Unit No.: 02 – Data Protection, Intelligent Storage system

Chapter 3 - Data Protection: RAID

Introduction
In the late 1980s, rapid adoption of computers for business processes stimulated the growth of new applications and databases, significantly increasing the demand for storage capacity. At that time, data was stored on a single large, expensive disk drive called Single Large Expensive Drive (SLED). Use of single disks could not meet the required performance levels, due to their limitations.

HDDs are susceptible to failures due to mechanical wear and tear and other environmental factors. An HDD failure may result in data loss. The solutions available during the 1980s were not able to meet the availability and performance demands of applications.

An HDD has a projected life expectancy before it fails. Mean Time Between Failure (MTBF) measures (in hours) the average life expectancy of an HDD. Today, data centers deploy thousands of HDDs in their storage infrastructures. The greater the number of HDDs in a storage array, the greater the probability of a disk failure.

RAID is an enabling technology that leverages multiple disks as part of a set, which provides data protection against HDD failures. In general, RAID implementations also improve the I/O performance of storage systems by storing data across multiple HDDs.

Chapter Objective
This chapter details RAID technology, RAID levels, and different types of RAID implementations and their benefits.

3.1 Implementation of RAID

There are two types of RAID implementation, hardware and software. Both have their merits and demerits.

3.1.1 Software RAID
Software RAID uses host-based software to provide RAID functions. It is implemented at the operating-system level and does not use a dedicated hardware controller to manage the RAID array.

Software RAID implementations offer cost and simplicity benefits when compared with hardware RAID.

Limitations of software RAID:
1. Performance: Software RAID affects overall system performance. This is due to the additional CPU cycles required to perform RAID calculations.
2. Supported features: Software RAID does not support all RAID levels.
3. Operating system compatibility: Software RAID is tied to the host operating system hence upgrades to software RAID or to the operating system should be validated for compatibility.

3.1.2 Hardware RAID

In hardware RAID implementations, a specialized hardware controller is implemented either on the host or on the array.

Controller card RAID is host-based hardware RAID implementation in which a specialized RAID controller is installed in the host and HDDs are connected to it. RAID Controller interacts with the hard disks using the PCI bus.

Manufacturers integrate RAID controllers on motherboards. This reduces the overall cost of the system, but does not provide the flexibility required for high-end storage systems.

The external RAID controller is an array-based hardware RAID. It acts as an interface between host and disks. It presents storage volumes to host, which manage the drives using the supported protocol.

Key functions of RAID controllers are:
1. Management and control of disk aggregations
2. Translation of I/O requests between logical disks and physical disks
3. Data regeneration in the event of disk failures.

3.2 RAID Array Components
A RAID array is an enclosure that contains a number of HDDs and the supporting hardware and software to implement RAID.
HDDs inside a RAID array are contained in smaller sub-enclosures. These sub-enclosures, or physical arrays, hold a fixed number of HDDs, and also include other supporting hardware, such as power supplies.

![RAID Array Components](image)

*A subset of disks within a RAID array can be grouped to form logical associations called logical arrays, also known as a RAID set or a RAID group (see Figure 3-1). Logical arrays are comprised of logical volumes (LV). The operating system recognizes the LVs as if they are physical HDDs managed by the RAID controller.*

3.3 RAID Levels
RAID levels are defined on the basis of 1) striping, 2) mirroring, and 3) parity techniques. These techniques determine the data availability and performance characteristics of an array. Some RAID arrays use one technique, whereas others use a combination of techniques.
3.3.1 Striping
A RAID set is a group of disks. Within each disk, a predefined number of contiguously addressable disk blocks are defined as *strips*. The set of aligned strips that spans across all the disks within the RAID set is called a *stripe* (Figure 3-2).

**Figure 3-2: Striping**
Strip size (also called stripe depth) describes the number of blocks in a strip, and is the maximum amount of data that can be written to or read from a single HDD in the set. All strips in a stripe have the same number of blocks.

Stripe width refers to the number of data strips in a stripe.
Striped RAID does not protect data unless parity or mirroring is used. Striping significantly improves I/O performance.

### Table 3-1: Raid Levels

<table>
<thead>
<tr>
<th>LEVELS</th>
<th>BRIEF DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAID 0</td>
<td>Striped array with no fault tolerance</td>
</tr>
<tr>
<td>RAID 1</td>
<td>Disk mirroring</td>
</tr>
<tr>
<td>RAID 3</td>
<td>Parallel access array with dedicated parity disk</td>
</tr>
<tr>
<td>RAID 4</td>
<td>Striped array with independent disks and a dedicated parity disk</td>
</tr>
<tr>
<td>RAID 5</td>
<td>Striped array with independent disks and distributed parity</td>
</tr>
<tr>
<td>RAID 6</td>
<td>Striped array with independent disks and dual distributed parity</td>
</tr>
<tr>
<td>Nested</td>
<td>Combinations of RAID levels. Example: RAID 1 + RAID 0</td>
</tr>
</tbody>
</table>

3.3.2 Mirroring
Mirroring is a technique whereby data is stored on two different HDDs, yielding two copies of data. In the event of one HDD failure, the data is intact on the surviving HDD (see Figure 3-3) And the controller continues to service the host’s data requests from the surviving disk.

![Figure 3-3: Mirrored disks in an array](image)

When the failed disk is replaced with a new disk, the controller copies the data from the surviving disk of the mirrored pair.

Mirroring provides complete data redundancy, and also enables faster recovery from disk failure. Mirroring is not a substitute for data backup. Mirroring constantly captures changes in the data, whereas a backup captures point-in-time images of data.

Mirroring involves duplication of data - amount of storage capacity needed is twice the amount of data being stored.

Therefore, mirroring is considered expensive and is preferred for mission-critical applications that cannot afford data loss.

Mirroring improves read performance because read requests can be serviced by both disks. However, write performance decreases, as each write request must perform two writes on the HDDs.

### 3.3.3 Parity

**Parity** is a method of protecting striped data from HDD failure without the cost of mirroring. An additional HDD is added to the stripe width to hold parity. Parity is a redundancy check that ensures full protection of data without maintaining a full set of duplicate data.

Parity information can be stored on separate, dedicated HDDs or distributed across all the drives in a RAID set. Figure 3-4 shows a parity RAID. The first four disks, labeled $D$, contain the data. The fifth disk, labeled $P$, stores the parity information, which is the sum of the elements in each row. Now, if one of the $D$s fails, the missing value can be calculated by subtracting the sum of the rest of the elements from the parity value.
The computation of parity is represented as a simple arithmetic operation on the data. Parity calculation is a bitwise XOR operation. Calculation of parity is a function of the RAID controller.

**Advantage:** Compared to mirroring, parity implementation considerably reduces the cost associated with data protection.

Consider a RAID configuration with five disks. Four of these disks hold data, and the fifth holds parity information. Parity requires 25 percent extra disk space compared to mirroring, which requires 100 percent extra disk space.

**Disadvantage:** Parity information is generated from data on the data disk. Therefore, parity is recalculated every time there is a change in data. This recalculation is time-consuming and affects the performance of the RAID controller.

### 3.3.4 RAID 0

In a RAID 0 configuration, data is striped across the HDDs in a RAID set. It utilizes the full storage capacity by distributing strips of data over multiple HDDs.

- To read data, all the strips are put back together by the controller.
- The stripe size is specified at a host level for software RAID and is vendor specific for hardware RAID.

Figure 3-5 shows RAID 0 on a storage array in which data is striped across 5 disks. When the number of drives in the array increases, performance improves because more data can be read or written simultaneously.

**Adv:** RAID 0 is used in applications that need high I/O throughput.

**Disadv:** RAID 0 does not provide data protection and availability in the event of drive failures.
3.3.5 RAID 1

In a RAID 1 configuration, data is mirrored to improve fault tolerance (Figure 3-6). A RAID 1 group consists of at least two HDDs. As explained in mirroring, every write is written to both disks. In the event of disk failure, the impact on data recovery is the least among all RAID implementations. This is because the RAID controller uses the mirror drive for data recovery and continuous operation.

RAID 1 is suitable for applications that require high availability.

3.3.6 Nested RAID

Most data centers require data redundancy and performance from their RAID arrays. RAID 0+1 and RAID 1+0 combine the performance benefits of RAID 0 with the redundancy benefits of RAID 1. They use striping and mirroring techniques and combine their benefits. These types of RAID require an even number of disks, the minimum being four (see Figure 3-7).

RAID 1+0 is also known as RAID 10 (Ten) or RAID 1/0. Similarly, RAID 0+1 is also known as RAID 01 or RAID 0/1.

RAID 1+0 performs well for workloads that use small, random, write-intensive I/O.

Some applications that benefit from RAID 1+0 include the following:
1. High transaction rate Online Transaction Processing (OLTP)
2. Large messaging installations
3. Database applications that require high I/O rate, random access, and high availability.
A common misconception is that RAID 1+0 and RAID 0+1 are the same. **RAID 1+0 is also called striped mirror.** The basic element of RAID 1+0 is a mirrored pair, which means that data is first mirrored and then both copies of data are striped across multiple HDDs in a RAID set. When replacing a failed drive, only the mirror is rebuilt, i.e. the disk array controller uses the surviving drive in the mirrored pair for data recovery and continuous operation. Data from the surviving disk is copied to the replacement disk.

**RAID 0+1 is also called mirrored stripe.** The basic element of RAID 0+1 is a stripe. This means that the process of striping data across HDDs is performed initially and then the entire stripe is mirrored. If one drive fails, then the entire stripe is faulted. A rebuild operation copies the entire stripe, copying data from each disk in the healthy stripe to an equivalent disk in the failed stripe.

**Disadv:** This causes increased and unnecessary I/O load on the surviving disks and makes the RAID set more vulnerable to a second disk failure.

### 3.3.7 RAID 3
RAID 3 stripes data for high performance and uses parity for improved fault tolerance. Parity information is stored on a dedicated drive so that data can be reconstructed if a drive fails. For example, of five disks, four are used for data and one is used for parity. Therefore, the total disk space required is 1.25 times the size of the data disks.

RAID 3 always reads and writes complete stripes of data across all disks, as the drives operate in parallel. There are no partial writes that update one out of many strips. Figure 3-8 illustrates the RAID 3 implementation. RAID 3 provides good bandwidth for the transfer of large volumes of data. RAID 3 is used in applications that involve large sequential data access, such as video streaming.

---

**Figure 3-7:** Nested RAID

A common misconception is that RAID 1+0 and RAID 0+1 are the same. **RAID 1+0 is also called striped mirror.** The basic element of RAID 1+0 is a mirrored pair, which means that data is first mirrored and then both copies of data are striped across multiple HDDs in a RAID set. When replacing a failed drive, only the mirror is rebuilt, i.e. the disk array controller uses the surviving drive in the mirrored pair for data recovery and continuous operation. Data from the surviving disk is copied to the replacement disk.

**RAID 0+1 is also called mirrored stripe.** The basic element of RAID 0+1 is a stripe. This means that the process of striping data across HDDs is performed initially and then the entire stripe is mirrored. If one drive fails, then the entire stripe is faulted. A rebuild operation copies the entire stripe, copying data from each disk in the healthy stripe to an equivalent disk in the failed stripe.

**Disadv:** This causes increased and unnecessary I/O load on the surviving disks and makes the RAID set more vulnerable to a second disk failure.

### 3.3.7 RAID 3
RAID 3 stripes data for high performance and uses parity for improved fault tolerance. Parity information is stored on a dedicated drive so that data can be reconstructed if a drive fails. For example, of five disks, four are used for data and one is used for parity. Therefore, the total disk space required is 1.25 times the size of the data disks.

RAID 3 always reads and writes complete stripes of data across all disks, as the drives operate in parallel. There are no partial writes that update one out of many strips. Figure 3-8 illustrates the RAID 3 implementation. RAID 3 provides good bandwidth for the transfer of large volumes of data. RAID 3 is used in applications that involve large sequential data access, such as video streaming.
3.3.8 RAID 4

Similar to RAID 3, RAID 4 stripes data for high performance and uses parity for improved fault tolerance (refer to Figure 3-8). Data is striped across all disks except the parity disk. Parity information is stored on a dedicated disk so that the data can be rebuilt if a drive fails. Striping is done at the block level. Unlike RAID 3, data disks in RAID 4 can be accessed independently so that specific data elements can be read or written on single disk without read or write of an entire stripe. RAID 4 provides good read throughput and reasonable write throughput.

3.3.9 RAID 5

RAID 5 is a very versatile RAID implementation. It is similar to RAID 4 because it uses striping and the drives (strips) are independently accessible. The difference between RAID 4 and RAID 5 is the parity location. In RAID 4, parity is written to a dedicated drive, creating a write bottleneck for the parity disk. In RAID 5, parity is distributed across all disks. The distribution of parity in RAID 5 overcomes the write bottleneck. Figure 3-9 illustrates the RAID 5 implementation.

RAID 5 is preferred for messaging, data mining, medium-performance media serving, and relational database management system (RDBMS) implementations in which database administrators (DBAs) optimize data access.
3.3.10 RAID 6

RAID 6 includes a second parity element to enable survival in the event of the failure of two disks in a RAID group (Figure 3-10). Therefore, a RAID 6 implementation requires at least four disks. RAID 6 distributes the parity across all the disks.

The write penalty in RAID 6 is more than that in RAID 5; therefore, RAID 5 writes perform better than RAID 6. The rebuild operation in RAID 6 may take longer than that in RAID 5 due to the presence of two parity sets.
3.4 RAID Comparison

<table>
<thead>
<tr>
<th>RAID</th>
<th>MIN. DISKS</th>
<th>STORAGE EFFICIENCY %</th>
<th>COST</th>
<th>READ PERFORMANCE</th>
<th>WRITE PERFORMANCE</th>
<th>WRITE PENALTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>100</td>
<td>Low</td>
<td>Very good for both random and sequential read</td>
<td>Very good</td>
<td>No</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>50</td>
<td>High</td>
<td>Good. Better than a single disk.</td>
<td>Good. Slower than a single disk, as every write must be committed to all disks.</td>
<td>Moderate</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>(n-1)*100/n where n= number of disks</td>
<td>Moderate</td>
<td>Good for random reads and very good for sequential reads.</td>
<td>Poor to fair for small random writes. Good for large, sequential writes.</td>
<td>High</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>(n-1)*100/n where n= number of disks</td>
<td>Moderate</td>
<td>Very good for random reads. Good to very good for sequential writes.</td>
<td>Poor to fair for random writes. Fair to good for sequential writes.</td>
<td>High</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>(n-1)*100/n where n= number of disks</td>
<td>Moderate</td>
<td>Very good for random reads. Good for sequential reads</td>
<td>Fair for random writes. Slower due to parity overhead. Fair to good for sequential writes.</td>
<td>High</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>(n-2)*100/n where n= number of disks</td>
<td>Moderate but more than RAID 5</td>
<td>Very good for random reads. Good for sequential reads.</td>
<td>Good for small, random writes (has write penalty).</td>
<td>Very High</td>
</tr>
<tr>
<td>1+0 and 0+1</td>
<td>4</td>
<td>50</td>
<td>High</td>
<td>Very good</td>
<td>Good</td>
<td>Moderate</td>
</tr>
</tbody>
</table>

3.5 RAID Impact on Disk Performance

When choosing a RAID type, it is imperative to consider the impact to disk performance and application IOPS. In both mirrored and parity RAID configurations, every write operation translates into more I/O overhead for the disks which is referred to as write penalty.

Figure 3-11 illustrates a single write operation on RAID 5 that contains a group of five disks. Four of these disks are used for data and one is used for parity.

The parity (P) at the controller is calculated as follows:

\[ Ep = E1 + E2 + E3 + E4 \] (XOR operations)

Here, D1 to D4 is striped data across the RAID group of five disks.

Whenever controller performs a write I/O, parity must be computed by reading the old parity (Ep old) and the old data (E4 old) from the disk, i.e. two read I/Os.
The new parity (Ep new) is computed as follows:

\[ \text{Ep new} = \text{Ep old} - \text{E4 old} + \text{E4 new} \] (XOR operations)

After computing the new parity, controller completes write I/O by writing the new data and new parity onto the disks, amounting to two write I/Os.

Therefore, controller performs two disk reads and two disk writes for every write operation, and the write penalty in RAID 5 implementations is 4.

![Diagram of RAID 5 parity computation](image)

**Figure 3-11: Write penalty in RAID 5**

### 3.5.1 Application IOPS and RAID Configurations

When deciding the number of disks required for an application, it is important to consider the impact of RAID based on IOPS generated by the application. The total disk load should be computed by considering the type of RAID configuration and the ratio of read compared to write from the host.

The following example illustrates the method of computing the disk load in different types of RAID.

Consider an application that generates 5,200 IOPS, with 60 percent of them being reads.

The disk load in RAID 5 is calculated as follows:

RAID 5 disk load = \(0.6 \times 5,200 + 4 \times (0.4 \times 5,200)\) [because the write penalty for RAID 5 is 4]

\[= 3,120 + 4 \times 2,080\]

\[= 3,120 + 8,320\]

\[= 11,440 \text{ IOPS}\]

The disk load in RAID 1 is calculated as follows:

RAID 1 disk load = \(0.6 \times 5,200 + 2 \times (0.4 \times 5,200)\) [because every write manifests as two writes to the disks]

\[= 3,120 + 2 \times 2,080\]

\[= 3,120 + 4,160\]

\[= 7,280 \text{ IOPS}\]

Computed disk load determines the number of disks required for the application.

If in this example an HDD with a specification of a maximum 180 IOPS for the application needs to be used, the number of disks required to meet the workload for the RAID configuration as follows:
RAID 5: 11,440 / 180 = 64 disks  
RAID 1: 7,280 / 180 = 42 disks (approximated to the nearest even number)

3.6 Hot Spares
A hot spare refers to a spare HDD in a RAID array that temporarily replaces a failed HDD of a RAID set. A hot spare takes the identity of the failed HDD in the array. One of the following methods of data recovery is performed depending on the RAID implementation:
1. If parity RAID is used, then the data is rebuilt onto the hot spare from the parity and the data on the surviving HDDs in the RAID set.
2. If mirroring is used, then the data from the surviving mirror is used to copy the data.

When the failed HDD is replaced with a new HDD, one of the following takes place:
1. The hot spare replaces the new HDD permanently. This means that it is no longer a hot spare, and a new hot spare must be configured on the array.
2. When a new HDD is added to the system, data from the hot spare is copied to it. The hot spare returns to its idle state, ready to replace the next failed drive.

A hot spare should be large enough to accommodate data from a failed drive. System can implement multiple hot spares to improve data availability.

A hot spare can be configured as automatic or user initiated, which specifies how it will be used in the event of disk failure.

In an automatic configuration, when the recoverable error rates for a disk exceed a predetermined threshold, the disk subsystem tries to copy data from the failing disk to the hot spare automatically. If this task is completed before the damaged disk fails, then the subsystem switches to the hot spare and marks the failing disk as unusable.

Chapter 4 - Intelligent Storage System
Introduction
Business-critical applications require high levels of performance, availability, security, and scalability. A hard disk drive is a core element of storage that governs the performance of any storage system. RAID technology made an important contribution to enhancing storage performance and reliability, but hard disk drives even with a RAID implementation could not meet performance requirements of today’s applications.

With advancements in technology, a new breed of storage solutions known as an intelligent storage system has evolved. The intelligent storage systems detailed in this chapter are the feature-rich RAID arrays that provide highly optimized I/O processing capabilities. These arrays have an operating environment that controls the management, allocation, and utilization of storage resources.

4.1 Components of an Intelligent Storage System
An intelligent storage system consists of four key components: front end, cache, back end, and physical disks. Figure 4-1 illustrates these components and their interconnections. An I/O request received from the host at the front-end port is processed through cache and the back end, to enable storage and retrieval of data from the physical disk. A read request can be serviced directly from cache if the requested data is found in cache.

4.1.1 Front End
The front end provides the interface between the storage system and the host. It consists of two components: front-end ports and front-end controllers.
1. **Front-end ports**: enable hosts to connect to the intelligent storage system. Each front-end port has processing logic that executes the appropriate transport protocol, such as SCSI, Fibre Channel, or iSCSI, for storage connections. Redundant ports are provided on the front end for high availability.

2. **Front-end controllers**: route data to and from cache via the internal data bus.

![Diagram of intelligent storage system](image)

**Figure 4-1:** Components of an intelligent storage system

**Front-End Command Queuing:** Command queuing is a technique implemented on front-end controllers. It determines the execution order of received commands and can reduce unnecessary drive head movements and improve disk performance.

When a command is received for execution, the command queuing algorithms assigns a tag that defines a sequence in which commands should be executed. With command queuing, multiple commands can be executed concurrently based on the organization of data on the disk, regardless of the order in which the commands were received.

The most commonly used command queuing algorithms are as follows:

1. **First In First Out (FIFO)**: This is the default algorithm where commands are executed in the order in which they are received (Figure 4-2 [a]). There is no reordering of requests for optimization; therefore, it is inefficient in terms of performance.

2. **Seek Time Optimization**: Commands are executed based on optimizing read/write head movements, which may result in reordering of commands. Without seek time optimization, the commands are executed in the order they are received. For example, as shown in Figure 4-2(a), the commands are executed in the order A, B, C and D. The radial movement required by the head to execute C immediately after A. With seek time optimization, the command execution sequence would be A, C, B and D, as shown in Figure 4-2(b).

3. **Access Time Optimization**: Commands are executed based on the combination of seek time optimization and an analysis of rotational latency for optimal performance.
4.1.2 Cache
Cache is an important component that enhances the I/O performance in an intelligent storage system. Cache is semiconductor memory where data is placed temporarily to reduce the time required to service I/O requests from the host. Cache improves storage system performance by isolating hosts from the mechanical delays associated with physical disks, which are the slowest components of an intelligent storage system. Accessing data from cache takes less than a millisecond. Write data is placed in cache and then written to disk. After the data is securely placed in cache, the host is acknowledged immediately.

Structure of Cache: Cache is organized into pages or slots, which is the smallest unit of cache allocation. The size of a cache page is configured according to the application I/O size. Cache consists of the data store and tag RAM. The data store holds the data while tag RAM tracks the location of the data in the data store (see Figure 4-3) and in disk.

Entries in tag RAM indicate where data is found in cache and where the data belongs on the disk. Tag RAM includes a dirty bit flag, which indicates whether the data in cache has been committed to the disk or not. It also contains time-based information, such as the time of last access, which is used to identify cached information that has not been accessed for a long period and may be freed up.
Read Operation with Cache: When a host issues a read request, the front-end controller accesses the tag RAM to determine whether the required data is available in cache. If the requested data is found in the cache, it is called a read cache hit or read hit and data is sent directly to the host, without any disk operation (see Figure 4-4[a]). This provides a fast response time to the host (about a millisecond).

If the requested data is not found in cache, it is called a read cache miss or read miss and the data must be read from the disk (see Figure 4-4[b]). The back-end controller accesses the appropriate disk and retrieves the requested data. Data is then placed in cache and is finally sent to the host through the front-end controller. Cache misses increase I/O response time.

A pre-fetch, or read-ahead, algorithm is used when read requests are sequential. In a sequential read request, a contiguous set of associated blocks is retrieved. Several other blocks that have not yet been requested by the host can be read from the disk and placed into cache in advance. When the host subsequently requests these blocks, the read operations will be read hits. This process significantly improves the response time experienced by the host.

The intelligent storage system offers fixed and variable pre-fetch sizes.

In fixed pre-fetch, the intelligent storage system pre-fetches a fixed amount of data. It is most suitable when I/O sizes are uniform.

In variable pre-fetch, the storage system pre-fetches an amount of data in multiples of the size of the host request.

Figure 4-4: Read hit and read miss

Write Operation with Cache: Write operations with cache provide performance advantages over writing directly to disks. When an I/O is written to cache and acknowledged, it is completed in less time (from the host’s perspective) than it would take to write directly to disk.

A write operation with cache is implemented in the following ways:

1. Write-back cache: Data is placed in cache and an acknowledgment is sent to the host immediately. Later, data from several writes are committed (de-staged) to the disk. Write response times are much faster, as the write operations are isolated from the mechanical delays of the disk. However, uncommitted data is at risk of loss in the event of cache failures.

2. Write-through cache: Data is placed in the cache and immediately written to the disk, and an acknowledgment is sent to the host. Because data is committed to disk as it arrives, the risks of data loss are low but write response time is longer because of the disk operations.

Cache Implementation: Cache can be implemented as either dedicated cache or global cache. With dedicated cache, separate sets of memory locations are reserved for reads and writes. In global cache, both reads and
writes can use any of the available memory addresses. Cache management is more efficient in a global cache implementation, as only one global set of addresses has to be managed.

**Cache Management:** Cache is a finite and expensive resource that needs proper management. When all cache pages are filled, some pages have to be freed up to accommodate new data and avoid performance degradation. Various cache management algorithms are implemented in intelligent storage systems:

1. **Least Recently Used (LRU):** An algorithm that continuously monitors data access in cache and identifies the cache pages that have not been accessed for a long time. LRU either frees up these pages or marks them for reuse. This algorithm is based on the assumption that data which hasn’t been accessed for a while will not be requested by the host.
2. **Most Recently Used (MRU):** In MRU, the pages that have been accessed most recently are freed up or marked for reuse. This algorithm is based on the assumption that recently accessed data may not be required for a while.

As cache fills, storage system must take action to flush dirty pages (data written into the cache but not yet written to the disk) to manage availability. Flushing is the process of committing data from cache to the disk. On the basis of the I/O access rate and pattern, high and low levels called **watermarks** are set in cache to manage the flushing process.

- **High watermark (HWM):** is cache utilization level at which the storage system starts highspeed flushing of cache data.
- **Low watermark (LWM):** is the point at which the storage system stops the high-speed or forced flushing and returns to idle flush behavior.

The cache utilization level, as shown in Figure 4-5, drives the mode of flushing to be used:

1. **Idle flushing:** Occurs continuously, at a modest rate, when the cache utilization level is between the high and low watermark.
2. **High watermark flushing:** Activated when cache utilization hits the high watermark. The storage system dedicates some additional resources to flushing. This type of flushing has minimal impact on host I/O processing.
3. **Forced flushing:** Occurs in the event of a large I/O burst when cache reaches 100 percent of its capacity, which significantly affects the I/O response time. In forced flushing, dirty pages are forcibly flushed to disk.

**Figure 4-5:** Types of flushing

**Cache Data Protection:** Cache is volatile memory, so a power failure or any kind of cache failure will cause the loss of data not yet committed to the disk. This risk of losing uncommitted data held in cache can be mitigated using **cache mirroring** and **cache vaulting**:

1. **Cache mirroring:** Each write to cache is held in two different memory locations on two independent memory cards. In the event of a cache failure, the write data will still be safe in the mirrored location and can be committed to the disk. Reads are staged from the disk to the cache; therefore, in the event of a cache failure, the data can still be accessed from the disk. As only writes are mirrored, this method results in better utilization of the available cache. In cache mirroring approaches, the problem of maintaining **cache coherency** is introduced.
Cache coherency means data in two different cache locations must be identical at all times. It is the responsibility of the array operating environment to ensure coherency.

2. **Cache vaulting:** In this case, Storage vendors use a set of physical disks to dump the contents of cache during power failure. This is called cache vaulting and the disks are called vault drives. When power is restored, data from these disks is written back to write cache and then written to the intended disks. Cache is exposed to the risk of uncommitted data loss due to power failure. This problem can be addressed in various ways: powering the memory with a battery until AC power is restored or using battery power to write the cache content to the disk. In the event of extended power failure, using batteries is not a viable option because in intelligent storage systems, large amounts of data may need to be committed to numerous disks.

4.1.3 **Back End**

The *back end* provides an interface between cache and the physical disks. It consists of two components: **back-end ports** and **back-end controllers**. The back end controls data transfers between cache and the physical disks. From cache, data is sent to the back end and then routed to the destination disk. Physical disks are connected to ports on the back end. The back end controller communicates with the disks when performing reads and writes and also provides additional, but limited, temporary data storage. The algorithms implemented on back-end controllers provide error detection and correction, along with RAID functionality.

4.1.4 **Physical Disk**

A physical disk stores data persistently. Disks are connected to the back-end with either SCSI or a Fibre Channel interface (discussed in subsequent chapters). An intelligent storage system enables the use of a mixture of SCSI or Fibre Channel drives and IDE/ATA drives.

**Logical Unit Number:** Physical drives or groups of RAID protected drives can be logically split into volumes known as logical volumes, commonly referred to as **Logical Unit Numbers (LUNs)**. The use of LUNs improves disk utilization.

For example, without the use of LUNs, a host requiring only 200 GB could be allocated an entire 1TB physical disk. Using LUNs, only the required 200 GB would be allocated to the host, allowing the remaining 800 GB to be allocated to other hosts.

In the case of RAID protected drives, these logical units are slices of RAID sets and are spread across all the physical disks belonging to that set. The logical units can also be seen as a logical partition of a RAID set that is presented to a host as a physical disk.

For example, Figure 4-6 shows a RAID set consisting of five disks that have been sliced, or partitioned, into several LUNs. LUNs 0 and 1 are shown in the figure.
LUNs 0 and 1 are presented to hosts 1 and 2, respectively, as physical volumes for storing and retrieving data. Usable capacity of the physical volumes is determined by the RAID type of the RAID set. The capacity of a LUN can be expanded by aggregating other LUNs with it. The result of this aggregation is a larger capacity LUN, known as a meta-LUN.

**LUN Masking:** LUN masking is a process that provides data access control by defining which LUNs a host can access. LUN masking function is typically implemented at the front end controller. This ensures that volume access by servers is controlled appropriately, preventing unauthorized or accidental use in a distributed environment.

For example, consider a storage array with two LUNs that store data of the sales and finance departments. Without LUN masking, both departments can easily see and modify each other’s data, posing a high risk to data integrity and security. With LUN masking, LUNs are accessible only to the designated hosts.

### 4.2 Intelligent Storage Array

Intelligent storage systems generally fall into one of the following two categories:

1. **High-end storage systems**
2. **Midrange storage systems**

#### 4.2.1 High-end Storage Systems

High-end storage systems, referred to as *active-active arrays*, are aimed at large enterprises for centralizing corporate data. These arrays are designed with a large number of controllers and cache memory. An active-active array implies that the host can perform I/Os to its LUNs across any of the available paths (see Figure 4-7).

To address the enterprise storage needs, these arrays provide the following capabilities:

1. Large storage capacity
2. Large amounts of cache to service host I/Os optimally
3. Fault tolerance architecture to improve data availability
4. Connectivity to mainframe computers and open systems hosts
5. Availability of multiple front-end ports and interface protocols to serve a large number of hosts
6. Availability of multiple back-end Fibre Channel or SCSI RAID controllers to manage disk processing
7. Scalability to support increased connectivity, performance, and storage capacity requirements
8. Ability to handle large amounts of concurrent I/Os from a number of servers and applications
9. Support for array-based local and remote replication

4.2.2 Midrange Storage System
Midrange storage systems are also referred to as *active-passive arrays* and they are best suited for small- and medium-sized enterprises. In an active-passive array, a host can perform I/Os to a LUN only through the paths to the owning controller of that LUN. These paths are called *active paths*. The other paths are passive with respect to this LUN.

As shown in Figure 4-8, the host can perform reads or writes to the LUN only through the path to controller A, as controller A is the owner of that LUN. The path to controller B remains passive and no I/O activity is performed through this path.

Midrange arrays are designed to meet the requirements of small and medium enterprises; therefore, they host less storage capacity and global cache than active-active arrays. There are also fewer front-end ports for connection to servers. However, they ensure high redundancy and high performance for applications with predictable workloads. They also support array-based local and remote replication.
Data Protection, Intelligent Storage system

1. What is RAID? Explain Implementation of RAID. (5m)
2. Explain the three major techniques used in RAID configuration. (9m)
3. How RAID 1+0 and RAID 0+1 are different? Explain why RAID 0 not an option for data protection and high availability? Justify. (8m)
4. Explain RAID 4 and RAID 5. (5m)
5. An application generates 7650 IOPS with 50% being READ operation with disk handling capacity of 180 IOPS. Determine the disk load and number of disks required in RAID 5 configuration. (Given write penalty of RAID 5 is 4). (5m)
6. Write a note on Hot Spares. (5m)
7. Explain with respect to cache in ISS:
   a. Read operations     b. Write operations. (8m)
8. Explain with neat diagram, organization of Physical disk using LUNs in ISS. (6m)
9. Explain with diagram, categories of ISA. What are the capabilities of High-end storage systems? (8m)