ensure ordering, he will have to continue stepping until the first operation is complete before adding the second operation to the disk buffer.



**Flow Diagram**

**Snapshot**

The snapshot uses the Copy on write technique. To simulate the time taken, we will simply add another disk access to the RAID buffer when a page is written. For increased efficiency, we will assume that the snapshot files start in the other striped disk (as compared to the original file), so that both can be written simultaneously.

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.

We will also provide interface functions to access the snapshot. Two files will be opened for the snapshot (the backup and the actual).

**Implementation Details**

raid.c: This C file will provide the RAID abstraction to the PF layer. Any page that needs to be accessed from the disk will be added to the buffer present in this file.

pf.c: The original pf.c file will be modified to maintain a buffer as well. Instead of directly accessing the files, the main program will access this buffer. If the required page is not available in the main memory, pf.c will call raid.c for disk access. Otherwise, pf.c will deliver the page from main memory.

Main program: The main program will have to add page accesses to the pf.c buffer. The main program can then keep querying the pf.c buffer to check if the page has been completed. All CPU and Main Memory operations will complete immediately. However, the disk operations will step forward in time only when the main program explicitly calls the timeStep function. Once a page has been fetched, the main program has to remove it from the buffer. Since the buffer size is fixed, there can only be a finite number of pending accesses beyond which the main program has to keep stepping until a page can be removed from the buffer.

**Test Workload**

We will write a script to generate a random workload with both independent and dependent disk access, so as to judge the performance improvements because of RAID. We will assess this performance by varying the main memory buffer size and the RAID buffer size so as to determine the most optimal buffer sizes for better performance vis-a-vis the memory usage.