

Configuring EMC CLARiiON for SAS Business Intelligence Platforms

Applied Technology

Abstract

Deploying SAS applications optimally with data stored on EMC[®] CLARiiON[®] systems requires a close collaboration between SAS application programmers, database administrators, and storage administrators. Understanding how SAS typically drives I/O from the server environment and how CLARiiON works internally in servicing different types of I/O patterns is the key to enabling that effective collaboration. This white paper provides the common ground knowledge to help achieve that objective.

June 2007

Copyright © 2007 EMC Corporation. All rights reserved.

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED “AS IS.” EMC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

For the most up-to-date listing of EMC product names, see EMC Corporation Trademarks on EMC.com

All other trademarks used herein are the property of their respective owners.

Part Number H2840

Table of Contents

Executive summary	4
Introduction	4
Audience	4
Typical I/O pattern expected for SAS deployments	4
I/O pattern of SAS WORK files	5
I/O pattern and CLARiiON storage system	5
Overview of CLARiiON system architecture	6
Random data access from CLARiiON disk drives	7
CLARiiON storage processor data caching	7
LUN cache setting.....	9
LUN prefetching	9
RAID protection type selection.....	10
Data striping on CLARiiON LUNs	12
Stripe element boundary alignment	12
Host Logical Volume Managers and CLARiiON metaLUNs	14
Drive type and performance	15
CLARiiON LUN migration	15
Performance monitoring for CLARiiON	15
Navisphere Quality of Service Manager.....	17
Conclusion	18
References	18

Executive summary

This white paper covers guidelines for configuring and deploying EMC® CLARiiON® storage systems in typical environments deploying SAS Business Intelligence data analysis applications. Deployments vary in how data is accessed and modified, so no single formula is guaranteed to work for all environments. The goal of this paper is to explain how certain configuration choices in the CLARiiON affect different I/O patterns directed against the system. By understanding the actual dominant I/O pattern in a specific SAS deployment environment, SAS system administrators will be able to collaborate with their storage administrators to properly configure the CLARiiON system layout to the best advantage for their particular deployment.

Introduction

SAS deployments use data stored as OS files within the deployment server platform. For example, data may be stored in UFS files under Solaris, ext3 files under Linux, and NTFS files under Windows. Other options such as NFS files in VERITAS file systems (Vxfs) are also frequently found in customer deployments.

Different file systems under different operating systems behave differently. And each file system has its own configuration options that may affect how it behaves exactly under that OS. These configuration options frequently affect how physical I/O is driven from the server down to the storage subsystem.

So, configuring the storage subsystem to the best advantage in supporting a SAS deployment cannot be done without understanding how the particular SAS workload is driving I/O against the file system, and in turn, how the file system is pushing I/O down to the storage system.

It is beyond the scope of this paper to cover the specifics of different file system configurations and usage for different OS platforms. Certain assumptions will be made to the typically anticipated I/O pattern likely to be driven from the host running the SAS deployment to the underlying storage system. Based on those assumptions, configuration considerations specific to the CLARiiON system will be discussed.

For readers who are already familiar with the EMC CLARiiON architecture and the details of system internals such as RAID type support, caching configurations, and so on, the “Conclusion” section contains the key recommendations made for the purpose of this paper. These readers may refer back to other sections for more specific details as needed.

Audience

This white paper is intended for SAS system and application administrators to collaborate more effectively with their storage administrators for planning the layout of CLARiiON storage to accommodate the different SAS deployments.

Typical I/O pattern expected for SAS deployments

SAS workloads tend to be oriented toward concurrent data reads from the different files holding the data being analyzed. In typical deployments, with the power of the servers and the need for supporting enterprise mission-critical work, it is increasingly rare that the data is not being accessed by multiple user sessions processing concurrently.

So, while a typical single user session may be frequently making a pass of a large segment of sequential data over the stored data, the typical multi-user environments tend to be driving a mix of large sequential and small random reads predominantly to the I/O subsystem in general.

How that mix of large and small random reads maps through the OS file system is very important.

Typically, an OS file system organizes file data in file system blocks or pages. A block or page is the finest unit of data that the file system will perform I/O on. A common block or page size for many file systems is 4 KB. That means whenever an application asks for data from a file, and the file system has to issue an I/O request to the storage system, at least 4 KB worth of data bits would have to be transferred between server and storage.

On many occasions, when a large chunk of data is needed, multiple data pages may be passed in a single server to storage request.

OS file systems frequently either have default options, or can be configured, to cache recently accessed and manipulated file data on server memory as part of the OS file system caching buffers. In these cases, some file systems may issue a single server to storage request to manipulate a chunk of file pages in a single request. These are commonly referred to as file system buffer gather/scatter page requests. Consecutive file pages with modified data may be “gathered” up from the file system buffers and sent down to the storage system as a single large write request. Alternatively, a single large chunk of data may be read as a single transfer from storage to server, then “scattered” into the different file system page buffers as separate file pages ready for access by the application.

Most of the data analysis performed by SAS involves multiple rows of data stored in multiple consecutive OS file pages on a particular OS file. In those cases, it is very likely that the SAS engine will be driving a series of host I/O reads that are in multiples of OS files, such as 128 KB reads, to try to prefetch the data, getting data read up from storage in place, and ready for the processing threads to grind full speed ahead with the data already pulled into the user process memory buffers. On other occasions, there may be queries that are leveraging preaggregated data, such as data in data analytics dimension clubs.

Also, complex queries involving joins of multiple data tables may be leveraging indices created for the join columns, or using the indices to facilitate filtering of the result sets to be returned in response to some selection qualification criteria in the query predicates.

I/O pattern of SAS WORK files

SAS WORK files’ I/O is frequently a key performance factor to any SAS deployments. Often, chunks of relevant user data are pulled from the database files, extracted, and written into the work files, where most of the data analytics processing are performed against. Data reads and writes from the standpoint of each SAS user process going into the work files are primarily sequential. However, when many user processes are running concurrently, the WORK file system will be seeing many concurrent large reads/writes into different part of the file system concurrently. From a storage system viewpoint, the I/O pattern would therefore appear as mostly large random reads/writes, since requests would frequently be switching back and forth between parts of the file system holding the different WORK files manipulated by different concurrent users.

I/O pattern and CLARiiON storage system

With any storage subsystem, there is almost never a single right answer as to how to configure the storage system. Even when one can qualify the usage of stored data as only for supporting SAS deployments, there are still enough variations among different types of analysis being run against different types of stored data to significantly affect the actual I/O pattern. But in general, it would be safe to assume that the most common I/O pattern that would likely give the best return for the effort would be for a mixed large (128 KB) and small (single SAS data page size, commonly 4 KB) random read load.

We will also assume that writes from SAS environments tend to include typically large (128 KB or higher) data writes, or single 4 KB page rewrites, with large “batch” writes being the more common of the two.

With those I/O pattern assumptions in mind, we can then match those patterns to the CLARiiON architecture, and provide some general CLARiiON configuration guidelines to meet those assumptions.

Overview of CLARiiON system architecture

Figure 1 shows a CLARiiON CX3-80 system architecture.

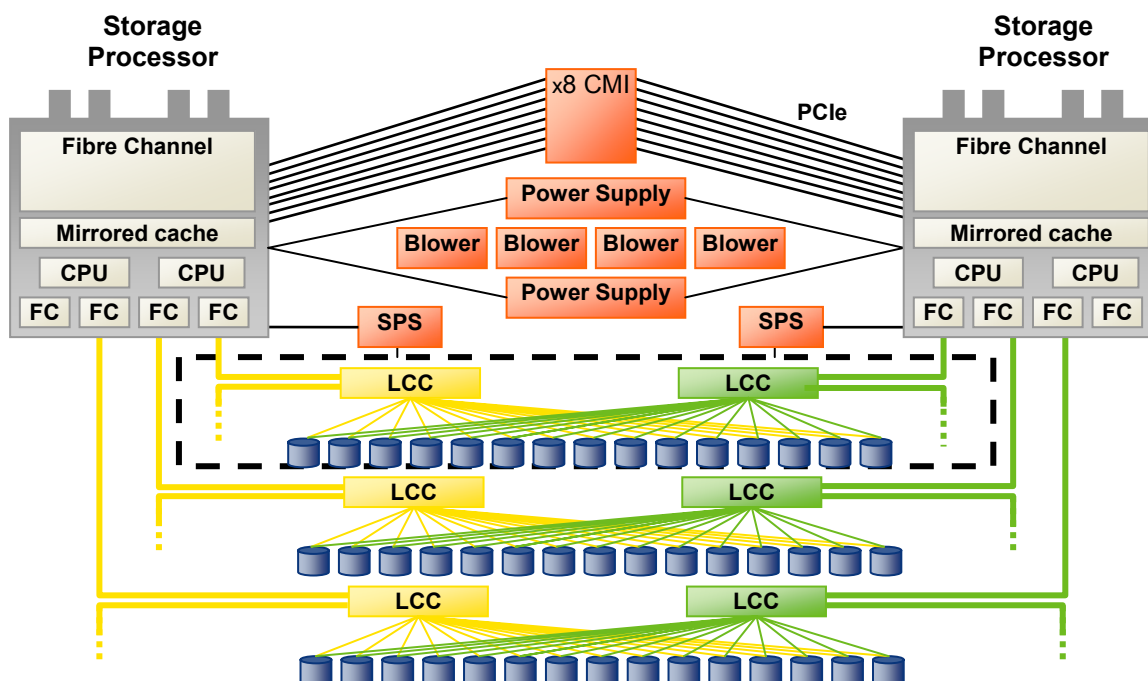


Figure 1. EMC CLARiiON CX3-80 system architecture

A CLARiiON storage system consists of many modules. The two key modules are the two storage processing units, typically referred to as Storage Processor – A (SP-A) and Storage Processor – B (SP-B). These processing units are supported by all redundant power supply units, battery backup units, and cooling. One or more disk enclosures, each capable of holding up to 15 disks, can be connected to the storage processors. Both storage processors will be connected to all the disk enclosures.

The disks in the different enclosures are managed by the processors. Different sets of disk drives in the different enclosures are grouped together to form a RAID protected data group. From each of the RAID group, one or more host usage storage objects, called logical units, identified as logical unit numbers (LUNs) can be sliced out of each RAID group. The LUNs can then be managed and exposed to different servers for use as SCSI disk I/O protocol conforming disk objects.

For SAS environments, the LUNs projected to servers as OS disks are typically formatted using OS disk formatting utilities. OS file systems are then created using OS file system creation utilities (for example, UNIX MKFS, Windows file system creation under DISK MANAGER), mounted, and used to hold SAS files. SAS file I/O then gets mapped by the OS file system layer to block data access I/O against the LUNs.

All I/O requests from the servers against the LUNs are processed through the storage processors. The processors manage the distribution of the request loads, handle RAID protection for drive failures, and

support other access optimization to the stored data such as data buffering and caching in the processors' memory.

The storage processors are capable of taking over the full request load for its respective peer processor in the event of a processor hardware failure, as each has logical and physical disk path access to all disks in our enclosures.

Hence the CLARiiON system is architected to support highly available access even in the event of certain unanticipated hardware failure, and the powerful processor power allows the system to scale effectively with each increment of physical disk storage enclosure and disk.

Random data access from CLARiiON disk drives

The CLARiiON disk enclosures, called disk-array enclosures (DAEs), support a variety of industry standard commodity disk drivers, including Fibre Channel high-performance drives such as the 146 GB 10k or 15k rpm drives from Seagate, Maxtor or Hitachi, as well as the more recent Serial ATA technology drives that are typically slower, but with much higher density, such as the 7200 rpm SATA drives with 500 GB.

The mechanics of the drives mandates a certain level of head seek latency whenever there is a random series of data access against data stored on the drive. For the bigger and slower drives, such as the SATA drives, the head seek latency is correspondingly higher. So, for a predominantly random drive access pattern, the ability to pull data in and out of a particular drive is limited by the practical amount of head seeks possible within a unit of time. The *EMC CLARiiON Best Practices for Fibre Channel Storage* white paper on EMC.com has very complete discussions on the different drive types and RAID configurations used.

As the head seek latency limits the number of distinct pieces of stored data within a single physical disk drive to be accessed and manipulated, it is important to keep in mind that disks with larger raw storage capacity do not seek any faster than disks with lower capacity. For data that requires high random access performance, it is preferable to store the data on smaller disks, and spread the data over more physical drives.

Because of the random disk seek overhead, the effective volume of data accessed increases with larger average request sizes. This is where CLARiiON storage processor caching usually become relevant.

CLARiiON storage processor data caching

The CLARiiON storage processors can be configured to dedicate a portion of their onboard memory for data caching purposes. Depending on the particular storage system, each storage processor can be populated with varying amount of memory, for example, up to 4 GB per processor on the top-end CX3-80 or CX700 class systems.

Some amount of the SP memory is assigned for running the storage system kernel software, plus optional data replication software is supported on these CLARiiON systems. The rest of the SP memory can be partitioned into segments of memory for data read and write caching.

When partitioning SP memory to support write caching, the same amount of write cache must be allocated from each SP. The SPs always mirror write cached data between the two segments of allocated SP memory for write caching. This ensures that in the event of a SP hardware failure, no writes that have been acknowledged to servers as committed would ever be lost.

Hence, if we allocate 2 GB worth of SP memory from each SP to support write caching, there will never be more than 2 GB worth of write cached data in total between the two SPs.

In contrast, if 1 GB of SP memory is configured on each SP to be for read caching, each SP can have a completely separate segment of data cached in the memory allocated for read caching. Read caching memory never holds any modified data.

Each I/O request incoming from the servers through the SP against the different LUNs will be checked against data cached in both the read and write cache memory areas. If the required data can be found in the SP cache memory, that copy is used, thereby avoiding having to dispatch the host-side request to the physical disk drive.

Given that SAS data sets typically are getting into multi-gigabyte ranges, and SAS manages its own data caching on the server memory that SAS is running, for pure random data reads, any host-side reads pushed down from the server by SAS to the storage system are not often getting a cache hit from the limited few gigabytes' worth of SP memory that can be configured. (For example, on a CX3-80, the most cache memory that can be configured is about 3-plus GB per SP. If SAS is managing a server-side data cache of 16 GB, and the total data set is 500 GB, the probability of getting a cached data random hit is only $(500-16)/500 \times 6/500 = 1.6\%$.) Configuring most of the available SP memory for read caching of SAS data frequently will be a waste of the SP cache memory resource. It is preferable to dedicate more of the available memory to write caching instead.

The real value of write cache is when there are bursts of changed data writes coming down from the server, there is enough SP memory to accommodate that burst. When the server pushes down a series of writes, and those can all be just memory moved into write cache, the server will see request responsiveness in milliseconds of low single digits. As long as the write burst does not overrun the available memory space in the write cache, all the writes coming down can be serviced quickly as far as the server is concerned. The SP can take much longer to physically write all the write cached data back to the physical drives, leveraging the time window between write bursts to get those writebacks to disks completed.

The read cache is typically most useful for supporting prefetches of data when the read requests arriving from the server exhibit a particular sequential pattern. When the SP receives a series of read requests against a LUN that are for consecutive segments of data stored on the drives, it can dispatch a single large request to the drive to "prefetch" additional data, in anticipation of the data being needed for subsequent requests from the host. By dispatching a single large request, we minimize the amount of head reseeking that is needed to support the requests individually. Subsequent read requests that can be satisfied from the data prefetched will be getting a relatively low response time.

Imagine a typical 10k rpm Fibre Channel drive that can typically support about 160 random head seeks and data retrievals per second. If we position to a new head position and read up to 8 KB of data, we can pull up 1.2 MB ($160 \times 8 \text{ KB} = 1.2 \text{ MB}$) worth of data per second from that drive.

If instead we pull out 256 KB on each request, we will be pulling out 40 MB/s ($256 \times 160 = 40 \text{ MB/s}$) of the same disk spindle.

So, when it comes to random reading data from the drives, the bigger the single request size, typically the more data can be read from the same physical drive.

Similarly, when we are doing writes, if we take every 8 KB write coming down from the server and dispatch that directly to a drive, we will be able to write only about 1 MB/s worth of data to the drive. However, if the 8 KB pieces of data written can be combined (coalesced) in the write cache into a single piece such as a 256 KB piece and written to the drive as one single write request, we will be able to achieve a much higher write per drive rate overall.

The *EMC CLARiiON Best Practices for Fibre Channel Storage* white paper on EMC.com provides further information and guidelines on configuring read and write cache sizes on a CLARiiON system.

LUN cache setting

Data caching is achieved by the storage processors, leveraging memory assigned as data caching memory. Data caching can be optionally enabled/disabled on a LUN-by-LUN basis. LUN caching is meaningful only if the SP is configured with data caching memory, and if SP caching is enabled.

However, a LUN can be configured such that requests against that particular LUN may or may not be allowed to take advantage of SP caching. For LUNs with very low levels of activities, or with less stringent request responsiveness, it is often the right strategy to configure them without caching to give the higher priority LUNs better overall service request levels.

SAS WORK files typically have one of the highest read/write activity levels in most SAS workloads. Also, data written to WORK is frequently reread in a subsequent stage of most SAS query operations. So, SAS WORK files typically can benefit from both read and write caching.

As read/write LUN caching can be dynamically enabled/disabled for each LUN, it is possible (and logical) to selectively enable write caching on DATA file LUNs when doing batch data loads into the DATA files, and disable write caching for those LUNs after the data loading is completed.

LUN prefetching

Storage processor predictable data prefetching in response to sequential access request patterns on LUNs can also be dynamically configured. By default, four times the typical data read size would be prefetched for a LUN when prefetching kicks in. For example, if the host requests have exhibited a sequential pattern for consecutive 4 KB DATA “OS file pages,” the SP will be dispatching a prefetch request of up to 16 KB to the LUN, assuming that LUN read caching is enabled for the LUN, and that SP cache memory is available.

This form of prefetching is called VARIABLE prefetch. The amount of data prefetched is dependent on the typical sequential data read segment sizes detected. The actual amount of data prefetched will be based on the typical read size, multiplied by a prefetch multiplier, which is configurable (default is four times).

There is also the option to control the amount of prefetch as a fixed amount. The third configuration option is to select the prefetch option of NONE, which means disabling prefetch for this particular case. All the options can be dynamically adjusted for any LUN, even when I/O is going continually against the LUNs. The change will be in effect as soon as the SP can properly adjust the prefetch behavior (that is, In-flight prefetch activities will not be disrupted, but further prefetch on the LUN will be adjusted based on the configuration change admitted).

LUN prefetches should typically be left alone unless there is a strong understanding that prefetches would hurt rather than help. This may be the case when most of the read accesses to a LUN are primarily expected to be random, and any adjacent segment access is mostly by coincidence. (The setting has the potential of triggering prefetches for data that may never be required). Alternatively, if there are many LUNs that may have clustered or sequential read patterns, and if prefetches are triggered for all the LUNs, they would be competing with each other, resulting in prefetched data from one LUN being displaced by prefetches done on other LUNs before the data can be leveraged to expedite the subsequent sequential access on the first LUN.

When such situations occur, it would then be useful to enable data prefetching only for the LUNs that have the highest data access efficiency requirements, and that tend to be accessed most frequently in true LUN sequential data pattern. The mechanism to determine which LUNs can benefit from prefetching, and whether data prefetching on particular LUNs may have become counterproductive, will be discussed in the “Performance monitoring for CLARiiON” section.

RAID protection type selection

The CLARiiON system currently supports the following RAID protection types:

- RAID 1
- RAID 3
- RAID 5
- RAID 1/0

These are true RAID protection types in the sense that some amount of raw storage capacity is expended to stored data in some redundant form, such that in the event of a hard disk drive failure, the lost data can be dynamically re-created by the storage processors leveraging the redundant data, and that server data access can be sustained without incurring a service outage.

RAID 3 and 5 are called parity RAID types, where the parity of certain segments of data stored on multiple drives are calculated and stored on an additional drive. A notation such as 4+P R5 would indicate that an extra drive worth of space is used up to hold the calculated parity for the data in the other four drives. In this example of 4+P R5, basically for every 100 GB of actual data, the total needed storage would be 125 GB, a 25 percent storage capacity overhead.

RAID 1 and RAID 1/0 types are commonly referred to as mirrored RAID. For these types, the actual data stored on one drive is bit-for-bit mirrored to another drive. The total used storage capacity compared to the actual user data stored is exactly 100 percent extra (that is, 200 GB of physical storage space required to store 100 GB of user data).

Parity RAID makes more total drive physical space usable for storing data, as less space is required for data redundancy, compared to mirrored RAID types. Parity RAID types have a much lower sustained random small data write performance level, since every data write would require triggering an associated parity data adjustment. To adjust the corresponding parity data, it may be necessary to first read in the parity data as well as the old version of the affected data, compare the old version against the modified version, compute the bit differences, and apply those to the parity data before the modified data can be properly destaged back to the disk area that it belongs. The same data change in a mirrored RAID type would just be destaged back directly to the corresponding disk areas on both spindles that form the mirrored pair.

For data LUNs that typically involve a large chunk of data modifications, or a lot of sequentially stored data modifications, such as batch data loads into tables, backup areas, transaction log archival areas, and so on, parity RAID write performance may be acceptable.

The parity data segment is typically calculated based on what is called a stripe element worth of data stored on each of the data spindles. For example, on a 4+P R5 LUN on CLARiiON, with a default stripe element size of 64 KB, the parity stripe will be 64 KB, representing the parity aggregate of the four different 64 KB data pieces stored on each of the other four spindles (Figure 2).

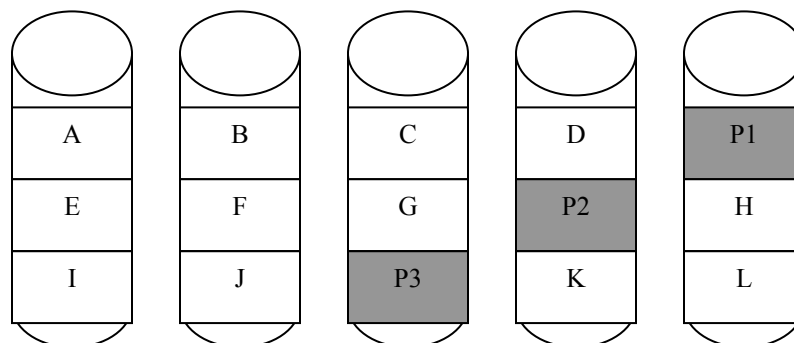


Figure 2. 4+P R5 data layout on drives

Each data stripe element (denoted by the letters, A, B, C, and so on) would be 64 KB worth of stored data. The parity stripes (denoted by P1, P2 and P3, and so on) would also be 64 KB. P1, for example, would be the bit parity of all data stored in A, B, C, and D.

If we make a change to A (for example, from A to A'), we would then have to read back in P1, figure out which bits in A have been changed, and make the adjustment to P1 accordingly.

However, if our typical data change is 256 KB, such that when SAS data is changed from the host in 256 KB chunks, then there is a chance that all of A, B, C, and D would now be changed all at the same time to say, A', B', C', and D'. In that case, the new parity value of P1' can be directly computed from the new data, and there would be no additional overhead required to reread the old value of P1 and the separate values of A, B, C, and D.

Even if our host-side data change is done typically in smaller sizes, such as 64 KB, if the data change pattern is sequential, then the first write from host may change A to A', the second from B to B', and so on. Since the CLARiiON processors can cache host-side writes in memory before destaging the new data to disk, this type of sequential write pattern allows CLARiiON to wait until a full stripe (all the four pieces, A', B', C', and D') worth of changes have been gathered up in the write cache. At that point, the new value of P1' can be directly computed, and then all the four pieces of 64 KB sequential writes will be done as a single 5 x 64 KB spindle destage of all five spindles in parallel.

Hence, parity RAID is frequently a good alternative if the typical random write rate is low, or when the writes from hosts are typically large, meaning in sizes equal to 256 KB or more.

Parity RAID performance for read is typically not an issue, except in the event that one of the spindles actually fail. In that case, the data that logically is stored in the failed spindle has to be derived from the data in the remaining four spindles, and so user read service performance would degrade quite noticeably until the failed physical drive has been replaced, and its logical content completely rebuilt from the data in the other spindles.

For mirrored RAID types, if a spindle should fail, the needed data is just read up from the mirrored spindle.

SAS data writes tends to be primarily sequential. Host file systems frequently buffer us pieces of data written sequentially, and dispatches them as a single large write data chunk to the storage system. The chunks as delivered to the storage system may further be storage cache captured and coalesced in the storage write cache. Consequently, full stripe data writes from the CLARiiON cache to back-end physical drives can readily be done in full stripes. As such, parity RAID, such as 4+P RAID 5, is usually quite satisfactory to be used for LUNs to house SAS data, as well as to support the SAS WORK file systems.

Mirrored RAID incurs a higher storage cost (100 percent more storage capacity deployed to support redundantly stored data for protection). If a particular SAS deployment is performance and mission-critical, and unexpected performance degradation due to physical drive failure is not acceptable, mirrored RAID type protection should be considered.

SAS WORK files can be configured to use any file system(s) available under the OS instance where the SAS workloads are running. Hence, if the performance of a particular parity RAID-based WORK file system has been compromised due to an unexpected drive failure (and while the drive content is still being dynamically rebuilt by the storage system), it is often possible from an operational standpoint to have all subsequently initiated SAS user jobs reconfigured to use WORK files from a *different* file system. Hence, it is preferable in general to try to leverage parity RAID LUNs to hold WORK file systems for more effective space utilization.

Data striping on CLARiiON LUNs

As in Figure 2, consecutive pieces of stored 64 KB data, such as the pieces A, B, C, and D, are alternately spread across the four disk spindles, and protected by the parity data on the fifth spindle (P1). The piece of 64 KB data is referred to as a stripe element.

If we gather 256 KB worth of data in our write cache, that entire segment of 256 KB worth of new data, plus their corresponding parity, is sent out in parallel to the five spindles. So, the total elapsed time to complete writing out the entire 256 KB worth of data is significantly reduced.

Similarly, by reading all four spindles holding data in parallel, we will be able to get a chunk of 256 KB typically in the time it would take to get 64 KB out of one spindle.

When a CLARiiON LUN with data striping is created (such as a 4+1 R5, or a 3+3 R1/0), the default stripe element size is 64 KB. It is often tempting to tinker with adjusting the stripe element size. For example, if the host application is known to frequently push 1 MB write I/O down to the CLARiiON, and read data back in 1 MB chunks, it may seem logical to change the stripe element size to 256 KB, such that we will be able to always access each of the spindles with the fewest number of requests.

There are a number of reasons why this is not necessarily the best approach. For example, most of the commodity disk drives used in the storage system have a certain amount of intelligence to queue and optimize the servicing of multiple requests at the drive head level. So, a series of four read requests for consecutive 64 KB chunks of data stored on a particular drive can actually be serviced almost as efficiently as a single read request for 256 KB from that same drive. The drive head would optimize the disk head seek position, servicing all requests in the queue with the minimal amount of head seek movement required. Truly consecutive requests are serviced in order without incurring disk head seeks, thereby eliminating the majority of the time required for each disk read.

In most practical SAS deployment scenarios, even if a lot of the typical I/Os tend to be large and sequential in nature for a particular database and application, there are always some small random I/Os involved, such as index page reads. If the drive spindles are constantly being monopolized driving 256 KB I/O, these single 4/8 KB page I/Os may end up being stuck behind the big I/Os, and would suffer far more erratic service responsiveness. The default stripe element size helps to minimize the likelihood of these small I/Os being subjected to long and unpredictable service delays, and delivers a smoother total I/O balance between all the different I/O size mixes.

Finally, because of the CLARiiON caching intelligence, even though our stripe element sizes may just be 64 KB, if the data access pattern warrants issuing a bigger single I/O to each spindle so that data can be manipulated more efficiently, those will often be implicitly triggered by the CLARiiON caching heuristics. For example, when the CLARiiON software detects a LUN serial data access pattern, and LUN prefetches are enabled, the SP will be issuing a bigger than 64 KB request down to the spindles to try to “prefetch” data in anticipation of needing it. The prefetch size determination is based on heuristics of data access and use pattern, and the CLARiiON software would adjust the size as needed to the optimal size, regardless of how the stripe element size was configured.

So, in general, it is CLARiiON performance engineering’s recommendation that the default stripe element be left unchanged.

Stripe element boundary alignment

A CLARiiON LUN appears to the host as an OS disk device. Typically, to use the disk device, it has to be formatted with disk device partitions. Then OS file systems are created in one or some of the disk partitions formatted. For example, under Windows, when Disk Manager discovers a new disk device, the system administrator is prompted to perform a disk formatting function on the new disk. Then, in each of the new disk partitions formatted, a Windows file system, such as an NTFS file system, may be created,

and assigned a drive letter. Applications then read and write to the different drives. Similarly, on UNIX systems such as Solaris 10, the disk device such as `/dev/dsk/c1t0d0` is formatted with different partitions, and the file utility command `mkfs` is used to create new file systems on the different partitions. The separate file systems are then mounted and used by applications for doing OS file I/O.

Imagine that we have created a striped LUN in CLARiiON as shown in Figure 3:

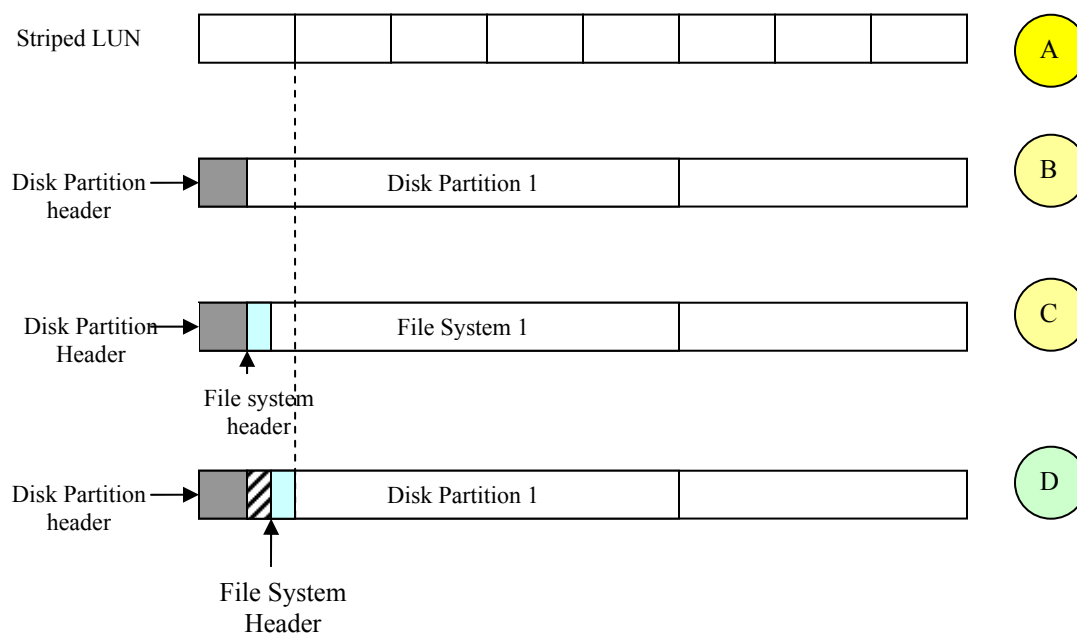


Figure 3. File system and disk partition alignment on stripe element boundaries

Starting from the striped LUN in (A), OS disk formatting would create a partition, with a disk partition header (B). Then, an OS file system is created on that disk partition. As shown in (C), as OS files are added to File System 1, the first file will have a piece sitting on the first stripe element (for example, 64 KB) of the LUN. So, if we try to do an I/O of 64 KB on this OS file, part of the data will end up going to the first stripe element, which belongs to one physical drive, and the rest to the second drive, making up this striped LUN.

This type of drive crossing is called a stripe crossing in CLARiiON terminology. Striping crossing results in less efficient dispatches of I/O requests from the CLARiiON storage processors to the back-end disk drives, thereby reducing I/O request service efficiency.

Most of the OS disk formatting utilities or tools have some provision for offsetting the space eaten up by the partition header. Suppose, as shown in (D), we manage to adjust the partition header offsets such that the boundary of the actual file in the OS file system begins just aligned to the LUN stripe boundary. Then all the file I/O of sizes 64 KB, or multiples of 64 KB, would have no (or minimum) number of times that LUN stripe boundaries are crossed. This delivers a higher overall data delivery rate, especially when the file data is accessed with typical sequential patterns.

Many file systems on different OSs have different, and frequently, variable size file system headers (depending on the number of files and directories, or folders created in the file system). So, it may not always be easy to figure out how much to offset or pad the partition header space to get the file data to stripe boundary align properly. In those cases, perhaps the best that can be achieved is to try to create the partition and pad out the partition header so that the file system itself can be stripe element boundary aligned.

How to pad out and adjust the partition header size depends on the particular disk formatting tool. For example, under Linux, the fdisk formatting utility has an expert mode option to support the partition header offset. Under Windows, a utility such as diskpart.exe may be used.

Stripe crossing and alignment are generally not an issue if most of the file I/Os are random or single file page type I/Os (such as 4 or 8 KB).

Host Logical Volume Managers and CLARiiON metaLUNs

Host Logical Volume Managers (LVMs) are commonly used with SAS deployments. Usable space from one or more storage LUNs is placed under the LVM to form a logical volume (LV). File systems are created out of the usable space in the LVs.

A key advantage of LVM is that when the file systems are full, more space can be allocated from the LV. When the LV itself is out of usable space, additional LUNs from the CLARiiON storage can be added to expand the LV itself. Also, the LVM will stripe and evenly distribute the file system data over the LUNs.

If the CLARiiON LUNs are themselves striping stored data over multiple disk spindles, the LV striping should be configured to be in multiples of the LUN full stripes. For example, if the CLARiiON LUN is a 4+P R5 with a default stripe element size of 64 KB, each full stripe of data stored on the LUN would be 4 x 64 KB = 256 KB. So, the LV stripe element size should be some multiples of 256 KB.

Also, if an LV is striping data over multiple CLARiiON LUNs, the CLARiiON LUNs should be from different CLARiiON RAID groups, occupying separate CLARiiON disks. Otherwise, when the LVM tries to push parallel requests against the different LUNs for the purpose of improving overall I/O push against the storage system, these parallel requests would end up spindle contending with each other in the underlying CLARiiON drive, thereby defeating the purpose of trying to push more I/O in parallel by the LVM.

The CLARiiON internally also supports the expansion of an existing LUN by concatenating or striping its data with another internal LUN. This is called the metaLUN expansion feature.

A metaLUN resulting from expanding a LUN currently used by host applications is just a LUN that appears to have more usable free space. If a host LUN with 100 GB is currently holding a file system that is using all the available space in the LUN, the expanded metaLUN may show an additional 100 GB worth of usable space, but the file system itself would still be just using the space from the 100 GB OS disk partition that was originally created. So, expanding the usable space in a LUN via the metaLUN feature is not the same as expanding an LV by the LVM on the host.

However, a LVM can use metaLUNs as its underlying LUNs. In this case, when the underlying LUN(s) is expanded, the LVM will be able to take advantage of the additional space, thereby expanding the LV striped over the underlying LUNs. The LV stripe element size in this case should be a multiple of the metaLUNs full stripe, which may already be some multiple of the underlying simple LUNs' full stripe. (For example, the metaLUN may have a stripe size that is four times the full stripe size of 256 KB of the underlying 4+P R5 LUN. In that case, the LV stripe element should be a multiple of 1 MB).

An LV striped over multiple CLARiiON LUNs allows the LVM to queue I/O requests against multiple "OS disks." A metaLUN is still just one "OS disk," and would be limited to supporting what can be maximally queued against that one device.

Drive type and performance

CLARiiON was the first midrange storage product supporting the simultaneous use of FC and ATA technology drives within the same system.

The FC drives are 10k and 15k rpm drives, with capacities of 73, 146, or 300 GB. The ATA technology drives typically have a higher per-drive capacity, such as 300, 500, or 750 GB per drive. They are typically 7200 rpm drives.

ATA drives are generally not the best choice for data that has a high frequency of random accesses, such as indices of databases. Data with a high frequency of random access, especially random access in relatively small I/O sizes such as 4 KB or 8 KB, should be stored on LUNs from FC drives.

ATA drive-based LUNs are good choices for backup copies of database files, or database files that are generally read and written in primarily sequential manner.

Data warehouse deployments frequently involve working with historical data that accumulates over time. As data ages, frequency of access tends to diminish. This is one common situation where FC and ATA LUNs may be used in tandem.

Current data with a high usage frequency is stored on FC LUNs for optimal access performance. Aged data can be moved to the slower ATA drives to lower storage costs.

CLARiiON LUN migration

Data residing on a particular LUN can be implicitly moved to another LUN of the same size inside the CLARiiON system with the LUN migration feature. While the data content is being implicitly migrated by the storage system, host applications continue to have full read/write access to the data content. After the migration is fully completed, the new LUN takes on the former identity of the original LUN, so there is no host-side application or operation procedure adjustment needed. Data previously stored on LUN 0 from a set of FC drives may be moved to a new LUN 0 on the ATA drives, without affecting any host applications.

Performance monitoring for CLARiiON

The CLARiiON system is managed via an out-of-band mechanism. The storage management and performance monitoring software run inside the storage processors. Storage administrators access this software either with a standard browser, or via commands issued from servers directed to the storage processors over the IP connection.

The performance monitoring facility is an optionally license-enabled software feature in the standard storage system administration software package. Figure 4 is an example of selecting the Navisphere® Analyzer option from the Navisphere Manager Browser window. Figure 5 shows the Performance Details screen in Navisphere Analyzer.

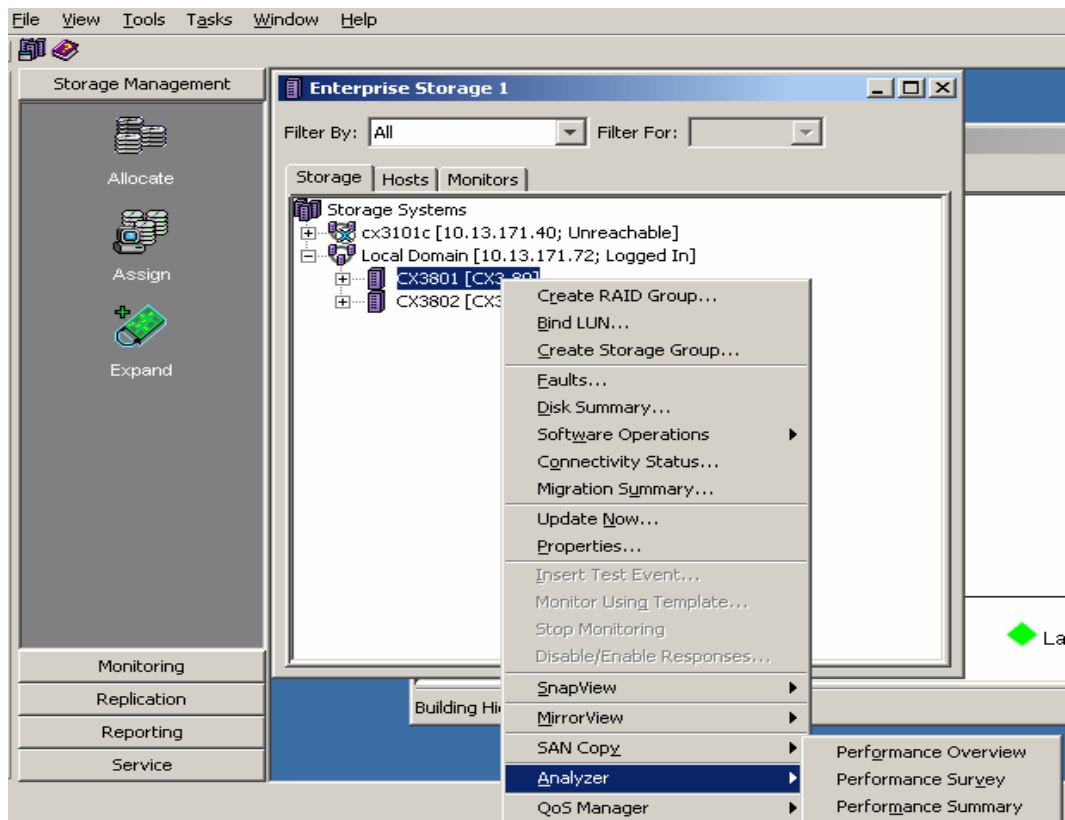


Figure 4. Invoking the Analyzer option from Navisphere Manager

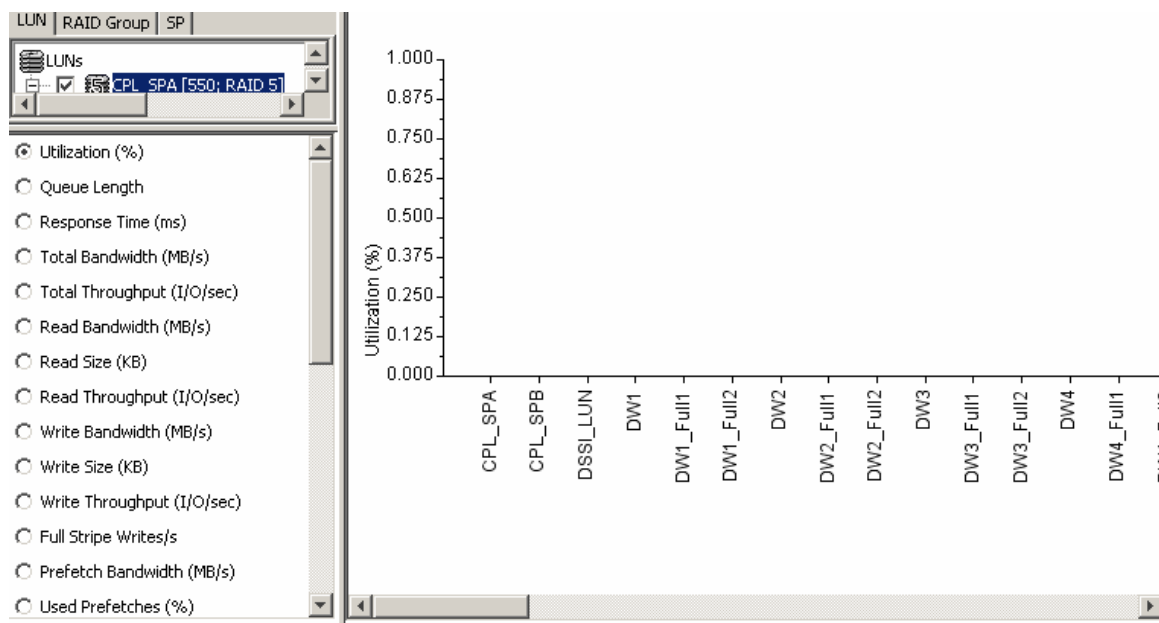


Figure 5. Performance Details option for Navisphere Analyzer

Navisphere Manager has the full set of capabilities for monitoring and managing the CLARiiON system, including creating and deleting LUNs, controlling which LUNs get projected for use to different servers, and others. This is primarily a storage administrator's tool. However, the storage system administrator can authorize access to SAS application database administrators to monitor how the different CLARiiON LUNs are being utilized, and to help them identify potential I/O performance issues.

There are many different levels of LUN access details, such as typical I/O sizes, I/O per seconds (on average), MB/s, amount of prefetches used, and others. By correlating the storage system's view of how I/O is behaving with host I/O views, such as with OS utilities like SAR, IOSTAT, and SAS's own logical I/O statistics, performance problems can be readily identified and isolated for resolution.

Navisphere Quality of Service Manager

The Navisphere Quality of Service Manager (NQM) option can also be invoked from the Navisphere Manager Browser window.

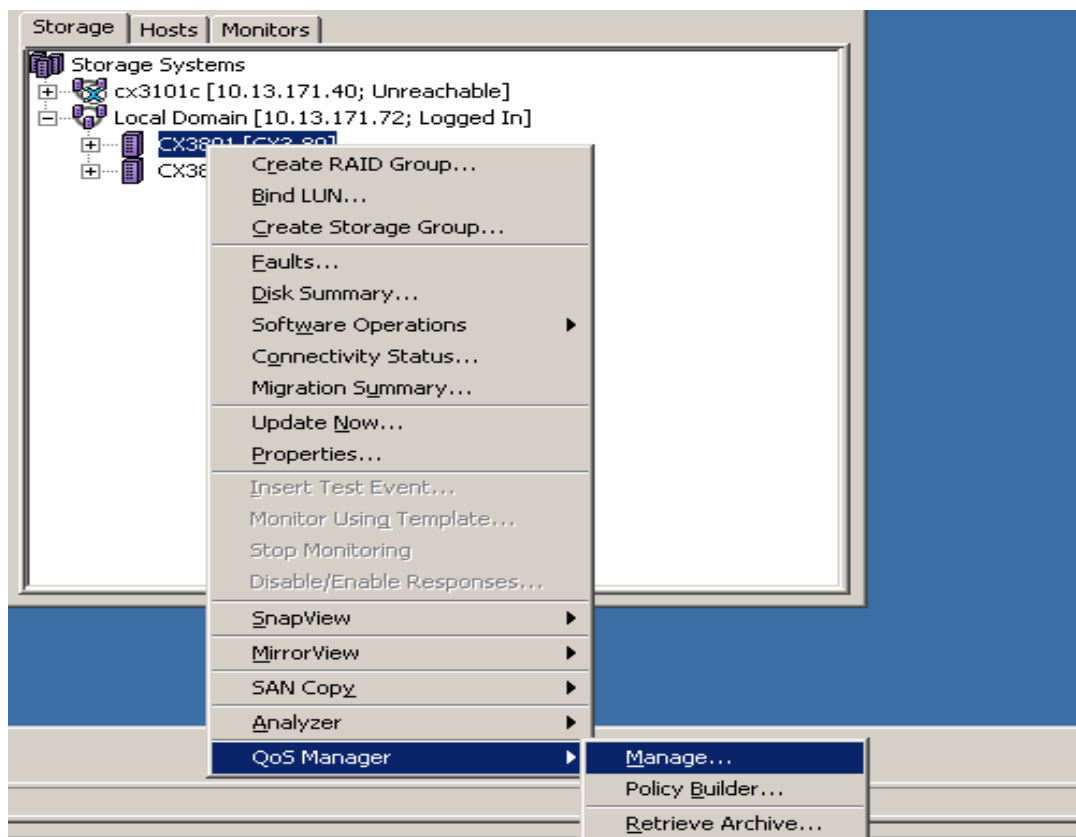


Figure 6. Invoking Navisphere QoS Manager

Ideally, if a LUN can be isolated to be the only LUN on a particular RAID group, spanning a distinct set of drives, it would be the easiest to monitor and manage I/O performance that the host application users are driving against a particular set of SAS files. In a large CLARiiON system used to consolidate and hold a variety of enterprise data, this is often impractical to achieve. When a LUN holding SAS data is sharing a RAID group and spindles with other enterprise data LUNs in the same RAID group, SAS system file I/O performance may vary due to interfering I/O that may be going against the other unrelated LUN activities.

NQM, a new CLARiiON feature from release 24, provides a policy-based mechanism for the storage administrator to set different levels of service goals for the different LUNs that may be sharing the storage system resources, including LUNs holding different types of enterprise data that may be sharing a set of disk drives. Priority may be set to ensure that the SAS data LUNs would be given preference for service levels, or may be limited to a particular service threshold (such as no more than 40 percent of storage resource).

Conclusion

The following are key recommendations to remember:

- Consider using parity RAID, such as 4+P R5, for LUNs used to hold SAS files.
- RAID 1/0 LUNs should be used for data files if I/O performance is critical and cannot afford to be compromised due to unexpected disk drive failures.
- Leverage CLARiiON's implicit LUN migration facility to move the contents of a parity RAID LUN to a mirrored RAID-based new LUN of equal size if write latency to files on the parity RAID LUN becomes a bottleneck for SAS.
- Ensure read/write caching is enabled for LUNs holding WORK files.
- Configure the CLARiiON write cache with size, and a high and low dirty mark, to ensure that the typical data loads and burst dirty data flushes from the host can be absorbed by the amount of write cache configured and enabled.
- Turn off read prefetching for data LUNs if the Navisphere Analyzer data indicated that prefetched data was not being used.
- Align the host OS file system and partitions with LUN striping boundaries.
- Consider leveraging the different disk types supported concurrently within the same CLARiiON system to achieve most cost-effective and best performance matched data placement.
- If LVMs are used, match the LV stripe size to full stripe size of the underlying LUNs.
- Use Navisphere Analyzer to examine I/O performance on the LUNs holding the SAS data and correlate those to the host I/O performance data.
- Leverage the CLARiiON NQM facility to address performance service prioritization among different enterprise data that may need to be sharing a common set of physical disk drives.

References

For more information, consult the *EMC CLARiiON Best Practices for Fibre Channel Storage* white paper on EMC.com.