Secondary Storage Management

CS315: Principles of Database Systems, Jan-Apr 2022
IIT Kanpur

Outline

Memory Hierarchy: outline

- Memory hierarchy of a computer system.
  - Cache hierarchy, memory, disks, tertiary storage (DVDs, tapes etc.).
- Focus on disks: secondary level storage.
- Speed of access: latency and data transfer rates.

Typical Memory Hierarchy

Briefly: levels from fastest-smallest towards slower-bigger.
Cache: Typical machines (e.g., Intel processor) has multi-level caches:

- L1 level-1 cache: fastest access cache, typically built into each core chip: 16KB-256KB.
- L2/L3 level-2/3 cache: second level cache, shared by cores. Today typically 4MB-20MB. Bandwidth: 0.5 TB/s.
- Data and instructions are moved to cache from main memory as needed.
- Access speed: few nanoseconds.

Main Memory

- *DRAM* technology and its variants (currently, DDR4).
- Today, typically has 1GB-128GB.
- Typical rates: Latency
  - Latency: 9-20 cycles (DDR4).
  - Typical clock speeds: 1-4 GHz.
  - Latency: of order of 10-100 nanoseconds.
- Typical rates: Bandwidth
  - Peak transfer rates (currently): 2-24GB/s.
  - Typical times to move data from main memory to processor or cache 5-5 ns.

Secondary Storage

- Typically, magnetic disk. Other technologies like SSD.
- E.g.,1 terabyte disk is common.
- A machine can have several disk units.
- Transfer times between disk and main memory is ~ 10 milliseconds.
- Large amounts of data can be transferred at high rates.
- We will see in a bit of detail later.

Tertiary Storage

- A collection of disk units may not be voluminous enough to store the data: e.g., astronomy dat, etc..
- may involve robot arms or conveyors.
- Storage media: magnetic tape, optical disks (DVDs).
- Retrieval can take seconds or minutes.
- Capacities in petabyte range possible.

Transfer of data between levels

- Data moves between adjacent levels of hierarchy.
- At secondary and tertiary levels: locating desired data is time consuming.
- i.e., latency is high.
- These levels are organized to transfer higher amounts of data to the level below.
- Typically, disk is organized into *disk blocks*.
- Each disk block is perhaps *4-64*KB.
- Memory to Cache:
  - Often by units of cache lines.
  - Typically, 32-128 consecutive bytes.

Volatile and Nonvolatile Storage

- A volatile device “forgets” what is stored in it when *power goes off*.
- Nonvolatile device typically keeps its contents intact for long periods when the device is turned off, or there is a power failure.
- DBMS characteristic: data is retained in the presence of power failures.
- Technology: magnetic and optical materials hold their data in absence of power.
- Secondary and tertiary storage devices are non-volatile.
- Main memory is generally volatile (e.g., DDR)
  - Flash memory can hold their data after power failure.

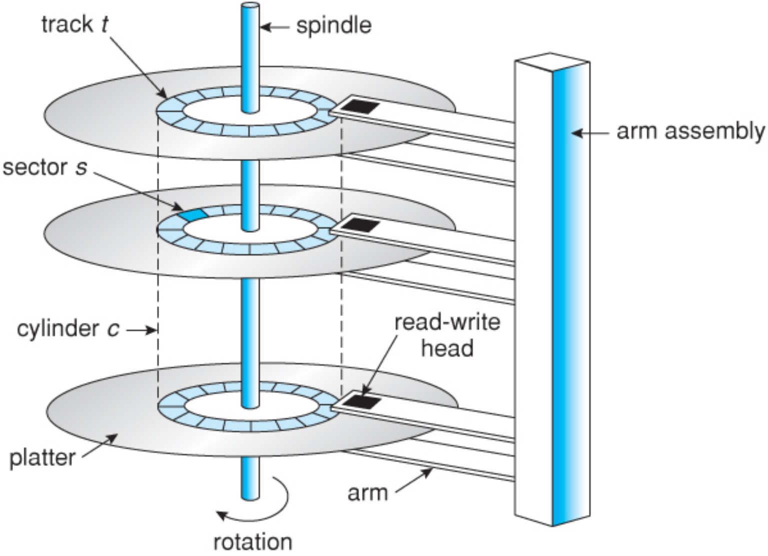
Moore's Law

- *Moore's Law*: Integrated circuits were following an exponential rise that doubles about every 18 months.
- Some parameters that follow "Moore's Law":
  1. Number of instructions per second. After 2005, multi-core chips were made.
  2. Number of memory bits that can be bought for unit cost,
  3. and the number of bits that ca be put on one chip.
  4. number of bytes per unit cost on a disk and the capacity of the largest disks.

Disks: Motivation

- Use of secondary storage (non-volatile) is an important characteristic of DBMS
- Secondary storage is mostly based on magnetic disks.

Disk Mechanics

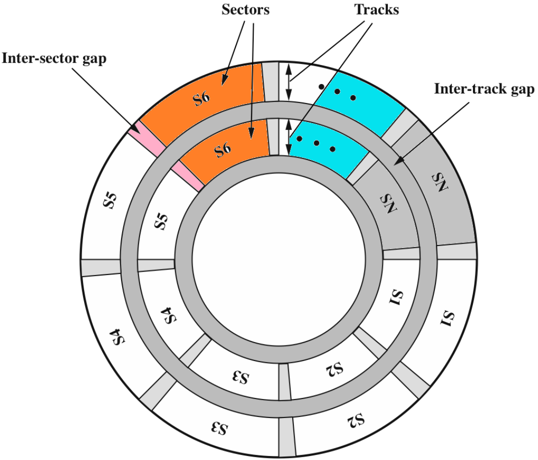


Disk Mechanics

- Two principal moving components: disk assembly and head assembly.
- Disk assembly: one or more circular platters that rotate with axis at spindle.
- Upper and lower layers of platter have magnetic material layer, where bits are stored.
- A common diameter for disk platters is 3.5inches.
- Disk is organized into tracks: concentric circles on a single platter.
- Tracks that are at a fixed radius from the center, among all surfaces, form one cylinder.

Disk Mechanics

- Tracks occupy most of the surface, except for region closest to spindle.



Disk Mechanics

- Typically, about 100,000 tracks per inch; stores about a million bits per inch along the tracks.
- *Sectors*: tracks are organized into sectors.
- Segments of the circle separated by *gaps* that are not magnetized to represent either 0's or 1's.
- Sector is an indivisible unit for reading or writing.
- It is also indivisible with respect to errors.
- Should a portion of the magnetic layer be corrupted in some way, so that it cannot store information, then the entire section containing this portion cannot be used.
- Gaps often use 10% of the total track. Used to identify beginnings of sectors.

Blocks

- *Blocks* are logical units of data that are transferred between disk and main memory.
- Blocks consist of one or more sectors.

Disk Heads

- Head assembly holds the *disk heads*, one per platter surface.
- For each surface there is one head, riding very close to the surface without touching it.
- or else, a head crash occurs and the disk is destroyed.
- Head reads the magnetism passing under it, and can also alter the magnetism to write information on the disk.
- Heads are attached to an arm and the the arms for all the surfaces move in and out together; part of the rigid head assembly.

Disks: Example

A *Megatron 747* disk has the following characteristics:

- There are 8 platters, providing 16 surfaces.
- There are 2^{16} or 65,536 tracks per surface.
- On average: $2^8 = 256$ sectors per track.
- There are $2^{12} = 4096$ bytes per sector.

Disks: Example

- Capacity: $2^4 = 16$ surfaces $\times$ 2^{16} tracks $\times$ 2^8 sectors per track $\times$ 2^{12} bytes per sector
  $= 2^{40}$ bytes or 1TB (terabyte disk).
- A single track holds $2^8 \times 2^{12} = 2^{20} = 1\text{MB}$.
- If blocks are $2^{14} = 16\text{KB}$, then one block uses 4 consecutive sectors.
- There would be (on average) 32 blocks per track.

Cylinder

- All the tracks that are at exactly the same radius is called a *cylinder*.
- If the disk arm is at that radius, then, any track in that cylinder can be read.
- Megatron 747 example:
  1. A cylinder has 1MB $\times 32 = 32$ MB capacity.

Disk Controller

A *disk controller* may control one or more disk drives. It is capable of

1. Controlling mechanical actuator that moves the head assembly, to position the heads at a particular radius, so that any track of one particular cylinder can be read or written.
2. Selecting a sector from among all those in the cylinder at which the heads are positioned.
3. Controller is responsible for knowing when rotating spindle has reached the point where the desired sector is beginning to move under the head.

Disk Controller -II

4. Transfers bits from the desired sector to the computer's main memory.
5. *Buffering*: possibly buffering an entire track or more in local memory of the disk controller,
   - hoping that many sectors of this track will be read soon, thereby,
   - avoiding additional accesses to the disk.

Disk Access Characteristics -I: seek time

Accessing, i.e., *reading* or *writing* a block (or sector) requires three steps. Each step has an associated delay.

1. The disk controller positions the head assembly at the cylinder containing the track on which the block is located.

   The time to do so is the *seek time*.

Rotational Latency

2. The disk controller waits until the first sector of the block moves right under the head. This time is called *rotational latency*.
3. All the successive sectors constituting the block as they pass under the head, are read/written by the disk controller. This delay is called *transfer time*.

Latency

- $\textit{Latency} = \textit{Seek time} + \textit{Rotational Latency} + \textit{Transfer time}$.

Latency: Comments

- Seek time depends on the distance the heads have to travel from where they are currently located.
  - If they are already at the desired cylinder, the seek time is 0.
  - It takes roughly 1ms to start the disk heads moving, and,
  - perhaps 10ms to move them across all the tracks.

Latency: Comments

- A typical disk rotates once in roughly 10ms.
- Then, rotational latency is in range 0-10ms, and the average is 5ms.
- Transfer times are typically much smaller, sub millisecond range.
- Adding all three delays, typical average latency is about 10ms, and the maximum latency about twice that.

Example: Latency

- Let us examine the time taken to read a 16KB block from *Megatron 747* disk (previous example). Following are more parameters:
- Disk rotates at 7200rpm; one rotation takes 60s/7200 = 1/120 s = 8.33ms.
- To move the head assembly between cylinders: *start* and *stop* takes 1ms.
- Plus, it takes 1ms to traverse 4000 cylinders.
  1. For head to traverse from inner most track to outermost track takes $1 + 65,536/4000 = 17.384$ms.
  2. *heads* move 1 track in $1 + 1/4000 = 1.00025$ms.
- Gaps (*inter-sector* gaps) occupy 10% of the space around the track.

Example: Times to read a block

- *Minimum time*: This is just transfer time. Why?
  1. Head is on the right track.
  2. The first sector of the block is just beginning to pass under the head.
  3. Hence no seek time, and no rotational latency.
- Sector size: 4KB, block size is 16KB $=$ 4 consecutive sectors (plus the 3 gaps; gaps are 10%).
- Total angle in terms of sector in degrees is

$$\frac{4+3\times 0.1}{256+256\times 0.1}\times 360^\circ = \frac{4.3}{281.6}\times 360^\circ = 5.497...^\circ$$

- Time taken for a full rotation 360° is 8.333ms, so transfer time for the block is

$$\frac{5.497}{360}\times 8.33\text{ms} = 0.1272\text{ms}$$

Maximum time to read a block

- Worst case: heads are positioned in the inner most cylinder, and,
- the block we want to read is in the outermost cylinder.
- It takes 1 ms to start the heads and stop at the destination cylinder.
- It takes 16.38 s (prev. calculation) to traverse 65,535 cylinders to reach the outermost cylinder.
- *Seek time* $=$ 17.38 ms.

Maximum time to read a block

- *Maximum rotational latency*: Worst case is that when the head arrives at the correct cylinder,
- the beginning of the desired block has just passed under the head.
- So the head waits until the desired block comes around after a full rotation.
- *Rotational Latency* $=$ 8.33ms.
- *transfer time* $=$ 0.1272ms (previous calculation).
- Maximum time to read a block

$$17.38 + 8.33 + 0.1272 \approx 25.8372\text{ms}$$

Average Latency

- Transfer time component is always fixed: 0.1272ms.
- On average, the rotational latency is for half a rotation, that would take $8.333/2 = 4.1667$ms.
- We can now calculate the average seek time.
- Let X and Y be uniform random variables in $(0, 1)$ and independent. We wish to find $\mathbf{E}[|X - Y|]$.

*Average Latency Calculation

$$
\begin{aligned}
\mathbf{E}\left[|X - Y|\right] &= \int_{x,y=0}^{1} |x - y| dxdy \\
&= \int_{x=0}^{1} \int_{y=0}^{x} (x - y) dydx + \int_{x=0}^{1} \int_{y=x}^{1} (y - x) dydx \\
&= \int_{x=0}^{1} \left[xy - \frac{1}{2} y^2 \right]_0^x dx + \int_{x=0}^{1} \left[\frac{y^2}{2} - xy \right]_x^1 dx \\
&= \int_{x=0}^{1} \frac{x^2}{2} dx + \int_{x=0}^{1} \left(\frac{1}{2} - \frac{x^2}{2} - x + x^2 \right) dx \\
&= \frac{1}{2} \cdot \frac{1}{3} + \int_{x=0}^{1} \left[\frac{1}{2} - x + \frac{x^2}{2} \right] dx \\
&= \frac{1}{6} + \frac{1}{2} - \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{3} = \frac{1}{3} \quad .
\end{aligned}
$$

Problems of Accessing Storage by DBMS

- Say disk takes an average of 10ms to access a block.
- Doesn't imply that each DBMS block request can be satisfied in 10ms.
- Suppose there is a series of block access requests made more frequently than 1 in 10ms.
- Then, the *queue* of requests will line up indefinitely: *scheduling latency* becomes infinite.
- How do we improve
  - average seek times?
  - throughput?

Simple techniques for speeding up database access

- Place blocks that are accessed together on the same cylinder.
  1. Reduces seek time, perhaps rotational latency.
- Divide data among multiple disks rather than placing all in one disk.
  1. Multiple head assemblies can access different blocks independently: *Parallel* access.
- **Mirror** a disk: make two or more copies of data on different disks.
  1. Increases failure resistance: *resilience*.
  2. Allows *parallel* access.
- Use a *disk scheduling algorithm*, either in the operating system, or in the DBMS, or in the disk controller.
- *Prefetch* blocks into main memory in anticipation of later use.

I/O model of computation

- A DBMS serves a number of users who are performing queries and database modifications.
- For great simplicity, assume, the computer has
    1. one processor, one disk controller and one disk.
- Database is too large to fit in memory.
    - Key blocks (parts of DB) may be *buffered* in main memory.
    - Each part of the DB that a user accesses is retrieved originally from disk.

I/O Dominant Costs

Dominance of I/O cost

Time taken to perform a disk access is much larger than the time likely to be used for performing CPU computations on that data in main memory. Thus, number of block accesses (*Disk I/O's*) mostly dominates substantially the total time needed by the algorithm.

Example of I/O dominant costs

- Suppose the database as a relation R and a query asks for a tuple of R that has a certain key value k.
- It would be desirable to have an *index on this key attribute of* R.
- The index identifies the disk block on which the tuple with key value k appears.
- On the e.g., Megatron 747 disk, it takes about 11ms to read a 16KB block.
- In 11ms, modern CPUs can execute tens of millions of instructions.
- Searching for the key from a block in main memory, and typical computations on it, likely takes only thousands of instructions.
- *Additional time to perform computations in main memory* $\leq 1\%$ *of block access time.*

Organizing Data by Cylinders

- Seek time is about half or more of the time it takes to access a block.
- For efficiency, we can store data that is likely to be accessed together on the same cylinder.
- This way, seek time is minimized. Example.

Example: Organizing data by cylinders

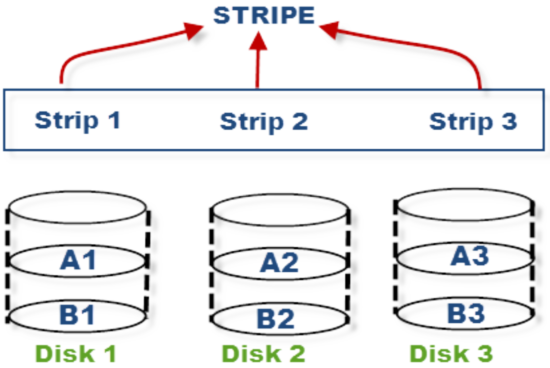
1. Relation R is stored in 1024 blocks of a Megatron 747 disk (prev.e.g.).
2. A query needs access to all the blocks (in some order).
3. Suppose blocks holding R are distributed around the disk at random.
    1. Average latency $\approx$ 10.76ms $\times$1024 $\approx$ 11s.
4. 1024 blocks fits in one cylinder: suppose we do the same.
5. Access $=$ 1 average seek (6.46ms), then read 16 tracks of cylinder in 16 rotations.
6. Time $= 6.46 \times 8.333 \times 16 = 133$ms.
7. About 80 times faster.

Multiple disks: Striping, mirroring and Parity

We will study them in more detail when we discuss *RAID*.

1. Suppose there are two tables A and B (for e.g.,
2. Instead of one disk, we have 3 disks, named disk 1, disk 2 and disk 3.
3. *Striping* for the table A goes like this.
4. Successive blocks of A are numbered $A1, A2, A3, A4, \ldots,$ so on.
5. *Striping*: A is striped across the disks as follows:
    1. Blocks $A1, A4, A7, \ldots, Ak$ are placed in disk 1, where, $k \mod 3 = 1$.
    2. Blocks $A2, A5, A8, \ldots, Ak$ are placed in disk 2, where, $k \mod 3 = 2$.
    3. Blocks $A3, A6, A9, \ldots, Ak$ are placed in disk 3, where, $k \mod 3 = 0$.

E.g., Striping of a table



- In the above figure, there are two tables *A* and *B* and both are *striped* across three disks.

Striping: Advantages

- E.g., Suppose there are 1024 blocks in table A.
  - Stripe 1 has blocks $A1, A4, A7, \ldots, A1024$, total 342 blocks.
  - Stripe 2 has blocks $A2, A5, \ldots, A1022$, total 341 blocks.
  - Stripe 3 has blocks $A3, A6, \ldots, A1023$,total 341 blocks.
- To retrieve all of table A, we have to retrieve 342, 341 and 341 blocks respectively from the stripes of A on disks 1,2 and 3.
- Assuming each stripe of A is on the same cylinder on that disk,
  1. a track can store 32 blocks, and so,
  2. a stripe is stored in at most $\lceil 342/32 \rceil = 11$ tracks.
  3. Last 11th track is not completely full: has respectively, 22, 21 and 21 blocks respectively in the three stripes.
- Recall, Megatron 747 has 32 tracks for a cylinder (16 platters, two-sided).

E.g., 3-way striping

- Time taken to access all blocks of *A* in some order:
- On each disk 1, 2 and 3:
  1. First seek to the right cylinder: avg time is 6.46ms.
  2. Read all blocks of that stripe of *A*: may take up to 11 rotations; 8.33ms $\times 11 = 90.667$ms.
  3. Time taken to finish reading a stripe $\approx 97.13ms$.
- On a single disk: the same calculation would be:

  avg. seek time 6.46ms $+$ 32 tracks/rotations $8.33 \times 32 \approx 274.13$ms

- Striping is close to factor of 3 times faster.
- *Note*: We have to wait till all 3 disks finish: there will be higher probability that one disk will have higher than average seek time.

Mirroring Disks for Reliability against Failure

- Keep say two copies of a table in two different disks.
- Both copies are exactly identical.
- These disks are said to be *mirrors* of each other.
- Advantages: Mainly, resilience to failure.
    - If there is a head crash on one disk, the other disk can still provide access to data.
    - While the other disk provides access to data, the first disk can be repaired or replaced.
    - System is designed to enhance reliability.

Mirroring: Pros and Cons

- Say we keep n copies of the data in n different disks.
- A read request can be satisfied by any of the n disks.
    - Read request may be directed to the least loaded disk. Hence, faster.
- Reliability against failure is enhanced:
    - A mirror fails, it is fixed/replaced and plugged back in.
    - In the meantime, $n - 1$ copies are used.
    - Once the failed disk is back online, n copies are now back.
- *Write/Update:* Updates are made to all n copies; one block is written on all copies.
- Time taken is close to the same as one update (seek times on all disks may not be the same, may be a little higher than one write).
- Write costs are not reduced.

Disk Scheduling and Elevator Algorithm

- Suppose at any time there are a number of pending disk accesses (read/writes).
- Assume that the accesses are from independent processes.
- *Elevator Algorithm*: for *fairness* and *no starvation*.
  1. Disk makes sweeps from the innermost cylinder to outermost and back.
  2. Similar to an elevator that makes sweeps from the bottom floor to the top floor and then back down.
  3. As the elevator passes a floor in a certain direction (up/down), it (a) drops passengers getting off on that floor, and, (b) picks up passengers going to floors along elevator's direction.
  4. Elevator records pending requests from floors and its direction (up/down).
  5. Elevator changes direction upon hitting the top-most floor beyond which there are no pending requests, or the bottom-most floor below which there are no pending requests.

Disk elevator algorithm

1. Disk makes sweeps from innermost cylinder to outermost cylinder and back.
2. As heads come upon a cylinder with a read/write request, the head stops and processes the request.
3. Heads proceed in the same direction they are travelling until the next cylinder with a processing request is reached.
4. When the heads reach a cylinder when there are no requests ahead of them in their direction of travel, they reverse direction.

Example: Disk elevator algorithm

- Megatron 747 disk: avg seek time 6.46ms, rotational latency 4.17ms, transfer time 0.13ms.
- Block Request timetable is given below.
- At time 0, head location is cylinder 8000.

Cylinder of Request	Time of Request
8000	0
24000	0
56000	0
16000	10
64000	20
40000	30

Ex: Elevator Algorithm

- Assume current position at time 0 is cylinder 8000.
- Each block incurs transfer time of 0.13ms and 4.17 for average rotational latency: total $= 4.17 + 0.13 = 4.30 ms$.
- Seek time: $1+$ number of tracks/4000 ms.
- Trace the elevator algorithm.

Tracing Elevator Algorithm

1. Time 0. Current position: cylinder 8000. Block is accessed. Time taken: 4.30ms.
2. Time: 4.30ms. Direction: Moves outward toward next request cylinder 24000.
3. Time taken: 1 + (24000-8000)/4000 = 5ms.
4. Time: 9.30ms. Position: Track 24000. Starts block access. Access takes 4.30ms.
5. Time: $9.30 + 4.30 = 13.60$ms. Direction outward. Request pending at track 56000. Continues moving outward to track 56000.
6. Time for seek: $1 + (56000 - 24000)/4000 = 9$ms.
7. Time: $13.60 + 9 = 22.60$ms. Starts block access, taking 4.30ms. Time: 26.90ms access completes.
8. Direction is outward. Request at cylinder 64000 pending,
9. Time for seek: $1 + (64000 - 56000)/4000 = 3$ms.
10. Time: $26.90 + 3 = 29.90$. Starts block access, taking 4.30ms. Completes at time 34.20ms.

Tracing Elevator Algorithm

1. Time: 34.20ms. No more pending requests with higher than 64000 cylinder number.
2. *Direction is reversed*. Now direction is inward.
3. In reverse direction, next pending request is at cylinder 40000.
4. Time for seek: $1 + (64000 - 40000)/4000 = 7$ms.
5. Time: $34.20 + 7 = 41.20ms$. Starts block access: 4.30ms. Time: 45.50ms, access completes.
6. Direction inward, pending request cylinder 16000.
7. Time for seek: $1 + (40000 - 16000)/4000 = 7$ms.
8. Time: $45.50 + 7 = 52.50$ms. Starts block access: 4.30ms. Time: 56.80ms, access completes.

Tracing Elevator Algorithm

Cylinder of Request	Time of Request	Time of Completion
8000	0	4.3
24000	0	13.6
56000	0	26.9
16000	10	56.8
64000	20	34.2
40000	30	45.5

Pre-fetching blocks

- In some applications, we can predict the order in which blocks will be requested from disk.
- If so, we can load them into memory buffers before they are actually needed.
- By requesting all these required blocks, the elevator algorithm may be better able to reduce seek times for each block required in the future.

Disk Failures

- *Intermittent failure*: attempt to read or write a sector is unsuccessful but with repeated tries (upto some max tries, say 100), the head is able to read or write successfully.
- *Media decay*. A sector has become bad; bits are permanently corrupted.
- *Write failure*: We can neither write successfully nor can we retrieve previously written sector. *Possible cause:* power failure during the writing of the sector.
- *Disk crash*: Entire disk becomes unreadable, suddenly and permanently.

Parity checks, Checksums

- *Parity bit*: Suppose a sector is of size 4200 bits.
- Out of these, the first 4096 bits = 512 bytes are used to store data in the sector.
- Let the sector be X, and successive bits are denoted as $X_1, X_2, \ldots, X_{4096}$.
- We define the *parity bit*, value of bit numbered 4097 as

$$X_{4097} = \text{parity bit} = X_1 \oplus X_2 \oplus X_3 \oplus \cdots \oplus X_{4096} \quad .$$

Review: Modulo 2 field

- The set $(\{0, 1\}, \oplus, \cdot)$ is accompanied by two operations:
- $\cdot$ for multiplication.
- $\oplus$ for addition (also called XOR in boolean operations).
- The operation tables are:

$\cdot$	0	1
0	0	0
1	0	1

$\oplus$	0	1
0	0	1
1	1	0

- Properties of field operations are satisfied.
  - $\cdot$ and $\oplus$ are each associative and commutative.
  - 0 is additive identity under $\oplus$, 1 is multiplicative identity over $\cdot$.
  - 0 and 1 are each of their respective inverses under $\oplus$.
  - Satisfies distributivity: $a \cdot (b \oplus c) = a \cdot b \oplus a \cdot c$.
  - Smallest finite field: commutative ring, $0 \neq 1$.

Back to Checksums

- A parity bit as the XOR of all the 4096 bits is defined.
- If *any one bit* among the 4097 bits is mis-read or is corrupted, then, the property that

$$X_{4097} = X_1 \oplus \cdots \oplus X_{4096}$$

will not be satisfied.
- Ex: Suppose there are 8 bits in the sector $X = 01101000$.
  - Its parity bit is $X_1 \oplus \cdots X_8 = X_9 = 1$.
  - X is replaced by $\bar{X} = 011010001$, the 9th bit is the parity bit.
  - Suppose any one of these 9 bits gets corrupted, e.g., 4th bit is read as a 1.
  - Then, the number of 1's in the read value of X is 4, and has parity even (XOR is 0).
  - Doesn't match X_9. Error detected.

Checksums and Error Detection

- As we saw, checksum can be used to detect a 1 bit error in an *n*-bit string.
- But if there was a 2 bit error in that *n*-bit string, parity would be the same.
- So checksum would not be able to detect 2 bit errors (more generally, even number of bit errors).
- When can we expect checksum to work?
- Suppose that bits may corrupt independently and each with probability p. (Some probability model of failure).

$$\mathbf{P}\left(1 \text{ bit failure }\right) = np(1-p)^{n-1}$$
$$\mathbf{P}\left(2 \text{ bit failure }\right) = \binom{n}{2} p^2 (1-p)^{n-2}$$

- $np \ll 1$ should hold for checksums to have effectiveness.

Checksums of smaller segments

- Prior example: sector size$=$ 4096 bits and 1 bit for checksum is added.
- Detects one error out of 4096, and is possibly effective only when $p \ll \frac{1}{4096}$.
- If probability of error is higher, use checksums for smaller segments of the sector.
- Ex. divide sector into words of size 32 bits each. 4096 bits $=$ 128 words each of size 32 bits.
- For each word, keep its checksum.
- We store $4096 + 128 = 4224$ bits, one bit checksum for each word.
- Sector reading: uses checksum for each word to check parity.
- Improves error-detection.

Correcting Errors: towards Stable Storage

- Checksums detect an error but do not correct the error.
- For disks, there is a notion of *stable storage*.
- Keep a pair of disks in sync with data stored on them.
- Each sector X is paired: call it X_L and X_R: two copies of X on each of the disks.
- Each sector of disk has sufficient parity bits to detect an error in that sector.
- In case of error, use the copy from the other disk,
  - while restoring/repairing/replacing the first disk.
- The stable storage method above reduces the probability of failure of the disk pair.

Simple probability model: Mean time to failure

- A very simple probability model for modeling failure of a device (disk, or sector) is the exponential distribution with parameter λ.
- Probability density function for failure at time t

$$f(t) = \lambda e^{-\lambda t}, \quad t > 0 \ .$$

- Probability that disk fails within $(0, T]$ is

$$\int_0^T \lambda e^{-\lambda t} = 1 - e^{-\lambda T} \ .$$

- Let T be lifetime of a disk whose failure is given by this distribution.

$$\mathbf{E}\left[T\right] = \int_{\lambda=0}^{\infty} \lambda t e^{-\lambda t} = \left[\lambda t(-)\frac{1}{\lambda}\lambda t\right]_0^{\infty} - \int_0^{\infty} e^{-\lambda t} dt = \frac{1}{\lambda} \ .$$

- This is often referred to as the *mean time to failure* MTTF.

Pair of disks: model

- Failure happens: when one disk fails and within its repair period, the second disk also fails.
- If a write was being done, and one disk fails, the write would succeed on the second, unless it fails during repair period.
- MTTF for disk pair model increases substantially.

Disaster Recovery model

- Model of failure: a data center has some major problem, e.g.,
  - Loss of network connectivity, (disks are often on a network), or,
  - natural disaster, or some other kind of disaster that afflicts the data center.
  - etc.
- All redundant copies of data in the data center may be inaccessible or unreliable.
- To recover from disasters, data is replicated in geographically distant parts.

Lower Level Error Correction: Example

- Back to lower level (sector, segments within sectors, etc.).
- Checksums allow us to detect a single bit error with reasonably high probability ($np(1-p)^{n-1}$, assuming $np \ll 1$).
- How can we recover from the error? How can we reconstruct the original bit vector?

Simple Example $(7, 4)$ code

- Suppose we have a four bit vector $x = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}$.
- Instead of sending 4 bits x, a 7-bit vector y is sent as follows. Matrix G:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}, \quad y = Gx = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_1 + x_2 + x_4 \\ x_1 + x_3 + x_4 \\ x_2 + x_3 + x_4 \end{bmatrix}$$

- Last three rows y_5, y_6, y_7 are called parity bits.

A (7,4) Hamming code

- Suppose x and x' differ in at least one bit, say $x_i \neq x'_i$.
- Then, $y = Gx$ and $y' = Gx'$ differ in at least 3 bits.
- We study two cases:
  1. Case 1: x and x' differ in exactly 1 bit.
  2. Case 2: x and x' differ in exactly 2 bits.

(7,4) Hamming: One bit differs

$$\text{Parity Matrix}: \quad \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

- In the three parity rows, each coordinate (column) has a 1 in at least two rows.
- If x, x' differ in that bit position (column) i, they will differ in at least two parity values.

(7,4) H-code: Case 2

$$\text{Parity Matrix}: \quad \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

- The parity matrix is the last three rows of G.
- Each pair of distinct column positions i, j have both 1's in at most two rows.
- Each pair of distinct col positions have at least one row that contains one but not the other.
- If x and x' differ in exactly two positions, then
- there is one row that has 1 in only one of these positions,
- Hence, the parity bit coming from that row will differ in x and x'.

Error Correction

- If x and x' differ by at least one bit, Gx and Gy differ by at least 3 bits.
- Correcting one bit error: Given a 7-bit y, find the x whose Gx differs from y by 1 bit.
- There cannot be two of those x's by triangle inequality.
- $|y - y'| =$ no of bits where y and y' differ. *Hamming Distance.*

Write failure

- Say we are writing a block X.
- During the process of writing the block, there can be power outage.
- Then, the write is incomplete and hence,
  1. The writing is incomplete and the block X is incorrect.
  2. We have lost the old copy of X as well.
  3. i.e., we have neither the old copy nor the new correct value of X.
- Two ways have come about to address this issue:
  1. Disk controller can use limited non-volatile memory store the new value of X. (RAID)
  2. Use mirroring of X into X_L and X_R (left and right copies of X). (RAID)

Write failure and mirroring

- Say failure occurred as we are writing a sector X_L.
- Then, checksums and other parity checks associated with the sector will fail.
- We will detect that X_L is “bad”.
- Writing to X_R is delayed, so X_R has the old value.
- Writing to X_R is slightly delayed, only after X_L is successfully written.
- This way, the old value of X_L is restored from X_R.
- Possibly, failure occurred after X_L was written but while X_R was being written.
- Then, X_R restores the new value from X_L.

Mirroring as a Redundancy Technique: RAID level 1

- Consider a simple case where data is replicated identically on two disks.
- Call the disks, *data disk* and *redundant disk* for notation.
- Data can be accessed from either disk, in case the data disk has failed,
  - data accesses are done using the mirror disk,
  - while, the original data disk is repaired/replaced.
  - If needed, data for the original disk is copied from the mirror disk.
- Failure of disk pair: During the fault period of disk 1, the other disk also fails.

Parity Blocks

- Summary: Mirroring disks reduces the probability of a disk crash,
  - number of redundant (mirror) disks is the same as data disks.
- *RAID level 4* approach: uses only one redundant disk. E.g.
- Assume three identical Megatron 747 disks:
  - Each disk has block size 16KB = $16,384 \times 8 = 131072$ bits.
  - blocks are numbered 1 to n in each of the disks.
  - Data disks however may store different data in their blocks.
- *RAID 4* keeps a fourth disk for parity.
- Parity block n, denoted P_n, keeps the parity checksum of the data in blocks B_n^1, B_n^2, B_n^3, the nth block of disks 1,2,3 respectively.

$$P_n = B_n^1 + B_n^2 + B_n^3$$

where, $+$ is co-ordinate wise $\oplus$ or XOR operation.

Example: Parity block

- Simplify a block to have 8 bits. There are three data disks.
- The first blocks on these three disks are

disk1	B_1^1	11110000
disk2	B_1^2	10101010
disk3	B_1^3	00111000

- The redundant or parity disk for block 1 is

Parity disk	P_1	01100010

Vector Addition is simply coordinate-wise using XOR ($\oplus$)

$$\begin{bmatrix}1 & 1 & 1 & 1 & 0 & 0 & 0 & 0\end{bmatrix} \oplus \begin{bmatrix}1 & 0 & 1 & 0 & 1 & 0 & 1 & 0\end{bmatrix}$$
$$\oplus \begin{bmatrix}0 & 0 & 1 & 1 & 1 & 0 & 0 & 0\end{bmatrix}$$
$$= \begin{bmatrix}0 & 1 & 1 & 0 & 0 & 0 & 1 & 0\end{bmatrix}$$

RAID level 4: reading

- Generally, there is no reason to read from the redundant disk.

RAID level 4: writing

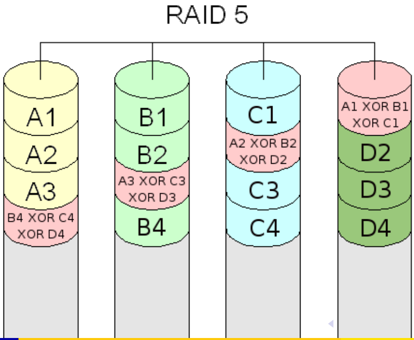
- Consider a block in a data disk that gets updated (written).
- The block has to be rewritten, and the corresponding parity block has to be updated.
- Say block B_1 of disk 1 is being updated.
- Let O_1 be the old value of this block, N_1 is the new value to be updated.
- Let P_1 be the old parity value for first block.
- The new parity block value is

$$\text{new parity block } 1 = P_1 + O_1 + N_1 \quad .$$

- There is no need to consult the other blocks.
- *Still:* Need to write 2 blocks, and parity disk is a bottleneck.
- This is improved in *RAID level 5*.

RAID level 5

- E.g., Four identical capacity disks: parity blocks are shared amongst all four disks.
- Data blocks in the "so called" data disks are divided as $A1, A2, A3, \ldots, An$; $B1, B2, B3, \ldots, Bn$; $C1, C2, \ldots, Bn$; $D1, \ldots, Dn$. Not all data blocks are present, they are replaced by parity block.



RAID level 5

- Parity block P_i is the XOR of the other three three data blocks A_i, B_i and C_i, e.g.,

$$P_1 = A_1 \oplus B_1 \oplus C_1 \qquad \text{Parity block is } D_1$$
$$P_2 = A_2 \oplus B_2 \oplus D_2 \qquad \text{Parity block is } C_2$$
$$P_3 = A_3 \oplus C_3 \oplus D_3 \qquad \text{Parity block is } B_3$$

- E.g., Number the disks $d = 0, 1, 2$ and 3. Disks are identical, the blocks in ith disk is $B^i_1, B^i_2, \ldots, B^i_n$.
- In above diagram j th numbered parity block P_j is stored in disk numbered $i = (j + 2) \mod 4$. P_j is the value of block B^i_j.
- For ith disk, data is stored sequentially in each block j except when $i = (j + 2) \mod 4$. This stores the parity block for the jth block sequence: $B^1_j, \ldots, B^4_j$, except $i = (j + 2) \mod 4$.

RAID level 5

- As in RAID level 4, any write operation to a data block must update the corresponding parity block using old copy and new copy.
- In general, two blocks are updated on two disks, and parity blocks are shared among the disks. *No disk is a bottleneck*.

Recovery from failure of one disk

- RAID level 4 can do failure recovery for one disk as follows:
  1. Suppose parity disk fails. A new disk is obtained.
  2. For each block i: parity block P_i is calculated as $B_i^1 \oplus \cdots \oplus B_d^i$, assuming d data disks.
- Suppose data disk numbered j fails.
  1. Block i of disk j is reconstructed as follows:

$$B_i^j = B_i^1 \oplus \cdots \oplus B_i^{j-1} \oplus B_i^{j+1} \oplus \cdots \oplus B_i^d \oplus P_i \quad .$$

- Assumption: During the process of failure recovery, no other disk fails.
- *RAID level 5:* Failure recovery is similar, except that disk j that fails has a combination of parity blocks and data blocks.
- Recovery process is the same. Assumption is the same.

RAID level 6: Correcting Multiple disk crashes

- As an example, suppose we use the $(7, 4)$ Hamming code.
- That is, use 4 data disks and 3 disks for parity. Parity bits are defined as per $(7, 4)$ Hamming code.
- Data disks are numbered 1 to 4, parity disks numbered 5,6,7. Disks have identical storage.
- Fix some bit index l among the disks. (i.e., we are considering the lth bit of disk 1, lth bit of disk 2, and so on, and for parity disks).

RAID 6: Hamming code $(7, 4)$

- For any fixed l, let x_i denote the lth bit of disk i, $i = 1, \ldots, 4$ are data disks, $i = 5, 6, 7$ are parity disks. The parity matrix P is:

$$\begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ 1 & 1 & 1 & 0 & 1 & & \\ 1 & 0 & 1 & 1 & & 1 & \\ 0 & 1 & 1 & 1 & & & 1 \end{array}$$

- The above parity matrix can be interpreted in multiple ways, e.g., Each row adds to 0, that is, $Px = 0$.

RAID 6: (7,4) Hamming code

- Another way:
  1. P in block form $P = \begin{bmatrix} D & I \end{bmatrix}$, where, D is the first 3×4 submatrix block, and I is 3×3 identity matrix.
  2. Accordingly $x = \begin{bmatrix} x_D \\ x_P \end{bmatrix}$, where,
  3. x_D is the data component of x, and
  4. x_P is the parity component of x.
  5. $x_D = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$ and $x_P = \begin{bmatrix} x_5 \\ x_6 \\ x_7 \end{bmatrix}$.
  6. $Px = 0$ is equivalent to $Dx_D = x_P$.

How to correct 2 disk crashes

- The data part 3×4 matrix D of the parity matrix P has the property:
  - for any pair (i, j) of distinct disk indices between 1 and 4, there is some row r in the parity matrix such that exactly one of $P_{r,i}$ or $P_{r,j}$ equals 1, the other is 0.
- E.g., Suppose disk 1 and 3 fails. Let $i = 1, j = 3$.
- The third row of P is $\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$. $P_{3,1} = 0$ and $P_{3,3} = 1$.
- Failure recovery: Using 3rd row of P, we have

$$P_3 x = 0 \text{ is equivalent to } x_2 + x_3 + x_4 = x_7$$

- Value of x_3 is reconstructed as:

$$x_3 = x_7 + x_4 + x_2 \quad .$$

Failure recovery: Example

- The other (simpler) property: for every pair (i, j) there is at least one row r where both coefficients $P_{r,i}$ and $P_{r,j} = 1$.
- E.g. continued: row 1 of P is $P_1 = \begin{bmatrix}1 & 1 & 1 & 0 & 1 & 0 & 0\end{bmatrix}$. Here, $P_{11} = P_{13} = 1$.
- The row equation $P_1 x = 0$ gives

$$x_1 + x_2 + x_3 = x_5$$

- Since, x_3 has been correctly reconstructed, x_1 is reconstructed as:

$$x_1 = x_5 + x_2 + x_3 \quad .$$

Failure recovery: Hamming distance

- We used the example of Hamming $(7, 4)$ code. Here 4 is the number of data bits, 7 is the total length of code, i.e., there are $7 - 4 = 3$ parity bits.
- Any pair u, v of distinct codes from among the $2^4 = 16$ possible 7-bit code vectors, they differ in at least 3 bits.
- *Hamming distance:* $|u - v| = \{$ number of bit positions where u and v differ $\}$.
- It satisfies the properties of a distance metric: (S, d), where, S is a set of items.
  1. $d(u, u) = 0$, for all $u \in S$,
  2. $d(u, v) = d(v, u)$, for any pair u, v from S and
  3. $d(u, v) \leq d(u, w) + d(w, v)$, for any vectors u, v, w from S: *Triangle Inequality*.

Failure recovery: Existence

- In coding theory terms, the Hamming code $(7, 4)$ has *minimum distance* $d = 3$.
  - Means that any two 7-bit codewords differ in at least 3 bits.
- In coding theory, generally a code is referred to as (n, d) to mean that codewords are of size n and the minimum distance is d.
  - Any two n-bit codewords have a distance of at least d. (underlying distance metric is assumed Hamming distance).
- Suppose RAID 6 uses (n, d) code. Then it can correct $d - 1$ disk failures.

RAID 6: (n, d) codes to correct $d - 1$ failures

- Suppose there are $d - 1$ (or less) disk failures.
- Let x be the n-vector and let F be the set of at most $d - 1$ indices in $1, 2, \ldots, n$ corresponding to disks that failed.
- The coordinates of x from x, denoted as x_F are unreliable.
- Suppose there are two codewords y and z such that
  1. y and z each agree on all coordinates of x except F.
  2. Then, y and z differ in at most coordinates of F, hence at most $d - 1$ coordinates.
  3. But minimum distance among any two codewords is d. *Contradiction*.
- Therefore, there is only one *completion* possible of x_F to restore x.