2

Intelligent Disk Subsystems

Hard disks and tapes are currently the most important media for the storage of data. When storage networks are introduced, the existing small storage devices are replaced by a few large storage systems (storage consolidation). For example, individual hard disks and small disk stacks are replaced by large disk subsystems that can store between a few hundred gigabytes and several ten petabytes of data, depending upon size. Furthermore, they have the advantage that functions such as high availability, high performance, instant copies and remote mirroring are available at a reasonable price even in the field of open systems (Unix, Windows, OS/400, Novell Netware, MacOS). The administration of a few large storage systems is significantly simpler, and thus cheaper, than the administration of many small disk stacks. However, the administrator must plan what he is doing more precisely when working with large disk subsystems. This chapter describes the functions of such modern disk subsystems.

This chapter begins with an overview of the internal structure of a disk subsystem (Section 2.1). We then go on to consider the hard disks used inside the system and the configuration options for the internal I/O channels (Section 2.2). The controller represents the control centre of a disk subsystem. Disk subsystems without controllers are called JBODs (Just a Bunch of Disks); JBODs provide only an enclosure and a common power supply for several hard disks (Section 2.3). So-called RAID (Redundant Array of Independent Disks) controllers bring together several physical hard disks to form virtual hard disks that are faster and more fault-tolerant than individual physical hard disks (Sections 2.4 and 2.5). Some RAID controllers use a cache to further accelerate write and read access to the server (Section 2.6). In addition, intelligent controllers provide services such as instant copy and remote mirroring (Section 2.7). The conclusion to this chapter

Storage Networks Explained: Basics and Application of Fibre Channel SAN, NAS, iSCSI, InfiniBand and FCoE, Second Edition
U. Troppens R. Erkens W. Müller-Friedt N. Haustein R. Wolafka

summarises the measures discussed for increasing the fault-tolerance of intelligent disk subsystems (Section 2.8).

2.1 ARCHITECTURE OF INTELLIGENT DISK SUBSYSTEMS

In contrast to a file server, a disk subsystem can be visualised as a hard disk server. Servers are connected to the connection port of the disk subsystem using standard I/O techniques such as Small Computer System Interface (SCSI), Fibre Channel or Internet SCSI (iSCSI) and can thus use the storage capacity that the disk subsystem provides (Figure 2.1). The internal structure of the disk subsystem is completely hidden from the server, which sees only the hard disks that the disk subsystem provides to the server.

The connection ports are extended to the hard disks of the disk subsystem by means of internal I/O channels (Figure 2.2). In most disk subsystems there is a controller between the connection ports and the hard disks. The controller can significantly increase the data availability and data access performance with the aid of a so-called RAID procedure. Furthermore, some controllers realise the copying services instant copy and remote mirroring and further additional services. The controller uses a cache in an attempt to accelerate read and write accesses to the server.

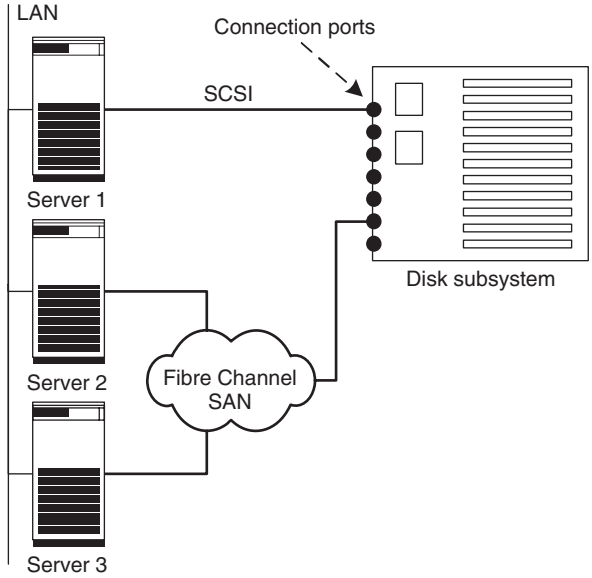

Figure 2.1 Servers are connected to a disk subsystem using standard I/O techniques. The figure shows a server that is connected by SCSI. Two others are connected by Fibre Channel SAN.

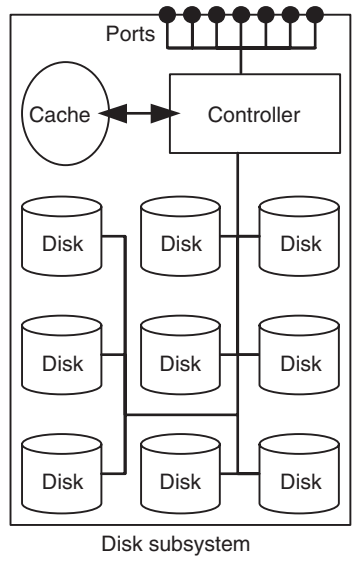

Figure 2.2 Servers are connected to the disk subsystems via the ports. Internally, the disk subsystem consists of hard disks, a controller, a cache and internal I/O channels.

Disk subsystems are available in all sizes. Small disk subsystems have one to two connection ports for servers or storage networks, six to eight hard disks and, depending on the disk capacity, storage capacity of a few terabytes. Large disk subsystems have multiple ten connection ports for servers and storage networks, redundant controllers and multiple I/O channels. A considerably larger number of servers can access a subsystem through a connection over a storage network. Large disk subsystems can store up to a petabyte of data and, depending on the supplier, can weigh well over a tonne. The dimensions of a large disk subsystem are comparable to those of a wardrobe.

Figure 2.2 shows a simplified schematic representation. The architecture of real disk subsystems is more complex and varies greatly. Ultimately, however, it will always include the components shown in Figure 2.2. The simplified representation in Figure 2.2 provides a sufficient basis for the further discussion in the book.

Regardless of storage networks, most disk subsystems have the advantage that free disk space can be flexibly assigned to each server connected to the disk subsystem (storage pooling). Figure 2.3 refers back once again to the example of Figure 1.2. In Figure 1.2 it is not possible to assign more storage to server 2, even though free space is available on servers 1 and 3. In Figure 2.3 this is not a problem. All servers are either directly connected to the disk subsystem or indirectly connected via a storage network. In this configuration each server can be assigned free storage. Incidentally, free storage capacity should be understood to mean both hard disks that have already been installed and have not yet been used and also free slots for hard disks that have yet to be installed.

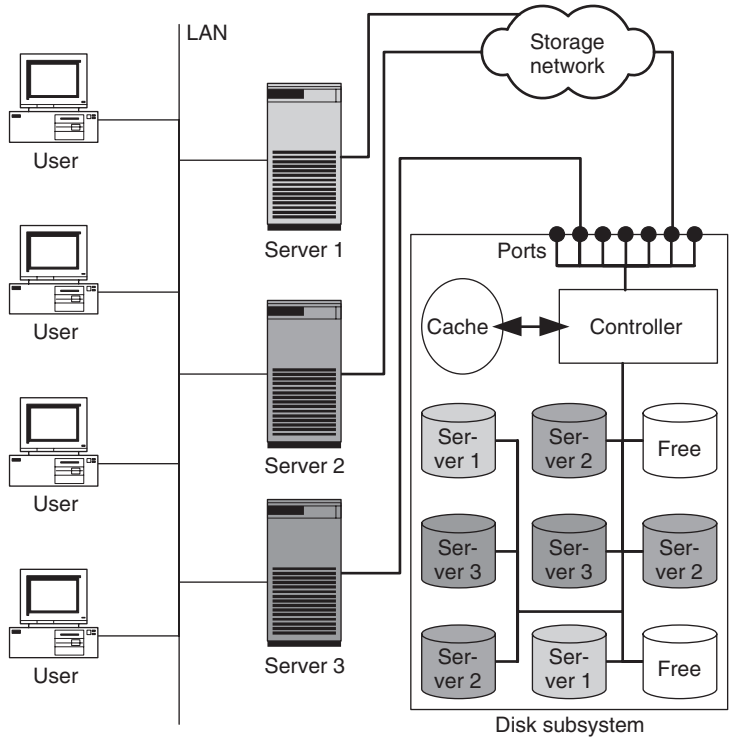

Figure 2.3 All servers share the storage capacity of a disk subsystem. Each server can be assigned free storage more flexibly as required.

2.2 HARD DISKS AND INTERNAL I/O CHANNELS

The controller of the disk subsystem must ultimately store all data on physical hard disks. Standard hard disks that range in size from 36 GB to 1 TB are currently (2009) used for this purpose. Since the maximum number of hard disks that can be used is often limited, the size of the hard disk used gives an indication of the maximum capacity of the overall disk subsystem.

When selecting the size of the internal physical hard disks it is necessary to weigh the requirements of maximum performance against those of the maximum capacity of the overall system. With regard to performance it is often beneficial to use smaller hard disks at the expense of the maximum capacity: given the same capacity, if more hard disks are available in a disk subsystem, the data is distributed over several hard disks and thus the overall load is spread over more arms and read/write heads and usually over more I/O channels (Figure 2.4). For most applications, medium-sized hard disks are sufficient. Only

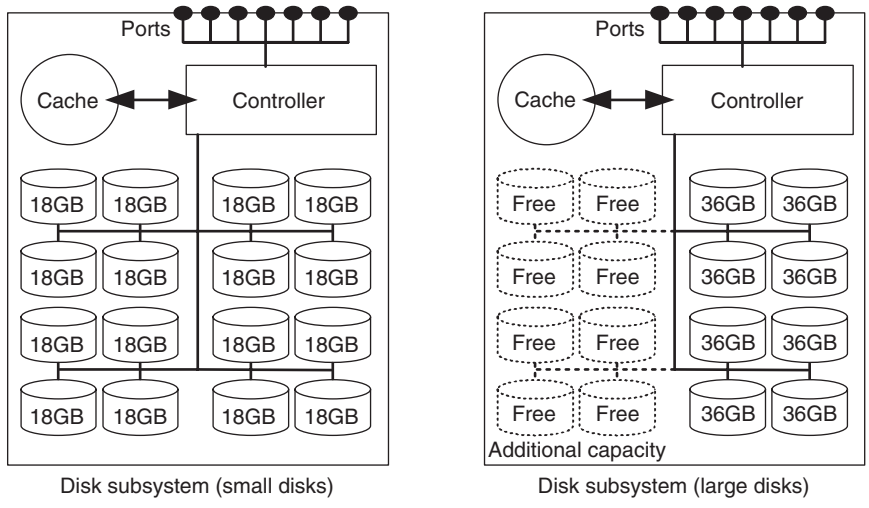

Figure 2.4 If small internal hard disks are used, the load is distributed over more hard disks and thus over more read and write heads. On the other hand, the maximum storage capacity is reduced, since in both disk subsystems only 16 hard disks can be fitted.

for applications with extremely high performance requirements should smaller hard disks be considered. However, consideration should be given to the fact that more modern, larger hard disks generally have shorter seek times and larger caches, so it is necessary to carefully weigh up which hard disks will offer the highest performance for a certain load profile in each individual case.

Standard I/O techniques such as SCSI, Fibre Channel, increasingly Serial ATA (SATA) and Serial Attached SCSI (SAS) and, still to a degree, Serial Storage Architecture (SSA) are being used for internal I/O channels between connection ports and controller as well as between controller and internal hard disks. Sometimes, however, proprietary – i.e., manufacturer-specific – I/O techniques are used. Regardless of the I/O technology used, the I/O channels can be designed with built-in redundancy in order to increase the fault-tolerance of a disk subsystem. The following cases can be differentiated here:

- *Active*
  In active cabling the individual physical hard disks are only connected via one I/O channel (Figure 2.5, left). If this access path fails, then it is no longer possible to access the data.
- *Active/passive*
  In active/passive cabling the individual hard disks are connected via two I/O channels (Figure 2.5, right). In normal operation the controller communicates with the hard disks via the first I/O channel and the second I/O channel is not used. In the event of the

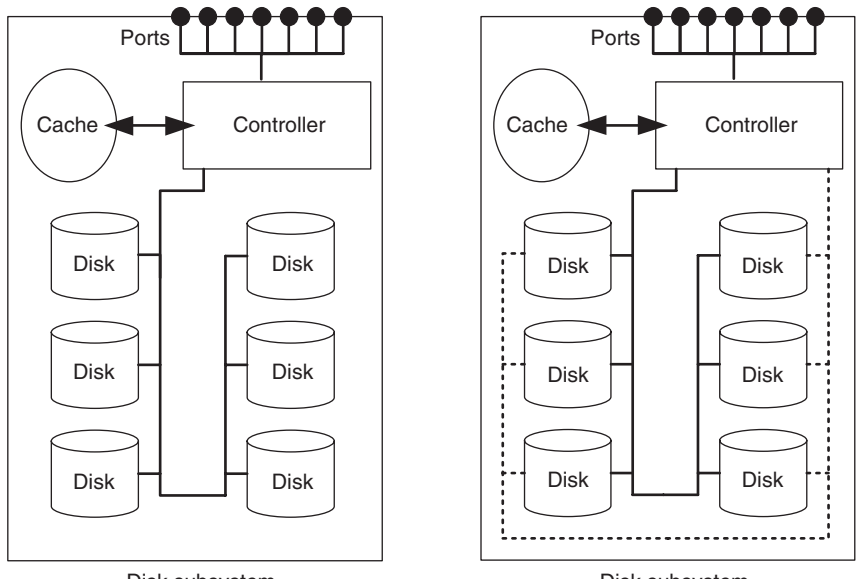

Figure 2.5 In active cabling all hard disks are connected by just one I/O channel. In active/passive cabling all hard disks are additionally connected by a second I/O channel. If the primary I/O channel fails, the disk subsystem switches to the second I/O channel.

failure of the first I/O channel, the disk subsystem switches from the first to the second I/O channel.

- *Active/active (no load sharing)*
  In this cabling method the controller uses both I/O channels in normal operation (Figure 2.6, left). The hard disks are divided into two groups: in normal operation the first group is addressed via the first I/O channel and the second via the second I/O channel. If one I/O channel fails, both groups are addressed via the other I/O channel.
- *Active/active (load sharing)*
  In this approach all hard disks are addressed via both I/O channels in normal operation (Figure 2.6, right). The controller divides the load dynamically between the two I/O channels so that the available hardware can be optimally utilised. If one I/O channel fails, then the communication goes through the other channel only.

Active cabling is the simplest and thus also the cheapest to realise but offers no protection against failure. Active/passive cabling is the minimum needed to protect against failure, whereas active/active cabling with load sharing best utilises the underlying hardware.

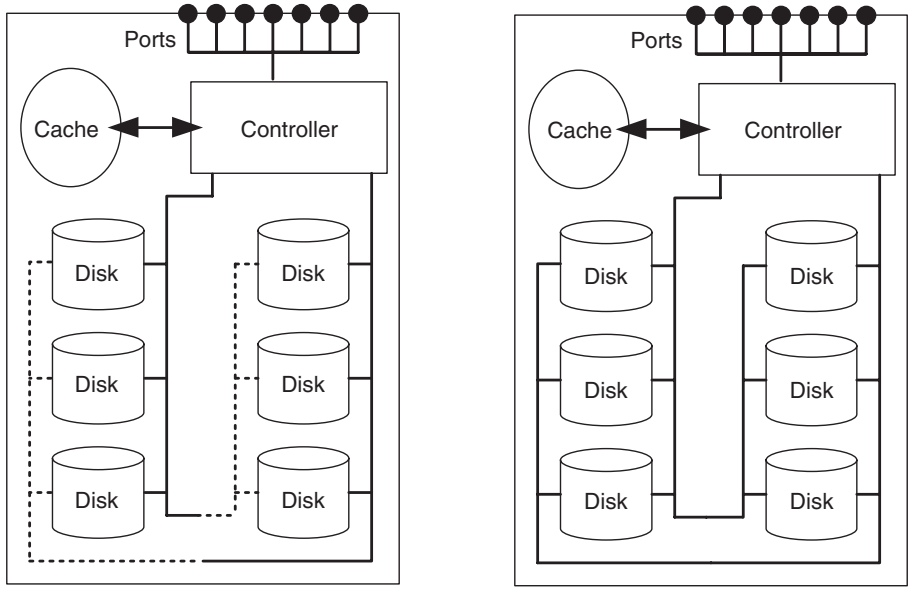

Figure 2.6 Active/active cabling (no load sharing) uses both I/O channels at the same time. However, each disk is addressed via one I/O channel only, switching to the other channel in the event of a fault. In active/active cabling (load sharing) hard disks are addressed via both I/O channels.

2.3 JBOD: JUST A BUNCH OF DISKS

If we compare disk subsystems with regard to their controllers we can differentiate between three levels of complexity: (1) no controller; (2) RAID controller (Sections 2.4 and 2.5); and (3) intelligent controller with additional services such as instant copy and remote mirroring (Section 2.7).

If the disk subsystem has no internal controller, it is only an enclosure full of disks (JBODs). In this instance, the hard disks are permanently fitted into the enclosure and the connections for I/O channels and power supply are taken outwards at a single point. Therefore, a JBOD is simpler to manage than a few loose hard disks. Typical JBOD disk subsystems have space for 8 or 16 hard disks. A connected server recognises all these hard disks as independent disks. Therefore, 16 device addresses are required for a JBOD disk subsystem incorporating 16 hard disks. In some I/O techniques such as SCSI (Section 3.2) and Fibre Channel arbitrated loop (Section 3.3.6), this can lead to a bottleneck at device addresses.

In contrast to intelligent disk subsystems, a JBOD disk subsystem in particular is not capable of supporting RAID or other forms of virtualisation. If required, however, these can be realised outside the JBOD disk subsystem, for example, as software in the server (Section 5.1) or as an independent virtualisation entity in the storage network (Section 5.6.3).

2.4 STORAGE VIRTUALISATION USING RAID

A disk subsystem with a RAID controller offers greater functional scope than a JBOD disk subsystem. RAID was originally developed at a time when hard disks were still very expensive and less reliable than they are today. RAID was originally called 'Redundant Array of Inexpensive Disks'. Today RAID stands for 'Redundant Array of Independent Disks'. Disk subsystems that support RAID are sometimes also called RAID arrays.

RAID has two main goals: to increase performance by striping and to increase fault-tolerance by redundancy. Striping distributes the data over several hard disks and thus distributes the load over more hardware. Redundancy means that additional information is stored so that the operation of the application itself can continue in the event of the failure of a hard disk. You cannot increase the performance of an individual hard disk any more than you can improve its fault-tolerance. Individual physical hard disks are slow and have a limited life-cycle. However, through a suitable combination of physical hard disks it is possible to significantly increase the fault-tolerance and performance of the system as a whole.

The bundle of physical hard disks brought together by the RAID controller are also known as virtual hard disks. A server that is connected to a RAID system sees only the virtual hard disk; the fact that the RAID controller actually distributes the data over several physical hard disks is completely hidden to the server (Figure 2.7). This is only visible to the administrator from outside.

A RAID controller can distribute the data that a server writes to the virtual hard disk amongst the individual physical hard disks in various manners. These different procedures are known as RAID levels. Section 2.5 explains various RAID levels in detail.

One factor common to almost all RAID levels is that they store redundant information. If a physical hard disk fails, its data can be reconstructed from the hard disks that remain intact. The defective hard disk can even be replaced by a new one during operation if a disk subsystem has the appropriate hardware. Then the RAID controller reconstructs the data of the exchanged hard disk. This process remains hidden to the server apart from a possible reduction in performance: the server can continue to work uninterrupted on the virtual hard disk.

Modern RAID controllers initiate this process automatically. This requires the definition of so-called hot spare disks (Figure 2.8). The hot spare disks are not used in normal operation. If a disk fails, the RAID controller immediately begins to copy the data of the remaining intact disk onto a hot spare disk. After the replacement of the defective disk,

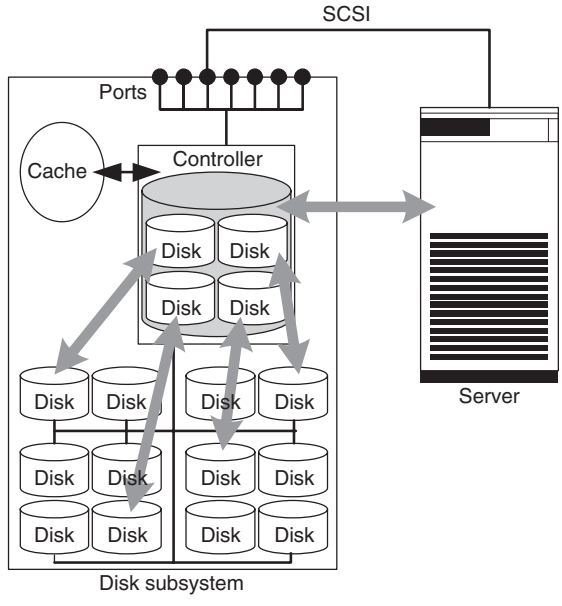

Figure 2.7 The RAID controller combines several physical hard disks to create a virtual hard disk. The server sees only a single virtual hard disk. The controller hides the assignment of the virtual hard disk to the individual physical hard disks.

this is included in the pool of hot spare disks. Modern RAID controllers can manage a common pool of hot spare disks for several virtual RAID disks. Hot spare disks can be defined for all RAID levels that offer redundancy.

The recreation of the data from a defective hard disk takes place at the same time as write and read operations of the server to the virtual hard disk, so that from the point of view of the server, performance reductions at least can be observed. Modern hard disks come with self-diagnosis programs that report an increase in write and read errors to the system administrator in plenty of time: 'Caution! I am about to depart this life. Please replace me with a new disk. Thank you!' To this end, the individual hard disks store the data with a redundant code such as the Hamming code. The Hamming code permits the correct recreation of the data, even if individual bits are changed on the hard disk. If the system is looked after properly you can assume that the installed physical hard disks will hold out for a while. Therefore, for the benefit of higher performance, it is generally an acceptable risk to give access by the server a higher priority than the recreation of the data of an exchanged physical hard disk.

A further side-effect of the bringing together of several physical hard disks to form a virtual hard disk is the higher capacity of the virtual hard disks. As a result, less device addresses are used up in the I/O channel and thus the administration of the server is also simplified, because less hard disks (drive letters or volumes) need to be used.

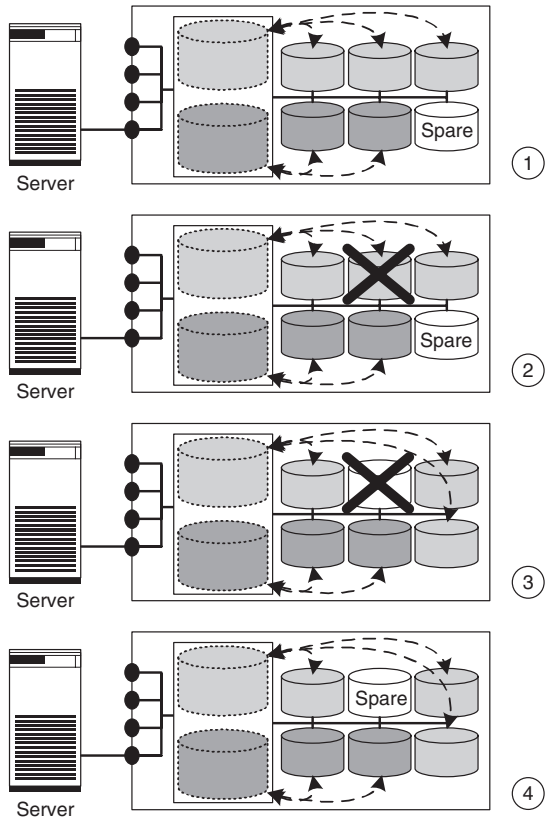

Figure 2.8 Hot spare disk: The disk subsystem provides the server with two virtual disks for which a common hot spare disk is available (1). Due to the redundant data storage the server can continue to process data even though a physical disk has failed, at the expense of a reduction in performance (2). The RAID controller recreates the data from the defective disk on the hot spare disk (3). After the defective disk has been replaced a hot spare disk is once again available (4).

2.5 DIFFERENT RAID LEVELS IN DETAIL

RAID has developed since its original definition in 1987. Due to technical progress some RAID levels are now practically meaningless, whilst others have been modified or added at a later date. This section introduces the RAID levels that are currently the most significant in practice. We will not introduce RAID levels that represent manufacturer-specific variants and variants that only deviate slightly from the basic forms mentioned in the following.

2.5.1 RAID 0: block-by-block striping

RAID 0 distributes the data that the server writes to the virtual hard disk onto one physical hard disk after another block-by-block (block-by-block striping). Figure 2.9 shows a RAID array with four physical hard disks. In Figure 2.9 the server writes the blocks A, B, C, D, E, etc. onto the virtual hard disk one after the other. The RAID controller distributes the sequence of blocks onto the individual physical hard disks: it writes the first block, A, to the first physical hard disk, the second block, B, to the second physical hard disk, block C to the third and block D to the fourth. Then it begins to write to the first physical hard disk once again, writing block E to the first disk, block F to the second, and so on.

RAID 0 increases the performance of the virtual hard disk as follows: the individual hard disks can exchange data with the RAID controller via the I/O channel significantly more quickly than they can write to or read from the rotating disk. In Figure 2.9 the RAID controller sends the first block, block A, to the first hard disk. This takes some time to write the block to the disk. Whilst the first disk is writing the first block to the physical hard disk, the RAID controller is already sending the second block, block B, to the second hard disk and block C to the third hard disk. In the meantime, the first two physical hard disks are still engaged in depositing their respective blocks onto the physical hard disk. If the RAID controller now sends block E to the first hard disk, then this has written block A at least partially, if not entirely, to the physical hard disk.

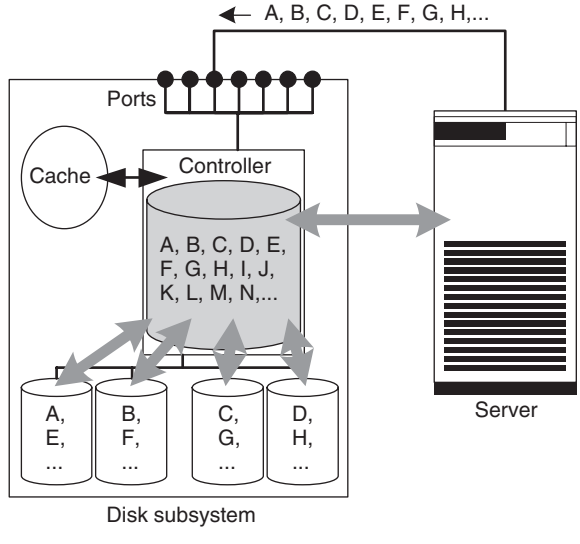

Figure 2.9 RAID 0 (striping): As in all RAID levels, the server sees only the virtual hard disk. The RAID controller distributes the write operations of the server amongst several physical hard disks. Parallel writing means that the performance of the virtual hard disk is higher than that of the individual physical hard disks.

In the example, it was possible to increase the throughput fourfold in 2002: Individual hard disks were able to achieve a throughput of around 50 MB/s. The four physical hard disks achieve a total throughput of around $4 \times 50\,\text{MB/s} \approx 200\,\text{MB/s}$. In those days I/O techniques such as SCSI or Fibre Channel achieve a throughput of 160 MB/s or 200 MB/s. If the RAID array consisted of just three physical hard disks the total throughput of the hard disks would be the limiting factor. If, on the other hand, the RAID array consisted of five physical hard disks the I/O path would be the limiting factor. With five or more hard disks, therefore, performance increases are only possible if the hard disks are connected to different I/O paths so that the load can be striped not only over several physical hard disks, but also over several I/O paths.

RAID 0 increases the performance of the virtual hard disk, but not its fault-tolerance. If a physical hard disk is lost, all the data on the virtual hard disk is lost. To be precise, therefore, the 'R' for 'Redundant' in RAID is incorrect in the case of RAID 0, with 'RAID 0' standing instead for 'zero redundancy'.

2.5.2 RAID 1: block-by-block mirroring

In contrast to RAID 0, in RAID 1 fault-tolerance is of primary importance. The basic form of RAID 1 brings together two physical hard disks to form a virtual hard disk by mirroring the data on the two physical hard disks. If the server writes a block to the virtual hard disk, the RAID controller writes this block to both physical hard disks (Figure 2.10). The individual copies are also called mirrors. Normally, two or sometimes three copies of the data are kept (three-way mirror).

In a normal operation with pure RAID 1, performance increases are only possible in read operations. After all, when reading the data the load can be divided between the two disks. However, this gain is very low in comparison to RAID 0. When writing with RAID 1 it tends to be the case that reductions in performance may even have to be taken into account. This is because the RAID controller has to send the data to both hard disks. This disadvantage can be disregarded for an individual write operation, since the capacity of the I/O channel is significantly higher than the maximum write speed of the two hard disks put together. However, the I/O channel is under twice the load, which hinders other data traffic using the I/O channel at the same time.

2.5.3 RAID 0+1/RAID 10: striping and mirroring combined

The problem with RAID 0 and RAID 1 is that they increase either performance (RAID 0) or fault-tolerance (RAID 1). However, it would be nice to have both performance and fault-tolerance. This is where RAID 0+1 and RAID 10 come into play. These two RAID levels combine the ideas of RAID 0 and RAID 1.

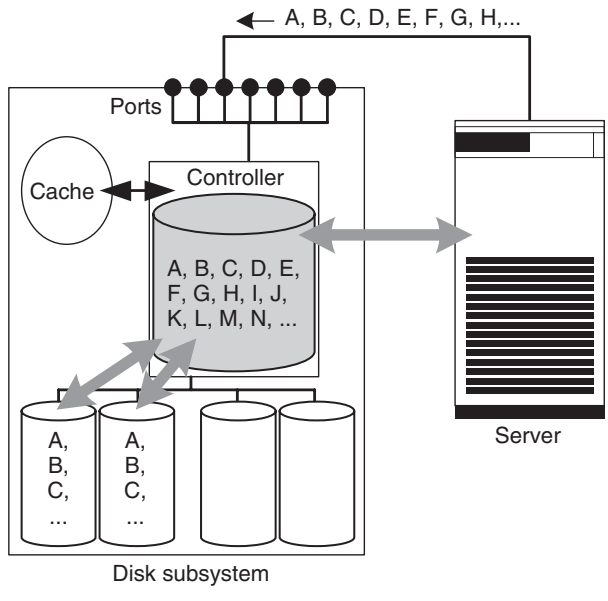

Figure 2.10 RAID 1 (mirroring): As in all RAID levels, the server sees only the virtual hard disk. The RAID controller duplicates each of the server's write operations onto two physical hard disks. After the failure of one physical hard disk the data can still be read from the other disk.

RAID 0+1 and RAID 10 each represent a two-stage virtualisation hierarchy. Figure 2.11 shows the principle behind RAID 0+1 (mirrored stripes). In the example, eight physical hard disks are used. The RAID controller initially brings together each four physical hard disks to form a total of two virtual hard disks that are only visible within the RAID controller by means of RAID 0 (striping). In the second level, it consolidates these two virtual hard disks into a single virtual hard disk by means of RAID 1 (mirroring); only this virtual hard disk is visible to the server.

In RAID 10 (striped mirrors) the sequence of RAID 0 (striping) and RAID 1 (mirroring) is reversed in relation to RAID 0+1 (mirrored stripes). Figure 2.12 shows the principle underlying RAID 10 based again on eight physical hard disks. In RAID 10 the RAID controller initially brings together the physical hard disks in pairs by means of RAID 1 (mirroring) to form a total of four virtual hard disks that are only visible within the RAID controller. In the second stage, the RAID controller consolidates these four virtual hard disks into a virtual hard disk by means of RAID 0 (striping). Here too, only this last virtual hard disk is visible to the server.

In both RAID 0+1 and RAID 10 the server sees only a single hard disk, which is larger, faster and more fault-tolerant than a physical hard disk. We now have to ask the question: which of the two RAID levels, RAID 0+1 or RAID 10, is preferable?

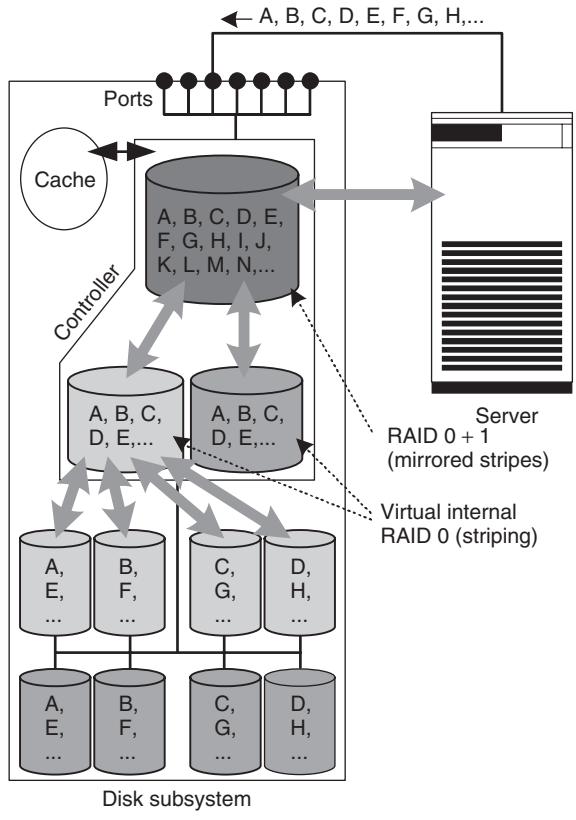

Figure 2.11 RAID 0+1 (mirrored stripes): As in all RAID levels, the server sees only the virtual hard disk. Internally, the RAID controller realises the virtual disk in two stages: in the first stage it brings together every four physical hard disks into one virtual hard disk that is only visible within the RAID controller by means of RAID 0 (striping); in the second stage it consolidates these two virtual hard disks by means of RAID 1 (mirroring) to form the hard disk that is visible to the server.

The question can be answered by considering that when using RAID 0 the failure of a hard disk leads to the loss of the entire virtual hard disk. In the example relating to RAID 0+1 (Figure 2.11) the failure of a physical hard disk is thus equivalent to the effective failure of four physical hard disks (Figure 2.13). If one of the other four physical hard disks is lost, then the data is lost. In principle it is sometimes possible to reconstruct the data from the remaining disks, but the RAID controllers available on the market cannot do this particularly well.

In the case of RAID 10, on the other hand, after the failure of an individual physical hard disk, the additional failure of a further physical hard disk – with the exception of the

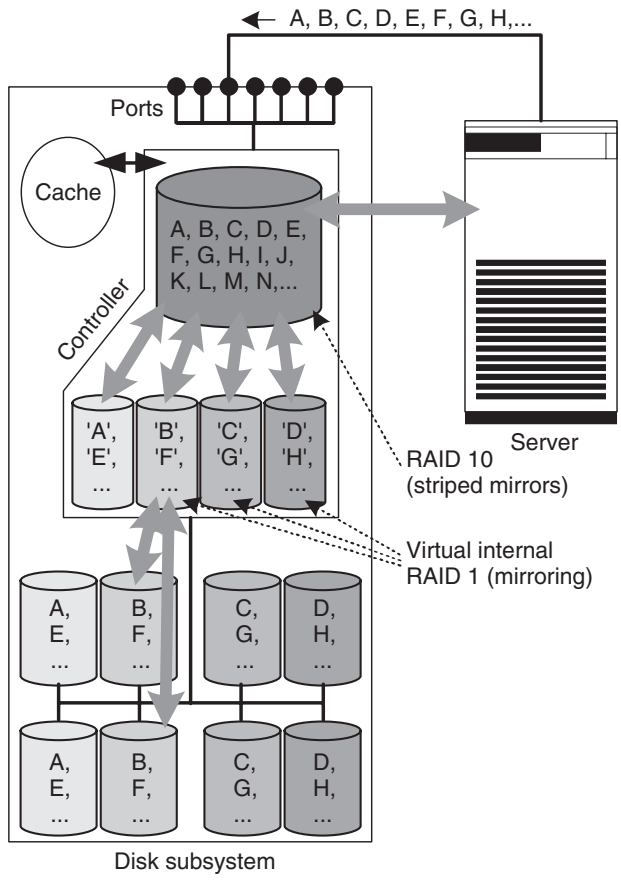

Figure 2.12 RAID 10 (striped mirrors): As in all RAID levels, the server sees only the virtual hard disk. Here too, we proceed in two stages. The sequence of striping and mirroring is reversed in relation to RAID 0+1. In the first stage the controller links every two physical hard disks by means of RAID 1 (mirroring) to a virtual hard disk, which it unifies by means of RAID 0 (striping) in the second stage to form the hard disk that is visible to the server.

corresponding mirror – can be withstood (Figure 2.14). RAID 10 thus has a significantly higher fault-tolerance than RAID 0+1. In addition, the cost of restoring the RAID system after the failure of a hard disk is much lower in the case of RAID 10 than RAID 0+1. In RAID 10 only one physical hard disk has to be recreated. In RAID 0+1, on the other hand, a virtual hard disk must be recreated that is made up of four physical disks. However, the cost of recreating the defective hard disk can be significantly reduced because a physical hard disk is exchanged as a preventative measure when the number of read errors start to increase. In this case it is sufficient to copy the data from the old disk to the new.

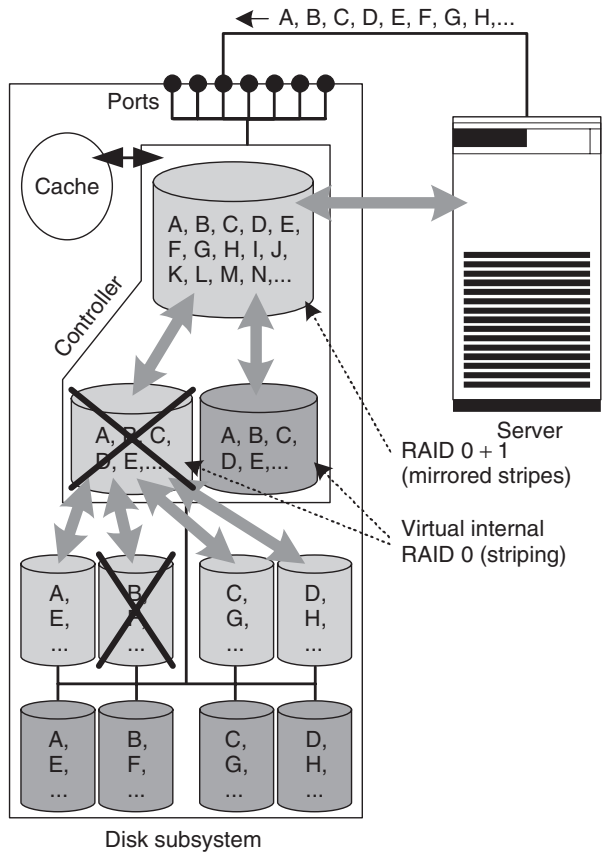

Figure 2.13 The consequences of the failure of a physical hard disk in RAID 0+1 (mirrored stripes) are relatively high in comparison to RAID 10 (striped mirrors). The failure of a physical hard disk brings about the failure of the corresponding internal RAID 0 disk, so that in effect half of the physical hard disks have failed. The recovery of the data from the failed disk is expensive.

However, things look different if the performance of RAID 0+1 is compared with the performance of RAID 10. In Section 5.1 we discuss a case study in which the use of RAID 0+1 is advantageous.

With regard to RAID 0+1 and RAID 10 it should be borne in mind that the two RAID procedures are often confused. Therefore the answer 'We use RAID 10!' or 'We use RAID 0+1' does not always provide the necessary clarity. In discussions it is better to ask if mirroring takes place first and the mirror is then striped or if striping takes place first and the stripes are then mirrored.

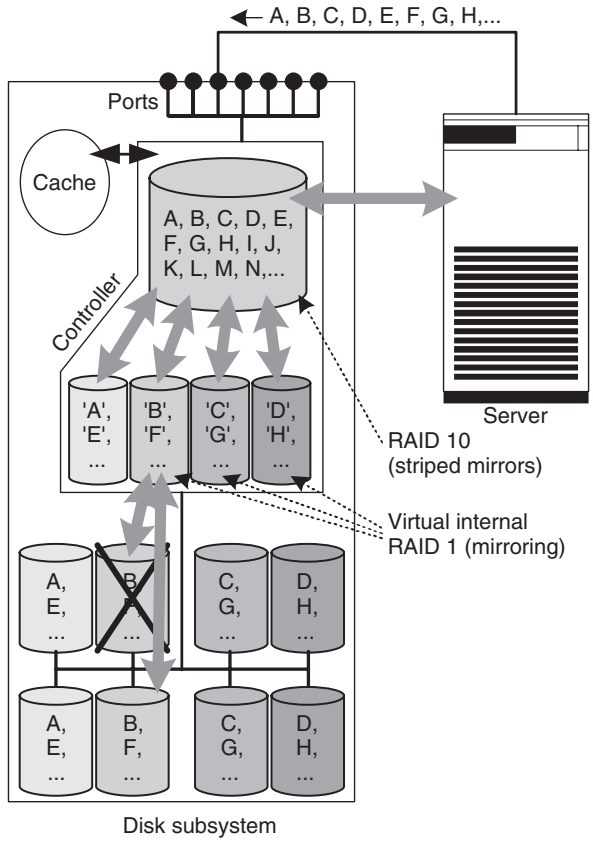

Figure 2.14 In RAID 10 (striped mirrors) the consequences of the failure of a physical hard disk are not as serious as in RAID 0+1 (mirrored stripes). All virtual hard disks remain intact. The recovery of the data from the failed hard disk is simple.

2.5.4 RAID 4 and RAID 5: parity instead of mirroring

RAID 10 provides excellent performance at a high level of fault-tolerance. The problem with this is that mirroring using RAID 1 means that all data is written to the physical hard disk twice. RAID 10 thus doubles the required storage capacity.

The idea of RAID 4 and RAID 5 is to replace all mirror disks of RAID 10 with a single parity hard disk. Figure 2.15 shows the principle of RAID 4 based upon five physical hard disks. The server again writes the blocks A, B, C, D, E, etc. to the virtual hard disk sequentially. The RAID controller stripes the data blocks over the first four physical hard disks. Instead of mirroring all data onto the further four physical hard disks, as in

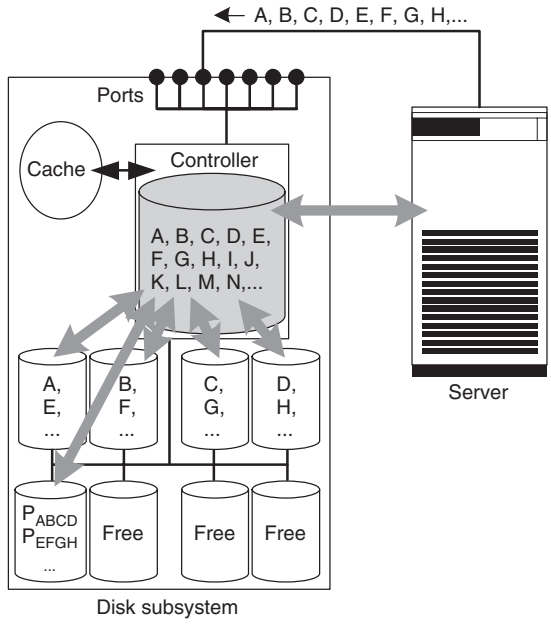

Figure 2.15 RAID 4 (parity disk) is designed to reduce the storage requirement of RAID 0+1 and RAID 10. In the example, the data blocks are distributed over four physical hard disks by means of RAID 0 (striping). Instead of mirroring all data once again, only a parity block is stored for each four blocks.

RAID 10, the RAID controller calculates a parity block for every four blocks and writes this onto the fifth physical hard disk. For example, the RAID controller calculates the parity block P_{ABCD} for the blocks A, B, C and D. If one of the four data disks fails, the RAID controller can reconstruct the data of the defective disks using the three other data disks and the parity disk. In comparison to the examples in Figures 2.11 (RAID 0+1) and 2.12 (RAID 10), RAID 4 saves three physical hard disks. As in all other RAID levels, the server again sees only the virtual disk, as if it were a single physical hard disk.

From a mathematical point of view the parity block is calculated with the aid of the logical XOR operator (Exclusive OR). In the example from Figure 2.15, for example, the equation $P_{ABCD} = A \text{ XOR } B \text{ XOR } C \text{ XOR } D$ applies.

The space saving offered by RAID 4 and RAID 5, which remains to be discussed, comes at a price in relation to RAID 10. Changing a data block changes the value of the associated parity block. This means that each write operation to the virtual hard disk requires (1) the physical writing of the data block, (2) the recalculation of the parity block and (3) the physical writing of the newly calculated parity block. This extra cost for write operations in RAID 4 and RAID 5 is called the write penalty of RAID 4 or the write penalty of RAID 5.

The cost for the recalculation of the parity block is relatively low due to the mathematical properties of the XOR operator. If the block A is overwritten by block $\tilde{A}$ and Δ is the difference between the old and new data block, then $\Delta = A \text{ XOR } \tilde{A}$. The new parity block $\tilde{P}$ can now simply be calculated from the old parity block P and Δ, i.e. $\tilde{P} = P \text{ XOR } \Delta$. Proof of this property can be found in Appendix A. Therefore, if P_{ABCD} is the parity block for the data blocks A, B, C and D, then after the data block A has been changed, the new parity block can be calculated without knowing the remaining blocks B, C and D. However, the old block A must be read in before overwriting the physical hard disk in the controller, so that this can calculate the difference Δ.

When processing write commands for RAID 4 and RAID 5 arrays, RAID controllers use the above-mentioned mathematical properties of the XOR operation for the recalculation of the parity block. Figure 2.16 shows a server that changes block D on the virtual hard disk. The RAID controller reads the data block and the associated parity block from the disk in question into its cache. Then it uses the XOR operation to calculate the difference

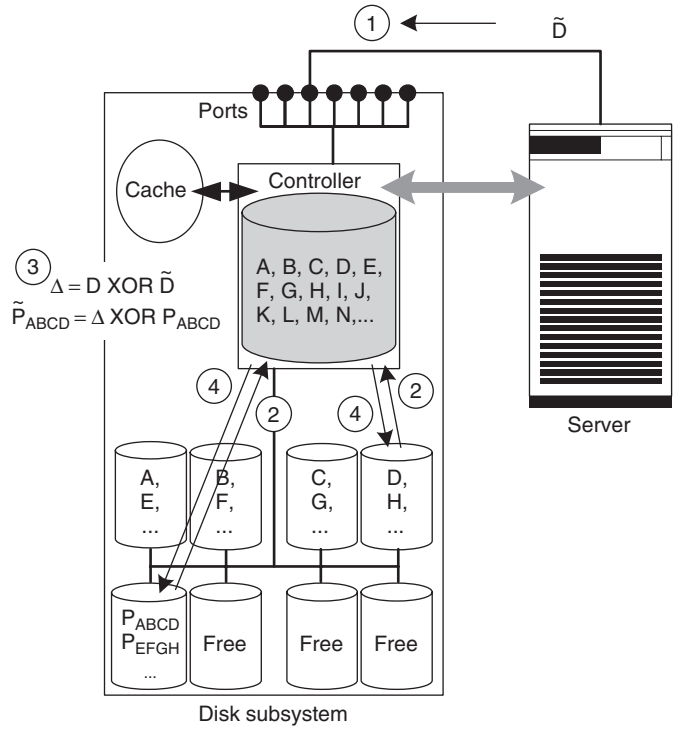

Figure 2.16 Write penalty of RAID 4 and RAID 5: The server writes a changed data block (1). The RAID controller reads in the old data block and the associated old parity block (2) and calculates the new parity block (3). Finally it writes the new data block and the new parity block onto the physical hard disk in question (4).

between the old and the new parity block, i.e. $\Delta = D \text{ XOR } \tilde{D}$, and from this the new parity block $\tilde{P}_{ABCD}$ by means of $\tilde{P}_{ABCD} = P_{ABCD} \text{ XOR } \Delta$. Therefore it is not necessary to read in all four associated data blocks to recalculate the parity block. To conclude the write operation to the virtual hard disk, the RAID controller writes the new data block and the recalculated parity block onto the physical hard disks in question.

Advanced RAID 4 and RAID 5 implementations are capable of reducing the write penalty even further for certain load profiles. For example, if large data quantities are written sequentially, then the RAID controller can calculate the parity blocks from the data flow without reading the old parity block from the disk. If, for example, the blocks E, F, G and H in Figure 2.15 are written in one go, then the controller can calculate the parity block P_{EFGH} from them and overwrite this without having previously read in the old value. Likewise, a RAID controller with a suitably large cache can hold frequently changed parity blocks in the cache after writing to the disk, so that the next time one of the data blocks in question is changed there is no need to read in the parity block. In both cases the I/O load is now lower than in the case of RAID 10. In the example only five physical blocks now need to be written instead of eight as is the case with RAID 10.

RAID 4 saves all parity blocks onto a single physical hard disk. For the example in Figure 2.15 this means that the write operations for the data blocks are distributed over four physical hard disks. However, the parity disk has to handle the same number of write operations all on its own. Therefore, the parity disk become the performance bottleneck of RAID 4 if there are a high number of write operations.

To get around this performance bottleneck, RAID 5 distributes the parity blocks over all hard disks. Figure 2.17 illustrates the procedure. As in RAID 4, the RAID controller writes the parity block P_{ABCD} for the blocks A, B, C and D onto the fifth physical hard disk. Unlike RAID 4, however, in RAID 5 the parity block P_{EFGH} moves to the fourth physical hard disk for the next four blocks E, F, G, H.

RAID 4 and RAID 5 distribute the data blocks over many physical hard disks. Therefore, the read performance of RAID 4 and RAID 5 is as good as that of RAID 0 and almost as good as that of RAID 10. As discussed, the write performance of RAID 4 and RAID 5 suffers from the write penalty; in RAID 4 there is an additional bottleneck caused by the parity disk. Therefore, RAID 4 is seldom used in practice because RAID 5 accomplishes more than RAID 4 with the same amount of physical resources (see also Section 2.5.6).

RAID 4 and RAID 5 can withstand the failure of a physical hard disk. Due to the use of parity blocks, the data on the defective hard disk can be restored with the help of other hard disks. In contrast to RAID 10, the failure of an individual sector of the remaining physical hard disks always results in data loss. This is compensated for with RAID 6, whereby a second parity hard disk is kept so that data is protected twice (Section 2.5.5).

In RAID 4 and RAID 5 the recovery of a defective physical hard disk is significantly more expensive than is the case for RAID 1 and RAID 10. In the latter two RAID levels only the mirror of the defective disk needs to be copied to the replaced disk. In RAID 4 and RAID 5, on the other hand, the RAID controller has to read the data from all disks, use this to recalculate the lost data blocks and parity blocks, and then write these blocks to the

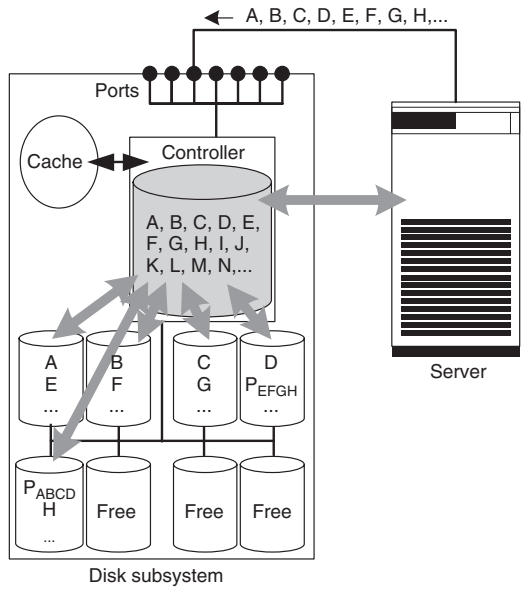

Figure 2.17 RAID 5 (striped parity): In RAID 4 each write access by the server is associated with a write operation to the parity disk for the update of parity information. RAID 5 distributes the load of the parity disk over all physical hard disks.

replacement disk. As in RAID 0+1 this high cost can be avoided by replacing a physical hard disk as a precaution as soon as the rate of read errors increases. If this is done, it is sufficient to copy the data from the hard disk to be replaced onto the new hard disk.

If the fifth physical hard disk has to be restored in the examples from Figure 2.15 (RAID 4) and Figure 2.17 (RAID 5), the RAID controller must first read the blocks A, B, C and D from the physical hard disks, recalculate the parity block P_{ABCD} and then write to the exchanged physical hard disk. If a data block has to be restored, only the calculation rule changes. If, in the example, the third physical hard disk is to be recreated, the controller would first have to read in the blocks A, B, D and P_{ABCD}, use these to reconstruct block C and write this to the replaced disk.

2.5.5 RAID 6: double parity

'How can it be that I am losing data even though it is protected by RAID 5?' Unfortunately, there are many storage administrators who do not deal with this question until *after* they have lost data despite using RAID 5 and not *before* this loss has taken place. With the size of the hard disks available today large quantities of data can quickly be lost. The main cause of this unpleasant surprise is that RAID 5 offers a high availability

solution for the failure of individual physical hard disks, whereas many people expect RAID 5 to provide a business continuity solution for the failure of multiple hard disks and, consequently, the entire RAID 5 array. Section 9.4.2 discusses this difference in detail.

Table 2.1 shows the failure probability for some RAID 5 configurations. The calculations are based on the technical data for the type of Fibre Channel hard disk commercially used in 2007. For example, the average serviceability of the disk was assumed to be 1,400,000 hours (approximately 160 years). In the model only the size of the hard disks has been varied. The first four columns of the table show the number of physical disks in a RAID array, the size of the individual disks, the net capacity of the RAID array and the ratio of net capacity to gross capacity (storage efficiency).

The fifth column shows the bit error rate (BER). The Fibre Channel hard disks commonly used in 2007 have a BER of 10^{-15} and the SATA hard disks a BER of 10^{-14}. A BER of 10^{-15} means that a sector of 100 TB of read-in data cannot be read, and a BER of 10^{-14} means that a sector of 10 TB of read-in data cannot be read. Hard disks with lower BER rates can be produced today. However, the advances in disk technology have mainly been focussed on increasing the capacity of individual hard disks and not on improving BER rates.

Table 2.1 The table shows the failure probability for different RAID 5 configurations (see text). In 2007, for example, a typical SATA disk had a size of 512 GB and a BER of 1E-15. At 93.8% a RAID array consisting of 16 such hard disks has a high level of storage efficiency, whereas the probability of data loss due to a stripe-kill error is around 22 years (line 9). This means that anyone who operates ten such arrays will on average lose one array every 2 years.

No. of disks	Disk size GB	Array size GB	Efficiency	BER	MTTDL BER	MTTDL DD	MTTDL Total
8	512	3.584	87.5%	1E-15	706	73.109	700
8	256	1.792	87.5%	1E-15	1.403	146.217	1.389
8	1.024	7.168	87.5%	1E-15	358	36.554	355
4	512	1.536	75.0%	1E-15	3.269	341.173	3.238
16	512	7.680	93.8%	1E-15	168	17.059	166
8	256	1.792	87.5%	1E-14	149	146.217	149
8	512	3.584	87.5%	1E-14	80	73.109	80
8	1.024	7.168	87.5%	1E-14	46	36.554	46
16	512	3.584	93.8%	1E-14	22	17.059	22
16	1.024	15.360	93.8%	1E-14	14	8.529	14
8	256	1.792	87.5%	1E-16	13.937	146.217	12.724
8	512	3.584	87.5%	1E-16	6.973	73.109	6.336
8	1.024	7.168	87.5%	1E-16	3.492	36.554	3.187
16	1.024	15.360	93.8%	1E-16	817	8.529	746

The last three columns show the probability of data loss in RAID 5 arrays. If a hard disk fails in a RAID 5 array, the data of all the intact disks is read so that the data of the faulty disk can be reconstructed (Section 2.5.4). This involves large quantities of data being read, thus increasing the probability of a corrupt sector surfacing during the reconstruction. This means that two blocks belonging to the same parity group are lost and therefore cannot be reconstructed again (mean time to data loss due to BER, MTTDL BER). Sometimes a second hard disk fails during the reconstruction. This too will lead to data loss (mean time to data loss due to double disk failure, MTTDL DD). The last column shows the overall probability of data loss in a RAID 5 array (mean time to data loss, MTTDL).

Line 1 shows the reference configuration for the comparison: a RAID 5 array with eight hard disks at 512 GB. The helper lines in table 2.1 show how different disk sizes or different array sizes have an effect on the probability of failure. The next two blocks show the probability of failure with a BER of 10^{-14}, which was common for SATA hard disks in 2007. In the model a life expectancy of 1,400,000 hours was assumed for these disks. In comparison, real SATA hard disks have a noticeably lower life expectancy and, consequently, an even higher probability of failure. Furthermore, SATA hard disks are often designed for office use (8×5 operations $= 5$ days $\times$ 8 hours). Hence, the likelihood of failures occurring earlier increases when these hard disks are used on a continuous basis in data centres (24×7 operations). Finally, the last block shows how the probability of failure would fall if hard disks were to have a BER of 10^{-16}.

RAID 5 arrays cannot correct the double failures described above. The table shows that there is a clear rise in the probability of data loss in RAID 5 arrays due to the increasing capacity of hard disks. RAID 6 offers a compromise between RAID 5 and RAID 10 by adding a second parity hard disk to extend RAID 5, which then uses less storage capacity than RAID 10. There are different approaches available today for calculating the two parity blocks of a parity group. However, none of these procedures has been adopted yet as an industry standard. Irrespective of an exact procedure, RAID 6 has a poor write performance because the write penalty for RAID 5 strikes twice (Section 2.5.4).

2.5.6 RAID 2 and RAID 3

When introducing the RAID levels we are sometimes asked: ‘and what about RAID 2 and RAID 3?’. The early work on RAID began at a time when disks were not yet very reliable: bit errors were possible that could lead to a written ‘one’ being read as ‘zero’ or a written ‘zero’ being read as ‘one’. In RAID 2 the Hamming code is used, so that redundant information is stored in addition to the actual data. This additional data permits the recognition of read errors and to some degree also makes it possible to correct them. Today, comparable functions are performed by the controller of each individual hard disk, which means that RAID 2 no longer has any practical significance.

Like RAID 4 or RAID 5, RAID 3 stores parity data. RAID 3 distributes the data of a block amongst all the disks of the RAID 3 system so that, in contrast to RAID 4 or RAID 5, all disks are involved in every read or write access. RAID 3 only permits the

reading and writing of whole blocks, thus dispensing with the write penalty that occurs in RAID 4 and RAID 5. The writing of individual blocks of a parity group is thus not possible. In addition, in RAID 3 the rotation of the individual hard disks is synchronised so that the data of a block can truly be written simultaneously. RAID 3 was for a long time called the recommended RAID level for sequential write and read profiles such as data mining and video processing. Current hard disks come with a large cache of their own, which means that they can temporarily store the data of an entire track, and they have significantly higher rotation speeds than the hard disks of the past. As a result of these innovations, other RAID levels are now suitable for sequential load profiles, meaning that RAID 3 is becoming less and less important.

2.5.7 A comparison of the RAID levels

The various RAID levels raise the question of which RAID level should be used when. Table 2.2 compares the criteria of fault-tolerance, write performance, read performance and space requirement for the individual RAID levels. The evaluation of the criteria can be found in the discussion in the previous sections.

CAUTION PLEASE: The comparison of the various RAID levels discussed in this section is only applicable to the theoretical basic forms of the RAID level in question. In practice, manufacturers of disk subsystems have design options in

- *the selection of the internal physical hard disks;*
- *the I/O technique used for the communication within the disk subsystem;*
- *the use of several I/O channels;*
- *the realisation of the RAID controller;*
- *the size of the cache;*
- *the cache algorithms themselves;*
- *the behaviour during rebuild; and*
- *the provision of advanced functions such as data scrubbing and preventive rebuild.*

Table 2.2 The table compares the theoretical basic forms of different RAID levels. In practice, huge differences exist in the quality of the implementation of RAID controllers.

RAID level	Fault-tolerance	Read performance	Write performance	Space requirement
RAID 0	None	Good	Very good	Minimal
RAID 1	High	Poor	Poor	High
RAID 10	Very high	Very good	Good	High
RAID 4	High	Good	Very very poor	Low
RAID 5	High	Good	Very poor	Low
RAID 6	Very high	Good	Very very poor	Low

The performance data of the specific disk subsystem must be considered very carefully for each individual case. For example, in the previous chapter measures were discussed that greatly reduce the write penalty of RAID 4 and RAID 5. Specific RAID controllers may implement these measures, but they do not have to.

Another important difference is the behaviour during rebuild. The reconstruction of a defective hard disk puts a heavy burden on the internal I/O buses of a disk subsystem and on the disks of the affected array. The disk subsystem must cope with rebuild in addition to its normal workload. In reality, RAID controllers offer no possibility for controlling rebuild behaviour, such as through a higher prioritisation of the application workload compared to the rebuild. The disk subsystem must be accepted as it is. This means that during capacity planning the behaviour of a disk subsystem in failure situations must be taken into account, such as in a RAID rebuild or in the failure of a redundant RAID controller.

Additional functions such as data scrubbing and preventive rebuild are also important. With data scrubbing, a RAID controller regularly reads all blocks and in the process detects defective blocks before they are actively accessed by the applications. This clearly reduces the probability of data loss due to bit errors (see Table 2.1).

Another measure is preventive rebuild. Hard disks become slower before they fail, because the controller of a hard disk tries to correct read errors – for example, through a repetitive read of the block. High-end RAID controllers detect this kind of performance degradation in an individual hard disk and replace it before it fails altogether. The time required to restore a defective hard disk can be substantially reduced if it is preventively replaced as soon as the correctible read error is detected. It is then sufficient to copy the data from the old hard disk onto the new one.

Subject to the above warning, RAID 0 is the choice for applications for which the maximum write performance is more important than protection against the failure of a disk. Examples are the storage of multimedia data for film and video production and the recording of physical experiments in which the entire series of measurements has no value if all measured values cannot be recorded. In this case it is more beneficial to record all of the measured data on a RAID 0 array first and then copy it after the experiment, for example on a RAID 5 array. In databases, RAID 0 is used as a fast store for segments in which intermediate results for complex requests are to be temporarily stored. However, as a rule hard disks tend to fail at the most inconvenient moment so database administrators only use RAID 0 if it is absolutely necessary, even for temporary data.

With RAID 1, performance and capacity are limited because only two physical hard disks are used. RAID 1 is therefore a good choice for small databases for which the configuration of a virtual RAID 5 or RAID 10 disk would be too large. A further important field of application for RAID 1 is in combination with RAID 0.

RAID 10 is used in situations where high write performance and high fault-tolerance are called for. For a long time it was recommended that database log files be stored on RAID 10. Databases record all changes in log files so this application has a high write component. After a system crash the restarting of the database can only be guaranteed if all log files are fully available. Manufacturers of storage systems disagree as to whether this recommendation is still valid as there are now fast RAID 4 and RAID 5 implementations.

RAID 4 and RAID 5 save disk space at the expense of a poorer write performance. For a long time the rule of thumb was to use RAID 5 where the ratio of read operations to write operations is 70:30. At this point we wish to repeat that there are now storage systems on the market with excellent write performance that store the data internally using RAID 4 or RAID 5.

RAID 6 is a compromise between RAID 5 and RAID 10. It offers large hard disks considerably more protection from failure than RAID 5, albeit at the cost of clearly poorer write performance. What is particularly interesting is the use of RAID 6 for archiving, such as with hard disk WORM storage, where read access is prevalent (Section 8.3.1). With the appropriate cache algorithms, RAID 6 can be as effective as RAID 5 for workloads with a high proportion of sequential write access.

2.6 CACHING: ACCELERATION OF HARD DISK ACCESS

In all fields of computer systems, caches are used to speed up slow operations by operating them from the cache. Specifically in the field of disk subsystems, caches are designed to accelerate write and read accesses to physical hard disks. In this connection we can differentiate between two types of cache: (1) cache on the hard disk (Section 2.6.1) and (2) cache in the RAID controller. The cache in the RAID controller is subdivided into write cache (Section 2.6.2) and read cache (Section 2.6.3).

2.6.1 Cache on the hard disk

Each individual hard disk comes with a very small cache. This is necessary because the transfer rate of the I/O channel to the disk controller is significantly higher than the speed at which the disk controller can write to or read from the physical hard disk. If a server or a RAID controller writes a block to a physical hard disk, the disk controller stores this in its cache. The disk controller can thus write the block to the physical hard disk in its own time whilst the I/O channel can be used for data traffic to the other hard disks. Many RAID levels use precisely this state of affairs to increase the performance of the virtual hard disk.

Read access is accelerated in a similar manner. If a server or an intermediate RAID controller wishes to read a block, it sends the address of the requested block to the hard disk controller. The I/O channel can be used for other data traffic while the hard disk controller copies the complete block from the physical hard disk into its cache at a slower data rate. The hard disk controller transfers the block from its cache to the RAID controller or to the server at the higher data rate of the I/O channel.

2.6.2 Write cache in the disk subsystem controller

In addition to the cache of the individual hard drives many disk subsystems come with their own cache, which in some models is gigabytes in size. As a result it can buffer much greater data quantities than the cache on the hard disk. The write cache should have a battery backup and ideally be mirrored. The battery backup is necessary to allow the data in the write cache to survive a power cut. A write cache with battery backup can significantly reduce the write penalty of RAID 4 and RAID 5, particularly for sequential write access (cf. Section 2.5.4 'RAID 4 and RAID 5: parity instead of mirroring'), and smooth out load peaks.

Many applications do not write data at a continuous rate, but in batches. If a server sends several data blocks to the disk subsystem, the controller initially buffers all blocks into a write cache with a battery backup and immediately reports back to the server that all data has been securely written to the drive. The disk subsystem then copies the data from the write cache to the slower physical hard disk in order to make space for the next write peak.

2.6.3 Read cache in the disk subsystem controller

The acceleration of read operations is difficult in comparison to the acceleration of write operations using cache. To speed up read access by the server, the disk subsystem's controller must copy the relevant data blocks from the slower physical hard disk to the fast cache before the server requests the data in question.

The problem with this is that it is very difficult for the disk subsystem's controller to work out in advance what data the server will ask for next. The controller in the disk subsystem knows neither the structure of the information stored in the data blocks nor the access pattern that an application will follow when accessing the data. Consequently, the controller can only analyse past data access and use this to extrapolate which data blocks the server will access next. In sequential read processes this prediction is comparatively simple, in the case of random access it is almost impossible. As a rule of thumb, good RAID controllers manage to provide around 40% of the requested blocks from the read cache in mixed read profiles.

The disk subsystem's controller cannot further increase the ratio of read access provided from the cache (pre-fetch hit rate), because it does not have the necessary application knowledge. Therefore, it is often worthwhile realising a further cache within applications. For example, after opening a file, file systems can load all blocks of the file into the main memory (RAM); the file system knows the structures that the files are stored in. File systems can thus achieve a pre-fetch hit rate of 100%. However, it is impossible to know whether the expense for the storage of the blocks is worthwhile in an individual case, since the application may not actually request further blocks of the file.

2.7 INTELLIGENT DISK SUBSYSTEMS

Intelligent disk subsystems represent the third level of complexity for controllers after JBODs and RAID arrays. The controllers of intelligent disk subsystems offer additional functions over and above those offered by RAID. In the disk subsystems that are currently available on the market these functions are usually instant copies (Section 2.7.1), remote mirroring (Section 2.7.2) and LUN masking (Section 2.7.3).

2.7.1 Instant copies

Instant copies can virtually copy data sets of several terabytes within a disk subsystem in a few seconds. Virtual copying means that disk subsystems fool the attached servers into believing that they are capable of copying such large data quantities in such a short space of time. The actual copying process takes significantly longer. However, the same server, or a second server, can access the virtually copied data after a few seconds (Figure 2.18).

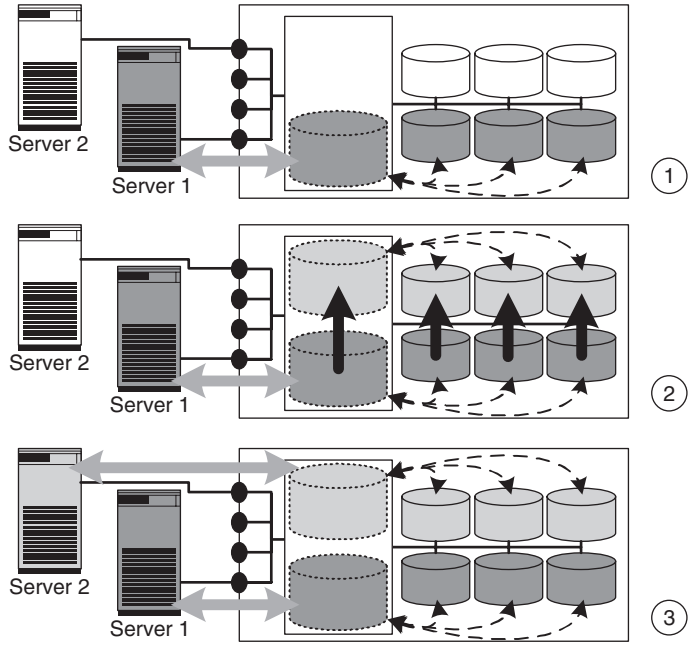

Figure 2.18 Instant copies can virtually copy several terabytes of data within a disk subsystem in a few seconds: server 1 works on the original data (1). The original data is virtually copied in a few seconds (2). Then server 2 can work with the data copy, whilst server 1 continues to operate with the original data (3).

Instant copies are used, for example, for the generation of test data, for the backup of data and for the generation of data copies for data mining. Based upon the case study in Section 1.3 it was shown that when copying data using instant copies, attention should be paid to the consistency of the copied data. Sections 7.8.4 and 7.10.3 discuss in detail the interaction of applications and storage systems for the generation of consistent instant copies.

There are numerous alternative implementations for instant copies. One thing that all implementations have in common is that the pretence of being able to copy data in a matter of seconds costs resources. All realisations of instant copies require controller computing time and cache and place a load on internal I/O channels and hard disks. The different implementations of instant copy force the performance down at different times. However, it is not possible to choose the most favourable implementation alternative depending upon the application used because real disk subsystems only ever realise one implementation alternative of instant copy.

In the following, two implementation alternatives will be discussed that function in very different ways. At one extreme the data is permanently mirrored (RAID 1 or RAID 10). Upon the copy command both mirrors are separated: the separated mirrors can then be used independently of the original. After the separation of the mirror the production data is no longer protected against the failure of a hard disk. Therefore, to increase data protection, three mirrors are often kept prior to the separation of the mirror (three-way mirror), so that the production data is always mirrored after the separation of the copy.

At the other extreme, no data at all is copied prior to the copy command, only after the instant copy has been requested. To achieve this, the controller administers two data areas, one for the original data and one for the data copy generated by means of instant copy. The controller must ensure that during write and read access operations to original data or data copies the blocks in question are written to or read from the data areas in question. In some implementations it is permissible to write to the copy, in some it is not. Some implementations copy just the blocks that have actually changed (partial copy), others copy all blocks as a background process until a complete copy of the original data has been generated (full copy).

In the following, the case differentiations of the controller will be investigated in more detail based upon the example from Figure 2.18. We will first consider access by server 1 to the original data. Read operations are completely unproblematic; they are always served from the area of the original data. Handling write operations is trickier. If a block is changed for the first time since the generation of the instant copy, the controller must first copy the old block to the data copy area so that server 2 can continue to access the old data set. Only then may it write the changed block to the original data area. If a block that has already been changed in this manner has to be written again, it must be written to the original data area. The controller may not even back up the previous version of the block to the data copy area because otherwise the correct version of the block would be overwritten.

The case differentiations for access by server 2 to the data copy generated by means of instant copy are somewhat simpler. In this case, write operations are unproblematic: the controller always writes all blocks to the data copy area. On the other hand, for read

operations it has to distinguish whether the block in question has already been copied or not. This determines whether it has to read the block from the original data area or read it from the data copy area and forward it to the server.

The subsequent copying of the blocks offers the basis for important variants of instant copy. Space-efficient instant copy only copies the blocks that were changed (Figure 2.19). These normally require considerably less physical storage space than the entire copy. Yet the exported virtual hard disks of the original hard disk and the copy created through space-efficient instant copy are of the same size. From the view of the server both virtual disks continue to have the same size. Therefore, less physical storage space is needed overall and, therefore, the cost of using instant copy can be reduced.

Incremental instant copy is another important variant of instant copy. In some situations such a heavy burden is placed on the original data and the copy that the performance within

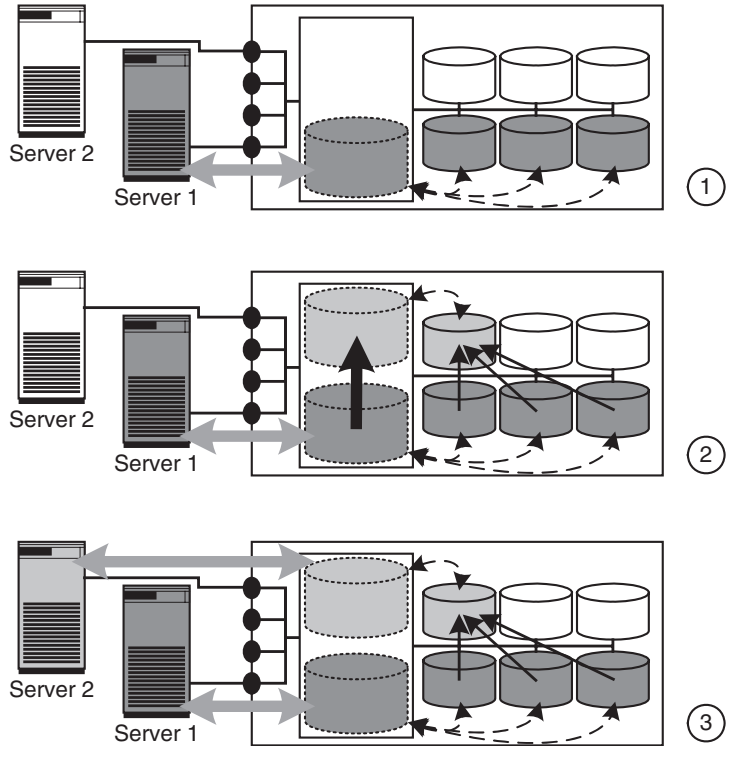

Figure 2.19 Space-efficient instant copy manages with fewer storage systems than the basic form of instant copy (Figure 2.18). Space-efficient instant copy actually only copies the changed blocks before they have been overwritten into the separate area (2). From the view of server 2 the hard disk that has been copied in this way is just as large as the source disk (3). The link between source and copy remains as the changed blocks are worthless on their own.

the disk subsystem suffers unless the data has been copied completely onto the copy. An example of this is backup when data is completely backed up through instant copy (Section 7.8.4). On the other side, the background process for copying all the data of an instant copy requires many hours when very large data volumes are involved, making this not a viable alternative. One remedy is incremental instant copy where data is only copied in its entirety the first time around. Afterwards the instant copy is repeated – for example, perhaps daily – whereby only those changes since the previous instant copy are copied.

A reversal of instant copy is yet another important variant. If data is backed up through instant copy, then the operation should be continued with the copy if a failure occurs. A simple approach is to shut down the application, copy back the data on the productive hard disks per a second instant copy from the copy onto the productive hard disks and restart the application. In this case, the disk subsystem must enable a reversal of the instant copy. If this function is not available, if a failure occurs the data either has to be copied back to the productive disks by different means or the operation continues directly with the copy. Both of these approaches are coupled with major copying operations or major configuration changes, and, consequently the recovery takes considerably longer than a reversal of the instant copy.

2.7.2 Remote mirroring

Instant copies are excellently suited for the copying of data sets within disk subsystems. However, they can only be used to a limited degree for data protection. Although data copies generated using instant copy protect against application errors (accidental deletion of a file system) and logical errors (errors in the database program), they do not protect against the failure of a disk subsystem. Something as simple as a power failure can prevent access to production data and data copies for several hours. A fire in the disk subsystem would destroy original data and data copies. For data protection, therefore, the proximity of production data and data copies is fatal.

Remote mirroring offers protection against such catastrophes. Modern disk subsystems can now mirror their data, or part of their data, independently to a second disk subsystem, which is a long way away. The entire remote mirroring operation is handled by the two participating disk subsystems. Remote mirroring is invisible to application servers and does not consume their resources. However, remote mirroring requires resources in the two disk subsystems and in the I/O channel that connects the two disk subsystems together, which means that reductions in performance can sometimes make their way through to the application.

Figure 2.20 shows an application that is designed to achieve high availability using remote mirroring. The application server and the disk subsystem, plus the associated data, are installed in the primary data centre. The disk subsystem independently mirrors the application data onto the second disk subsystem that is installed 50 kilometres away in the backup data centre by means of remote mirroring. Remote mirroring ensures that the application data in the backup data centre is always kept up-to-date with the time

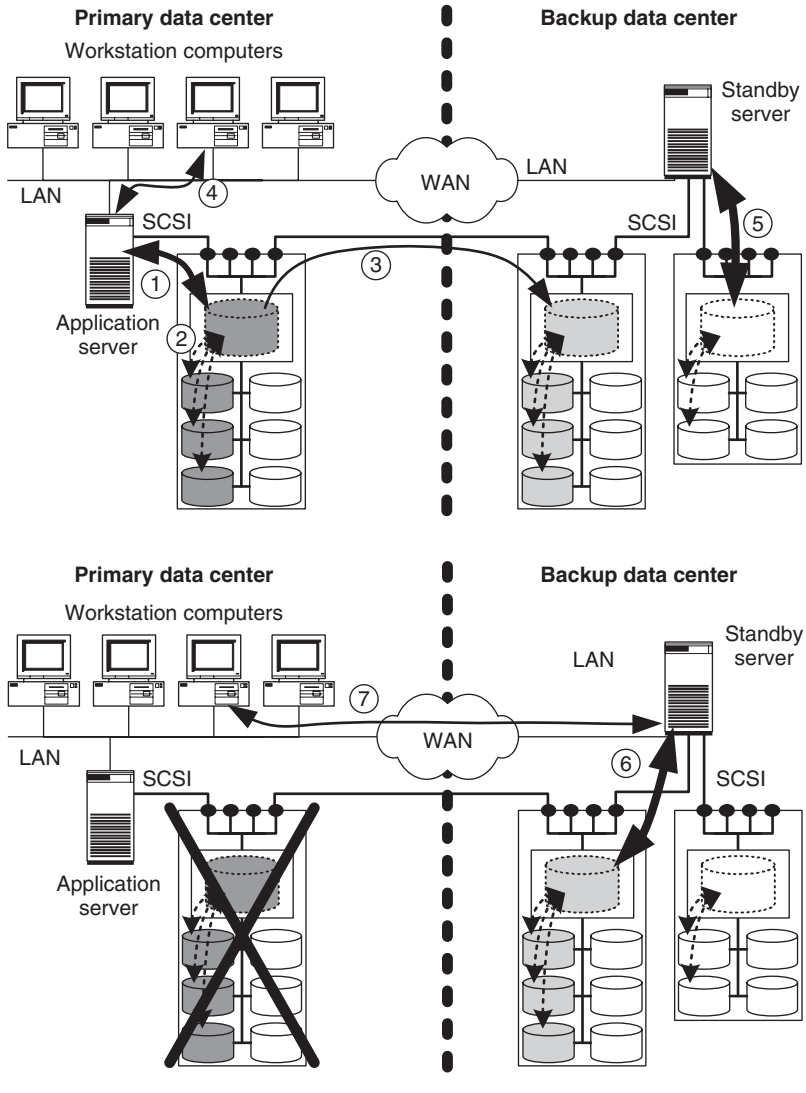

Figure 2.20 High availability with remote mirroring: (1) The application server stores its data on a local disk subsystem. (2) The disk subsystem saves the data to several physical drives by means of RAID. (3) The local disk subsystem uses remote mirroring to mirror the data onto a second disk subsystem located in the backup data centre. (4) Users use the application via the LAN. (5) The stand-by server in the backup data centre is used as a test system. The test data is located on a further disk subsystem. (6) If the first disk subsystem fails, the application is started up on the stand-by server using the data of the second disk subsystem. (7) Users use the application via the WAN.

interval for updating the second disk subsystem being configurable. If the disk subsystem in the primary data centre fails, the backup application server in the backup data centre can be started up using the data of the second disk subsystem and the operation of the application can be continued. The I/O techniques required for the connection of the two disk subsystems will be discussed in the next chapter.

We can differentiate between synchronous and asynchronous remote mirroring. In synchronous remote mirroring the first disk subsystem sends the data to the second disk subsystem first before it acknowledges a server's write command. By contrast, asynchronous remote mirroring acknowledges a write command immediately; only then does it send the copy of the block to the second disk subsystem.

Figure 2.21 illustrates the data flow of synchronous remote mirroring. The server writes block A to the first disk subsystem. This stores the block in its write cache and immediately sends it to the second disk subsystem, which also initially stores the block in its write cache. The first disk subsystem waits until the second reports that it has written the block. The question of whether the block is still stored in the write cache of the second disk subsystem or has already been written to the hard disk is irrelevant to the first disk subsystem. It does not acknowledge to the server that the block has been written until it has received confirmation from the second disk subsystem that this has written the block.

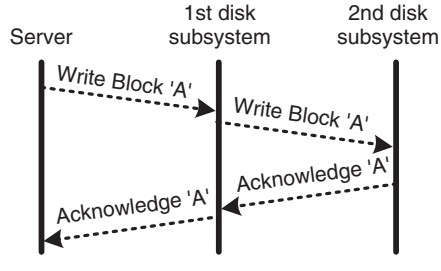

Figure 2.21 In synchronous remote mirroring a disk subsystem does not acknowledge write operations until it has saved a block itself and received write confirmation from the second disk subsystem.

Synchronous remote mirroring has the advantage that the copy of the data held by the second disk subsystem is always up-to-date. This means that if the first disk subsystem fails, the application can continue working with the most recent data set by utilising the data on the second disk subsystem.

The disadvantage is that copying the data from the first disk subsystem to the second and sending the write acknowledgement back from the second to the first increases the response time of the first disk subsystem to the server. However, it is precisely this response time that determines the throughput of applications such as databases and file systems. An important factor for the response time is the signal transit time between the two disk subsystems. After all, their communication is encoded in the form of physical

signals, which propagate at a certain speed. The propagation of the signals from one disk subsystem to another simply costs time. As a rule of thumb, it is worth using synchronous remote mirroring if the cable lengths from the server to the second disk subsystem via the first are a maximum of 6–10 kilometres. However, many applications can deal with noticeably longer distances. Although performance may then not be optimal, it is still good enough.

If we want to mirror the data over longer distances, then we have to switch to asynchronous remote mirroring. Figure 2.22 illustrates the data flow in asynchronous remote mirroring. In this approach the first disk subsystem acknowledges the receipt of data as soon as it has been temporarily stored in the write cache. The first disk subsystem does not send the copy of the data to the second disk subsystem until later. The write confirmation of the second disk subsystem to the first is not important to the server that has written the data.

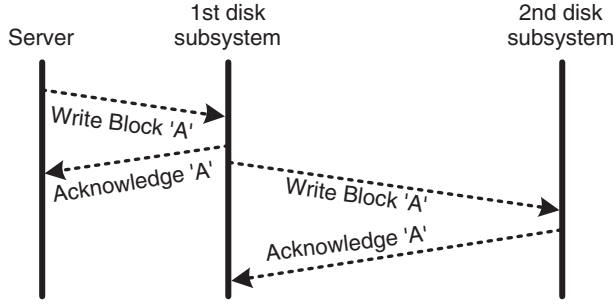

Figure 2.22 In asynchronous remote mirroring one disk subsystem acknowledges a write operation as soon as it has saved the block itself.

The price of the rapid response time achieved using asynchronous remote mirroring is obvious. In contrast to synchronous remote mirroring, in asynchronous remote mirroring there is no guarantee that the data on the second disk subsystem is up-to-date. This is precisely the case if the first disk subsystem has sent the write acknowledgement to the server but the block has not yet been saved to the second disk subsystem.

If we wish to mirror data over long distances but do not want to use only asynchronous remote mirroring it is necessary to use three disk subsystems (Figure 2.23). The first two may be located just a few kilometres apart, so that synchronous remote mirroring can be used between the two. In addition, the data of the second disk subsystem is mirrored onto a third by means of asynchronous remote mirroring. However, this solution comes at a price: for most applications the cost of data protection would exceed the costs that would be incurred after data loss in the event of a catastrophe. This approach would therefore only be considered for very important applications.

An important aspect of remote mirroring is the duration of the initial copying of the data. With large quantities of data it can take several hours until all data is copied from

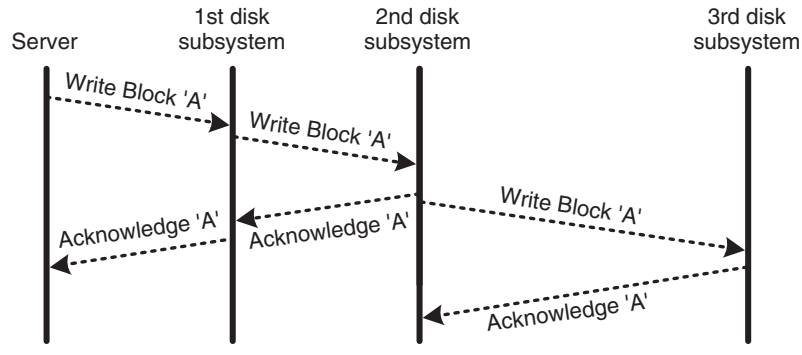

Figure 2.23 The combination of synchronous and asynchronous remote mirroring means that rapid response times can be achieved in combination with mirroring over long distances.

the first disk subsystem to the second one. This is completely acceptable the first time remote mirroring is established. However, sometimes the connection between both disk subsystems is interrupted later during operations – for example, due to a fault in the network between both systems or during maintenance work on the second disk subsystem. After the appropriate configuration the application continues operation on the first disk subsystem without the changes having been transferred to the second disk subsystem. Small quantities of data can be transmitted in their entirety again after a fault has been resolved. However, with large quantities of data a mechanism should exist that allows only those blocks that were changed during the fault to be transmitted. This is also referred to as suspending (or freezing) remote mirroring and resuming it later on. Sometimes there is a deliberate reason for suspending a remote mirroring relationship. Later in the book we will present a business continuity solution that suspends remote mirroring relationships at certain points in time for the purposes of creating consistent copies in backup data centres (Section 9.5.6).

For these same reasons there is a need for a reversal of remote mirroring. In this case, if the first disk subsystem fails, the entire operation is completely switched over to the second disk subsystem and afterwards the data is only changed on that second system. The second disk subsystem logs all changed blocks so that only those blocks that were changed during the failure are transmitted to the first disk subsystem once it is operational again. This ensures that the data on both disk subsystems is synchronised once again.

2.7.3 Consistency groups

Applications such as databases normally stripe their data over multiple virtual hard disks. Depending on data quantities and performance requirements, the data is sometimes even distributed over multiple disk subsystems. Sometimes, as with the web architecture,

multiple applications that are running on different operating systems manage common related data sets (Section 6.4.2). The copies created through instant copy and remote mirroring must also be consistent for these types of distributed data sets so that they can be used if necessary to restart operation.

The problem in this case is that, unless other measures are taken, the copying from multiple virtual hard disks through instant copy and remote mirroring will not be consistent. If, for example, a database with multiple virtual hard disks is copied using instant copy, the copies are created at almost the same time, but not exactly at the same time. However, databases continuously write time stamps into their virtual hard disks. At restart the database then checks the time stamp of the virtual hard disks and aborts the start if the time stamps of all the disks do not match 100%. This means that the operation cannot be restarted with the copied data and the use of instant copy has been worthless.

Consistency groups provide help in this situation. A consistency group for instant copy combines multiple instant copy pairs into one unit. If an instant copy is then requested for a consistency group, the disk subsystem makes sure that all virtual hard disks of the consistency group are copied at exactly the same point in time. Due to this simultaneous copying of all the virtual hard disks of a consistency group, the copies are given a consistent set of time stamps. An application can therefore restart with the hard disks that have been copied in this way. When a consistency group is copied via instant copy, attention of course has to be paid to the consistency of the data on each individual virtual hard disk – the same as when an individual hard disk is copied (Section 1.3). It is also important that the instant copy pairs of a consistency group can span multiple disk subsystems as for large databases and large file systems.

The need to combine multiple remote mirroring pairs into a consistency group also exists with remote mirroring (Figure 2.24). Here too the consistency group should be able to span multiple disk subsystems. If the data of an application is distributed over multiple virtual hard disks or even over multiple disk subsystems, and, if the remote mirroring of this application has been deliberately suspended so that a consistent copy of the data can be created in the backup data centre (Section 9.5.6), then all remote mirroring pairs must be suspended at exactly the same point in time. This will ensure that the time stamps on the copies are consistent and the application can restart smoothly from the copy.

Write-order consistency for asynchronous remote mirroring pairs is another important feature of consistency groups for remote mirroring. A prerequisite for the functions referred to as journaling of file systems (Section 4.1.2) and log mechanisms of databases (Section 7.10.1), which are presented later in this book, is that updates of data sets are executed in a very specific sequence. If the data is located on multiple virtual hard disks that are mirrored asynchronously, the changes can get ahead of themselves during the mirroring so that the data in the backup data centre is updated in a different sequence than in the primary data centre (Figure 2.25). This means that the consistency of the data in the backup centre is then at risk. Write-order consistency ensures that, despite asynchronous mirroring, the data on the primary hard disks and on the target hard disks is updated in the same sequence – even when it spans multiple virtual hard disks or multiple disk subsystems.

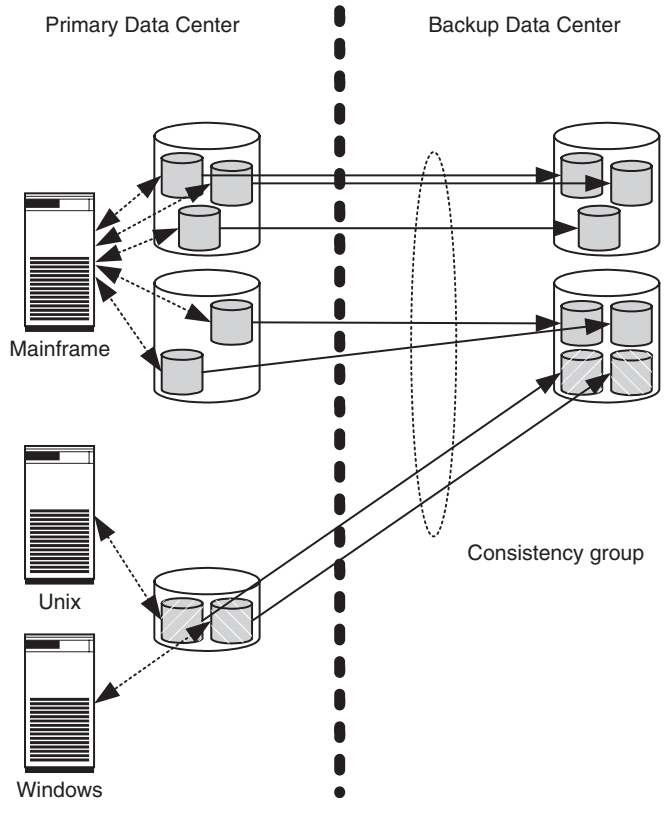

Figure 2.24 The example shows a distributed application that manages shared data over multiple servers. A large database needing two disk subsystems is running on one of the servers. For data consistency it is important that changes in the database are synchronised over all the servers and disk subsystems. Consistency groups help to maintain this synchronisation during copying using instant copy and remote mirroring.

2.7.4 LUN masking

So-called LUN masking brings us to the third important function – after instant copy and remote mirroring – that intelligent disk subsystems offer over and above that offered by RAID. LUN masking limits the access to the hard disks that the disk subsystem exports to the connected server.

A disk subsystem makes the storage capacity of its internal physical hard disks available to servers by permitting access to individual physical hard disks, or to virtual hard disks created using RAID, via the connection ports. Based upon the SCSI protocol, all hard disks – physical and virtual – that are visible outside the disk subsystem are also known as LUN.

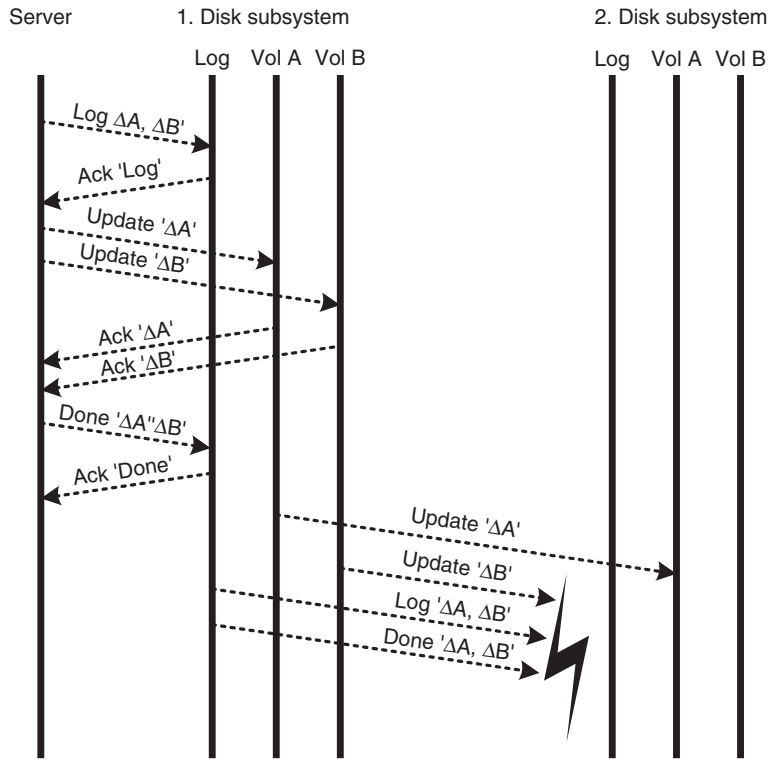

Figure 2.25 The consistency mechanism of databases and file systems records changes to data in a log, requiring adherence to a very specific sequence for writing in the log and the data. With asynchronous remote mirroring this sequence can be changed so that the start-up of an application will fail if a copy is used. The write-order consistency for asynchronous remote mirroring ensures that the copy is changed in the same sequence as the original on all the disks.

Without LUN masking every server would see all hard disks that the disk subsystem provides. Figure 2.26 shows a disk subsystem without LUN masking to which three servers are connected. Each server sees all hard disks that the disk subsystem exports outwards. As a result, considerably more hard disks are visible to each server than is necessary.

In particular, on each server those hard disks that are required by applications that run on a different server are visible. This means that the individual servers must be very carefully configured. In Figure 2.26 an erroneous formatting of the disk LUN 3 of server 1 would destroy the data of the application that runs on server 3. In addition, some operating systems are very greedy: when booting up they try to draw to them each hard disk that is written with the signature (label) of a foreign operating system.

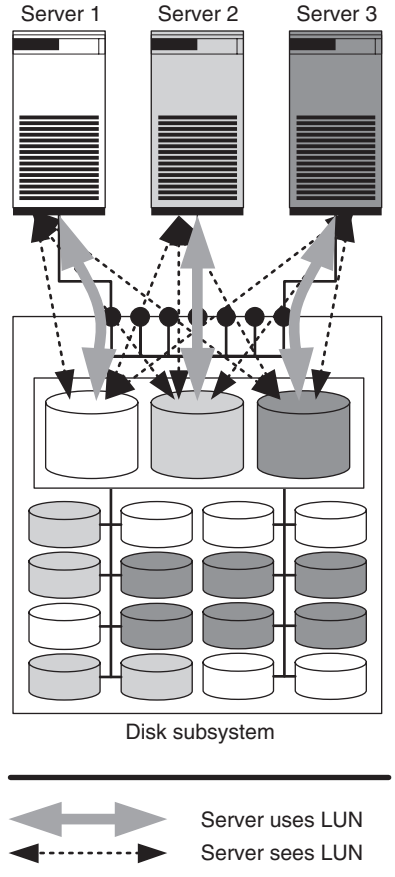

Figure 2.26 Chaos: Each server works to its own virtual hard disk. Without LUN masking each server sees all hard disks. A configuration error on server 1 can destroy the data on the other two servers. The data is thus poorly protected.

Without LUN masking, therefore, the use of the hard disk must be very carefully configured in the operating systems of the participating servers. LUN masking brings order to this chaos by assigning the hard disks that are externally visible to servers. As a result, it limits the visibility of exported disks within the disk subsystem. Figure 2.27 shows how LUN masking brings order to the chaos of Figure 2.26. Each server now sees only the hard disks that it actually requires. LUN masking thus acts as a filter between the exported hard disks and the accessing servers.

It is now no longer possible to destroy data that belongs to applications that run on another server. Configuration errors are still possible, but the consequences are no longer so devastating. Furthermore, configuration errors can now be more quickly traced since

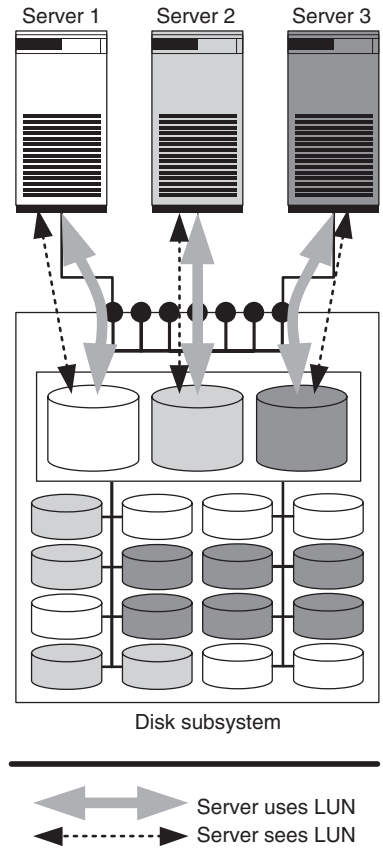

Figure 2.27 Order: each server works to its own virtual hard disk. With LUN masking, each server sees only its own hard disks. A configuration error on server 1 can no longer destroy the data of the two other servers. The data is now protected.

the information is bundled within the disk subsystem instead of being distributed over all servers.

We differentiate between port-based LUN masking and server-based LUN masking. Port-based LUN masking is the 'poor man's LUN masking', it is found primarily in low-end disk subsystems. In port-based LUN masking the filter only works using the granularity of a port. This means that all servers connected to the disk subsystem via the same port see the same disks.

Server-based LUN masking offers more flexibility. In this approach every server sees only the hard disks assigned to it, regardless of which port it is connected via or which other servers are connected via the same port.

2.8 AVAILABILITY OF DISK SUBSYSTEMS

Disk subsystems are assembled from standard components, which have a limited fault-tolerance. In this chapter we have shown how these standard components are combined in order to achieve a level of fault-tolerance for the entire disk subsystem that lies significantly above the fault-tolerance of the individual components. Today, disk subsystems can be constructed so that they can withstand the failure of any component without data being lost or becoming inaccessible. We can also say that such disk subsystems have no 'single point of failure'.

The following list describes the individual measures that can be taken to increase the availability of data:

- The data is distributed over several hard disks using RAID processes and supplemented by further data for error correction. After the failure of a physical hard disk, the data of the defective hard disk can be reconstructed from the remaining data and the additional data.
- Individual hard disks store the data using the so-called Hamming code. The Hamming code allows data to be correctly restored even if individual bits are changed on the hard disk. Self-diagnosis functions in the disk controller continuously monitor the rate of bit errors and the physical variables (e.g., temperature, spindle vibration). In the event of an increase in the error rate, hard disks can be replaced before data is lost.
- Each internal physical hard disk can be connected to the controller via two internal I/O channels. If one of the two channels fails, the other can still be used.
- The controller in the disk subsystem can be realised by several controller instances. If one of the controller instances fails, one of the remaining instances takes over the tasks of the defective instance.
- Other auxiliary components such as power supplies, batteries and fans can often be duplicated so that the failure of one of the components is unimportant. When connecting the power supply it should be ensured that the various power cables are at least connected through various fuses. Ideally, the individual power cables would be supplied via different external power networks; however, in practice this is seldom realisable.
- Server and disk subsystem are connected together via several I/O channels. If one of the channels fails, the remaining ones can still be used.
- Instant copies can be used to protect against logical errors. For example, it would be possible to create an instant copy of a database every hour. If a table is 'accidentally' deleted, then the database could revert to the last instant copy in which the database is still complete.
- Remote mirroring protects against physical damage. If, for whatever reason, the original data can no longer be accessed, operation can continue using the data copy that was generated using remote mirroring.

- Consistency groups and write-order consistency synchronise the copying of multiple virtual hard disks. This means that instant copy and remote mirroring can even guarantee the consistency of the copies if the data spans multiple virtual hard disks or even multiple disk subsystems.
- LUN masking limits the visibility of virtual hard disks. This prevents data being changed or deleted unintentionally by other servers.

This list shows that disk subsystems can guarantee the availability of data to a very high degree. Despite everything it is in practice sometimes necessary to shut down and switch off a disk subsystem. In such cases, it can be very tiresome to co-ordinate all project groups to a common maintenance window, especially if these are distributed over different time zones.

Further important factors for the availability of an entire IT system are the availability of the applications or the application server itself and the availability of the connection between application servers and disk subsystems. Chapter 6 shows how multipathing can improve the connection between servers and storage systems and how clustering can increase the fault-tolerance of applications.

2.9 SUMMARY

Large disk subsystems have a storage capacity of up to a petabyte that is often shared by several servers. The administration of a few large disk subsystems that are used by several servers is more flexible and cheaper than the administration of many individual disks or many small disk stacks. Large disk subsystems are assembled from standard components such as disks, RAM and CPU. Skilful combining of standard components and additional software can make the disk subsystem as a whole significantly more high-performance and more fault-tolerant than its individual components.

A server connected to a disk subsystem only sees those physical and virtual hard disks that the disk subsystem exports via the connection ports and makes available to it by LUN masking. The internal structure is completely hidden to the server. The controller is the control centre of the disk subsystem. The sole advantage of disk subsystems without controllers (JBODs) is that they are easier to handle than the corresponding number of separate hard disks without a common enclosure. RAID controllers offer clear advantages over JBODs. They combine several physical hard disks to form a virtual hard disk that can perform significantly faster, is more fault-tolerant and is larger than an individual physical hard disk. In addition to RAID, intelligent controllers realise the copying services instant copy and remote mirroring, consistency groups for copying services and LUN masking. Instant copy enables large databases of several terabytes to be virtually copied in a few seconds within a disk subsystem. Remote mirroring mirrors the data of a disk subsystem to a second disk subsystem without the need for server resources. Consistency groups synchronise the simultaneous copying of multiple related virtual hard disks.

Disk subsystems can thus take on many tasks that were previously performed within the operating system. As a result, more and more functions are being moved from the operating system to the storage system, meaning that intelligent storage systems are moving to the centre of the IT architecture (storage-centric IT architecture). Storage systems and servers are connected together by means of block-oriented I/O techniques such as SCSI, Fibre Channel, iSCSI and FCoE. These techniques are handled in detail in the next chapter. In the second part of the book we will be explaining the use of instant copy, remote mirroring and consistency groups for backup (Chapter 7) and for business continuity (Chapter 9). It is particularly clear from the discussion on business continuity that these three techniques can be combined together in different ways to optimise protection from small and large outages.