Click here to print this article.

Re-Printed From SLCentral

RAID: An In-Depth Guide To RAID Technology
Author: Tom Solinap
Date Posted: January 24th, 2001
URL: http://www.slcentral.com/articles/01/1/raid

Introduction

In the last 20 years, computers have revolutionized the way the world works. Whether you're talking about business, education, or leisure, computers have just about taken over every facet of life as we know it. We all know that pretty much every aspect of the computer system has evolved to meet the growing demands of the people. This is certainly true when dealing with data storage devices. Information has become a commodity in today's world, and protecting that information from being lost is mission critical. The internet has helped push this information age forward. Popular websites process so much information, that any type of slowdown or downtime can mean the loss of millions of dollars. Clearly, just a bunch of hard disks won't be able to cut it anymore. So, something called Redundant Array of Independent (or Inexpensive) Disks (RAID) was developed to increase the performance and reliability of data storage by spreading data across multiple drives. The term was first coined by researchers at UC-Berkeley. RAID technology has grown and evolved throughout the years to meet these ever growing demands for speed and data security. What I'm going to try to do in this article is to get into the guts of what RAID is all about. The average computer user is probably only familiar with IDE or software RAID solutions. I'm going to deal with all flavors of RAID and the underlying concepts behind the technology.

What is RAID?

So what exactly is RAID? Nope, it's not the bug spray I'm talking about here. It is a technique that was developed to provide speed, reliability, and increased storage capacity using multiple disks, rather than single disk solutions. RAID basically takes multiple hard drives and allows them to be used as one large hard drive with benefits depending on the scheme or level of RAID being used. Depending on your needs, there are many different RAID variations and implementations available with prices ranging from less than $100 to over $25,000. Of course, the better the RAID implementation, the more expensive it's probably going to be. There is really no one best RAID implementation. Some implementations are better than others depending on the actual application.

It used to be that RAID was only available in expensive server systems. However, with the advent of inexpensive RAID controllers, it seems it has pretty much reached the mainstream market. Performance nuts, such as myself, are always looking for the latest technology to give us that edge we need. Of course, the mainstream implementation has it's limitations, and RAID isn't really for everyone. There are many levels of RAID being used today. Before I go into the different levels, I'd like to discuss the basic concepts behind these levels in a little more detail.

The Array and RAID Controller Concept

First let's make sure we know exactly what we're dealing with here. A drive array is a collection of hard disk drives that are grouped together. When we talk about RAID, there is often a distinction between physical drives and arrays and logical drives and arrays. Physical arrays can be divided or grouped together to form one or more logical arrays. These logical arrays can be divided into logical drives that the operating system sees. The logical drives are treated like single hard drives and can be partitioned and formatted accordingly. I know it's confusing but often times most RAID setups are relatively simple. However, they can get complicated when dealing with multiple nested RAID levels in the high-end RAID implementations.

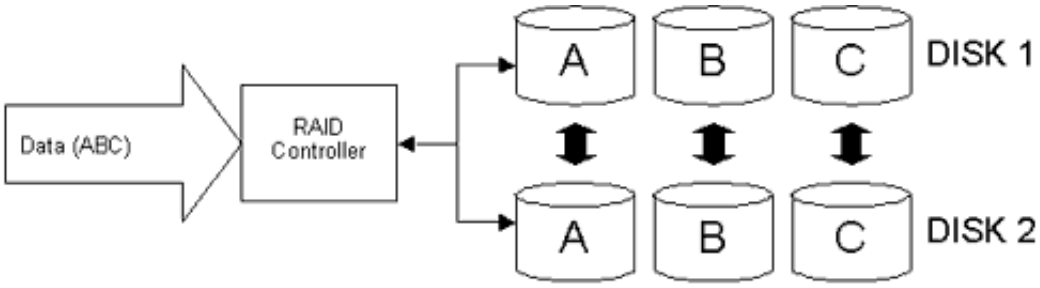
The RAID controller is what manages how the data is stored and accessed across the physical and logical arrays. It ensures that the operating system only sees the logical drives and does not need to worry about managing the underlying schema. As far as the system is concerned, it's dealing with regular hard drives. A RAID controller's functions can be implemented in hardware or software. Hardware implementations are better for RAID levels that require large amounts of calculations. With today's incredibly fast processors, software RAID implementations are more feasible, but the CPU can still get bogged down with large amounts of I/O. Later in the article, I'll discuss which applications and RAID levels are more suited to hardware or software RAID.

Mirroring

Mirroring involves having two copies of the same data on separate hard drives or drive arrays. So basically the data is effectively mirrored on another drive. The system basically writes data simultaneously to both hard drives. This is one of the two data redundancy techniques used in RAID to protect from data loss. The benefit is that when one hard drive or array fails, the system can still continue to operate since there are two copies of data. Downtime is minimal and data recovery is relatively simple. All you need to do is rebuild the data from the good copy.

The following diagram illustrates how mirroring actually works. Of course, this is a simplified diagram. A, B, and C are separate files that reside on each disk. Disk 1 and 2 in this diagram don't necessarily have to be disks themselves. They can be arrays of drives. The main thing to remember here is that the RAID controller writes the same data blocks to each mirrored drive. Each drive or array has the same information in it. You can even add another level of complexity by introducing striping, which will be discussed shortly. I'm not going to get into what striping is now if you're not familiar with it, but just know it increases performance. If you have one striped array, you can mirror the array at the same time on a second striped array. It can get very complicated. To set up mirroring the number of drives will have to be in the power of 2 for obvious reasons.

The drawback here is that both drives are tied up during the writing process which limits parallelism and can hurt performance. In contrast, mirroring has a performance increase when dealing with reads. A good RAID controller will only read from one of the drives since the data on both are the same. While the other is used to read, the free drive can be used for other requests. This increases parallelism, which is pretty much the concept behind the performance increase of RAID.



Mirroring may seem like a good solution to ensure the safety of data. However, the trade off here is the cost and wasted space involved with having two copies of the same data. You will need twice as much storage space to mirror the data. It can be cost effective for some applications where downtime can mean the downfall of a company or even loss of human life. Most of the time, however, it might not be worth it. Data might not be that critical to warrant spending twice as much on storage. The alternative to mirroring is parity, which is what the next section deals with.

Parity

Parity is the other data redundancy technique used in RAID. The term parity is often used when dealing with error detection in communication, such as modems. It is also used in memory. Parity in RAID uses a similar concept to parity in memory. For example, you have X number of data elements. You would then use those X data elements to create a parity element, and end up with X+1 total data elements. If any ONE of those X+1 elements is lost, it can be recovered as long as X number of elements remain. You might have heard the term "parity bit" being used to refer to the extra information. In the case of RAID, the parity data is much larger than just one bit. Confusing you yet?

Well, the extra parity data is created typically using a logical operation called exclusive OR (XOR). If you're not familiar with the XOR operation, here's a truth table to refresh your memory:

p	q	p XOR q
T	T	F
T	F	T
F	T	T
F	F	F

Basically, the XOR operation is true if and only if one of its operands is true, unless both operands are true then the output is false.Obviously, you can carry this operation over to binary by letting true equal

1 and false equal 0.

For example:

10101010 XOR 11111111 = 01010101

11111111 XOR 01010101 = 10101010

10101010 XOR 01010101 = 11111111

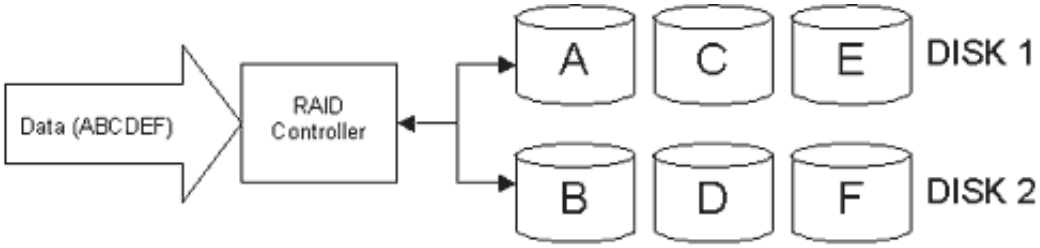
Basically, you can get any of the three values as long as you have the other two, which is what parity is all about in the first place. The operation works on any number of bits, so you can do the operation on whole hard drives. Now if you think about it, you can protect from any single hard drive being lost without having to keep two copies of data. You only need an extra hard drive. This is one advantage parity has over mirroring. However, the level of fault tolerance is not as high as in mirroring. The parity data doesn't necessarily have to be stored on a single hard drive. It can be distributed across the entire array, which is often referred to as distributed parity. As is the case with mirroring, striping with parity can also be implemented. The main limitation here is that parity algorithms usually need larges amounts of computing power. The parity data has to be computed every time a read/write takes place. This means that a hardware RAID controller is required. Software RAID would not be practical in this case because the operations would tie up the CPU. In addition, recovering from a failure is more complicated than in mirroring. Automatic rebuilding is often reserved for hardware controllers, and even then it takes longer than mirroring.

Striping

So far, the concepts I have discussed deal with improving the reliability of the data. I have mentioned striping in the earlier sections but have not explained thoroughly what it is. Striping improves the performance of the array by distributing the data across all the drives. The main principle behind striping is parallelism. Imagine you have a large file on a single hard drive. If you want to read the file, you have to wait for the hard drive to read the file from beginning to end. Now, if you break the file up into multiple pieces and distribute it across multiple hard drives, you have all these drives reading a part of the file at the same time. You only have to wait as long as it takes to read each piece since the drives are working in parallel. The same is true if you were writing a large file to a disk. Transfer performance is greatly increased. The more hard drives you have, the greater the increase in performance. The number of drives is also the same as the stripe width, that is the number of simultaneous stripes that can be transferred simultaneously. How does this actually work though?

Every piece of data that comes into the RAID controller is divided into smaller pieces. There two levels of striping that use different techniques to divide the data, byte level and block level striping. Byte level striping involves breaking up the data into bytes and storing them sequentially across the hard drives. For example, if the data is broken into 16 bytes and there are 4 hard drives, the first byte is stored in the first hard drive, the second byte in the second drive, and so on. The fifth byte is stored in the first hard drive and the cycle continues. Sometimes byte level striping is done using 512 bytes at a time. Block level striping involves breaking up the data into a given size block. These blocks are then distributed the same way across the array as in byte level striping. The size of these blocks is called the stripe size. A variety of stripe sizes are usually available depending on the RAID implementation used.

The stripe size is a largely debated topic. There is no ideal stripe size but certain sizes work best with certain applications. The performance effects of increasing or decreasing stripe size are apparent. Using a small stripe size will enable files to be broken up more and distributed across the drives. The transfer performance will increase due to the increased parallelism. However, this also increases the randomness of the position of each piece of the file. As you probably guessed already, using a large stripe size will do the opposite of decreasing the size. The data will be less distributed and transfer performance is decreased. The randomness is decreased as well. The best way to find out the right stripe size for your particular application is to experiment. Start out with a medium stripe size and try decreasing or increasing the size and recording the difference in over-all performance.



The above diagram is another simplified model of how striping works. The data file that comes in is broken up into 6 blocks (A,B,C,D,E,F) and distributed across those two drives. If you had more hard drives, each block would have been distributed to those as well. Now if you want to move or transfer the file somewhere, the controller accesses both drives simultaneously, which is where the performance gain kicks in. It only takes half the time to transfer the file. If you increase the number of hard drives, the file will be transferred in 1/Nth the time it takes to transfer from 1 hard drive (where N is the number of drives). The next section will involve the various levels of RAID, so you will see how these concepts fit into all this.

Levels of RAID

In the previous sections I dealt with the basic concepts behind RAID. All those concepts are used to make up the various RAID levels out there now. I'm covering the standard RAID levels here. Some companies have developed their own proprietary RAID levels which are not covered here. Also, others might have different interpretations of the levels of RAID. The explanation I present here are what I consider to be these levels. These merely the single RAID levels. These levels can be combined in many different ways to provide more functionality. I'll get into multiple or nested RAID levels in the next section.

RAID 0
This is the simplest level of RAID, and it just involves striping. Data redundancy is not even present in this level, so it is not recommended for applications where data is critical. This level offers the highest level of performance out of any single RAID level. It also offers the lowest cost since no extra storage is involved. At least 2 hard drives are required, preferably identical, and the maximum depends on the RAID controller. None of the space is wasted as long as the hard drives used are identical. This level has become popular with the mainstream market for it's relatively low cost and high performance gain. This level is good for most people that don't need any data redundancy. There are many SCSI and IDE/ATA implementations available. Finally, it's important to note that if any of the hard drives in the array fails, you lose everything.

RAID 1
This level is usually implemented as mirroring. Two identical copies of data are stored on two drives. When one drive fails, the other drive still has the data to keep the system going. Rebuilding a lost drive is very simple since you still have the second copy. This adds data redundancy to the system and provides some safety from failures. Some implementations add an extra RAID controller to increase the fault tolerance even more. It is ideal for applications that use critical data. Even though the performance benefits are not great, some might just be concerned with preserving their data. The relative simplicity and low cost of implementing this level has increased its popularity in mainstream RAID controllers. Most RAID controllers nowadays implement some form of RAID 1.

RAID 2
This level uses bit level striping with Hamming code ECC. The technique used here is somewhat similar to striping with parity but not really. The data is split at the bit level and spread over a number of data and ECC disks. When data is written to the array, the Hamming codes are calculated and written to the ECC disks. When the data is read from the array, Hamming codes are used to check whether errors have occurred since the data was written to the array. Single bit errors can be detected and corrected immediately. This is the only level that really deviates from the RAID concepts talked about earlier. The complicated and expensive RAID controller hardware needed and the minimum number of hard drives required, is the reason this level is not used today.

RAID 3
This level uses byte level striping with dedicated parity. In other words, data is striped across the array at the byte level with one dedicated parity drive holding the redundancy information. The idea behind this level is that striping the data increasing performance and using dedicated parity takes care of redundancy. 3 hard drives are required. 2 for striping, and 1 as the dedicated parity drive. Although the performance is good, the added parity does slow down writes. The parity information has to be written to the parity drive whenever a write occurs. This increased computation calls for a hardware controller, so software implementations are not practical. RAID 3 is good for applications that deal with large files since the stripe size is small.

RAID 4
This level is very similar to RAID 3. The only difference is that it uses block level striping instead of byte level striping. The advantage in that is that you can change the stripe size to suit application needs. This level is often seen as a mix between RAID 3 and RAID 5, having the dedicated parity of RAID 3 and the block level striping of RAID 5. Again, you'll probably need a hardware RAID controller for this level. Also, the dedicated parity drive continues to slow down performance in this level as well.

RAID 5
RAID 5 uses block level striping and distributed parity. This level tries to remove the bottleneck of the dedicated parity drive. With the use of a distributed parity algorithm, this level writes the data and parity data across all the drives. Basically, the blocks of data are used to create the parity blocks which are then stored across the array. This removes the bottleneck of writing to just one parity drive. However, the parity information still has to be calculated and written whenever a write occurs, so the slowdown involved with that still applies. The fault tolerance is maintained by separating the parity information for a block from the actual data block. This way when one drive goes, all the data on that drive can be rebuilt from the data on the other drives. Recovery is more complicated than usual because of the distributed nature of the parity. Just as in RAID 4, the stripe size can be changed to suit the needs of the application. Also, using a hardware controller is probably the more practical solution. RAID 5 is one of the most popular RAID levels being used today. Many see it as the best combination of performance, redundancy, and storage efficiency.

Combining Levels of RAID

The single RAID levels don't address every application requirement that exist. So, to get more functionality, someone thought of the idea of combining RAID levels. What if you can combine two levels and get the advantages of both? Well that was the motivation behind creating these new levels. The main benefit of using multiple RAID levels is the increased performance. Usually combining RAID levels means using a hardware RAID controller. The increased level of complexity of these levels means that software solutions are not practical. RAID 0 has the best performance out of the single levels and it is the one most commonly being combined. Not all combinations of RAID levels exist. The most common combinations are RAID 0+1 and 1+0. The difference between 0+1 and 1+0 might seem subtle, and sometimes companies may use the terms interchangeably. However, the difference lies in the amount of fault tolerance. Both these levels require at least 4 hard drives to implement. Let's look at RAID 0+1 first.

This combination uses RAID 0 for it's high performance and RAID 1 for it's high fault tolerance. I actually mentioned this level when I talked about adding striping to mirroring. Let's say you have 8 hard drives. You can split them into 2 arrays of 4 drives each, and apply RAID 0 to each array. Now you have 2 striped arrays. Then you would apply RAID 1 to the 2 striped arrays and have one array mirrored on the other. If a hard drive in one striped array fails, the entire array is lost. The other striped array is left, but contains no fault tolerance if any of the drives in it fail.

RAID 1+0 applies RAID 1 first then RAID 0 to the drives. To apply RAID 1, you split the 8 drives into 4 sets of 2 drives each. Now each set is mirrored and has duplicate information. To apply RAID 0, you then stripe across the 4 sets. In essence, you have a striped array across a number of mirrored sets. This combination has better fault tolerance than RAID 0+1. As long as one drive in a mirrored set is active, the array can still function. So theoretically you can have up to half the drives fail before you lose everything, as opposed to only two drives in RAID 0+1.

The popularity of RAID 0+1 and 1+0 stems from the fact that it's relatively simple to implement while providing high performance and good data redundancy. With the increased reduction of hard drive prices, the 4 hard drive minimum isn't unreasonable to the mainstream anymore. However, you still have the 50% waste in storage space whenever you are dealing with mirroring. Enterprise applications and servers are often willing to sacrifice storage for increased performance and fault tolerance. Some other combinations of RAID levels that are used include, RAID 0+3, 3+0, 0+5, 5+0, 1+5, and 5+1. These levels are often complicated to implement and require expensive hardware. Not all of the combinations I mentioned above are used

Benefits

Ok now that you know the different RAID levels and configurations, why would you even bother? Well it really all depends on your application and the RAID level you use. However, in general using RAID provides data redundancy, fault tolerance, increased capacity, and increased performance. Data redundancy protects the data from hard drive failures. This benefit is good for companies or individuals that have critical or important data to protect, or just anyone that's paranoid about losing their gigabytes of MP3s or pr0n. Fault tolerance goes hand in hand with redundancy in providing a better over-all storage system. The only RAID level that does not have any form of redundancy or fault tolerance is RAID 0.

RAID also provides increased capacity by combining multiple drives. The efficiency of how the total drive storage is used depends on the RAID level. Usually, levels involving mirroring need twice as much storage to mirror the data. And lastly, the reason most people go to RAID is for the increase in performance. Depending on the RAID level used, the performance increase is different. For applications

that need raw speed, RAID is definitely the way to go.

Hardware RAID Implementations

Let's look at some of the common ways RAID is implemented in hardware. Hardware RAID can use either SCSI or IDE/ATA to interface with the system and hard drives. SCSI is the preferred interface when dealing with the high-end servers. IDE/ATA RAID controllers have recently introduced RAID into the mainstream. There are obvious limitations to using IDE, and for the more complicated and high-end RAID implementations SCSI is used. However, with the increased performance and functionality of SCSI comes a higher price tag.

Hardware implementations of RAID are basically divided into internal and external RAID controllers. The internal RAID controllers are usually controller cards that are installed in the bus system of the computer. Typically it resembles any SCSI or IDE adapter that connects drives to the system. Some motherboards have these RAID controllers built in. Only recently have IDE RAID controllers become some what of a standard for high performance motherboards. Depending on the RAID controller and levels available, a certain amount of cache memory is present. Most of the time, the more memory you have, the better and the more expensive the controller is.

External RAID controllers involve moving the actual controller to a case of its own with the hard drives. In high-end servers, you'll often see a separate enclosure for the RAID controller and hard drives. The controller handles all the RAID functions and presents all the logical drives to the system. An external controller is usually more complex and has more memory than an internal one. This is because of the large number of hard drives and complex RAID levels it needs to work with. The interface used for all this is usually some form of SCSI. This makes it easier to have large numbers of hard drives hot swappable, and also avoids having to touch the actual system. The system might be some super computer that needs to be isolated somewhere, but the drives don't have to be.

Cost wise, internal controllers will be much cheaper than external controllers. You definitely get more flexibility with the external controllers but often times the cost doesn't warrant it. Even though IDE/ATA RAID controllers are becoming more popular with the masses, SCSI and the like is still the interface of choice for high-end machines. The cost of a hardware RAID implementation is still fairly high for your average Joe, so many just go with software RAID. The choice is pretty much just financial. Software RAID uses processor time, so the more complex RAID levels will slow down a system considerably. If you can pay for it, hardware RAID is superior to software RAID.

Conclusion

So what have we learned here? Well we've learned that RAID is not just a bug spray. RAID is a good solution for companies or individuals craving more transfer performance, redundancy, and storage capacity in their data storage systems. There are many levels of RAID, which range from very simple and cheap to extremely complex and expensive. The benefits of having RAID in your system are obvious, however, RAID is not for everyone. Performance freaks like myself will always like what RAID has to offer, but the price tag of the better RAID implementations is still a hurdle to overcome. I tried to cover as much as I possibly could about what RAID is, it's benefits, and the various implementations. I tried not to focus on any specific company or hardware implementation, but looked at RAID in general. I hope you've enjoyed reading this article and maybe even learned something along the way. This isn't the definitive guide to RAID, but I hope it's helped you understand RAID more.

Re-Printed From SLCentral