Contents

RAID

RAID is a technology that is used to increase the performance and/or reliability of data storage.

The rate of improvement in secondary storage performance has been considerably less than the rate for processors and main memory. This mismatch has made the disk storage system perhaps the focus of concern in improving overall computer system performance.

In the case of disk storage, arrays of disks that operate independently and in parallel were developed to improve performance. Disk I/O performance may be increased by spreading the operation over multiple read/write heads, or multiple disks. With multiple disks separate I/O requests can be handled in parallel as long as the data required is on separate disks.

Industry agreed on a standardised ***scheme for multiple-disk database design***, known as RAID (redundant array of independent disks).

RAID storage uses multiple disks in order to provide fault tolerance, to improve overall performance, and to increase storage **capacity** in a system. A RAID system consists of two or more drives working in parallel. These disks can be hard discs, but there is a trend to also use the technology for SSD (solid state drives).

The RAID scheme consists of seven (universally agreed upon) levels and these levels do not imply a hierarchical relationship but designate different design architectures that share the same three common characteristics. Data is distributed across the drives in one of several ways, referred to as 7 levels of RAID levels (RAID 0 to RAID 6), depending on the required level of redundancy and performance.

There are other levels i.e. RAID 10 which is a combination of RAID 1 and RAID 0 (easier to remember if you think of it as RAID 1+0, written as RAID 10), but these are not standardised by an industry group or standardisation committee. This explains why companies sometimes come up with their own unique numbers and implementations.

The software to perform the RAID-functionality and control the drives can either be located on a separate controller card (a hardware RAID controller) or it can simply be a driver.

- RAID is a set of physical disk drives viewed by the OS as a single logical drive
- Data are distributed across the physical drives of an array (striping)

- Redundant disk capacity is used to store parity information which provides recoverability from disk failure

[Many RAID levels employ an error protection scheme called "parity", a widely used method in information technology to provide fault tolerance in a given set of data.

A ***parity drive*** is a hard drive used in a RAID array to provide fault tolerance. For example, RAID 3 uses a parity drive to create a system that is both fault tolerant and, because of data striping, fast. Basically, a single data bit is added to the end of a data block to ensure the number of bits in the message is either odd or even.

One way to implement a parity drive in a RAID array is to use the exclusive or, or XOR, function - The XOR of all of the data drives in the RAID array is written to the parity drive. If one of the data drives fails, the XOR of the remaining drives is identical to the data of the lost drive. Therefore, when a drive is lost, recovering the drive is as simple as copying the XOR of the remaining drives to a fresh data drive.]

The unique contribution of the RAID proposal is to address effectively the need for redundancy. RAID makes use of stored parity information that enables the recovery of data lost due to a disk failure. Very briefly the levels of RAID are defined here:

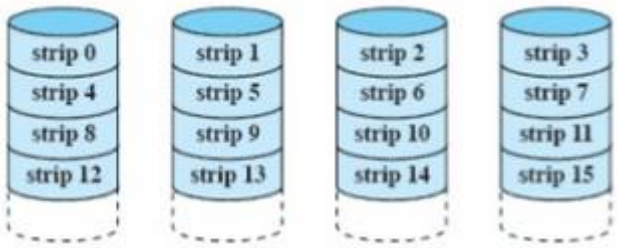
RAID 0 - Stripped

Not a true RAID – no redundancy; Disk failure is catastrophic; Very fast due to parallel read/write

In a RAID 0 system data are split up into blocks that get written across all the drives in the array. By using multiple disks (at least 2) at the same time, this offers superior I/O performance. This performance can be enhanced further by using multiple controllers, ideally one controller per disk.

- RAID 0 offers great performance, both in read and write operations. There is no overhead caused by parity controls.
- All storage capacity is used, there is no overhead.
- The technology is easy to implement.

RAID 0 is ideal for non-critical storage of data that have to be read/written at a high speed, such as on an image retouching or video editing station i.e. if it's a non-mission-critical systems



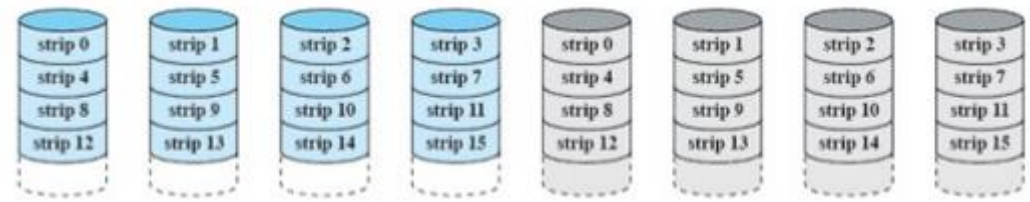
RAID 1 - Mirrored

Redundancy through duplication instead of parity; Read requests can made in parallel; Simple recovery from disk failure

Data are stored twice by writing them to both the data drive (or set of data drives) and a mirror drive (or set of drives). If a drive fails, the controller uses either the data drive or the mirror drive for data recovery and continues operation. You need at least 2 drives for a RAID 1 array.

RAID 1 is a very simple technology but the main disadvantage is that the effective storage capacity is only half of the total drive capacity because all data get written twice.

RAID-1 is ideal for mission critical storage, for instance for accounting systems. It is also suitable for small servers in which only two data drives will be used.



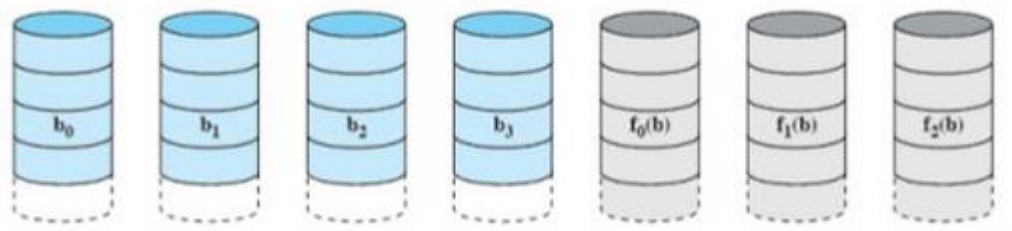
RAID levels 2, 3, 4 and 7 are not that common (RAID 3 is essentially like RAID 5 but with the parity data always written to the same drive) so those four are described more briefly here:

RAID 2 (Using Hamming code)

Synchronised disk rotation

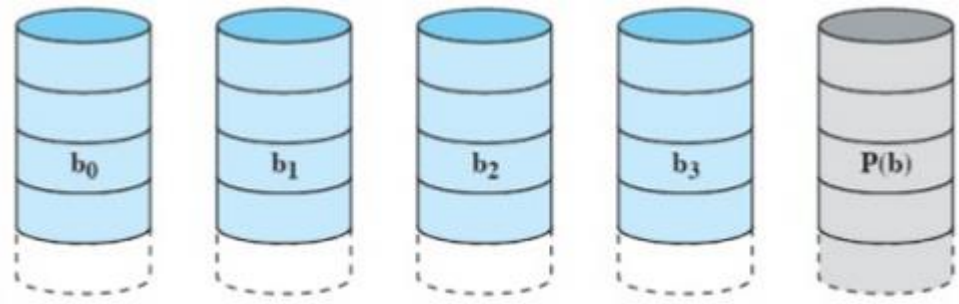
Data stripping is used (extremely small)

Hamming code used to correct single bit errors and detect double-bit errors



RAID 3 bit-interleaved parity

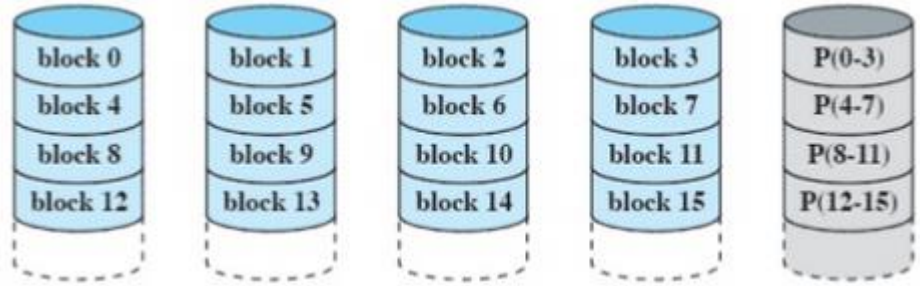
Similar to RAID-2 but uses all parity bits stored on a single drive



RAID 4 Block-level parity

A bit-by-bit parity strip is calculated across corresponding strips on each data disk

The parity bits are stored in the corresponding strip on the parity disk.



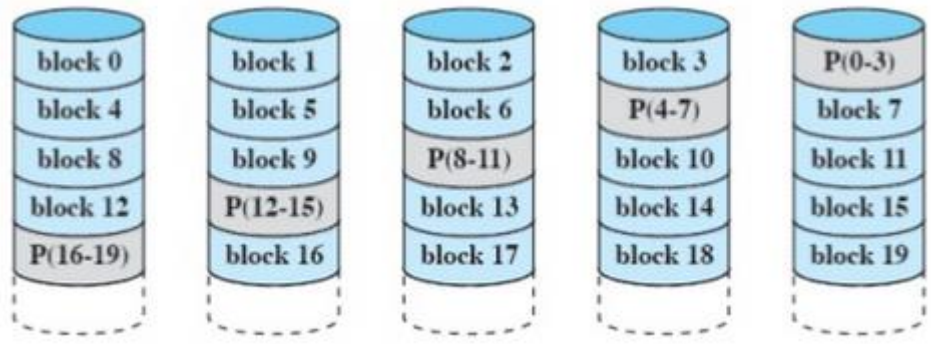
RAID 5 Block-level Distributed parity

RAID 5 is the most common secure RAID level. It requires at least 3 drives but can work with up to 16. Data blocks are striped across the drivesand on one drive a parity checksum of all the block data is written. The parity data are not written to a fixed drive, they are spread across all drives (Similar to RAID-4 but distributing the parity bits across all drives)

Using the parity data, the computer can recalculate the data of one of the other data blocks, should those data no longer be available. That means a RAID 5 array can withstand a single drive failure without losing data or access to data.

If a drive fails, you still have access to all data, even while the failed drive is being replaced and the storage controller rebuilds the data on the new drive. But drive failures have an effect on throughput, although this is still acceptable, and it is a complex technology

RAID 5 is a good all-round system that combines efficient storage with excellent security and decent performance. It is ideal for file and application servers that have a limited number of data drives.



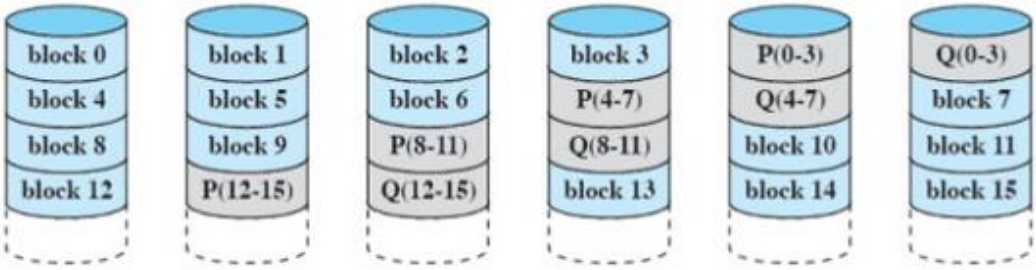
RAID 6 Dual Redundancy (Striping with double parity)

Can recover from two disks failing

RAID 6 is like RAID 5, but the parity data are written to two drives. That means it requires at least 4 drives and can withstand 2 drives dying simultaneously.

> The chances that two drives break down at exactly the same moment are of course very small. However, if a drive in a RAID 5 systems dies and is replaced by a new drive, it takes hours or even more than a day to rebuild the swapped drive. If another drive dies during that time, you still lose all of your data. With RAID 6, the RAID array will even survive that second failure.

If two drives fail, you still have access to all data, even while the failed drives are being replaced. So RAID 6 is more secure than RAID 5 but it has similar disadvantages in throughput and complexity as RAID 5 above.



RAID level 10 – combining RAID 1 & RAID 0

It is possible to combine the advantages (and disadvantages) of RAID 0 and RAID 1 in one single system. This is a nested or hybrid RAID configuration. It provides security by mirroring all data on secondary drives while using striping across each set of drives to speed up data transfers.

Advantages

- If something goes wrong with one of the disks in a RAID 10 configuration, the rebuild time is very fast since all that is needed is copying all the data from the surviving mirror to a new drive. This can take as little as 30 minutes for drives of 1TB.

Disadvantages

- Half of the storage capacity goes to mirroring, so compared to large RAID 5 or RAID 6 arrays, this is an expensive way to have redundancy.

RAID is no substitute for back-up!

All RAID levels except RAID 0 offer protection from a single drive failure. A RAID 6 system even survives 2 disks dying simultaneously. For complete security, you do still need to back-up the data from a RAID system.

- That back-up will come in handy if all drives fail simultaneously because of a power spike.
- It is a safeguard when the storage system gets stolen.
- Back-ups can be kept off-site at a different location. This can come in handy if a natural disaster or fire destroys your workplace.
- The most important reason to back-up multiple generations of data is user error. If someone accidentally deletes some important data and this goes unnoticed for several hours, days or weeks, a good set of back-ups ensure you can still retrieve those files.

Disk Cache

Cache Memory

With exponential advancement in field of faster processors popping up every day, the usage of this terminology has increased rapidly. This has also been the most major parameter in faster processing.

Cache memory (***cache***, or ***CPU memory***) is a type of fast, relatively small memory that is stored on computer hardware. It is classed as random access memory which computer microprocessors can access more quickly than regular RAM.

It is typically directly integrated with the CPU chip, or is placed on a separate chip that can connect to the CPU via a separate bus interconnect.

The purpose of cache memory is to store program instructions that are frequently used by software during its general operations, this is why fast access is needed as it helps to keep the program running quickly. In effect, caching involves keeping a ***copy*** of data in a faster-access location than where the data is normally stored

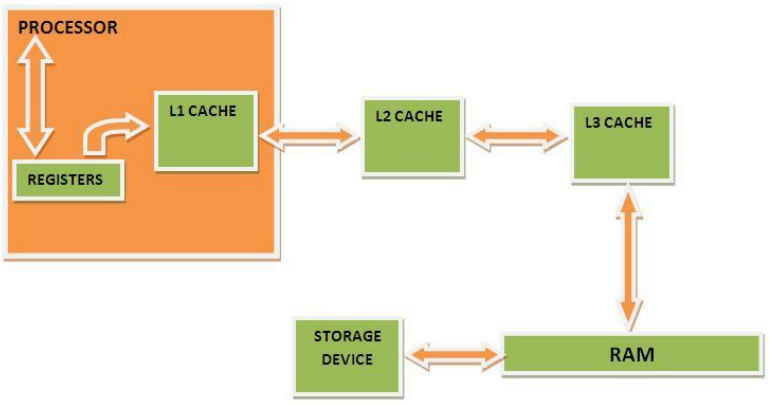


Figure 1: How cache works

Whenever the processor needs to perform an action or execute any command then it first checks the state of the data registers. If the required instruction/data is not present over there, then it looks in the first level of cache memory – L1, and if there also data is not present it further goes to second and further third level of cache memory. Whenever the data needed by processor is not found in the cache it is known as **CACHE MISS** and it leads to delay in the execution thus making the system slow. If the data is found in cache memory it is known as **CACHE HIT.**

> If the data needed is not found in any of the cache memory, the processor checks in RAM. And if this also fails then it goes to look onto the slower storage device.

Often caches can be regarded as fast buffers

- Buffering and caching are very similar, except that a buffer may hold the only copy of a given data item, whereas a cache is just a duplicate copy of some other data stored elsewhere.
- Buffering and caching go hand-in-hand, and often the same storage space may be used for both purposes. For example, after a buffer is written to disk, then the copy in memory can be used as a cached copy, (until that buffer is needed for other purposes)

So, these buffers may be used for communication between disk and memory or memory and CPU. The CPU memory caches may used for instructions or data. In case cache is used for instructions, then a group of instructions may be pre-fetched and kept there. This helps in overcoming the latency experienced in instruction fetch. In the same manner, when it is used for data it helps to attain a higher locality of reference. As for the main memory to disk caches, one use is in disk rewrites. The technique is used almost always to collect all the write requests

for a few seconds before actually a disk is written into. Caching is always used to enhance the performance of systems.

Disk Memory

The same principle can be applied to disk memory – a disk cache is a buffer in main memory for disk sectors. The cache contains a copy of some of the sectors on the disk. When an I/O request is made for a particular sector, a check is made to determine if the sector is in the disk cache

If it is, the request is completed via the cache

If it isn't, the requested sector is read into the disk cache from the disk.

Because of *locality of reference*, when a block of data is fetched into the cache to satisfy a single I/O request, it is likely that there will be future references to that same block.

Design Considerations

When an I/O request is satisfied from the disk cache, the data in the disk cache must be delivered to the requesting process by:

- Transferring the block of data within memory from the disk cache to memory assigned to the user process
  or
- Using a shared memory capability and passing a pointer to the appropriate slot in the disk cache. This would save time of a memory-to-memory transfer and also allowed shared access by other processes

When a new sector is brought into the disk cache, one of the existing blocks have to be replaced, to populate the cache, using some replacement algorithm. Some options to populate the cache include:

1. **Least Recently Used (LRU)**

The block that has been in the cache the longest with no reference to it is replaced

A stack of pointers reference the cache: the most recently referenced block is on the top of the stack. When a block is referenced or brought into the cache, it is placed on the top of the stack (it is not necessary to move these blocks around in memory but a stack of pointers can be associated with the cache)

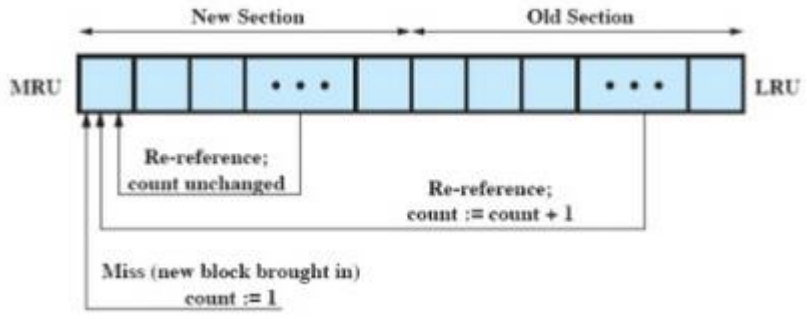
2. **Least Frequently Used (LFU)**

The block that has experienced the fewest references is replaced. This is implemented by associating a counter to each block: the counter is incremented each time block accessed. When replacement is required, the block with the smallest count is selected.

Problems with this is that certain blocks are referenced relatively infrequently overall but when they are referenced, there are short intervals of repeated references due to locality, therefore building up a (misleading) high reference count and therefore cause LFU to make poor choices

3. **Frequency-Based Replacement**

This was developed to overcome to above problem.



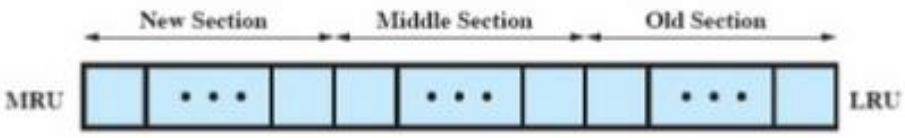
Cache Hit: In the image above, the blocks are logically organised in a stack (same as LRU) and when there's a cache hit, the referenced block is moved to the top of the stack

If the block was already in the new section, then it reference count is not changed, otherwise it's incremented by 1 so blocks that are repeatedly re-referenced are unchanged.

Cache miss: a new block is brought to the new section with a count of 1 (the block with the smallest reference count that is not in the new section). This count remains at 1 as long as the block remains in the new section. Eventually the block ages out of the new section (count still as 1) and will get replaced if not re-referenced, even if they were relatively frequently referenced.

4. LRU Disk Cache Performance

This is a refinement on above: the stack is divided into three sections



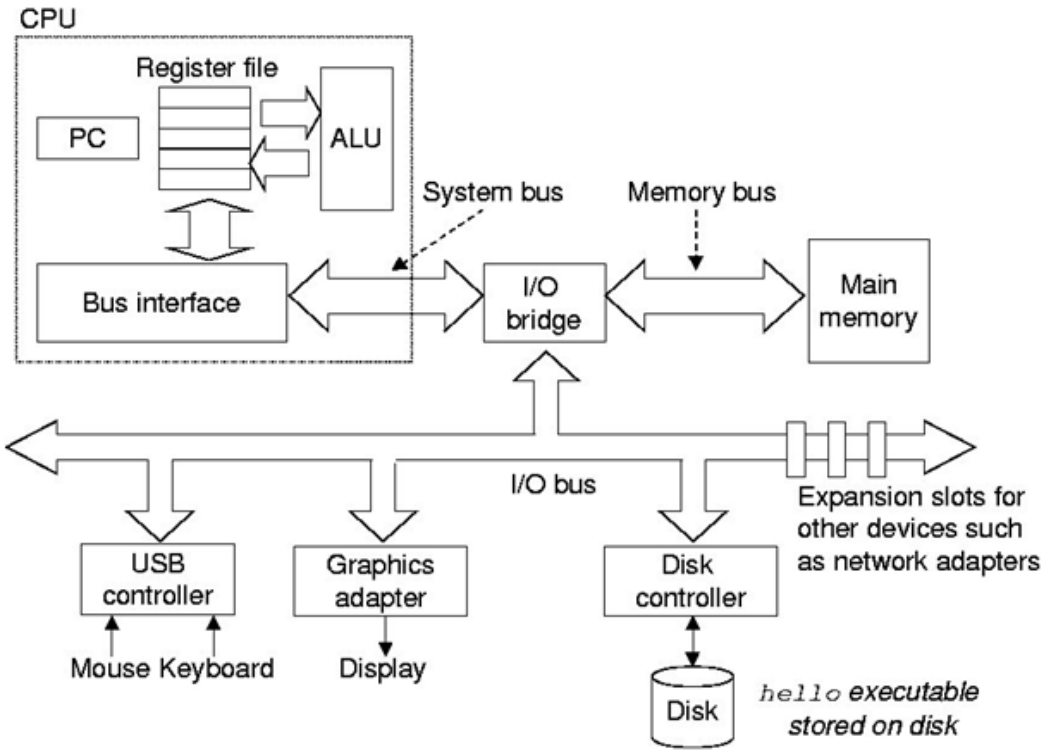
As before, reference counts are not incremented on blocks in the new section but only blocks in the old section are eligible for replacement.

Assuming a sufficiently large middle section, this allows relatively frequently referenced blocks a chance to build up their reference counts before becoming eligible for replacements, which can be significantly better than LFU.

A number of slots are released at a time and this is related to the need to write back sectors.

If a sector is brought into cache and only read, then when it's replaced it's not necessary to write it back to before replacing it – ordering the writing will minimise seek time.

In Summary



Goals for I/O

• Users should access all devices in a uniform manner.

• Devices should be named in a uniform manner.

• The OS, without the intervention of the user program, should handle recoverable errors.

• The OS must maintain security of the devices.

• The OS should optimize the performance of the I/O system.

The CPU and its supporting circuitry provide memory-mapped I/O that is used in low-level computer programming in the implementation of device drivers. An I/O algorithm is one designed to exploit locality and perform efficiently when data reside on secondary storage, such as a disk drive.