

Deploying EMC CLARiiON CX4-960 for Data Warehouse/Decision Support System (DSS) Workloads

Best Practices Planning

Abstract

The EMC® CLARiiON® CX4-960 storage system offers performance, flexibility, and scalability for storage infrastructures supporting data warehouse/decision support systems at a totally new level unparalleled in the midrange storage industry. This white paper provides recommendations for how to deploy this new offering to your best advantage when implementing a mission-critical supporting DW/DSS deployment for the business enterprise.

October 2009

Copyright © 2008, 2009 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 H5548.2

Deploying EMC CLARiiON CX4-960
for Data Warehouse/Decision Support System (DSS) Workloads
Best Practices Planning

Table of Contents

Executive summary	4
Introduction	4
Audience	5
Terminology	5
Overview	6
CX4 UltraFlex I/O modules	7
Multiple storage tiers support	8
Configuration guides	8
Balancing the configuration	8
Balancing I/O activities among the two storage processors	9
Balancing data distributions among LUNs	9
Balancing traffic across the back-end buses	10
Balancing I/O traffic among the UltraFlex I/O modules	12
RAID choices for DW/DSS workloads	13
RAID stripe width and stripe element depth	14
Configuring cache memory for DW/DSS workloads	16
Write cache configuration considerations for DW data LUNs	16
Read cache configuration considerations for DW data LUNs	17
Leveraging storage cache for other data	18
Transaction log placement and caching considerations	19
Deploying Enterprise Flash Drives	20
Balancing EFDs on CX4 back-end buses	21
RAID choices for LUNs on EFDs	22
Cache settings for LUNs based on EFDs	23
High speed CX4-960 front-side access support	23
The 10 GigE iSCSI connection option	24
10 GigE iSCSI bandwidth expectation for read-intensive data warehouse deployments	25
Summary of recommendations	26
Conclusion	27
References	27

Executive summary

Mission-critical decision support (DSS) and business intelligence (BI) systems based on enterprise information organized and maintained in sophisticated and comprehensive data warehouses are rapidly becoming a standard and critical part of most modern enterprise IT functions. Demands on data warehouses grow at an alarming rate, posing severe challenges to the IT support teams. This is especially true with successfully deployed data warehouses, where enterprises are reaping major benefits from their DSS/BI systems, and in turn are hungry for more by further extending their capabilities. Data warehouses typically grow in data volume managed over time, with a continually increasing number of business users wanting to be able to leverage the systems to help them with improving their efficiency and profits.

The set of new features in the EMC® CLARiiON® CX4 product family is designed to specifically support this type of rapid growth. The top-end member of the family, the CX4-960, delivers unparalleled levels of performance and scalability, easily supporting multi-threaded large random read requests on a sustained basis from many complex concurrent users. The system supports up to 960 drives, and accommodates a multitude of drive types ranging from extremely high-performance drives to extremely dense drives, offering not only good scaling in total capacity, but also storage tiering to allow for the most cost-effective way of storing and managing enterprise data warehouse content.

This white paper provides specific recommendations for configuring and managing a CX4-960 system to provide the most optimal support for deploying typical data warehouses for DSS/BI applications.

Introduction

The next generation of the CLARiiON storage system family, the CX4 product family, delivers not only higher performance at a more competitive and attractive price point, but many other new features and advantages. Major enhancements have been made to the product architecture to improve ease of use and provide flexibility to support growth and scaling for both performance and capacity.

A key enhancement is the new flexible I/O module design. The CX4 allows additional connection ports to be added to expand connection paths from servers to the CLARiiON, to handle more incoming I/O requests as more servers are connected, or if concurrent I/O traffic increases from the servers due to heavier load demands.

To ensure that the increased I/O load arriving from the servers can be effectively handled, ports from the flexible I/O modules can also be configured as additional back-end storage ports to support more connection paths for dispatching I/O to the physical disks.

Each member of the CX4 product family can support two or more of these flexible I/O cards, referred to as a pair of UltraFlex™ I/O modules for each of the two storage processors. The I/O cards have ports that can be adapted for front-side Fibre Channel (FC) or iSCSI connections, and ports for doing FC connections to the back-end disk drives. For all the standard CX4 systems delivered, the first I/O card will always be a FC I/O module, with at least one port (port 0 out of four ports) dedicated as a connection to the back-end drives. The second I/O card will support iSCSI connection to the array from the front side. All CX4 systems will be delivered with both FC and iSCSI connections as a standard configuration.

The FC I/O cards have four ports. The top-end CX4-960 can be populated with up to six I/O modules per storage processor (SP). EMC offers a number of different supported configuration options for partitioning up the ports to provide for balanced front- and back-end port bandwidth scaling. The iSCSI I/O modules have two ports each with a copper Ethernet speed of 1 Gb/s.

The CX4-960 also uses a totally new generation of storage processor CPUs, memory, and PCI bus architecture. As such, the storage processors are able to sustain more than twice of the peak data transfer bandwidth between the front- and back-end ports of the storage processors compared to the CX3 high-end systems.

Most DSS/BI applications frequently demand high sustained read I/O bandwidth from the underlying storage systems. Complex business decision support queries tend to need to access a significant volume of

data stored in the warehouse using parallel data reading streams, and the concurrent streams create a demand for high sustained read I/O through the storage. The new storage processor hardware architecture and the flexible I/O modules for configuring additional front- and back-end connections are all key new factors that make the CX4-960 the ideal storage infrastructure for supporting any significant and rapidly growing data warehouse.

Audience

This white paper is intended for IT staff responsible for planning and managing their storage infrastructure for supporting their enterprise data warehouse deployments. This paper discusses the best practices for configuring and deploying the CX4-960 to provide the most effective I/O subsystem support for DW/DSS type workloads.

Terminology

- **Business Intelligence (BI):** The effective use of information assets to improve the profitability, productivity, or efficiency of a business. Frequently, IT professionals use this term to refer to the business applications and tools that enable such information usage. The source of information is frequently the enterprise data warehouse.
- **Data Warehouse (DW):** The process of organizing and managing information assets of an enterprise. IT professionals often refer to the physically stored data content in some databases managed by database management software as the Data Warehouse. They refer to applications that manipulate the data stored in such databases as DW applications.
- **Decision Support System (DSS):** A set of business applications and processes that provide answers in response to different queries pertaining to the business, based on the business's information assets, to help direct or facilitate key business decisions.
- **Disk Array Enclosure (DAE):** The physical enclosure with disk drive slots to support up to 15 drives to be accessed from the CLARiiON storage processors using the UltraPoint™ connection CLARiiON technology. DAEs support the ability to grow the total number of disk drives used in a CLARiiON system in a modular fashion.
- **Enterprise Flash Drive (EFD):** Highest performance disk drives supported by the CLARiiON storage systems leveraging solid state technology. Because there are no moving components, these drives have very low per I/O request service latency compared to typical rotating disk drives.
- **I/O Cards:** The flexible I/O modules that can be added to the CX4 systems to expand the connection ports for increased front-side connections from the servers on the SAN, or back-end ports to provide for more I/O paths for the storage system processors to dispatch I/O to the physical drives inside the CLARiiON system.
- **Logical Unit Number (LUN):** A storage system object that can be made visible and usable as a server OS "disk device" from the underlying storage system.
- **Redundant Array of Inexpensive Disks (RAID):** A method of organizing and storing data distributed over a set of physical disks, which logically appear to be one single storage disk device to any server host and OS performing I/O to access and manipulate the stored data. Frequently, redundant data would be distributed and stored inside this set of physical disks to protect against loss of data access should one of the drives in the set fail.
- **RAID 5 (R5):** A RAID option where the actual data distributed and stored inside a set of drives is effectively protected by an additional set of parity data of the distributed content across the drives computed and stored in an additional drive. Under EMC CLARiiON implementation, the extra parity data is systematically rotated among all the drives in that RAID set to avoid any particular write hot spots when parity data adjustment has to be made against any piece of data stored inside LUN or LUNs created from this RAID group. RAID 5 protects against loss of data, or data inaccessibility, in the event that one of the drives in the RAID set should experience a drive failure.

-
- **RAID 6 (R6):** Another RAID option where two, instead of one, additional drives are used to augment all the drives holding data to create two independent set of parity data. RAID 6 provides higher data protection compared to RAID 5. There will be no data loss, or data inaccessibility, even if two of the drives in the RAID set should fail.
 - **RAID 10 (R10):** A RAID option with data striped over mirrored drive pairs.

Overview

The CX3 product family storage systems have been very successful in support of many major customer new data warehouses. The modular architecture of the system, the centralized single point management support for a group of CX3 systems under one browser view, and the five 9s reliability of these systems have established the CLARiiON product family as a solid storage infrastructure choice ideal for meeting the high-bandwidth requirements imposed by many of these major customer data warehouses.

A survey of IT professions by IDC and TDWI suggested that the trends of existing and new data warehouses would be moving into tens to hundreds of terabytes as the most common sizes in the later part of 2008. This represents a significant growth in capacity from the 1 TB typical size of new data warehouses from a survey taken in mid-2006.

Along with the expanded size, the way by which data warehouses are being used has also been changing radically. More and more, data warehouses are supporting mission-critical functions through BI/DSS applications. Many of the more prominent applications in this space are third-party applications with extremely intuitive and easy-to-use interfaces for business users as opposed to DBAs or SQL application programmers (for example, Cognos, BusinessObjects, MicroStrategy).

The upside of these powerful BI tools is that they enable a lot more business users to be able to directly benefit from the information assets. The challenge, however, is that it also becomes very easy for business users to generate very complex queries that are extremely taxing to the database service without always being fully aware of the implications. The database service now needs more intelligence and automatic resource management control to cope with these “ad hoc” queries that may not always have been well organized for leveraging available indices.

A common approach that is now used by most database service engines for supporting the complex “ad hoc” queries is to leverage data partitioning. Most of the business queries involve “facts” from the business data, which are frequently numeric in nature, such as sales revenue, units of products manufactured/sold/shopped, number of customer subscriptions, and so on. These “facts” are frequently summarized, analyzed, and compared for trends and statistical significance along certain “dimensions,” such as time (year, month, week, days), business geography, and product categories.

Most of today’s advanced database engines supporting DSS/BI queries have provisions for supporting partitioning of the business data stored inside the database. Business data that is frequently involved in certain business queries by virtue of some business criteria would be stored physically close. For example, rows of sales transaction data for all sales in the month of January 2008 may be stored adjacent to each other on a database “container.” In this way, when the DBMS engine performs I/O to access the “container,” the most amount of data useful to satisfy a query to determine how much sales revenue was achieved in that month would be accessed with the fewest physical I/O requests.

Partitioning typically helps to optimize the I/O needed to satisfy complex “ad hoc” queries.

First, by recognizing that the sales data has been partitioned by month, a query that tries to determine the total revenue year-to-date, for example, can be split into multiple subtasks, each working against just the partitions of data for a month. So, a query that needs the total revenue for 2008 can be processed as 12 subquery tasks, each working with the partition for a different month of the year in parallel. This helps to reduce the total time required to arrive at the needed result in 1/12 of time required, assuming there is sufficient processing and I/O resources to support all 12 tasks running in parallel.

An intelligent query optimizer with knowledge of how data has been partitioned can also speed up queries by avoiding unnecessary data reads. If the sales data is stored in a single large table, and the business query is interested only in the business trends in the first quarter of 2008, unless the sales data has been indexed by date of sales, we typically would have to pass through all the sales data to sort out the rows that are related to sales for the quarter.

With data stored and partitioned by months, the query optimizer would only generate subtasks to work on the data partitions for January, February, and March in parallel. If it takes 5 minutes to read up a month's worth of data for processing, it would have taken the query an hour to process the entire sales data table with 12 months of data (since we have to read all the data in the table). By parallelizing and limiting the I/O to the needed partition, the query can now be performed in 5 minutes, with one-fourth of the data passing through the I/O channel.

As mentioned, a key to effectively parallelizing the complex query into a subtask is that there are enough processing and I/O resources to handle all the concurrent work. With the rate server CPU power and supporting memory and I/O bus bandwidth have been growing compared to their cost, acquiring and scaling processing resources with extra server blades and CPUs is generally relatively inexpensive with commodity servers today. But to fully harness the power of these new processing resources, it is imperative that the underlying storage support can also grow and scale in a manner that would not end up breaking the bank.

This is one of the major benefits of the new CLARiiON CX4 system architecture.

CX4 UltraFlex I/O modules

A major innovation in the new CX4 product family architecture is the flexible I/O port modules. The new system architecture allows some number of additional I/O port modules to be added to the storage system to increase the number of front-side connection paths for the servers to drive more concurrent I/Os to the storage system. To match that increase for I/O service demand, additional ports can be configured to create more paths from the storage processors to the back-end physical drives by using the back-end expansion kit for CX4-960. (More information can be found on Powerlink[®] in *EMC CLARiiON CX4 Series Ordering Information and Configuration Guidelines*. Access may be limited.)

This flexibility offers two major benefits. First, even if the origin system procured has more limited front-end and back-end I/O bandwidth and path support, it would be relatively easy to expand the system by adding more of the I/O modules as the growing workload in the environment justifies the additional investment.

Second, hardware technology continues to evolve and deliver more power at more competitive prices. By going to the flexible port expansion architecture, as new technology enhancements in both hardware and software are added into the CX4 products, there will not be a need to completely replace the physical storage processors to expand the connectivity in order to take full advantage of the new capabilities. Since the initial CX4 introduction, high bandwidth connection support including 8 Gb FC and 10 GigE iSCSI has been added to the product family. CX4-960 systems already in deployment with 4 Gb FC or GigE iSCSI connections can readily be upgraded without affecting anything else already established and configured inside the array.

This new architecture provides strong protection investment, leaving ample leeway for adjustment of the system as needed, when a successfully deployed data warehouse should grow at a rate that is difficult to predict when sizing the initial system.

Multiple storage tiers support

Typical data warehouses tend to continually grow in size over time. At the same time, aged data stored in the warehouse often gets referenced with decreasing frequency, while more recent data tends to be referenced constantly, and often by many users or applications in parallel. Key data structures, such as indices, summary and rollup data tables, dimension cubes, and others, need to be accessed from the underlying storage with the lowest possible access service latency.

Transparent support for multiple disk types within the CX4-960 is therefore a key advantage for implementing data warehouses. Aged data that is accessed infrequently should be moved into the denser but slower SATA drives. The more frequently referenced data should be stored on FC drives to ensure reasonable access performance. Finally, the data with the most critical access and manipulation I/O service latency may be targeted for Enterprise Flash Drives (EFDs).

Virtual LUN migration technology supported by the CX4-960 allows data that is initially stored on one type of drive, or a particular “tier” of storage, to be migrated on the fly by the storage system processors, without impacting their continual accessibility from the database system. No operational outage is required to migrate the segment of data between the storage “tiers” leveraging this particular CX4 system capability.

Configuration guides

Balancing the configuration

Almost every data warehouse deployed by different businesses would exhibit some unique characteristics reflecting the business requirement, practice, and culture of that particular company. But most BI/DSS-type deployments these days tend to leverage query partitioning and parallel sub-query executions to handle the difficult-to-anticipate “ad hoc” queries that involve significant portions of the data stored in the warehouse.

In addition to the DBMS having the intelligence to partition the query optimally, and engaging all the resources available, it is very important that in the execution of the parallelized subtasks all the subtasks would execute in roughly the same amount of time. The time to complete a complex query that has been decomposed and processed as a set of parallelized subtasks is essentially the time to complete the slowest of all these subtasks.

It may not always be practical to ensure that each business partition of data would be equal in size. Seasonal retail store sales, for example, would likely have a much larger volume for times such as around holidays like Christmas and Easter, end of summer for back-to-school sales, and after Thanksgiving. Certain months of the year would typically have a higher concentration of sales data gathered. So, if we are partitioning the data by months and the subtasks are each working against data from a different month, chances are the subtask working with the December data may take the longest time to finish.

In recognizing the business implications, the DBA may select a partitioning scheme to break December’s sale data into two halves, so that the volume of data in each partition would match up better with the other months. Alternatively, multiple months with low activities may be logically grouped into a single partition to create a better balance.

Balance is therefore often the key. If we manage to balance the volume of data and the relative amount of work each subtask has to perform, then the next step is to make sure that a sufficient and balanced amount of processing and I/O resources can be dedicated to each of the subtasks to ensure that all the subtasks can complete in roughly the same amount of elapsed time, running in parallel.

Balance also extends to matching the rate by which data can be accessed from the storage infrastructure to keep the server processing as optimally engaged as possible. For the CX4-960 system architecture, there are a number of configuration considerations around different storage system components.

Balancing I/O activities among the two storage processors

The CX4-960 follows the traditional system architectural model of having I/O requests from servers against LUNs being serviced by one of the two storage system processors (SPs) that “owns” the LUN at any one instance in time. *To ensure that all the separate LUNs used by the DBMS engine to support partitioned and parallelized queries are balanced, the LUNs used to form the different database partitions should be divided as evenly as possible between the two storage processors.* For example, if there are 12 distinct data partitions for each of the 12 months of sales data, it may be logical to put all the LUNs corresponding to the odd number months on SP-A, and all the even ones on SP-B. Alternatively, if there is more historical knowledge about the distribution of volumes of data for the different months, it may be logical to put the month with historically the most volume on SP-A, the second and third on SP-B, and the fourth on SP-A, and so on, to ensure that each storage processor would be responsible for serving up a comparable volume of data.

Balancing data distributions among LUNs

While it may not always be operationally viable, to the extent possible, the goal would be to try to isolate the data partitions that would be logical and meaningful to the business queries into separate LUNs, and keep each LUN on a separate RAID group. To ensure that the typical data read rate for a comparable number of I/O requests, with comparable request sizes, can be achieved for each LUN used, the LUNs for the different partitions should be derived from RAID groups with comparable geometry. For example, if LUN 1 used as one of the “containers” by the DBMS to hold January’s sale data is a 2+1 R5 LUN from spindles 0, 1 and 2, created within RAID group 1, with 15k 146 GB FC drives, then it would generally be a good practice to create LUN 2 as the second container, also from a 2+1 R5 group, using spindles 3, 4 and 5, for example, on that same DAE. Then LUN 1 may be assigned to SP-A, and LUN 2 to SP-B, as suggested in the previous section.

The practical data read rate for a LUN is a function of the physical disk drives in the RAID group, and the pattern by which the data will be accessed from the drives. If the LUNs used for holding the data from different partitions are made up of different number of drives, or the drives are of different characteristics (such as 15k rpm FC drives versus 10k rpm drives, FC versus SATA, and others), the practical sustained data read rate from the different LUNs would be different. This may in turn unbalance the concurrent progress of the different parallel query subtasks going against partitions of data on different LUNs.

Creating multiple LUNs on the same RAID group, which share the same set of drives, to be used as “distinct partitions” by the query engine, is generally not recommended. The parallelized subtasks would each be trying to drive I/O against their respective data partitions, and concurrent I/O against each of the data partitions will in fact be competing for data access. The drive head will generally end up doing considerably more seeking, and delivering a lower total sustained MB/s of data read.

If it is decided for operational considerations to create more than one LUN per RAID group for use by the database engine for the partitions of warehouse data, it is recommended that two LUNs of roughly equal size be created per RAID group. One of the LUNs should be assigned to SP-A, the other to SP-B. A similar layout should be done for the second RAID group, as shown in Figure 1.

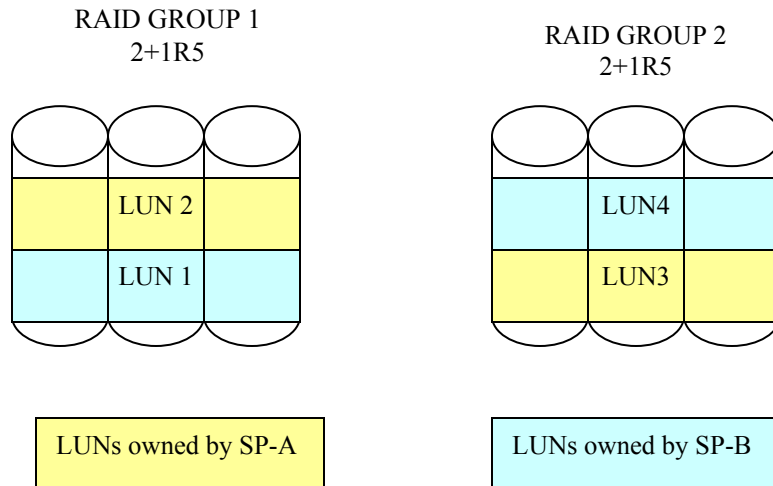


Figure 1. Example of multiple balanced LUNs per RAID group

Balancing traffic across the back-end buses

In the CLARiiON system architecture, the disk drives on each DAE are supposed to be always simultaneously accessible from both SP-A and SP-B. Each SP goes through one of its back-end I/O ports to communicate with the Link Control Card (LCC) of the DAE.

A back-end I/O port from each SP, connecting into the same DAE, is considered as a *back-end bus*. A back-end bus can be extended to include more disks by chaining the first DAE through the LCC links to a second DAE. Since the CX4 product family uses a modular and flexibly expandable I/O port connection architecture, we can adjust the number of I/O ports used as back-end ports for each of the SP, with the proviso that there must be matching pairs of ports, one from each SP, to connect into a DAE, in forming a separate back-end bus.

A back-end bus can include up to eight fully populated DAEs, or 120 drives. As a general recommendation, each bus should preferably carry roughly the same number of total drives. Assuming that data will be as evenly spread over all the drives being used on all buses, having roughly the same number of physical drives would ensure that there would not be unbalanced I/O traffic distributions among all the back-end I/O ports of the array SPs. Figure 2 on page 11 depicts a back-end bus layout.

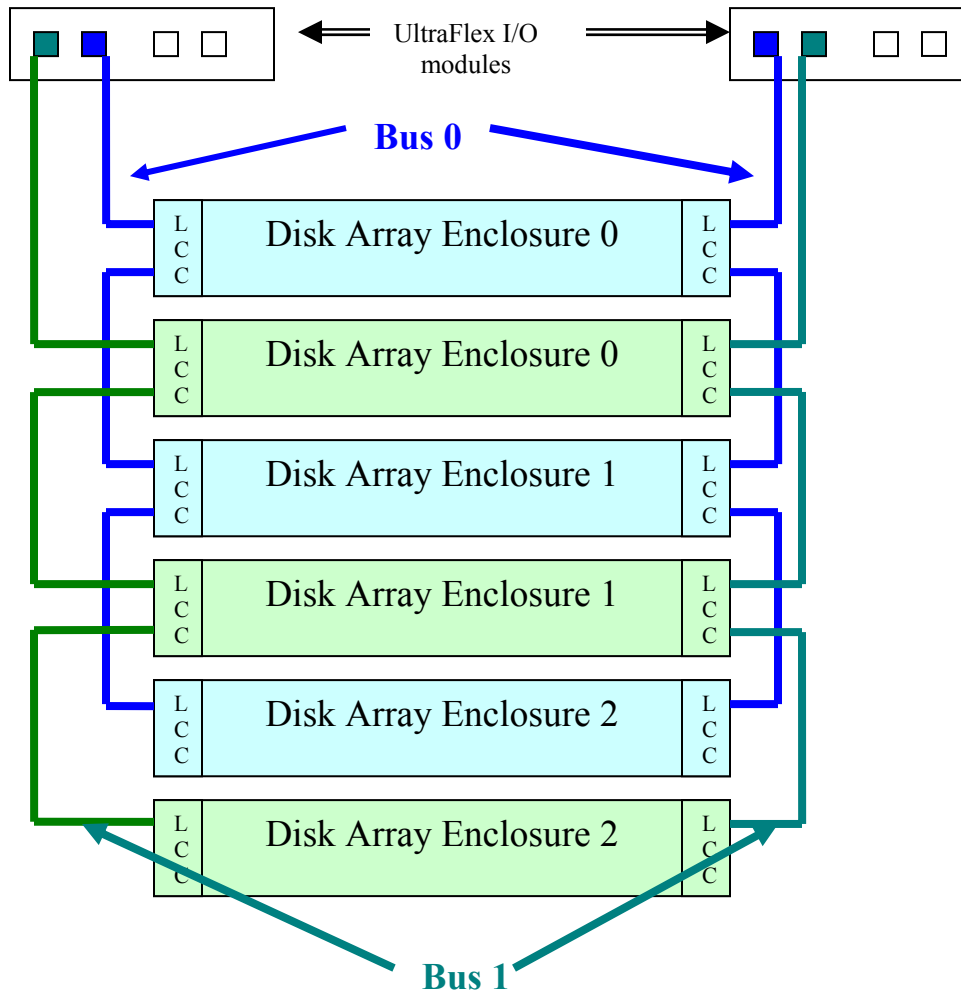


Figure 2. Back-end bus examples with two buses, each with three DAEs

In this example, each back-end bus includes three DAEs. If all the DAEs are fully populated, each will have 15 drives. And each bus will then have 45 drives in total (3 x 15) where I/O and data will be going through the corresponding back-end I/O ports on the two I/O module cards, one on SP-A, and one on SP-B.

Since all the disks are accessed through the one back-end port from each SP, and these are 4 Gb FC ports at present, the theoretical maximum number of bytes of data that can be pulled from all the drives on a single bus is limited to no more than 2 x 4 Gb/s achievable through the two I/O ports. As a general rule of thumb, 720 MB/s is roughly the practical bandwidth that can be used for the more conservative sizing estimates. Once we start to close in on that amount of data being pulled through those two back-end FC I/O ports, adding more DAE trays to the bus will not necessarily net more MB/s of I/O data being pulled from the additional drives. The top sustained MB/s will not increase for accessing data from that bus. Only the total volume of data that can be accessed through that particular bus will increase.

To increase the total net bandwidth of accessing data, generally, more back-end buses should be used. For this example, using the 720 MB/s rule of thumb, we can sustain logically 2 x 720 or 1440 MB/s of data pull from all the data spread over the 6 x 15, or 90, drives.

If the same 90 drives are redistributed over six separate buses, so that each DAE will be on a separate bus, it may seem like we can scale up to 6 x 720 or 4320 MB/s.

In practice, this will not hold up. Once we add more back-end buses by using a back-end expansion kit to create more back-end I/O ports, other limiting factors will start to come into the equation. To pull up to 720 MB/s from the 15 drives in a DAE, each drive must offer up on average 68 MB/s per drive. This is typically unrealistic with the amount of random data reads pushed against the drives in response to the different concurrent host I/O reads against different parts of the different LUNs spread over these 90 drives.

While there is no absolute requirement that each bus should have exactly the same number of DAEs and drives, it is generally recommended that the number of drives and DAEs per bus should be balanced as evenly as possible. Assuming that the data partitions are going to be organized to be spread out over the available drives, the drives should be distributed evenly among the different buses. At any rate, if the data partitions are going to be distributed over N drives, and there are M buses used, the objective would be to try to organize the back-end data distributions so that N/M drives on each bus will be holding data for the different data partitions.

The CLARiiON system software by architecture reserves a certain amount of disk space on the “vault drives,” or the first five drives from bus 0, enclosure 0. The net usable space from these five drives will therefore typically be less than the rest of the drives, assuming the same type of drives are used throughout the array in the different buses and enclosures.

A common practice is to include an additional DAE on bus 0. For example, in Figure 2 on page 11, we may add another DAE to bus 0, behind enclosure 2. Instead of trying to evenly spread the data over the six enclosures (bus 0, enclosures 0 – 2 and bus 1, enclosures 0 – 2), we will skip bus 0, enclosure 0, and put its share of data to bus 0, enclosure 3. This makes the job of distributing the data a bit easier, as all drives on all enclosure will have identically usable capacity.

The drives on bus 0, enclosure 0, can be reserved for use for other than the key data warehouse data partitions. Also, a few of the drives in that enclosure can be used as hot spares to provide for hot spare stand in, should any of the drives fail in the enclosures containing the DW data partitions. Also, if the space is not needed, aside from the hot spares and the vault drives, the base enclosure (bus 0, enclosure 0) does not have to be fully populated.

Balancing I/O traffic among the UltraFlex I/O modules

The FC port in the I/O modules in the standard CX4 system configurations are partitioned evenly into front and backup ports. This ensures a nice balance of I/O traffic through the CLARiiON storage processors. The CX4-960 offers expansion options that allow more front-end or back-end ports to be added to meet operation needs.

When it comes to DW/DSS workloads where I/O bandwidth is often the key, it is important to remember that most of the time, the data that is being requested by the DBMS engine(s) from the host(s) will in fact be read from the physical drives. The CLARiiON storage cache will not be holding most of the data requested (except for the segment of data triggered by LUN prefetching). The data read pushed down from the host by the DBMS data is for the data sitting out on the physical drives, which the DBMS engine has not been able to locate on the server side I/O buffer memory of the server. Mostly, what the DBMS has cached on the server memory is DB data that has previously been read from the storage. So, if we have anything in the CLARiiON cache, these are probably the very same DB data pages that the DBMS has already cached on the server side. So, the likelihood of us actually finding the needed DB page in the CLARiiON cache is low, and we have to go to the physical drives to get the needed data.

Hence, as a general guideline, to the extent possible, it is recommended that when considering upgrades to front-end and back-end ports the front-end and back-end ports for DSS deployments should be kept as balanced as possible. For example, if we end up with a configuration of four back-end buses and eight ports, and we choose to expand the front ends and deploy 10 out of 16 ports, we may end up with occasional unbalanced data flow between the different I/O modules.

RAID choices for DW/DSS workloads

In the world of OLTP workloads, mirrored RAID options, such as R1 or R10 or R 0/1, are frequently preferred over parity RAID options such as R3, R5, R6, and others. Typically, the key concern is with random DBMS data page writes resulting from update transactions.

Parity RAID implementations typically maintain data parity for the stored data in a data stripe-by-stripe basis. The CLARiiON system default data striping size is 64K. On a 4+1 R5 configuration, the first 256 KB worth of stored data is stored as four pieces of 64 KB across four of the five drives in the RAID group. On the remaining drive, a parity piece of 64 KB is computed and stored, based on those four pieces of 64 KB stored in the other four drives.

If a database transaction results in all 256 KB of stored data being changed, and the entire 256 KB is written back to the storage system, the parity 64 KB can be directly recomputed from the new data. While the parity computation does add some CPU time, and requires more SP memory buffer resources to be performed, the overall time to actually write the new data back to the respective drives would still be dominated by the time of the physical disk write. This is usually in order of milliseconds by comparison to the CPU parity computation times, which may be in order of hundreds of microseconds or low milliseconds.

For OLTP workloads that tend to frequently update and rewrite many single 4 KB or 8 KB database pages, each such write would have to be done with a parity adjustment. The old image of the DB page, as well as the corresponding 4 KB or 8 KB of parity data, frequently has to be read back in from the two different disks in that RAID group. Then, the bit delta between the old and revised DB page image is determined, and the parity page content adjusted. Finally, the revised new DB page and adjusted parity page would have to be sent back to the respective disks.

Hence, parity RAID is generally avoided for extremely high update rate OLTP workloads.

DSS workloads, on the other hands, tend to focus on reads. Data writes to the LUNs holding the database are generally from the loading of new data into the data warehouse. Most DW new data loads tend to be batched together. Random updates to existing data are typically rare. And even with the trend of moving more toward “close to real time continual data loads,” new data loads tend to be done as controlled batches at well-defined operational windows.

As such, the actual OS level write requests being sent by the DBMS to the storage system tend to be bursts of large writes. In fact, for most DBMS engines supporting DW data load functions, the engine can often be configured to batch up consecutive new DB pages, sending the entire batch as a “coalesced” big server write request that spans a full RAID stripe of the storage LUN (or multiples of full stripes).

Hence, for DW purposes, parity RAID is often a good choice.

As data volume increases, full mirrored RAID options such as R10 requires a 2x increase of physical disk space for every additional of X GB of data growth. With parity RAID such as 4+1 R5, the protection data overhead is only 0.25x.

Because of the way parity is rotated in R5, actual warehouse data is in fact stored in every drive in the RAID group. So, when it comes to retrieving data from the LUN in the RAID group, all the drives in the RAID group in fact participate in contributing the data required. When there are enough concurrent read threads trying to read different stripes of data stored in the LUN at different LUN offsets, as is common these days with a DW supporting a number of concurrent user queries, all the drives will end up participating and delivering a comparable amount of data overall.

Let’s say we have the choice of creating eight 4+1 R5 LUNs, or five 4+4 R10 LUNs, out of the same 40 physical drives we have available in the CLARiiON system. Assume that each drive is a 146 GB drive, with about 130 GB of real drive space readily usable.

With the R5 configuration, we can store up to $4 \times 8 \times 130 = 4120$ GB worth of data. With the R10 configuration, we can only store at most $4 \times 5 \times 130 = 2600$ GB. With likely more data stored in the same 40 physical drives, drives will likely be pushed harder by the concurrent demand for data stored on the drives by the DBMS processes.

At the same time, the DBMS sees eight distinct “OS disk devices” that I/O requests can be queued against, as opposed to five with the R10 configuration. As long as we can keep up with that harder concurrent read data I/O push from the host, we will likely end up getting more effective total data push through the CLARiiON system.

The bottom line is that parity RAID, including both R5 and even possibly R6, is a reasonable RAID choice to consider for the database that is used primarily for supporting DSS/BI-type query-focused workloads, particularly if the data load pattern is in fact bursty batches of large data chunk writes against the underlying storage LUNs holding the database.

Certain data warehouses/datamarts may include more than pure “business facts” data. Often, redundant data may be carried in the warehouse. The most common form of redundant data would be different indices. Others might include data like materialized views, and dimension cubes, which are additional information derived from the “facts” data.

Whereas the raw “facts” data are often added to the data warehouse in large batches, the ingestion of the new batches of data likely end up triggering revisions to the derived data as stored in a fashion that approximate random single DB page updates. For example, when adding in the sales receipt data for this past month, the year-to-date total sales revenue for the country, region, state, and county aggregate data may need to be adjusted in a sales data cube with a geographical information drilling dimension in a typical dimension cube for doing sales data analysis.

For these types of data, with their expected update characteristics, it may be appropriate to isolate them into a separate set of LUNs organized as mirrored RAID, to ensure faster update latency.

RAID stripe width and stripe element depth

Intuitively, if the DBMS is frequently performing large data scans of the stored data, it would seem that it would be a good idea to spread the stored data out as wider stripes across more physical drives, so that more drives can be kept busily engaged delivering the data streams. On that reasoning, an 8+1 R5 would be preferable to a 4+1 R5, for example.

Practical experience working with different DSS/BI query-heavy deployments is proving more to the contrary.

With the large drives, a very wide RAID group has considerably more usable capacity. As such, considerably more data will typically be stored inside a LUN that spans a wider RAID group.

The real world data warehouses today are rarely the private, single, or limited user domain of a privileged few. When there are heavy concurrent user queries running against the data warehouses, it is common to have multiple user queries trying hard to pull the data they need from different part of the LUN.

If two concurrent user requests of 1 MB each are being pushed against the same LUN that is eight-way striped (for example, in an 8+1 R5 LUN), the two host-side 1 MB requests will be split up into two sets of 128 KB requests to be sent to each drive in the eight-way striped set. The drive head has to spend a certain amount of time seeking the right disk position, pull in the first 128 KB, then reseek to the second position, to secure the next 128 KB. The time to actually deliver both pieces would be equal to $2 \times (\text{seek} + 128 \text{ KB data read time})$. While the disk head is seeking, no data is being delivered. The seek time is physically

governed by the disk drive hardware mechanics, and is relatively large (and sometimes variable) compared to the time to actually pull the 128 KB of bits out of the disk drive.

In contrast, if we have striped and stored the data in a 2+1 R5 LUN, these two same requests would have possibly expended just about the same amount of time seeking at the drive head. But once at the right disk head position, 512 KB worth of data would be pulled from the spindle on each read from the drive.

So, in effect, the “thin” RAID allows more sustained MB/s of data being read from the drives after each seek, thereby delivering more effective MB/s per drive.

With the same number of drives, configuring with “thin” RAID groups will result in more RAID groups, and likely more LUNs. An 8+1 R5 LUN spanning an entire nine drives of 146 GB provides a usable capacity of about 8 x 133 GB, or 1 TB. Three LUNs from three 2+1 R5 groups from the same nine drives offer only 6 x 133 GB, or 7.5 TB.

But when the DBMS can organize the parallel I/O to push three concurrent 1 MB read requests, with each going to a separate “thin” LUN, these get serviced a lot more efficiently than if the three requests have been sent as three “competing” 1 MB requests against the eight-way-striped “thick” LUN.

This is true as long as the DBMS is capable of fully exploiting the increased degree of parallelism. For most DBMS engines with smart support of partitioning and parallel optimization, this is indeed the case.

With the mechanical nature of the disk drive head seeks, most of the typical 15k FC drives would have an average head seek of about 2.5-3 ms per drive vendor specifications. Let’s assume that the average data transfer time for 128 KB is about 3 ms, and to transfer 512 KB would take 12 ms.

For the single wide LUN (eight-way striped) case, it takes 3 ms to move the head to the right place on a drive, and another 3 ms to pull off 128 KB. So, in one second, we can pull off 20 MB/s on that drive (1000 x 128 KB / 6).

For the thin LUN (two-way striped), it takes the same 3 ms to move the head, and 12 ms to move the 512 KB worth of bits. The net MB/s from that drive is therefore 33 MB/s (1000 x 512 KB / (3 + 12)).

So, in fact, from the same nine drives, we can pull off considerably more MB/s by going to the thin RAID configuration, as long as we can manage the increased number of LUNs presented to the DBMS engine, and that it can fully leverage the higher number of “OS disk containers” that the data warehouse is distributed across to drive higher concurrent host level parallel read I/O requests.

But what about the extra usable capacity lost by going to a thin RAID LUN configuration? The good news is that with the advance in drive technology, denser and denser drives with comparable performance characteristics are now available. For example, 300 GB FC drives at 15k rpm are now available in the marketplace, and as time goes on, more and more of the CX4 class systems will in fact be ordered and shipped with the 300 GB FC drives, just like the 146 GB drives overtaking the 73 GB drives not very long ago.

Assume that the 300 GB drives currently cost about 50 percent more per drive compared to the 146 GB. Ninety drives of the 300 GB organized as “thin RAID” will deliver in theory 3000 MB/s (90 x 33.3). To get to the equivalent rate with the 146 GB drives organized as the 8+1 R5 LUNs, we will need 150 drives.

With 90 drives of 2+1 R5, we have about 60 drives of roughly 270 GB usable each, or about 16+ TB. With the 8+1 R5 on the 146 GB drives, we have about 17 TB usable.

The relative cost of the 90 drives of 300 GB would be $1.5 \times 90 = 135$ compared to 150 for the 146 GB drive.

What that means is that by leveraging thin RAID and larger density drives, we can achieve a comparable sustained read performance level with fewer drives, more concentrated data usage per drive, and a lower overall system cost.

This is based on the assumptions that the performance focus is typically highly parallel and concurrent large (such as 256 KB, 512 KB or 1 MB) reads being pushed by the host software, and that the host applications can fully leverage the higher number of distinct “OS disk devices” to drive a higher degree of concurrent read I/O. This happens to be exactly the typical paradigm these days with most of the DSS/BI workloads running against many of the DW DBMS engines.

A number of key FLARE® changes have been included in release 28, the array system software release in support of the CX4 family, to ensure that we can drive the underlying disk drives to the considerably higher level of data delivery rate using the “thin” RAID striping configurations. The larger capacity drives, such as the 300 GB 15k drive, are supported in the CX4 family product release as standard drive types.

So, a new recommendation for RAID choice with the CX4-960 deployed for the DW/DSS environment is to consider using thin parity RAID, such as 2+1 R5, or 2+2 R6, and to also leverage the 300 GB drives to help with reducing drive count, footprint, and power consumption.

Configuring cache memory for DW/DSS workloads

One of the key enhancements in the CX4-960 system architecture is the expanded support by the kernel to use up to 16 GB of physical system memory per storage processor. Allowing for system resident kernel software and drivers, there is over 13 GB of free storage processor memory per SP that can be configured to data cache use for the different LUNs. So, much larger read and write caches can now be configured (up to over 10 GB of mirrored write cache, a significant increase from the 3 GB maximum supported in the CX3-80).

The CLARiiON cache configuration supports a range of cache page size options. DW databases tend to use larger database page sizes such as 8 KB or 16 KB. In general, it is a reasonable approach to choose the cache page size to match the database page size. Choosing the 16 KB size, the maximum setting, tends to allow the storage system software to push the large single disk I/O request through the back end to the physical drives.

Write cache configuration considerations for DW data LUNs

The memory allocated for write data caching continues to be memory mirrored on each new data write from the write cache memory location on the owning SP of the LUN (that the write I/O is directed to) across to the peer corresponding cache memory area in the peer SP, before the host write is considered committed. This protects against the loss of any committed write data, even in the event of an SP failure.

However, major innovative rearchitecture of how dirty write cache is handled in the SP mirrored memory cache now reduces significantly the number of situations where the storage system may have to disable write caching to avoid the potential risk of losing committed write data from the SP cache. For example, array software non-disruptive upgrades, which would bring down one SP at a time to perform the upgrade, no longer require the write cache to be disabled for the duration of the new system software installation and upgrade process.

DSS/BI workloads tend to emphasize data reads. Data writes to the LUNs holding the DW data are generally associated with new load loads (or old data archival and deletion from the warehouse for further active use).

As such, data writes also tend to be done frequently as a single host write to “add” a new batch of database pages into the warehouse. Mostly, the DBMS engine handling the data loads batches the new data together into a larger batch, to be sent down to the storage as a single (or few) larger, and perhaps full array LUN stripe size, OS write request.

Thus, the writes from the data load tend to come to the storage system as write bursts. The extended write cache now allows a much larger burst of writes to be absorbed quickly, and be subsequently destaged to the physical drives in a more orderly manner that would provide greater write efficiency to the physical drives.

Write caching is especially crucial for large batch data writes to the DW with data stored on R5 or R6 LUNs. Even though it is frequently possible to configure the DBMS to ensure that new data pages are batched together, and sent down to the array as single large writes such as 1 MB or 512 KB writes that would span one or more LUN stripes, it is often very difficult, if not impossible, to force the data that is being written to be aligned perfectly on a LUN full stripe boundary.

For example, let's assume that our database LUN is a 2+1 R5 LUN. We manage to get the DBMS to batch up a number of new pages, and send down to us a 512 KB, which would have been spanning four full stripes of the LUN (each stripe is 2 x 64, or 128 KB for this LUN). If the data is to be written back to the LUN at a LUN byte offset of 128 KB, for example, we know that the new data is going to be exactly the next four full LUN stripes of 128 KB for this LUN, starting at byte address 128 KB. We will not have to worry about doing any partial stripe writes, and contending with the read-before-write parity adjustments. In fact, if the host writes are 1 MB, and these are always fully LUN stripe aligned in a sustained rate, it is actually more efficient for the writes to be done as cache bypassed writes (an option that can be configured on a LUN-by-LUN basis under Navisphere[®] Manager), since we can bypass all the extra effects of having to mirror the write data to the peer SP, and so on.

However, if the database files are stored on OS file system files, such as NTFS under Windows, or on cluster file systems such as VxFS, and there are multiple DB files created into the file system, it would be impossible to force the OS file system to cause the data to be always laid out with the LUN, perfectly aligned to a LUN stripe boundary.

When there is no viable way to enforce host data writes to align with storage LUN stripes, writing through cache is the best way to achieve the most efficient data destage for R5 or R6 LUNs holding the DW database.

Read cache configuration considerations for DW data LUNs

In spite of the predominant read pattern typical with DSS/BI workloads, storage read caching for DW data LUNs does not always produce as much of a high payoff as one may expect.

Database servers today frequently have a considerable amount of memory for DBMS data buffering purposes. The DBMS engine will only be pushing read requests down to the storage subsystem against the data LUNs if the required data is not found in the DBMS memory buffers at the host. Since most of the data cached in the DBMS server buffer memory is database pages that had been previously read from the storage, chances are that whatever can be found in the storage system cache overlaps with what the DBMS has already memory cached on the server side. So, the chance of a DBMS read that can be satisfied from the storage cache for the data LUNs is really not that high.

About the only exception would be if the storage system has been implicitly initiating predictive LUN data prefetching from the back-end drives for the LUN triggered by detection of an apparent sequential data access pattern from successive read requests against the LUN from the host.

In most parallel query paradigms, different concurrent threads are scanning different data LUNs. Concurrent queries may have threads that are going after data from different parts of some common LUNs. With many of the DBMS engines managing data placement of the stored data using their own logical striping and hashing schemes, a true LUN sequential read pattern is not really that common.

Each user thread trying to read data from a particular LUN may in fact be reading data in somewhat clustered and consecutive LUN locations. But when these are run together against the same LUN concurrently, the net I/O access pattern against the LUN is effectively random reads.

The reads pushed down against the LUNs are essentially already logical table data “prefetches” from the DBMS’s perspective. A parallel query thread working on one partition of data from the DW is usually driving an OS read asynchronously to start getting the next 256 pages of data for the current table of interest ready into the DBMS memory buffers, while it is working through the last 256 pages of table data. As long as we have that next set of 256 pages of data ready for this thread by the time it is done working through the current set, trying to get the following set of 256 KB very quickly (leveraging the storage implicitly prefetched LUN data) is often not particularly important or relevant to the overall effectiveness of the parallelized subtask executions. The thread is not really ready to use that next set of 256 pages anyway.

In fact, implicit LUN data prefetching can frequently have the counterproductive effect of tying up the drive unnecessarily, slowing down the completion of some of the subtasks, which have to wait longer for their share of data.

It is therefore usually not necessary to configure a large amount of read cache for the benefit of data caching for the data LUNs. If the typical workload is driving a high degree of parallel queries, it is usually right to disable implicit LUN prefetching on the data LUNs.

The CLARiiON performance analysis tool, Navisphere Analyzer, includes two metrics, the MB/s of data prefetched for a LUN, and the % of prefetched LUN actually used. These two metrics can provide a guide as to whether the implicit LUN prefetching should be disabled.

To disable LUN prefetching, which can be done dynamically, the prefetch policy for that particular LUN can be set to NONE from Navisphere Manager, or through NAVISECCLI.

Leveraging storage cache for other data

While read caching for database data LUNs may not always pay off great dividends, targeting storage cache memory for LUNs used to hold TEMP table spaces for the DBMS engines can be very beneficial. Business SQL queries that include clauses like GROUP BY or ORDER BY frequently involve sorting. Also, while most sophisticated query optimizers try to avoid SORT-MERGE joins, there are situations where those cannot be completely avoided.

Data sorting is often on performance-critical paths for executing the set of parallelized subtasks. Data sorts are by definition serial processes. A task depending on the result of a data sort must wait for all the relevant data to be collected and sorted before the task can move forward. The more data that has to be sorted, the longer is the wait for the completion of the data sorting required. When the volume of data to be sorted is large enough that it cannot fit within the DBMS server memory buffers, the intermediate, partially sorted data is frequently spilled to temporary disk areas. Then, the data will typically be read back from these temporary disk areas in chunks, so that they can be merged into a final, fully sorted data stream.

Intermediate sorts spilled to disks are generally done to TEMP table spaces managed by the DBMS. TEMP table spaces are often created as files in the same storage object (LUN) along with the database data files. While this approach may simplify operational management issues (fewer storage LUNs and fewer server level “disk devices” and/or file systems), it does give up the opportunity of more effectively optimizing the performance of the overall space usage.

It is actually a good idea to try to separate out the TEMP table spaces into separate CX4 system LUNs if possible. The pattern of I/O to the TEMP areas is bursty reads and writes. I/O service latency is often far more critical, since most of the I/Os involved are likely synchronous as opposed to asynchronous I/O. Query subtasks frequently block waiting for I/O to the TEMP areas.

SP cache memory assigned to the LUNs holding the TEMP table spaces can make a tremendous amount of difference. The data being written out to TEMP as part of a sort action invariably will be read back

subsequently, and perhaps within a relatively short time window, be used for merging, and so on. As the data chunks are being read back, primary in a sequential order, LUN implicit prefetch will booster the data read back rate significantly.

So, TEMP space LUNs should always have read/write caching enabled. Then, whatever read/write cache memory allocated to the SPs will be used by the SP to bias towards expediting the performing of I/O for the TEMP LUNs.

Transaction log placement and caching considerations

It is an often debated topic among storage administrators and DBAs whether database transaction logs should coexist in the same RAID group as the database data LUNs, sharing usage of the same set of physical drives. For data warehouses, the transaction logs are mostly active while new data is being added to the warehouse, or while old data is being archived and evicted from the data warehouse active use. It is not critical to separate transactions logs from data on separate physical drives.

However, log write I/O service time is important. It is important to make sure that transaction log LUNs have read and write caching enabled. For that reason, it is generally a good idea to have distinct LUNs holding the transaction log data. The transaction log LUNs do not have to be isolated necessarily to their own RAID group and set of physical drives. The log LUN(s) can share the drives of a RAID group with data LUNs. But as distinct LUNs, read/write caching can be fully enabled for the log LUN(s), without affecting the data LUNs on those same drives.

If the decision is to create thin parity RAID groups, such as 2+1 R5, for housing the data, and it is decided that it would be operationally desirable to have the logs and data sharing drives, a possible option may be as shown in Figure 3.

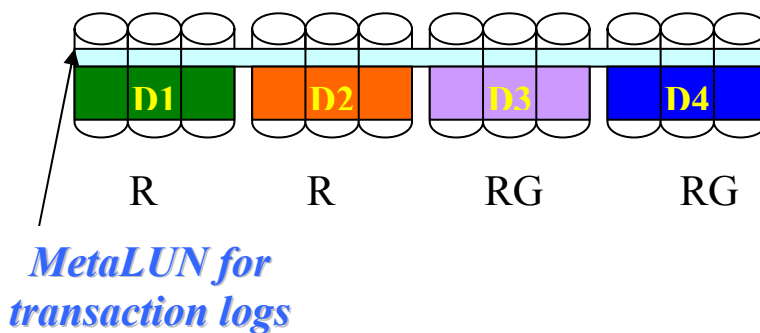


Figure 3. MetaLUN layout for transaction log option

The transaction log LUN (or one of many transaction log LUNs) can be created as a “thin” LUN slice from multiple RAID groups (for example, four in Figure 3). The bulk of the space for each of the RAID groups will be used to carve out a LUN (for example, D1, D2, D3, D4) to be used for housing the warehouse data.

But before creating the data LUNs, a thin slice of each RAID group (for example, 4 GB) will be taken out of each of the groups to create a set of LUN slices to be used for the transaction logs. The “skinny log data LUN slices” are grouped together to form a striped metaLUN (in this example, a four-way striped metaLUN of four slices of 4 GB).

The metaLUN stripe element width will be set to one instead of the normal default of four. What this means is that for our example above, we have a log LUN of 16 GB that spans the four pieces of 4 GB from each RAID group. The first 128 KB worth of log data written will go into the beginning of the slice in RG1, the second 128 KB into the slice in RG2, and so on.

So, the log data will be spanning all 12 drives. Each write of 128 KB or less will be focused on one of the slices, while bigger writes will be spread over all the slices. The log data, when destaged from the writer cache, will typically be just 128 KB full stripe writes to each of the slices. This ensures the destaging writes of the log would have the lowest impact on the I/O going to the drives that the log data is sharing with the database data.

Read and write caching should be enabled for the log LUN. If the CLARiiON Navisphere Quality of Service Manager (NQM) option is installed and enabled on the CX4-960 system, it is recommended that the log LUNs be biased high (please refer to NQM documentation for further information) relative to the data LUNs to ensure that it will get preferred I/O service when competing against the I/Os to data LUNs.

Deploying Enterprise Flash Drives

Introduced into the CX4 product offering in Q4 2008, EFDs have been gaining rapid adoption worldwide by many operational databases with high IOPS service latency requirements. Without any moving parts, EFDs are capable of servicing many random I/O requests per unit time, with low milliseconds I/O service time.

Bandwidth (as opposed to IOPS and service latency) is often the more crucial I/O characteristic for most data warehouse deployments. Unlike the case with rotating disks, accessing a 1 MB chunk of stored data versus 128 requests of 8 KB randomly scattered over the EFD actually is very comparable in time. So, even if a variety of data partitions are sharing an EFD, and there are many concurrent queries going after different parts of the data stored inside the drive, the aggregate rate of data that can be read from an EFD tends to be relatively constant. In fact, the more concurrent accesses are driven against the same EFD, the more likely the drive will be able to reach its full bandwidth delivery potential.

However, with DW applications, a common model is for the database engine to be doing a considerable amount of data processing with the data already pulled up from the warehouse, while keeping a number of asynchronous I/O data “prefetching” threads in the background to move further data up from the warehouse to complete the full query task. It is important to keep the warehouse data feed pipeline filled as much as possible, so that the CPUs will not become idle waiting for further data when they are done with the current set. However, as long as the data gets delivered to the server in time before the processing threads run out of work to do, it is often not as crucial how fast each individual I/O gets completed.

For this reason, the bulk of the detail data, often referred to as “facts” data in many star schema warehouse implementations, may not necessarily benefit from being moved in its entirety from rotating disks to the EFDs.

On the other hand, warehouses that are implemented to include summary tables, or dimension cubes, that are derived from the facts data are frequently strong candidates for EFDs. After all, the whole reason for going through the extra effort of summarizing and aggregating data into special tables while loading in the facts data is to avoid repeatedly recalculating these values from the detail data to satisfy repeated queries for the summarized data values. The summarized data thus becomes a relative “hot spot” in the schema.

For example, as detailed sales data is loaded, dimension tables of sales revenue may be built up showing daily sales total, weekly total, monthly total, and total by stores, by product types, and other values. These summary or aggregation data values are built into additional tables and stored within the database with the expectation that many user queries will be repeatedly targeting those summarized data values.

So, it would be reasonable to consider targeting just these tables for EFDs. Many more concurrent user requests for this table data can be satisfied from EFDs, with low millisecond service times for each request from the drive.

As the scope of business queries leveraging the warehouse data continues to expand for each successful deployment, the size and number of these types of dimension tables and so on also grow rapidly. It is now

not uncommon to have dimension cubes that are in the same order of magnitude as the raw business data details. The rapid advance of EFDs, both in speed and in total usable capacity, is the perfect solution to match the same rapid growth in the cubes.

Many well-managed queries (by knowledgeable DBAs and application developers) continue to heavily leverage indices. Like dimension cubes and summary tables, special effort is made during load ingestion into the warehouse to extract, update, and maintain the different indices with the goal of being able to significantly speed up subsequent queries. So, subsequent queries that go through the indices should be able to get at the stored indices with the highest performance emphasis. While it is true that intelligent database management systems tend to cache as much of the frequently used indices in server memory as possible to avoid actually doing repeated storage I/O, the fact remains that as the size of data and complexity of queries continue to expand, it comes to a point where the indices may get large enough (or there are enough indices used for speeding up *different* typical business queries) that it becomes impractical to cache all the key index entries in server memory. At that point, EFDs are again reasonable candidates for targeting the frequently used indices.

Contrary to the gut feel of many DBAs, it has been generally found that EFDs are not necessarily the right storage for transaction logs. This is especially true for some data warehouse deployments. Even though the trend is gradually shifting, there are still many warehouses today that operationally load new data into the warehouse in batches periodically. In many cases, new data loads are done without transaction logging. If the batch of data load fails, all partially loaded data is discarded by the database management system, and the loading of the batch is just restarted.

Even if the data load is done with logging, the log data generated is really written out serially to the database log. The new data being loaded is often not immediately required from the common business query perspective (for example, a comparison of last month's business revenue to the year before does not require the business revenue detail for the business from the last hour), and a typical transaction commit would be for a set of new business data as opposed to a single new business data entry. So, transaction responsiveness is frequently not as critical a business system success factor in these types of deployments.

In contrast, temporary table areas used frequently by the database management system as "scratch pads" for intermediate work can potentially benefit from EFDs. Complex queries with table joins and data ranking such as GROUP BY or ORDERED BY type SQL statement clauses often involving sorting of significant volume of data may not fit completely in server memory reserved by the DBMS. By organizing the DBMS temporary table areas on EFDs, the large sorts that spill over from server memory now effectively benefit from a "secondary" sort data buffer that can be significantly larger in size, and yet can provide low millisecond access efficiency.

To determine if EFDs are appropriate or needed for this purpose, more in-depth analysis of performance statistics provided by the DBMS should be done. Most DBMSs provide insight as to how frequently data sorts are occurring in a typical operation time window, and the amount of temporary table disk area read/writes are done in support of those sorting activities. Some DBMSs provide a further view of how responsive the "disk sorting" I/O is performing from the DBMS point of view. If there are substantial sorting activities with high I/O service latencies (for example, over 15 ms average I/O service time), it may be worthwhile to explore the viability of moving the temporary table areas to an EFD-based storage region.

Balancing EFDs on CX4 back-end buses

As discussed in an earlier section, it should generally be a configuration rule to ensure that data access is balanced over the different front-side and back-end storage buses. Each EFD delivers on average considerably more sustained IOPS and MB/s compared to rotating disks. Per the EFD vendors' performance specifications, EFDs often can deliver close to 200-plus MB/s, though such limits are not always practically reached per the nature of actual deployments. Nevertheless, closer attention should be paid to distribute the EFDs as evenly as possible across the back-end buses.

For example, if there are four back-end buses configured for the CX4-960 system, and there are two DAEs on each bus, the most logical approach, given a quantity of eight EFDs to be added to the system, is to try to add one EFD onto each of the eight DAEs. The second alternative may be to just put two EFDs on the first DAE of each bus.

It would not be a good idea of put all eight EFDs on one DAE on just one back-end bus, for example. The performance peak threshold that the eight EFDs can theoretically hit would far exceed what a single back-end bus can deliver, and the EFDs could not be driven anywhere close to their theoretical top performance level, limited by how much data can be moved between the storage processor memory and the drives through that one back-end I/O bus.

RAID choices for LUNs on EFDs

As mentioned earlier, for most DW deployments, data query processing efficiency generally tends to be the main focus. Consequently, storage data *read* performance is also the typical focus.

With rotating disks, because of the head seek latencies when we need to move around the drive to get at different data stored within a drive, the way to ensure the highest bandwidth of data retrieval from these drives would be to read in as large a chunk of data as possible before moving the drive head to a different position. This argues for going with “narrower” RAID striping choices, such as 2+1R5, discussed earlier.

In contrast, EFDs have no associated “repositioning of drive for data read” penalty. So, in the case of EFDs, it is reasonable to create a wider RAID striping configuration, such as 6+1R5 or 8+1R5, for example. This provides more effective usable capacity from the EFDs.

Wider RAID striping generally is less desirable for write-intensive workloads. There is higher likelihood of needing to do partial stripe writes, which would result in more parity read-adjust-write operations required. Also, in the event of a drive failure, it takes more resources, and longer time, for the failed drive content to be recovered through RAID rebuild.

With DW deployments, continual and high rate data changes are not as frequent. Most of the changes in the stored data are results of new data loads, and these tend to be often coalesced into larger size chunk writes into the storage. Even with wider RAID striping, the incoming data write chunks are often large enough, and clustered together, allowing the CX4-960 to drive full stripe writes to the back-end drives supporting the LUNs. In the case of EFDs, even if occasionally it does become required to perform smaller, parity adjusted data writes, the cost is also less prohibitive, as the parity data reread latency from the EFD is relatively low compared to rotating disks.

Hence the use of wider R5 is generally quite adequate for EFD-based LUNs in these type of deployments.

Because of the frequent need to do parallel chunk data reads from the different LUNs supporting the data warehouse, it is generally not a good idea to create multiple LUNs on the same RAID group from a set of rotating disks. The parallel concurrent reads in large data chunk against LUNs co-existing in the same RAID group, and from the same physical drive, results in severe disk head thrashing between the parallel data read streams, lowering the data retrieval rate achievable from the LUNs.

In contrast, it is generally a good idea to create multiple, smaller, LUNs on the same RAID group from the same set of EFDs. The more distinct LUNs that the host OS and the DBMS have visibility into, the more concurrent I/O requests can be presented to the array to be processed in parallel. This in turn ensures that each EFD in the RAID group gets the most number of simultaneous requests presented to the drive. Without any head movement, the maximum number of megabytes that can be pulled out of the drive per unit time is directly proportional to the number of requests presented to the drive per second. Generally, the more concurrent requests we can present to the drive, the higher will be the MB/s of data retrieval from each drive.

Cache settings for LUNs based on EFDs

The EMC CLARiiON default setting for LUNs bound on EFDs is NO READ/WRITE caching for LUN. For LUNs used in typical data warehouse deployments, that default should not be changed.

With data reads being the dominant I/O type, the advantage of having an incoming host read request being satisfied from the storage processors' read cache as opposed to going to the EFD for the data is very much reduced because of the low service latencies for retrieving the data from the EFD. Mostly, the chance of getting a data read cache hit in typical DW deployments is a result of the host read pattern that happens to trigger the CLARiiON predictive LUN data read ahead. General random data reads from the host against a large data set stored in the warehouse typically have a very low probability of getting a random rehit opportunity from read cache. When the data is stored in the EFD LUN, predictive read ahead from the drives into read cache memory (with the cached content now needing to be managed within the cache) shows little advantage compared to just dispatching the read request directly to the EFDs as needed, and avoids the extra storage processors' overhead for managing the cached data (and searching the cache to try to satisfying new incoming read requests).

Also, as stated earlier, the better use of the EFD-based LUNs inside the storage is to support the DW objects that have much higher hit rate and generally are modified with higher frequencies, such as indices, dimension cube entries, and others. These types of data objects in the warehouse tend to have far more typical single DB page (for example, 4-8 KB) read/write access patterns, where service latencies are really the key. Rather than shuffling these objects through the storage processor cache memory (and having to deal with possible contention with other stored data objects in the cache), it is often simply more effective to channel all read/writes directly to the drives.

High speed CX4-960 front-side access support

The UltraFlex I/O architecture allows the array to be upgraded readily to take advantage of the interconnect technology advances, including 8 Gb FC connects and 10 GigE iSCSI that are now becoming more widely adopted throughout the industry.

It is important to keep in mind that while high-speed interconnect technology advances allow data to be moved between the connection points more rapidly, it does not mean that one should necessarily be expecting to get *more* bandwidth delivered from a particular CX4-960 configuration because the front-side connections to the storage are upgraded from 4 Gb FC to 8 Gb, or from GigE to 10 GigE.

The analogy is like opening up more toll lanes on a freeway. Just because there are more lanes that vehicles can use to move through the toll machines, it does not follow that the county is going to end up collecting more toll revenue.

The total amount of data that can be pushed through a particular CX4-960 configuration, as discussed earlier, is determined by how all the components are balanced within the configuration. When, the configuration has reached a limit (for example, the distribution of data over a certain number of physical drives, on a certain number of DAEs and back-end buses, through the different I/O modules), the fact that we have upgraded all the pathways from the array SP front-side ports to the server hosts (where the I/O requests originate) to the higher bandwidth technology, does not necessarily imply that we will end up delivering more data to the servers per unit time.

In the typical DW deployment, the query processing logic of the DBMS system spends a certain amount of CPU time processing the data (needed by the query), while kicking off I/O, often in parallel, in the background, to “prefetch” the next set of data logically needed for the query. If the processing of the current batch of data is not completed until 20 ms have elapsed, then having the entire next batch of needed data delivered and be ready on the host buffers in 9 ms as opposed to 18 ms really does not affect the query responsiveness observable by the users.

Only in cases where the data link speed is the key limiting factor, such as when the processing typically can complete in 16 ms, and has to wait for 2 more ms before the next batch of data is ready to be used, would the upgrade from a 4 Gb connection to an 8 Gb connection result in “noticeable” improvement in the query processing time. (In this example, the query process would have shaved off 2 ms out of every 18 ms taken to process the query, or a 11 percent speed up in query time.)

The more direct advantage of upgrading to the higher interconnect technology is the simplification of the physical interconnect required, thereby reducing the management and support effort for that connection infrastructure.

Instead of needing to run four 4 Gb FC cables from the server into a SAN switch, zoning them to at least four other connections from the CX4-960 front-end FC 4 GB connections, in order to try to get close to being able to pull more than 1,500 MB/s of data from the array to the server, we can now upgrade the server to deploy with 8 Gb FC HBAs, and also reduce the number of front-side connections from the CX4-960 to two connections of 8 Gb. The total number of cables that have to be run and physically managed as part of the deploying infrastructure has just been reduced by half. The SAN switch (or switches) only needs to support four physical connections as opposed to eight. Consequently, the same set of SAN switches can be extended to support more server and storage system connections after the infrastructure upgrade.

Similarly, by leveraging the 10 GigE infrastructure and the recently added 10 GigE iSCSI front-side connect UltraFlex I/O modules, servers can now be connected to the same CX4-960, sharing the set of LUNs, and be able to drive I/O to the set of LUNs at a speed comparable to the other servers using 8 Gb FC connections.

The 10 GigE iSCSI connection option

The 10 GigE iSCSI connection option offers further interesting possibilities for simplifying the DW deployment infrastructure.

Since the introduction of the CX4 product line, the UltraFlex I/O architecture has supported the ability to allow servers to access any LUN inside the array with either the FC I/O protocol or iSCSI protocol. It was possible to have one server accessing a LUN using FC I/O, and another accessing the LUN through an Ethernet base connection structure doing iSCSI I/O.

However, sharing LUN I/O access in this mixed mode was not frequently done. The FC I/O will be going through a 4 Gb FC SAN connection infrastructure, while the iSCSI I/O would be driving through an IP-based infrastructure with a much lower limit on the maximum achievable bandwidth.

For two servers, such as an Oracle Real Application Cluster (RAC), to be clustered together to support a DW deployment service, the node running with the FC I/O infrastructure would be dominating the I/O system access to the storage, resulting in a high disparity between the ability of each node to effectively service database access requests. The cluster will therefore not scale well when moving from one node to a two-node configuration.

With the advent of 10 GigE iSCSI support, the raw speed disparity for pushing I/O down the 8 Gb FC channel from one server versus down an IP network pathway to the 10 GigE network using the iSCSI I/O protocol is now drastically reduced. It is therefore far more practical to augment an existing infrastructure that has been relying on the FC I/O connections with a new server leveraging purely a 10 GigE network connection, and expect the new addition to function quite effectively as a new member of this database service cluster.

Virtual network (VLAN) technology has been standardized in the IP world for a long time. The CX4-960 newly enhanced iSCSI connection supports 10 GigE iSCSI connections, with configuration options such as IP jumbo frames. VLAN tagging is also supported.

By creating virtual connection ports on the UltraFlex 10 GigE iSCSI connection module, it is possible to tag the server to array iSCSI I/O traffic to a particular VLAN ID. This ensures the data transmission security by enforcing data traffic isolation through the VLAN concept. The ability to create an operational quality-of-service distinction between traffic through the different VLAN classes also helps to ensure that under heavy load, the I/O traffic will not be unpredictably compromised as other network traffic travels through the different VLANs, riding on the same physical IP network, where unexpected spikes upward can result from exceptional operating situations.

It is generally acknowledged that IP-based network connectivity is more easily managed by IT staff, compared to trying to have a staff managing both IP and FC SAN networks. The very flexible connection architecture, and the phased-in 10 GigE iSCSI support, opens up all options to facilitate the establishing of a new database warehouse anchored on the CX4-960.

10 GigE iSCSI bandwidth expectation for read-intensive data warehouse deployments

When it comes to expected bandwidth through the iSCSI pathway, it is important to keep in mind that the practical bandwidth driven through the physical network connections is not just a function of the hardware itself. Whereas the 10 GigE cable may theoretically be able to allow close to 1000 MB/s of data packets to move through the wires, that does not imply that the server can sustain driving 1,000 MB/s of I/O through the CX4-960 front-side 10 GigE iSCSI connections.

Our engineering experience has generally led to the conclusions that more often than not, when host-side iSCSI software drivers are used, and the servers are not deploying expensive network interface cards with built-in intelligence to support TCP offloading (TOE) or additionally iSCSI frame offloading, all I/O requests being pushed through the server OS I/O layer tend to demand more CPU and memory resources from the server to drive the I/O requests and harvest the results. Consequently, with iSCSI software drivers such as Open-iSCSI drivers for Linux distributions and iSCSI drivers under MS Windows, we often would top out at around 450 MB/s of sustained I/O read rate through one iSCSI initiator to target connection. This tends to be less influenced by the physical hardware involved, though it does tend to be on the higher end (500-plus) with the more expensive network adapter hardware that may include TCP offload support.

With that in mind, when two or more 10 GigE NICs are used from the servers, it is recommended that the NICs should each be used as a separate iSCSI initiator connecting to the CX4-960. When each NIC initiates an iSCSI connection to the same CX4-960 iSCSI port, the storage processors will be supporting multiple connections against the array port. This connection approach allows more than the typical 450-500 MB/s of I/O bandwidth being pushed through a single 10 GigE iSCSI array port.

NIC teaming is often deployed on servers with multiple NICs to provide HA as well as network load balancing. However, a virtual network device from teamed physical NICs is still considered a single iSCSI connection initiator. So, a connection established between the server-side logical network device from a teamed physical NIC would still be just a single connection, and will typically be limited by the single connection bandwidth. So, teamed NICs should not be used as iSCSI initiators for connecting into the 10 GigE iSCSI front-side connection ports for the CX4-960.

With multiple connection sessions from different host-side NICs to the iSCSI port, each LUN will appear multiple times as distinctly addressable SCSI disk devices, even though these are really just different iSCSI paths from the different NICs to the same LUN through the same set of array ports. To ensure that the server applications are referencing the LUNs properly, host-side OS multipath support should be installed and configured. EMC PowerPath[®] multipath support is recommended, as this server capability is not only formally qualified thoroughly by the EMC interoperability testing process, but will offer the best performance for I/O access from all the qualified servers and OS to all of the EMC storage systems, including the CX4-960.

Summary of recommendations

Most of the recommendations discussed in the previous sections are based on experience with typical DW deployments. The assumption is that most of these deployments are query-oriented, with emphasis on multiple concurrent user queries, and that each complex user query is often broken down further into multiple parallel subtasks. The key to supporting such deployments effectively is to ensure that server and storage resources are properly balanced, so as much of the available supporting infrastructure resources can be brought to bear as possible.

To that end, the many innovative enhancements have raised that bar of balance significantly. Nonetheless, a good understanding of the actual deployment, and properly leveraging the different configuration options to provide the proper balance for the particular need identified, is obviously the key to success.

The following is a brief summary of the recommended configuration options discussed in the previous sections.

- Leverage the scalability offered by the UltraFlex I/O modules and size for the expected bandwidth. Keep front- and back-end port traffic balance with a comparable number of front- and back-end ports when possible.
- Choose parity RAID as opposed to mirrored RAID for most DW data LUNs unless there are obvious high rate data update requirements (as opposed to periodic new data loads).
- Consider 2+1 R5, 2+2 R6 “thin” parity RAID as long as the DBMS and the typical queries can drive parallel I/O read streams against all the LUNs. The thin RAID results in more distinct “data partition containers” that the DBMS can drive concurrent I/O streams against. The thinner striping results in more sustained MB/s of data accessed per drive supporting the RAID LUNs.
- With 2+1 R5 or 2+2 R6, if host read requests are sustained read requests that are typically 256 KB or higher, it would be reasonable to use 25 MB/s per drive as a relatively conservative guide for sizing the number of drives (and therefore RAID groups) for the storage system.
- Balance the number of drives evenly over the back-end buses. Use 720 MB/s as a general guide for expectable sustained data read rate per bus. Configure additional back-end buses with adding FC SLICs as needed.
- Distribute the data LUNs evenly across both storage processors.
- Focus cache for TEMP and logs. Data LUNs tend not to get as much advantage out of read cache. Consider disabling read cache and implicit LUN prefetches for data LUN. Disable write caching for data LUNs during loads only if the host writes can be guaranteed to be storage LUN stripe boundary aligned.

Conclusion

The CX4-960, the top end of the innovatively re-engineered CLARiiON system architecture, provides tremendous flexibility to support scaling and growth. This is a key attribute for any storage system to be able to support the type of rapid and often difficult-to-anticipate growth with today's business mission-critical data warehouses.

The system configuration considerations discussed in this paper are derived from EMC engineering experiences working with different DBMS systems supporting typical DW deployments leveraging the new CX4-960 systems. While it is too simplistic a view to assume that all DW deployments would fit under a single I/O profile, our joint engineering experience working with many of the key DBMS vendors and real-world customer DSS workloads has reinforced our belief that this system is an ideal choice as the storage system foundation to anchor any new mission-critical data warehouse deployment.

References

The following titles can be found on Powerlink, EMC's password-protected customer- and partner-only extranet. Access may be limited.

- *EMC CLARiiON CX4 Model 960 Networked Storage System* specification sheet
- *EMC CLARiiON CX4 Series Ordering Information and Configuration Guidelines*
- *Introduction to the EMC CLARiiON CX4 Series Featuring UltraFlex Technology* white paper