

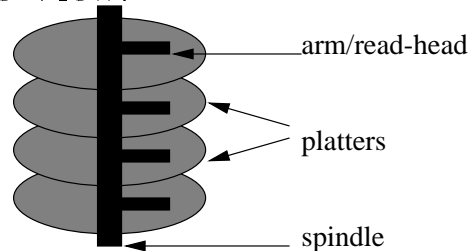
# Retrieving Multimedia Data from Disks

---

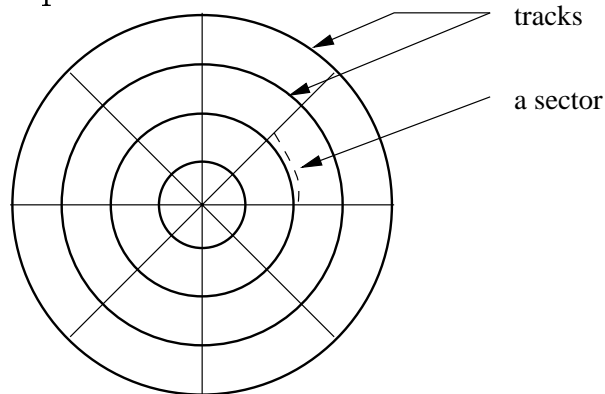
## Overview of Disks

---

- Disk drive Side View.



- Disk drive Top View.



- **Tracks:** Each disk platter consists of a number of concentric “tracks.”
- **Cylinders:** Suppose we are interested in track number  $i$  for some  $i$ . Each platter in our disk device contains such a track. The set of such tracks, one from each platter, is called a *cylinder*.
- **Regions:** Each disk platter is divided into  $k$  regions (for some fixed  $k$ ). Each region represents a wedge of the platter with angle  $\frac{360}{k}$ .

- **Sectors:** That part of a track that intersects a wedge is called a sector. Clearly, if we have  $n$  tracks altogether, then we will have  $n$  sectors per wedge.

## Disk Retrieval

---

- Associated with each disk platter is a disk arm that contains a read-write head to read/write from/to that platter.
- When a disk address is to be accessed, the disk controllers (the programs that control the position of the disk head relative to the disk) start by pursuing two steps:
  1. **Seek Operations:** First, find the track (and hence the cylinder) on which the address is located. Seek time has 4 phases:
    - (a) acceleration phase
    - (b) constant velocity/coast phase
    - (c) deceleration phase
    - (d) settle phase.
  2. **Rotational Operation:** Once the head is positioned over the right track, the disk spindle rotates so that the sector containing the desired physical address is located directly under the read/write head. The time taken for this is called *rotational latency*.
- **Read Head:** Associated with each disk arm is a read/write head that contains the necessary hardware to read data from a sector, or write data onto a sector.
- **Transfer Rate:** Rate at which data is read/written. Varies based on whether reading or writing.

## Notation

Symbol	Meaning
$t_{num}$	total number of tracks
$r_{num}$	total number of regions
$itd$	distance between two tracks
$ss$	spin speed of disk in rotations per minute
$rv$	average radial velocity = average movement of disk head along arm
<b>dtr</b>	The transfer rate of the disk
$rd$	Recording density in Mbytes per sector

- Suppose we wish to read sector  $i$  (on track  $t_i$ ) on a given platter, and the read head on that platter is currently over sector  $j$  in track  $t_j$ .

- 

$$Readtime(i, j) = \frac{rd}{dtr} + spintime(i, j) + Sk(t_i, t_j)$$

where  $spintime(i, j)$  is the amount of time required to spin from sector  $i$  to sector  $j$  and  $Sk(t_i, t_j)$  is the amount of time for the read head to move from track  $t_i$  to track  $t_j$ .

- The seek time required to find track  $i$  from track  $j$  (assuming the head is currently positioned at track  $j$ ) is given (crudely) by:

$$Sk(t_i, t_j) = \frac{abs(t_i - t_j)}{rv}.$$

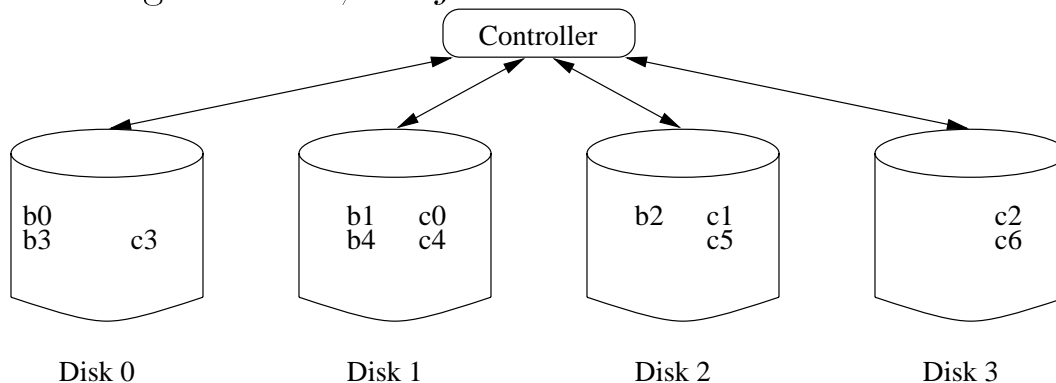
- Similarly,

$$spintime(i, j) = \text{abs}((i - j) \bmod rnum) \times \frac{360}{rnum} \times \frac{1}{av}.$$

## Raid-0 Architecture

---

- We have a set of  $n$  disks, labeled  $0, 1, \dots, (n - 1)$ .
- A  $k$ -stripe is a set of  $k$  drives for some integer  $k < n$  which divides  $n$ .
- When storing a set  $b_0, b_1, \dots, b_{r-1}$  of contiguous blocks in terms of a  $k$  striped layout, what we do is the following. We store block  $b_0$  on disk 0, block  $b_1$  on disk 1, block  $b_2$  on disk 2, and so on.
- Example striping: 2 movies. The blocks of the first movie are denoted by  $b_0, b_1, b_2, b_3, b_4$ . These are striped with  $k = 3$  starting at disk 0. Second movie has six blocks, denoted by  $c_0, \dots, c_5$ , and these are being striped with  $k = 4$  and starting at disk 1, i.e.  $j = 1$ .



## Raid-1 Architecture

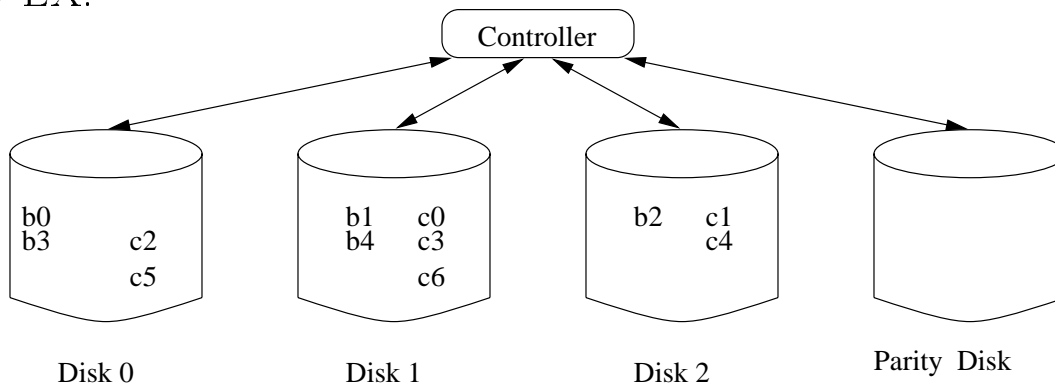
---

- If there are  $N$  disks available altogether, then  $n = \frac{N}{2}$  disks are utilized.
- For each disk, there is a “mirror” disk.
- Raid-1 is predicated on the assumption that there is a very low probability that a disk and its mirror will fail simultaneously.
- When we wish to read from a disk, we read from the disk (if it is active) or we read from its mirrored disk (if the disk has crashed). When writing to disk  $d$ , we must write on both the disk and its mirror.

## Raid-5 Architecture

---

- Each cluster of  $k$  disks has one disk reserved as a “parity” disk.
- Suppose that  $k = n$ , i.e. we have only one cluster (RAID-5 also applies when  $k \neq n$ , this is just an assumption for illustrative purposes).
- Let us further assume that these disks are numbered  $0, 1, \dots, (n-1)$ .
- EX:



- In this case, movie blocks are striped across  $(n - 1)$  of the  $n$  disks available, and disk number  $(n - 1)$  is reserved as a “parity” disk.
- Suppose we use  $D_{i.j}$  to denote the value of the  $j$ 'th bit of disk  $i$  and suppose disk  $n$  is the parity disk. If the symbol  $\oplus$  denotes the exclusive-or operator, then

$$D_{n-1.j} = D_{0.j} \oplus \dots \oplus D_{n-2.j}.$$

- I.e. the  $j$ 'th bit of the parity disk is obtained by taking the exclusive-or of the  $j$ 'th bits of all the other disks.
- This can be used to recover data if one disk crashes.

## Example

---

- Consider a simple example, where  $n = 3$ .
- The truth table for exclusive or is:

$D_1$	$D_2$	$D_3$	(Parity disk) $D_p = D_1 \oplus D_2 \oplus D_3$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1
0	0	0	0

- Suppose disk  $D_2$  crashes and we wish to find the value for a specific bit  $j$ .
- Read the value of bit  $j$  in disks  $D_1, D_3$  and the parity disk  $D_p$ .
- For instance, these values might be 0, 1, 1 respectively.
- Then examine the above truth table to see which row has  $D_1 = 0, D_3 = 1$  and  $D_p = 1$ .
- The second last row in the table satisfies this condition.
- From this, we can infer the value of this bit in disk  $D_2$  to be 0.
- How can we do this in general?

## Service Algorithms

---

- Given a set of clients each of whom wants to read some data from disk, how do we do schedule their reads?
- Several algorithms proposed for this task.
- Most such algorithms *must* execute very fast, i.e. it cannot take too long for the algorithm to determine order of reads.
- Some well known algorithms:
  - First Come First Serve (FCFS)
  - SCAN
  - SCAN Earliest Deadline First (SCAN-EDF)

## First Come First Served

---

- Each client retrieval request has an associated time stamp.
- Clients are serviced in order of their associated time-stamp.
- **EXAMPLE:** Suppose the disk read head is currently over track  $i$ , sector  $j$ . The table below shows some numbers specifying client requests, when they were made, and how long it takes to reposition the disk head from its current location to the desired location.

RequestID	ReqTime	Est. Seek	Est. Rotational Delay
r1	10	24	3
r2	8	12	5
r3	14	30	6
r4	11	18	4

- In this case, FCFS will serve the requests in the order: r2,r1,r4,r3.
- The last 2 columns are completely ignored by FCFS.

## SCAN Algorithm

---

- Suppose the disk read head is currently over track  $i$ , sector  $j$ .
- In this case, we order requests in the order of the number of tracks to be traversed from track  $i$ , moving either outwards first and then inwards, or vice versa, but not both. (Of course, the further away the tracks
- We then service the requests in the order prescribed.
- EXAMPLE: Suppose we return to the last example, and each track requires 3 units of time to be traversed (estimate). Then

RequestID	ReqTime	Est. Seek (Num of tracks)	Est. Rotational Delay
r1	10	24(8)	3
r2	8	12(4)	5
r3	14	30(10)	6
r4	11	18(6)	4

- If we assume that all of r1–r4 are in tracks beyond track  $i$  (i.e. between track  $i$  and the outer rim of the disk) then it is easy to see that we will provide service in the order r2,r4,r1,r3.

## SCAN-EDF Algorithm

---

- EDF stands for “Earliest Deadline First”.
- Here, we first group all requests in ascending order of their deadline.
- Suppose the resulting groups are  $G_1, G_2, \dots, G_n$ .
- Each group is serviced using SCAN.
- EXAMPLE:

RequestID	ReqTime	Est. Seek	Est. Rotational Delay	Deadline
		(Num of tracks)		
r1	10	24(8)	3	100
r2	8	12(4)	5	120
r3	14	30(10)	6	120
r4	11	18(6)	4	100

- Here we have two groups,  $G_1$  which contains  $r_1, r_4$ , and  $G_2$  which contains  $r_2, r_3$ .
- $G_1$  is serviced first using SCAN. By the same assumptions as on the previous slide, we first service  $r_4$ , then  $r_1$ .
- $G_2$  is then serviced using SCAN. By the same assumptions as on the previous slide, we first service  $r_2$ , then  $r_3$ .
- Thus, the overall order of service is  $r_4, r_1, r_2, r_3$ .

## Building Disk-based Media Servers

---

- Must service multiple clients simultaneously.
- Clients do not want just playback functionality, they also want to perform interactive operations like rewind, fast forward, pause, etc.
- For each client, the server must:
  - provide continuous playback
  - this requires filling his buffer at just the “right” rate.
  - Too fast → buffer might get overwritten.
  - Too slow → client might experience service interruption.

Symbol	Meaning
$\text{bnum}(\mathcal{M}_i)$	Number of blocks in movie $\mathcal{M}_i$
$\text{buf}(i)$	The total buffer space associated with disk server $i$
$\text{cyctime}(i, t)$	the total cycle time for server $i$ at time $t$
$\text{dtr}(i)$	The total disk bandwidth associated with disk server $i$
$\text{switchtime}(i, t)$	the time required for disk server $i$ to switch from one client's job to another client's job at time $t$
$\text{cons}(i, t)$	The consumption rate of client $C_i$ at time $t$
$\text{data}(i, t)$	The event specification for the client $C_i$ at time $t$
$\text{timealloc}(i, j, t)$	The time-slice allocated to client $j$ at time $t$
$\text{active}(t)$	The set of all clients that are active at time $t$
$\text{d\_active}(i, t)$	The set of all clients that have been assigned a non-zero time-allocation by disk server $i$
$\varphi(\mathcal{M}_i, b)$	The set of servers that contain block $b$ of movie $\mathcal{M}_i$ according to placement mapping $\varphi$
$\mu_t(i)$	The set of servers handling requests by client $C_i$ at time $t$
$\text{bufreq}(j, i, t)$	The buffer space needed at server $i$ to match the consumption rate of client $j$
$\mathcal{S}(t)$	The state of a movie-on-demand system

Figure 1: Notation and terminology

## Parameters

---

- We assume that we have  $n$  disk servers,  $d_1, \dots, d_n$  and  $m$  movies  $m_1, \dots, m_k$ .
- Each movie is a contiguous sequence of blocks. Block size can be fixed in any way.

## Client Request

---

- **data**( $i, t$ ) is a set of pairs of the form  $(m, b)$  where  $m$  is a movie, and  $b$  is a block in that movie.
- Playback:

$$\mathbf{data}(i, t) = \{(m, b), (m, b + 1), \dots, m(b + r - 1)\}.$$

Here  $r$  is the number of blocks the client watches per time unit.

- **Fast forward:** Suppose  $ffs$  is a positive integer called the “fast forward step.”

$$\mathbf{data}(i, t) = \{(m, b + i \times ffs) \mid i < r \ \& \ (b + i \times ffs) < \mathbf{bnum}(m)\}.$$

- **Rewind:** Suppose  $rws$  is the rewind step.

$$\mathbf{data}(i, t) = \{(m, b - i \times rws) \mid i < r \ \& \ (b - i \times rws) > 1\}.$$

- **Pause:** If when the user pauses, he was watching block  $b$  of movie  $m$ , then:

$$\mathbf{data}(i, t) = \{(m, b)\}.$$

## Client Request

---

- Reformulate  $\mathbf{data}(i, t)$  as a single quadruple

$$\mathbf{data}(i, t) = (m, b, len, step)$$

- This means that client  $C_i$  wishes to view the following blocks of movie  $m$ :

$$b, (b + step), (b + 2 \times step), \dots, (b + (len - 1) \times step).$$

- **Play - normal viewing:** In this case,  $step = 1$ .
- **Pause:** In this case,  $step = 0$ .
- **Fast Forward with speed ffs:** In this case,  $step = ffs$ .
- **Rewind at speed rws:** In this case,  $step = -rws$ .

## Algorithm to Support VCR-Functionality (Sketch)

---

- When a set of events (transactions) occur, each of these transactions must have an associated priority.
- Initial priority assignments.

Transaction	Priority
Exiting client	5
Continuing Client - normal viewing	4
Continuing Client - fast forwarding	4
Continuing Client - rewind	4
Continuing Client - pause	4
New (entering) client	2

- **Splitting:** This causes a user's transaction to be split into two or more pieces (called **twins**). Transaction is satisfied iff both twins can be satisfied.
- **Switching:** Causes a user's transaction (or its descendent subtransactions) to be switched from the server that was originally handling the request, to another server.

## Algorithm to Support VCR-Functionality (Sketch)

---

- Algorithm considers clients in decreasing order of priority.
- Note that exiting clients have top priority (as they free up resources) and continuing clients have high priority too.
- Satisfy clients in decreasing order of priority.
- If a client request cannot be satisfied, and it cannot be switched to another server, then split it, reinsert into client list with slightly lower priority (3 for split continuing clients).
- New client's priority increased if not served in the loop, but can never reach 3.