Redbooks Paper

JFS2/DIO Sequential Input/Output Performance on IBM pSeries 690

In this Redpaper we discuss the capabilities of the newest IBM® pSeries® 690 processor and its I/O subsystem. In the main section, we review the JFS2 (Enhanced Journaled File System) with Direct I/O implementation and provide details about our disk subsystem and disk adapters selections. We also provide guidance on selecting the proper RAID configuration, and give other performance and tuning tips.

In Appendix A, we explain array controller concepts; in Appendix B, we address FAStT setup concepts; and in Appendix C, we provide a brief overview of some of the available technologies and products that we considered for our benchmark, and explain why we chose a FAStT600 Turbo configuration for this sequential I/O workload benchmark.

Note: Because each environment is unique, and because technologies and capabilities are continuously changing, this discussion should only be used as a broad guideline of considerations to keep in mind when choosing storage.

The goal of this high performance I/O benchmark was to demonstrate that the IBM pSeries 690 has a very powerful I/O subsystem for handling sequential I/O workloads. The study shows that an appropriately configured pSeries 690 system, combined with a properly sized disk subsystem, is capable of sustaining 18.5 gigabytes per second (GB/s) sequential I/O when utilizing disk controller cache. When performing disk I/O, the server can sustain 16 GB/s aggregate sequential read or 14.8 GB/s aggregate write performance using either striped logical volumes or Enhanced Journaled File System (JFS2) using Direct I/O.

The team that wrote this Redpaper

Matt Accapadi is a Senior Technical Staff Member for IBM Server Group in Austin, TX. He has almost 17 years of experience in AIX®, and has specialized in AIX Performance for the last 13 years.

Mark Parmer is a Certified Senior Technical Sales Specialist for the eServer Brand at IBM. He has 21 years of experience in the technology field, spending the first seven years as a software engineer for IBM's Federal Systems Division and General Electric's Space Systems

Division, and the last 14 years as an IBM Systems Engineer/IT Specialist in the field. He specializes in large system tuning, application benchmarking, and selling the benefits of pSeries/AIX to the customers in his territory.

Tim Simon is a System Architect with IBM Sales and Distribution who works with customers in the US Western Region. He has 23 years of experience with IBM in various field technical support roles, and most recently has specialized in designing storage solutions for UNIX® and Windows®-based systems.

James Wang is a Senior Engineer with the IBM Design Center for e-business on demand™ in Poughkeepsie, NY. He is an IBM Certified IT Specialist with more than ten years of database experience in pSeries and AIX. He is certified in UDB 8.1, UDB for clusters, Oracle9i, SAP, UNIX/Oracle, AIX, HACMP™ and Business Intelligence. His areas of expertise include database system design, implementation, tuning and high availability.

Introduction

There are many applications today, in both industry and in the public sector, which read, write, and analyze large amounts of data with high throughput. Seismic processing in the oil and gas industry, credit analysis and prospect identification in the finance and banking industry, and image processing and digital video rendering in the media and entertainment industries are just a few examples.

To compete in this environment, a processing system requires both a high performance processing capability and a high capacity input/output infrastructure in order to provide the ability to quickly access the large volumes of data required.

Designing for performance

IBM develops systems with the goal of balanced performance, because it is of little value to have high performance processing in a system if the I/O subsystem is not capable of getting data into the system to be processed. In the IBM pSeries 690 subsystem, the I/O architecture is designed to provide high sequential I/O bandwidth to complement the high performance processing capabilities built into the system.

While it is possible to look at an I/O subsystem design and theoretically determine its throughput capabilities, this does not always present an accurate picture of actual system performance. Prior to this study there was no empirical data showing the system-level throughput of a large pSeries 690 loaded up with 2 Gbps fibre channel adapters and disks. A study of this type was needed to examine the actual performance and answer questions such as the following:

- Are there unseen bottlenecks in the hardware or operating system? If so, where are they?
- What is the actual maximum sequential I/O filesystem performance achievable? Can a single pSeries 690 system sustain more than 10 GB/s sequential read or write?
- Will the device drivers for the adapters experience contention in memory, or will lock contention in the file systems prevent the system from scaling?

Study goals

In this benchmark, a pSeries 690 system was assembled to provide the base for evaluating its sequential I/O capabilities. The processor we chose was the pSeries 690, which is the latest model of the pSeries family with 1.9 GHz POWER4+ processors. We used seven RIO2 drawers, each populated with 16 Fibre Channel adapters (IBM Feature Code 6239), to attach the disk subsystem.

Since most customer environments today typically include a SAN-oriented file system, with the ability to do very large sequential I/Os at very high data rates, we included IBM 2109 SAN switches in this pSeries 690 high performance I/O study. The 112 adapters were used to attach 56 FAStT600 disk subsystems using IBM 2109 Fibre Channel switches.

Using this test system, a series of performance tests were executed to provide I/O workload sizing information and to:

- Study the throughput capacity of a single RIO2 drawer, and profile its internal read and write capabilities and limitations
- Determine how system throughput scales as additional RIO2 drawers and adapters are added into the system, from one drawer up to the maximum of seven drawers
- Study the sequential throughput capacity of the system-wide striped logical volumes

- Study the JFS2 with DIO sequential throughput capacity of single file and the aggregate throughput
- Study JFS2 with DIO read-behind-write performance

IBM pSeries 690 system architecture

IBM extends the power of its flagship IBM eServer® pSeries 690 by adding the POWER4™+ 1.9 GHz processor–its fastest ever–and up to 1 TB of memory.

The 1.9G Hz POWER4+ processor options enable a broad range of pSeries 690 configurations. The 8-way 1.9 GHz Multichip Module (MCM) allows 8-, 16-, 24-, and 32-way pSeries 690 system configurations. The 8-way with 4-way active 1.9GHz Capacity Upgrade on Demand (CUoD) processor option enables clients to install standby processing capacity that can be activated permanently or temporarily to meet rapidly changing workload requirements. Capacity on Demand pSeries 690 configurations require that at least eight active processors be installed. The 128 GB memory card provides up to 1 TB of memory on the pSeries 690, doubling the capacity previously available.

The new pSeries 690 also includes the following, faster 633 MHz features that allow pSeries 690 systems to take full advantage of the performance of 1.9 GHz POWER4+ processors:

- 128 MB Level 3 (L3) cache
- Memory cards in increments of 4, 8, 16, 32, 64, and 128 GB
- Two-loop and four-loop Remote I/O-2 (RIO-2) adapters for attaching IBM7040-61D I/O Expansion Drawers

pSeries 690 I/O subsystem

The pSeries 690 has a leading-edge I/O subsystem that complements the POWER4+ CEC. Schematics for the pSeries 690, including the I/O subsystems for the POWER4++ systems, are shown in Figure 1 on page 5. The figure shows the pSeries 690 I/O subsystem default cabling, maximum bandwidth per PCI slot, all drawers double loop, 7 drawers maximum. The maximum I/O configuration is seven 7040-61D I/O drawers.

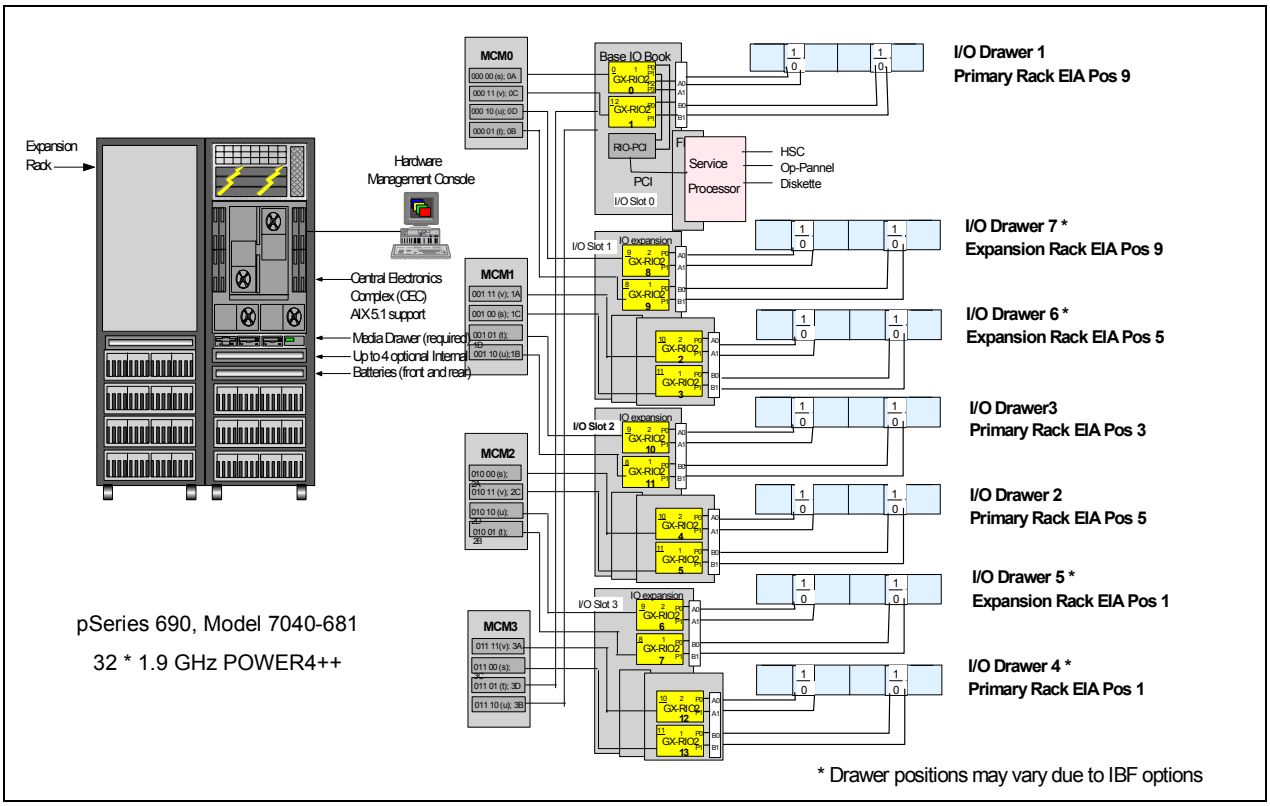

Figure 1 pSeries 690 I/O subsystem default cabling, maximum bandwidth per PCI slot, all drawers double loop, 7 drawers

RIO2 drawer benchmarked), is a 4 EIA drawer that contains support for 20 full-length PCI-X adapters and 16 disk bays. It contains two PCI I/O planars that have ten 64-bit PCI-X slots, and two integrated Ultra3 SCSI controllers each.

All of the slots are 64-bit, and support both PCI-X and PCI adapters. Each I/O planar has two RIO2 ports. One RIO2 port of each I/O planar is always connected to an RIO2 port on an I/O book in the CEC.

The other RIO2 can be connected to an RIO2 port on an I/O book in the CEC (see Figure 2 on page 6), or can be connected to other I/O planar. Each integrated Ultra3 SCSI adapter is internally hardwired to a 4-slot DASD back plane in the front of the I/O drawer, so that there are four groups of four hard drives.

The IBM pSeries 690 I/O architecture allows the bandwidth supported by the system to scale with the number of drawers attached. The total bandwidth required by a system will vary depending on the application. With the RIO2 I/O planar (FC 6571), 10 high performance adapters are supported per planar with 1300 MB/s maximum simplex bandwidth. System-wide, the pSeries 690 has the capability to sustain 18 GB/s simplex.

For this benchmark, each of the benchmarked RIO2 drawers is connected to an IBM 2109 F32 switch and 8 FAStT600 Turbo and EXP700 pairs.

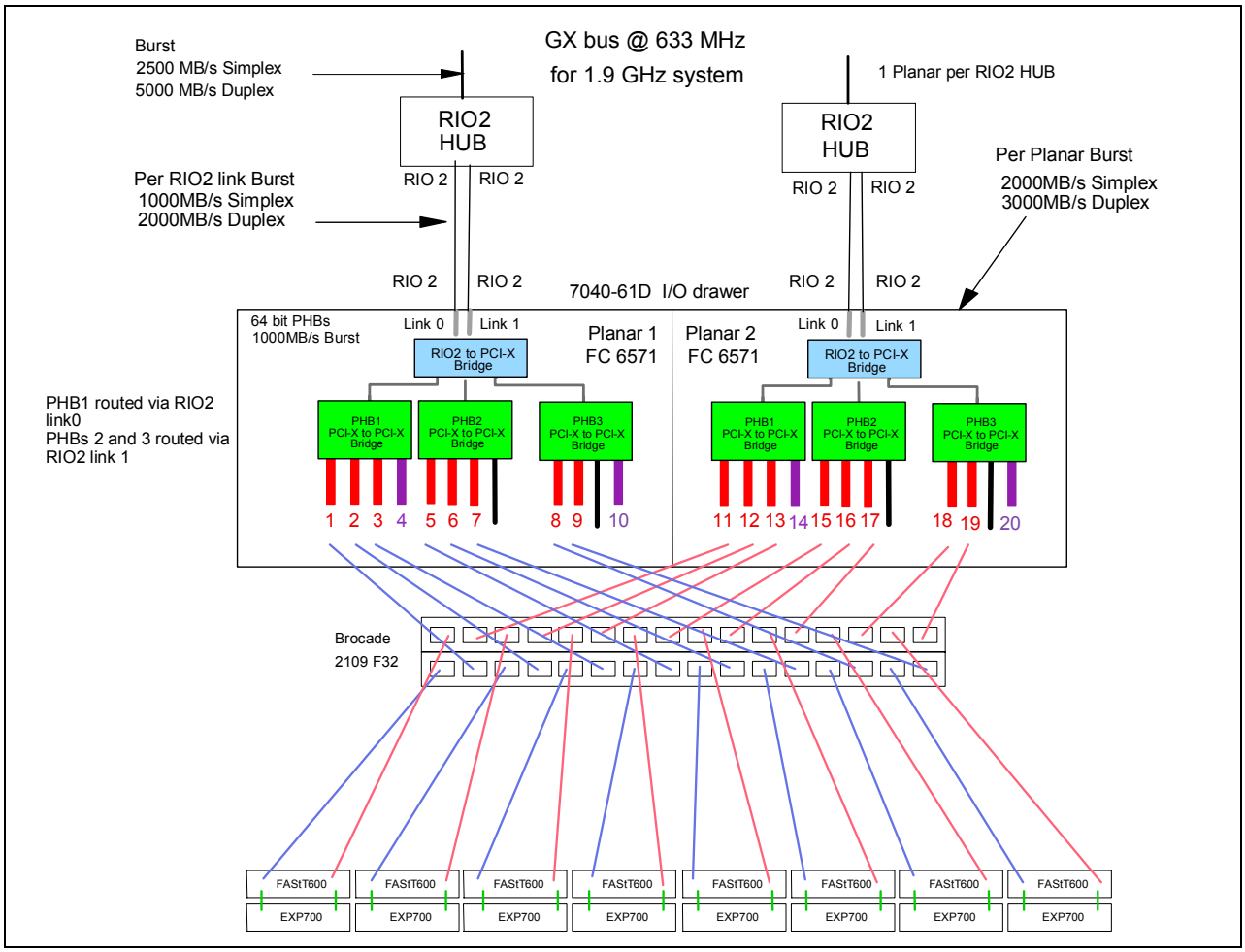

Figure 2 pSeries 690 I/O subsystem, I/O planars with default dual RIO2 loop cabling

Large pages support

The POWER4+ and POWER4 processor in the IBM pSeries 690 system supports two virtual page sizes. They support the traditional POWER™ architecture 4 KB page size and also a new 16 MB page size. AIX support of large pages, and how it was used in this benchmark, is discussed in “AIX support of large pages” on page 12.

Hardware configuration and sizing

In this section we describe the hardware configuration sizing justification of the fibre channel adapters selected, and explain why FAStT600 Turbo with RAID5 was chosen.

Hardware configuration benchmarked

The following hardware was used for this pSeries 690 I/O benchmark. The summary diagram is shown in Figure 3 on page 7.

- One pSeries 690 (IBM model 7040-681). The machine was equipped with four 8-way POWER4+ Turbo 1.9 Ghz processors, for a total of 32 processors. Memory consisted of eight 32 GB memory cards, for a total of 256 GB memory.

- One Hardware Maintenance Console (IBM Model 6792-LPU) was built with Release 3 Version 2.6, and Hardware Maintenance Console build level 20040113.1. The Hardware Maintenance Console was used to configure the pSeries 690 into one LPAR.
- Seven 7040-61D I/O drawers—Feature Code 6571 with dual RIO2 loop cabling.
- 56 FAStT600 turbo controller enclosures with 14 drives (36 GB, 15K RPM).
- 56 drive enclosures, EXP 700 with 14 drives (36 GB, 15K RPM).
- 112 one port 2 Gbps Fibre Channel HBA – Feature Code 6239.
- 16 HBAs per RIO2 drawer (slots 1, 2, 3, 5, 6, 7, 8, 9; 11,12,13,15,16,17,18,19).
- Seven Brocade 2109 F32 32 ports switches.

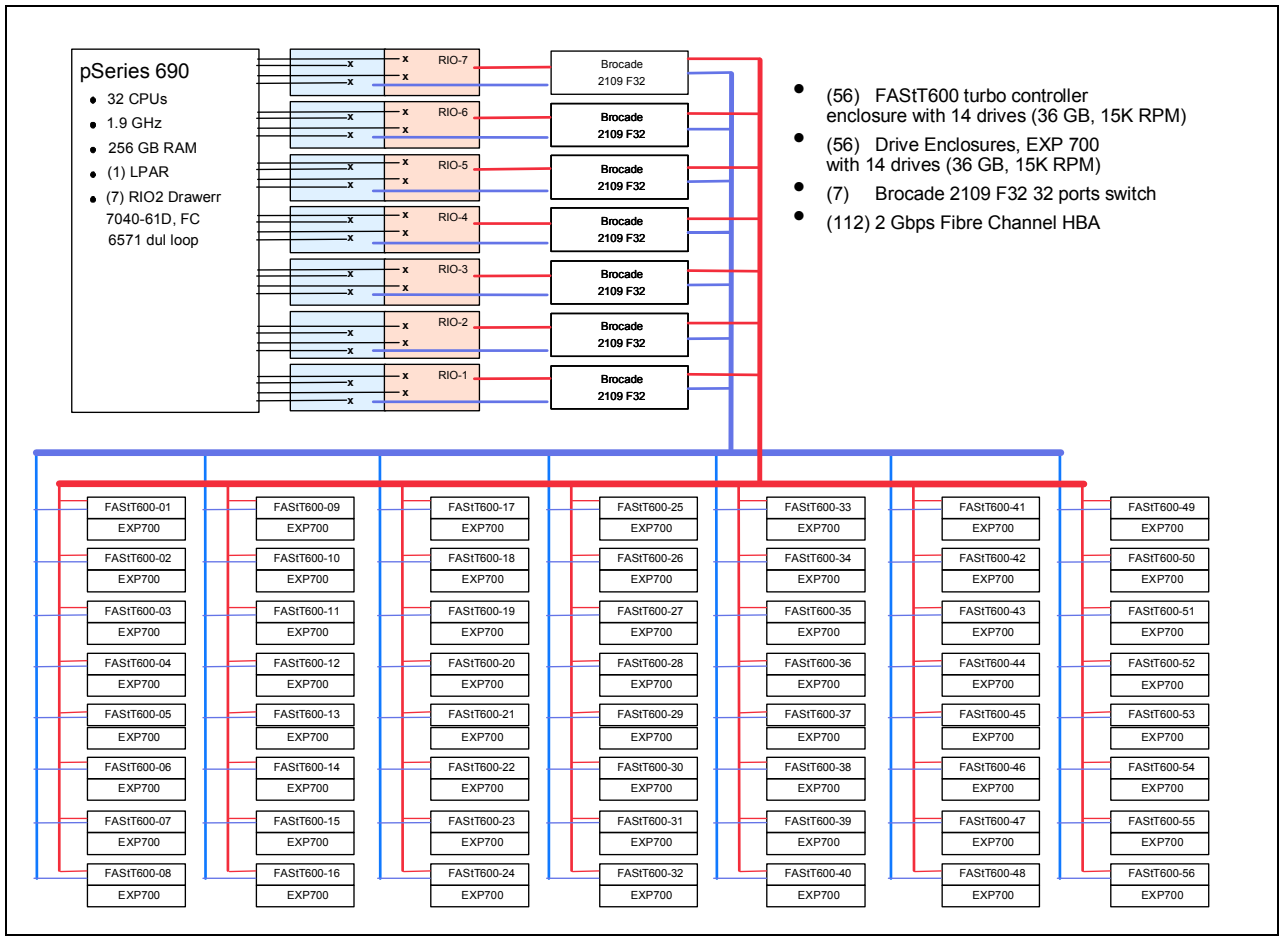

Figure 3 pSeries 690 I/O benchmark hardware configuration

Choosing the number and slot location of the Fibre Channel HBAs

Fibre Channel adapters are the bridges between the pSeries 690 hardware I/O subsystem and disks. If the number of adapters is undersized, then the aggregate adapter bandwidth becomes a system bottleneck. We used the following three steps to perform adapter sizing.

1. Determine the average adapter throughput

The current HBAs supported in the pSeries products are designed to operate at 2 Gbps. This means any adapter, given unlimited system resources, can transmit and receive data at approximately 200 MB/s.

However, when multiple HBAs within a system are all busy transferring data, other system limitations may become the bottleneck and lower the combined throughput of the HBAs; that is, the average 2 Gbps adapter throughput in an I/O-intensive system will be lower than the rated 200 MB/s. For sizing purposes, we use a more conservative 150 MB/second per adapter as our anticipated average throughput.

2. Determine the adapters required for each RIO2 drawer

The next step was to determine the number of HBAs that should be used in each RIO2 drawer. From engineering tests and the design specifications of the RIO2 drawer, we found that its theoretical maximum throughput was approximately 2.4 GB/s.

Using our planning number of 150 MB/second for each adapter, this said that we should plan on using 16 adapters in each RIO2 drawer.

3. Determine the RIO2 slots for adapter placement

For a 10-slot I/O planar tied to one GX bus, we expect 1200 MB/s total. For the 2 I/O planars per RIO2 drawer, which use two GX buses, we expect around 2.4 GB/s. After examining the design specification, we decided to place the HBAs in the RIO2 drawer in the following ways:

- Each RIO2 drawer has two planar boards with 10 PCI slots.
- Each planar has two connections back into a RIO2 hub, called link0 and link1. Each link can each support about 750 MB/s throughput, as a rule of thumb, for all of the devices attached.
- Within each planar there are three PCI Host Bridges (PHB), each supporting a group of PCI slots. Each PHB is designed to provide 450 MB/s, as a rule of thumb.
- RIO2 link0 feeds only PHB1. PHB1 supports slots 1, 2, 3 and 4 in planar0, and 11, 12, 13, and 14 in planar1.
  - Installing three Fibre Channel adapters in these slots will saturate PHB1 (assuming 150 MB/second per adapter as determined in step 1).
  - For this benchmark, we placed adapters at slots 1, 2, and 3 in planar1, and 11, 12, and 13 in planar 2.
- RIO2 link1 feeds both the PHB2 and PHB3 (slots 5, 6, 7, 8, 9, 10 in planar0, slots 15 through 20 in planar1, and the integrated SCSI ports).
  - Installing 5 adapters in these slots will saturate the RIO link (5 adapters at 150 MB/s or 750 MB/second).
  - For this benchmark, we selected PHB2 slots 5, 6, and 7 in planar1 and 15, 16, and 17 in planar2 to place adapters. For PHB3, we placed adapters at slots 8 and 9 in planar1, and at slots 18 and 19 in planar2.

Selection of FAStT600 Turbo for this sequential workload

To explore the pSeries 690's full I/O capability, we needed to select a storage environment that could best support the required data throughput. There are many devices available today for storing data in the pSeries environment, and our evaluation was very straightforward for this sequential I/O workload. After comparing the various options available, and considering all of the specific requirements (such as capacity, availability, manageability and cost), we determined that FAStT600 would provide the capacity and speed required at a price that would be attractive to the customers.

"Appendix C: Considerations for choosing storage" on page 38 contains a brief overview of some of the available technologies and products that we considered for our benchmark, and explains why we chose a FAStT600 Turbo configuration for this sequential I/O workload

benchmark. Because each environment is unique, however, and because technologies and capabilities are continuously changing, these discussions should only be used as broad guidelines in choosing storage for any specific environment.

Using a SAN-attached disk controller

Our primary goal in choosing a disk controller as the basis for our I/O subsystem was to choose a system that met our throughput requirements. However, a significant secondary goal was to create an environment that would be able to be adapted and installed across a broad group of customer environments.

Using a SAN-attached disk controller provided many benefits. For example, many customers are already using SAN technology and our solution could easily be integrated into those environments. Furthermore, the use of fiber in the SAN simplifies the cabling requirements and, by allowing extended distances between components, greatly simplifies the physical planning required to install the equipment.

In this study, we evaluated the IBM Enterprise Storage Subsystem (ESS) Model 800 and two members of the IBM FAStT controller family.

ESS Model 800

The ESS Model 800 offers specific advantages in the way it handles load balancing and failover across multiple physical fibers, which made it a good potential candidate for our study. However, the ESS is designed as a more monolithic system, and it lacks the modularity of the FAStT controller solutions. To support the 18 GB/s pSeries 690 bandwidth, we would have needed thirty ESS Model 800s, with each Model 800 supporting up to 525 MB/s sequential read. Its floor space and power requirements were too large for the space available for this study.

Comparing the FAStT Model 900 and FAStT Model 600 Turbo

The two models of the FAStT family of disk controllers that most closely met the requirements for this benchmark were the Model 600 with the Turbo feature and the Model 900. A more comprehensive overview of this family of disk controllers is provided in “Appendix C: Considerations for choosing storage” on page 38.

The Model 900, being the fastest model of the family, would initially appear to be the best choice for our environment, because each Model 900 can support up to 775 MB/second throughput in a sequential workload environment using four fiber connections into the SAN. The Model 600 controller provides approximately half of that throughput, using two fiber connections into the SAN.

The Model 900 controller drawer, however, requires disk expansion drawers to hold any disk drives in the system—while the Model 600 Controller drawer includes space for up to 14 disk drives in the controller drawer. This difference in design means that two Model 600 controller subsystems can be used to provide the same level of performance as a single Model 900. In addition, because of the integrated disks in the controller drawer, the Model 600 solution requires fewer disk drawers and less rack space.

When these factors were taken into account, the Model 600 solution was considered to be the more effective solution for this sequential workload environment.

The benchmarked FAStT600 Turbo configuration

A FAStT600 Turbo system can provide a sustained sequential throughput of 370 MB/s when using large I/O block sizes. In order to provide this throughput capability, a sufficient number of disks must be installed. Each FAStT600 drawer has slots to install 14 disks, but we had determined that 14 drives would not be enough to support the required throughput, so we

added one ESP700 drive enclosure to each FAStT600 Turbo system, to provide a total of 28 disks per system. We connected eight of these FAStT600 Turbo systems to each RIO2 drawer to support our target of 2400 MB/s.

Each FAStT600 Turbo was configured with two 2 Gigabit Fibre Channel adapters. Since these two 2 Gbps adapters are capable of sustaining 400 MB/s, they were more than adequate to support the FAStT600 Turbo 370 MB/s sequential read. The benchmarked RIO2 drawer is shown in Figure 2 on page 6, and the system configuration is shown in Figure 3 on page 7.

Selection of RAID5

The use of RAID5 was predetermined for this I/O benchmark. However, since there are 28 disks per FAStT600 turbo and EXP 700 pair, there are quite a few possible ways to create the RAID5 arrays. We needed to determine the number of RAID5 arrays to create and the number of disks to be used for each RAID5 array.

In the following section, we explain our rationale for selecting a RAID5 array.

AIX 5.2 limits 128 PV per volume group

From the sizing previously discussed, we concluded that the pSeries 690 needed 112 two Gigabit Fibre Channel adapters to drive the 56 FAStT600 turbo with EXP 700. To fully utilize all the adapter bandwidth, at least one RAID5 array had to be assigned to each of the 2 Gbps adapters.

Since our ultimate goal was to explore a single JFS2 aggregate sequential read and write bandwidth, this restriction also required all the RAID5 arrays to be included into one Volume Group (VG). However, AIX 5.2 has a 128 Physical Volumes (PV) limitation per VG. In order to stay within the limitation, each of the 112 adapters needed to be configured with only one RAID5 array. In other words, only two arrays could be used on each FAStT600 Turbo controller.

RAID5 8+P with 128 KB segment size gives the best I/O performance

There are many different ways to configure the RAID5 arrays out of the 28 disks available from each FAStT600 Turbo and EXP700 pair. But the RAID5 selection should satisfy the following constraints:

- Only two RAID5 arrays are allowed per FAStT Turbo and EXP 700 pair.
- The RAID5 configuration that you select needs to best match the AIX-supported 1 MB maximum logical track group (LTG) size to achieve the best hardware parallel striping.
- The combined throughput of all RAID5 arrays needs to be able to drive the pSeries 690 18 GB/s I/O bandwidth.

To fit the 1 MB LTG size, the most logical RAID5 configuration is either the RAID5 4+P with 256 KB segment size, or the RAID5 8+P with 128 KB segment size. These two RAID5 arrays can best fit the hardware striping when 1 MB data is evenly distributed among either the four disks in 256 KB segments, or the eight disks in 128 KB segments.

Note: Per a customer request, we also considered the RAID5 6+P with 256 KB segment size as a possible candidate, even though it does not satisfy the best hardware parallelism under the AIX-supported maximum 1 MB LTG. The only reason that the RAID5 6+P was suggested was because a drawer has 14 disks, and two RAID5 6+Ps is a perfect fit for a drawer of disks. The RAID5 6+P with 128 KB segment size was not considered because it will perform less efficiently.

Prior to the benchmark, we did a separate I/O test on a pSeries 630 to compare the FAStT600 Turbo read and write performance of these three candidates.

The preliminary study on p630 showed that although a single 2 Gbit Fibre Channel adapter can deliver close to 195 MB/s performance, the actual performance of both 2 Gbit adapters operating together under a FAStT600 Turbo and EXP pair does not linearly scale up to 390 MB/s. For this reason, the performance comparison was made using both controllers of the FAStT600 Turbo. The performance was measured using Xdd (the Xdd program is discussed in detail in “The Xdd I/O application” on page 14) with two threads, with each thread read or written against a RAID5 array with four MB application I/O blocks. The results are listed in Table 1 on page 11.

Note: The MB definition under Xdd is 1,000,000 bytes.

Table 1 The RAID5 performance comparison of a FAStT600 Turbo and EXP700 pair

FAStT600 Turbo + EXP700 Pair, 4 MB Application I/O Block	**2 RAID5 4+P, 256 KB Segment (10 disks used)**	**2 RAID5 6+P, 256 KB Segment (14 disks used)**	**2 RAID5 8+P, 128 KB Segment (18 disks used)**
Read, MB/s	331	347	357
Write, MB/s	246	274	285

The analysis reveals that the read performance of all three RAID5 configurations can meet the minimum 300 MB/s sequential read performance, which was calculated to support the 2400 MB/s RIO2 drawer performance target. However, RAID5 8+P LUNS deliver the highest write performance (285 MB/s) for each FAStT Turbo.

According to the results shown in Table 1 on page 11, if customers only need high read performance, then both RAID5 4+P and RAID5 6+P can still be considered. However, in order to explore pSeries 690 full I/O capability, we chose the RAID5 8+P with a 128 KB segment size for the I/O study.

Benchmark environment setup

In the following sections, we describe the setup we used for this benchmarking study.

Configure the pSeries 690 as one LPAR for highest performance

In SMP mode, AIX requires the TCE table to be continuous at the top of system memory. For maximum hardware speed, the GX TCE base address register (TCEBAR) is implemented as a simple mask, requiring the TCE table to be an even power of two in size and aligned to its size. These two interact to cause much of the memory taken by the TCE table to be unused for TCE entries.

To preserve customer investment in memory, the operating system is allowed to use the area not used by TCEs. The GX controller will “snoop” the TCEBAR region on behalf of I/O bridges below it and send invalidation operations if the TCEBAR region is altered.

In a worst case scenario, these invalidations can be sent to all GX buses in the system due to operating system usage of memory within the TCEBAR region. This, combined with transaction ordering restrictions on the invalidation operations, can have a negative effect on performance, manifested as higher system CPU usage.

For partitioned mode, the hypervisor maps the TCEs on behalf of the operating system, and the TCE table can be optimally placed to avoid the extra invalidation operations. A side benefit derived from running in LPAR mode is much quicker system rebooting, since only the operating system has to be reinitialized rather than the full I/O subsystem, and so on.

Operating system AIX 5.2 ML2

The pSeries 690 was built with the AIX 5.2 ML2 64-bit kernel.

Logical track group size 1 MB

The logical track group (LTG) is a contiguous block contained within the logical volume. It corresponds to the maximum allowed transfer size for disk I/O.

The LTG size can be set at the volume group. In the past AIX releases, only 128 KB was allowed. To take advantage of these larger transfer sizes and realize better disk I/O performance, starting from AIX 5L™, values of 128 KB, 256 KB, 512 KB, and 1024 KB for the LTG size are accepted. Much larger values will be available in future releases. The maximum allowed value is the smallest maximum transfer size supported by all disks in a volume group. The default size of creation of a new volume group is 128 KB.

In addition, the LTG should be less than or equal to the maximum transfer size of all disks in the volume group. To have an LTG size greater than 128 K, the disks contained in the volume group must support I/O requests of this size from the disk's strategy routines.

The mkvg SMIT screen shows all four values in the selection dialog for the LTG. The chvg SMIT screen shows only the values for the LTG supported by the disks. The supported sizes are discovered using an ioctl (IOCINFO) call.

Note: Before creating the volume group by using the mkvg command, the max_transfer_size of all hdisks needs to be set to 1 MB with the following command:

```
chdev -l hdiskX -a max_transfer=0x100000
```

An analysis that was completed prior to this benchmark showed that a larger LTG helps the large application block sequential read and write performance.

AIX support of large pages

Large page usage is primarily intended to provide performance improvements to memory access-intensive applications that use large amounts of virtual memory. These applications may obtain performance improvements by using large pages. The large page performance improvements are attributable to reduced translation lookaside buffer (TLB) misses due to the TLB being able to map a larger virtual memory range. Large pages also improve memory prefetching by eliminating the need to restart prefetch operations on 4 KB boundaries.

AIX support of large pages began with AIX 5.1.0.0-02. The large page architecture requires that all virtual pages in a 256 MB segment be the same size. AIX uses this architecture to support a “mixed mode” process model. Some segments in a process are backed with 4 KB pages, while 16 MB pages can be used to back other segments. Applications may request that their heap segments be backed with large pages. Applications may also request that shared memory segments be backed with large pages. Other segments in a process are backed with 4 KB pages.

In addition, there is further optimization for I/Os to raw LVM files when using large pages for shared memory (assuming that the application I/O buffers are in shared memory). With APAR

IY56002 for AIX 5.2, applications can benefit from this additional optimization for I/Os to JFS2 filesystems if using Direct I/O. Also with this APAR, the limitation to make the I/O buffers be in large page shared memory has been removed, so that the I/O buffers can use large page malloc/valloc buffers in the process heap as well.

Configuring AIX large pages

For this benchmark, we preallocated 96 GB of large pages with the **vmo** command and rebooted the system as shown:

```
vmo -p -o lgpg_size=16777216           # value in bytes for a 16 MB page
vmo -p -o lgpg_regions=6144             # number of regions (in this case, 96 GB)
bosboot -a; reboot -q
```

You can make an application use large pages for its heap/data area by either setting an environment variable or by using a link option (or having the application binary-edited), as described:

- To use an environment variable, set the env variable in the script that starts the I/O application:

  ```
  LDR_CNTRL=LARGE_PAGE_DATA=Y
  ```

  A value of Y says to use large pages as much as it can—but if an allocation fails for large pages, it will go ahead and use normal pages without an application failure. A value of M says that it is mandatory to use large pages, and the application will fail if not enough large pages are available.

  Note: We discovered that you have to use advisory mode (a value of Y) in order to benefit from the DIO optimization when using large pages.

- Alternatively, you can use the **-blpdata** link option when compiling the application, or edit the binary by using **ldedit -blpdata binary_name**

If using large pages for shared memory segments, the shmget() call should specify the SHM_LGPAGE | SHM_PIN flags. Also, **vmo -p -o v_pinshm=1** must be run, which indicates to the virtual memory manager that an application can use pinned memory for shared memory segments. The **–p** option makes it persist across reboots.

If using a non-root user id, then the user id must be given permission to use large pages:

```
chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE    username
```

The use of large pages can be monitored with the **svmon** command (either run **svmon** without any arguments, or use **svmon -P <pid>**). Usage can also be monitored by using **vmstat –l** and adding the other vmstat options (like the interval value).

FAStT600 benchmark settings

We used the following FAStT settings for this high performance I/O benchmark:

- RAID level: 5
- Segment Size: 128 KB
- Modification Priority: High
- Read Cache: Enabled
- Write Cache: Enabled
- Write Cache without Batteries: Enabled
- Write Cache with mirroring: Disabled
- Flush write after (in seconds): 10.00
- Cache read ahead multiplier: 8
- Enable background media scan: Disabled

- Media scan with redundancy check: Disabled
- Cache block size: 16 KB
- Cache flush settings: start = 50% stop = 50%
- AVT turned off

The Xdd I/O application

Xdd is a tool for measuring and characterizing disk subsystem I/O on single systems and clusters of systems. It was designed by Thomas M. Ruwart from I/O Performance, Inc. to provide consistent and reproducible performance of a sustained transfer rate of an I/O subsystem. It is a free software program distributed under a GNU General Public License. Xdd creates one thread for every device or file under test. Each I/O operation is either a read or write operation of a fixed size known as the “request size”.

Multiple passes feature

An Xdd run consists of several “passes”. Each pass will execute some number of I/O requests on the specified target at the given request size. In general, each pass is identical to the previous passes in a run with respect to the request size, the number of requests to issue, and the access pattern. Passes are run one after another with no delays between passes unless a pass delay is specified. Multiple passes within an Xdd run are used to determine the reproducibility of the results.

In this benchmark, three passes were used for each measurement.

de-skew feature

Xdd also has a “de-skew” feature. During the de-skew window, all targets are active and transferring data. The amount of data transferred by any given target during the de-skew window is simply the total amount of data it actually transferred minus the data it transferred during the front-end skew period, and minus the data it transferred during the back-end skew period. The de-skewed data rate is the total amount of data transferred by all targets during the de-skew window, divided by the de-skew window time.

In this benchmark, the de-skewed feature is used to show the highest sequential read or write throughput.

shmget/shmat or valloc/malloc memory allocation feature

The Xdd program allows two ways to allocate the memory:

- valloc/malloc

  malloc is a general purpose memory allocation package. The subroutine returns a block of memory of at least the number of bytes specified by the Size parameter.

  valloc has the same effect as malloc, except that allocated memory is aligned to a multiple of the system page size.

- shmget and shmat

  shmget and shmat allow processes to explicitly map files directly into memory, to avoid buffering and system call overhead. They are used by multiple processes/threads to share data.

  shmget allows applications to get or create a shared memory segment.

  shmat attaches a shared memory segment to the address space of a process.

In addition, Xdd is coded to use AIX shared memory as an option; it creates shared memory segments with the SHM_LGPAGE and SHM_PIN flags.

In this benchmark, we tested both valloc/malloc and shmget/shmat. For the valloc/malloc case, we set the LDR_CNTRL environment variable so that large pages would be used. For the shmget/shmat case, we set the v_pinshm VMM tuning parameter to 1 by using the command **vmo –p –o v_pinshm=1**.

In both cases, the use of large pages exploited the benefit of the additional optimization available for the raw LVM path and in the JFS2 Direct I/O path (DIO or CIO).

Read-behind-write feature

Xdd also has a read-behind-write feature. For the same target, Xdd can launch two threads: a writer thread, and a reader thread. After each record written by the writer thread, it will block until the reader thread reads the record before it continues.

pSeries 690 RIO2 drawer internal bandwidth profiling

The profiling of the RIO2 drawer internal bandwidth not only marks the highest possible throughput that the RIO2 can handle, but also reveals the bottlenecks. In this benchmark, we chose RIO2 drawer 1 and measured the sequential I/O performance of the following hardware components one at a time in the dedicated pSeries 690 environment (refer to Figure 2 on page 6 for the location of each hardware component):

- 2 Gigabit Fibre Channel adapter
- PHB1, PHB2 and PHB3 of the PCI-X to PCI-X Bridge
- link0 and link1 of the RIO2 to PCI-X Bridge
- planar 0 and planar 1
- RIO2 drawer
- Scalability of RIO2 drawers from 1 to 7

The RIO2 drawer internal bandwidth profiling can only be done if there is adequate hardware attached to the RIO2 drawer; in other words, sizing of the number of Fibre Channel adapters and disk subsystems must be done properly. The maximum hardware component throughput can only be measured if the aggregate bandwidth of the all adapters and disk subsystem (including controllers) exceeds what the hardware can sustain.

The hardware internal bandwidth can be profiled by either reading or writing two copies of the rhdisks or raw devices (logical volumes) from all the disks under the profiled component. Reading or writing two copies of the same rhdisks or raw devices forces a high level of FAStT controller cache hits, which allows more data transfer. In this benchmark, we used Xdd to read or write with 4 MB application I/O blocks. Three passes were measured for each data point, and only the average I/O rate is reported.

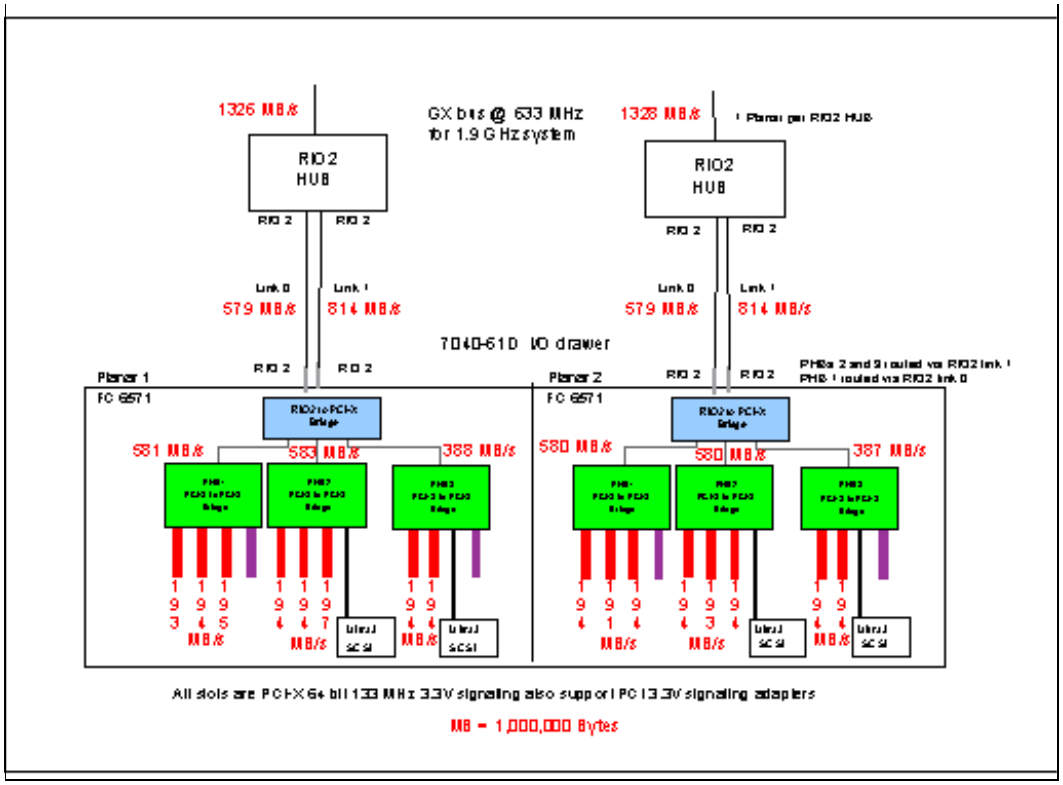

Figure 4 pSeries 690 RIO2 internal simplex bandwidth profile

For example, to profile a 2 Gbps Fibre Channel adapter bandwidth, we used Xdd to read or write two copies of rhdisk from the associated RAID5 array. Similarly, to profile the sequential I/O bandwidth at PHB1 level, we used Xdd to read or write two copies of each of the three rhdisks associated with the profiled PHB1. The same technique was used to characterize the hardware bandwidth at the link level and the drawer level.

The RIO2 internal simplex bandwidth profiling is shown in Figure 4 on page 16. If a single adapter is measured in the system, it shows that each of the 2 Gbps FIbre Channel adapters can sustain around 195 MB/s read or write.

At the PHB level, since there are three adapters under each PHB1 or PHB2, the measurement shows it can sustain around 585 MB/s sequential read or write. Since we placed only two adapters under PHB3, the profiling shows it can support around 390 MB/s simplex. All PHB measurements show that each 2 Gigabit Fibre Channel adapter needs to sustain 195 MB/s.

Although link0 is a 1 GB/s connection, it can saturate at about 820 MB/s. Since link0 only handles PHB1, its simplex profiling agrees with the PHB1 measurements (which is around 585 MB/s). That is, the throughput of the link0 and PHB1 path is restricted by the PHB1 sustain rate. Adding the fourth adapter to PHB1 may not improve the bandwidth.

On the other hand, since link1 feeds both PHB2 and PHB3, the bottleneck is in the link1 speed. Our profiling shows it is capable of 814 MB/s. Since link1 simplex throughput has been saturated, adding additional adapters under either PHB2 or PHB3 cannot improve the overall bandwidth. This is why we placed only three adapters under PHB2 and two adapters under PHB3.

At the planar level, the engineering projected speed is about 1350 MB/s simplex. The profiling shows it is capable of 1332 MB/s.

After the detailed profiling of RIO2 drawer 1, we also measured the read and write bandwidth of each of the seven RIO2 drawers. The profiling shows each RIO2 drawer can sustain 2650 MB/s simplex, as shown in Table 2.

Table 2 RIO2 profiling at drawer level and system level

RIO2 Drawer, Location Code	Read Profile MB/s	Write Profile MB/s
Drawer 1, U1.9	2655	2651
Drawer 2, U1.5	2656	2654
Drawer 3, U1.1	2656	2652
Drawer 4, U1.13	2657	2650
Drawer 5, U2.1	2649	2660
Drawer 6, U2.5	2656	2650
Drawer 7, U2.9	2656	2651
All 7 RIO2 Drawers	18558	18512

After the detailed profiling of RIO2 drawer 1, we also measured the read and write bandwidth of each of the seven RIO2 drawers. The profiling shows each RIO2 drawer can sustain 2650 MB/s simplex, as shown in Table 2.

The sequential read/write profiling of the pSeries 690 with seven RIO2 drawers reveals that the system is capable of 18512 MB/s simplex. Dividing the seven-drawer measurement 18512 MB/s by 7 gives 2645 MB/s, which is the single RIO2 drawer sustained throughput. This shows that the RIO2 bandwidth is linearly scalable from 1 to 7 drawers.

pSeries 690 striped logical volume performance

In previous section, we profiled the RIO2 internal bandwidth by reading and writing data mostly from the FAStT controller cache. Since most data is transferred from cache memory, not much disk I/O was exercised. In this section, we focus on the disk I/O performance by exploring AIX striped logical volume performance.

To trigger disk I/O, the application needs to at least read or write data several times of the 112 GB FAStT600 Turbo controller cache available from the 56 FAStT600 Turbo. In this part of the measurement, each measurement reads or writes multiple terabytes.

Each of the 112 RAID5 8+P arrays is a physical volume (PV) under AIX. There are many ways to create volume groups and striped logical volumes. However, in order to get a performance reference point for the “single large” JFS2 study, we decided to create only one system-wide volume group to include all RAID5 arrays, and create striped logical volumes horizontally spread across all RAID5 arrays.

In the following steps, we describe how we created the striped logical volumes.

System-wide striped logical volume

Since our main focus was to study the aggregated sequential I/O performance of a “single large” JFS2 with DIO, in this section, we only studied the performance of “system-wide” striped logical volumes. The aggregated sequential I/O performance of several striped logical volumes together—although not completely matching the JFS2 concurrent read or write

behavior—is the closest approximation. The intent of studying the raw striped LV performance is to reveal the JFS2 with DIO overhead over raw devices.

The striping mechanism, also known as RAID 0, is a technology that was developed to achieve I/O performance gain. The basic concept of striping is that in the write phase, each data operation is chopped into small pieces (referred to as a “striped unit”, or “chunk”), and these chunks are written to separate physical volumes in parallel. In the read phase, these chunks are read from those separate physical volumes in parallel, and re-assembled into the actual data.

Note the following:

- From the high-availability point of the view, the striping function does not provide redundancy by any means, except for the mirroring and striping function.
- From the performance point of view, a slow physical volume or RAID5 array may slow down the overall striped LV performance a great deal.

One volume group with 1 MB LTG size

Since a striped LV can only be allocated across *all* available physical volumes within a volume group, we created one volume group to include all the available RAID5 arrays. The volume group was 10.8 TB in size with a 512 MB Physical Partition (PP) and a 1 MB logical track group (LTG) size. In this volume group, we created 16 striped logical volumes. Each striped LV was 689 GB in size, and was spread across all the RAID5 arrays available.

Stripe width: all available RAID5 8+P arrays

The number of physical volumes that accommodate the striped logical volume is known as the “stripe width”. In this benchmark, all striped LVs were allocated across all 106 RAID5 arrays available at the time of measurement.

Stripe size: 1 MB

When the application accesses the striped logical volumes, the storage area for each physical partition is not used contiguously. These physical partitions are divided into chunks. The chunk size may be 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB or 1024 KB. The size is determined at the creation of the striped logical volumes, and cannot be changed afterward. The chunk size is also called the “stripe unit size” or the “stripe length”.

For this benchmark, since the application sequential I/O block size can be very large, we used a 1024 KB stripe size to take advantage of the LVM striping capability.

Combining LVM striping and RAID5 striping

To achieve high degrees of parallel I/O, the application takes advantage of two layers of striping. The first layer of the striping is at the LVM level. With LVM striping, a large application block can be parallelized into 1 MB stripes and distributed to all the physical volumes (RAID5 arrays) associated with the striped LV.

The second level of striping is at the hardware level. After each RAID5 array receives the 1 MB from LVM, it further breaks down the data into hardware stripes and distributes to disks in parallel. In our case, since the RAID5 is configured as 8+P, the 1 MB data received from LVM was distributed as eight 128 K segment to the eight disks in the RAID5 array. For very large I/O block sizes, this is the most efficient way to do parallel I/O.

Striped logical volume performance

To study the raw device performance, we created a volume group of 10.8 TB with a 512 MB Physical Partition (PP) size.

As shown in Figure 5 and Figure 6, the highest single striped LV read performance is 3816 MB/s read and 3542 MB/s write with application read or write of 256 MB block size. The highest system read performance is 15926 MB/s with 16 striped LVs concurrently read, and 14506 MB/s with 16 striped LVs concurrently write. Xdd treats each striped LV as a target, and it reads or writes each target concurrently with a separate thread.

Note: The MB definition under Xdd is 1,000,000 bytes.

In this part of the study, we had only 106 RAID5 arrays available for our study. If all 112 RAID5 8+P arrays can be used, the aggregated throughput shown in Figure 5 and Figure 6 should be another 5% higher.

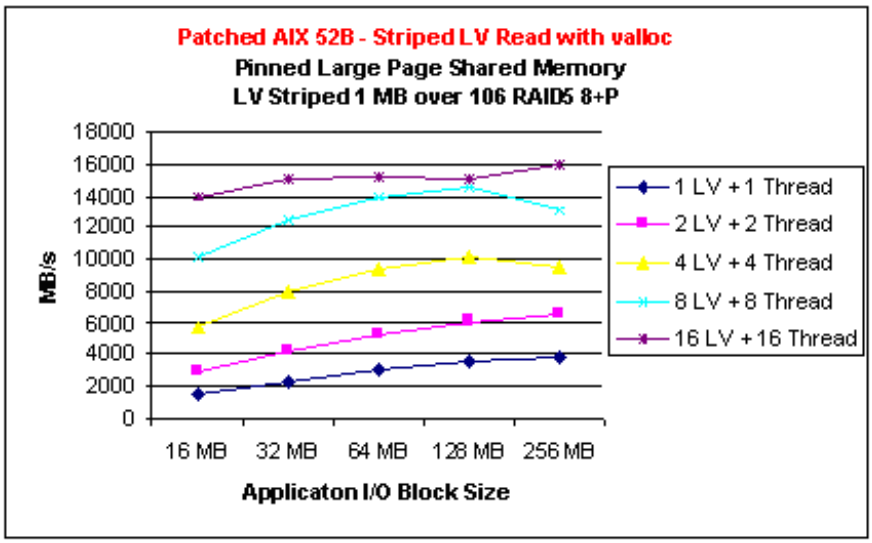

Figure 5 Striped logical volume read performance

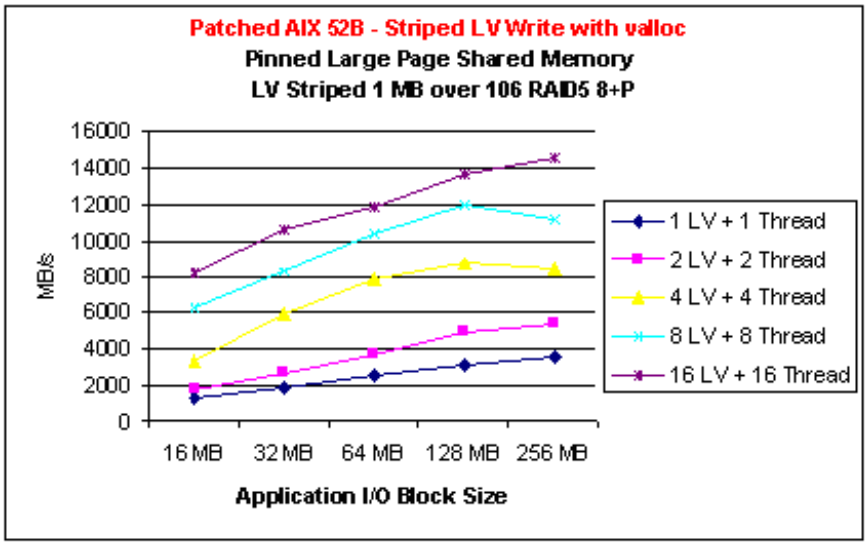

Figure 6 Striped logical volume write performance

The JFS2 filesystem

JFS2 is optimized for a 64-bit environment, and it provides high performance, reliability, and flexibility. JFS2 allows a system administrator to dynamically increase the size of the filesystem while it is being used. In AIX 5.3 (due to be available in 2004), the capability to dynamically shrink the size of a filesystem will be available without taking the filesystem offline.

JFS2 is architected for filesystems up to 4 Petabytes, but it has currently only been tested up to 16 Terabyte-sized filesystems. Also, file sizes are limited to 16 Terabytes. The number of inodes that can be created in a filesystem is dynamic and is only limited by the amount of free space in the filesystem.

JFS2 supports buffered I/O, synchronous I/O (the file is opened with O_SYNC or O_DSYNC flags), kernel asynchronous I/O (through the use of the Async I/O system calls), Direct I/O (on a per-file basis if the file is opened with O_DIRECT, or on a per-filesystem basis when the filesystem is mounted with the dio mount option), and Concurrent I/O (on a per-file basis if the file is opened with O_CIO or when the filesystem is mounted with the CIO mount option).

Direct I/O (non-buffered I/O) bypasses the file caching that is done by the Virtual Memory Manager (VMM). This elimination of the VMM layer saves CPU overhead and prevents memory from being filled up with file pages and potentially causing page replacement from occurring. Direct I/O is useful when the pages being read or written will not be re-used immediately, that is, when it is not expected to get file cache hits while accessing file pages. Using Direct I/O also bypasses the JFS2 readahead algorithms, but this can be compensated for by issuing large-sized I/Os or through the use of asynchronous I/O.

The use of Direct I/O also requires that the application's I/O size is a multiple of the filesystem block size. The JFS2 filesystem supports filesystems that can be created with 512-byte to 4096-byte blocks. Any I/O that does not meet the block size requirement will be demoted and will go through the file cache. However, this is done only on a per-request basis and will not cause permanent demotion; therefore, if other I/Os are a multiple of the filesystem block size, then those I/Os will go through the Direct I/O path.

The application can query the filesystem requirements for Direct I/O through the ffinfo() system call. The application can either open the file with the O_DIRECT flag specified in the open() system call, or it can rely on the mount option dio. To use the mount option, specify **`mount -o dio`** when mounting the filesystem, or else specify **`dio`** in the options line in /etc/filesystems.

Direct I/O can also be restricted to a subset of the files in a filesystem by placing the files that require Direct I/O in a separate directory and then using a named mount in order to mount this subdirectory over that filesystem. For example, if the files that need to be opened with Direct I/O are in a directory called direct files which is in a filesystem called /fs1, then the following mount option can be used:

```
mount -v namefs -o dio /fs1/direct_files  /mydirect_files
```

If a directory called /mydirect_files is created, then the subdirectory in /fs1 called direct_files can be mounted over /mydirect_files using Direct I/O, but the rest of /fs1 filesystem can be mounted without using Direct I/O.

By default, JFS2 serializes accesses to a file using inode locking. Multiple reads can occur simultaneously to a file, but only one write at a time can be done.

If an application is doing its own serialization (such as a database application), then the filesystem does not need to serialize at the file level using the inode lock.

The Concurrent I/O (CIO) feature of JFS2 provides the capability to not do inode locking of a file except in the case where the inode itself needs to change (such as when the file size is changing). Concurrent I/O also will implicitly use the Direct I/O path; therefore, CIO is the equivalent of Direct I/O without inode locking.

To enable CIO, the application can open a file with the O_CIO flag in the open() system call, or the filesystem can be mounted with the CIO mount option. The use of named mounts can also be done with Concurrent I/O as with Direct I/O mounts. Application I/O size and alignment requirements with CIO are the same as with DIO.

JFS2 with DIO sequential read or write performance

Very large sequential read or write applications derive no benefits from the JFS2 file buffer cache. Some technical workloads, for instance, never reuse data due to the sequential nature of their data accesses, resulting in poor buffer cache hit rates. Therefore, the extra CPU overhead of doing buffered I/O can be eliminated using Direct I/O for large sequential reads or writes.

JFS2 with the DIO mount option was used in this benchmark to explore the full pSeries 690 I/O capability. Since each measurement reads or writes multiple terabytes of data, with a 256 GB total memory, we anticipate no data reuse from the file system buffer cache.

In addition, since most customer large sequential I/O workloads tend to use large application I/O block sizes to reduce I/O overhead, we believe that JFS2 with DIO, which can fully exploit both LVM striping and RAID5 striping, is the best way to do parallel I/O.

For this part of the performance study, we created one 10.8 TB volume group with a 512 MB Physical Partition (PP) and 1 MB logical track group (LTG). In this volume group, only one striped logical volume was created. The benchmarked JFS2 was built upon this striped LV and we mounted it with the direct IO (DIO) option.

JFS2/DIO 1 MB striped read performance

The read performance of JFS2 with the DIO mount option is shown in Figure 8. The single file read performance varies from 1.7 GB/s to 4.6 GB/s. As the application block size gets larger, the read performance improves.

To study the aggregated JFS2 with DIO throughput, we used Xdd to concurrently read or write from different files. The aggregate file system read throughput peaked at 15977 MB/s when 16 files were read simultaneously. In this benchmark configuration, due to availability, we only used 106 RAID5 8+P arrays. If all 112 arrays can be used, we project that the aggregate read performance may be 5% higher.

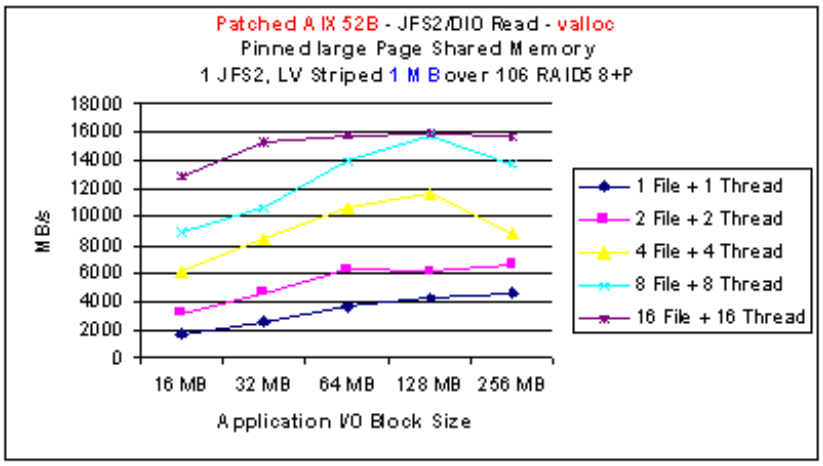

Figure 7 JFS2/DIO 1 MB striped read performance

As shown in Figure 7, read performance peaks when the application issues 128 MB block I/O. Since the JFS2 filesystem was built upon the 1 MB striped LV, the 128 MB application block size can be efficiently distributed to the 106 RAID5 arrays with 1 MB per RAID5 array. The 1 MB at the RAID5 array layer is further parallelized into eight 128 KB segments. The most ideal application I/O block size would be 106 MB, but this block size is not supported by Xdd, which requires the block size to be power of 2.

Figure 8 shows “nmon64” throughput monitoring results of JFS2 with DIO with 16 files concurrently read with 128 MB block size; the 16 files read sustained at 16 GB/s.

Note: The KB definition under “nmon64” is 1024 bytes.

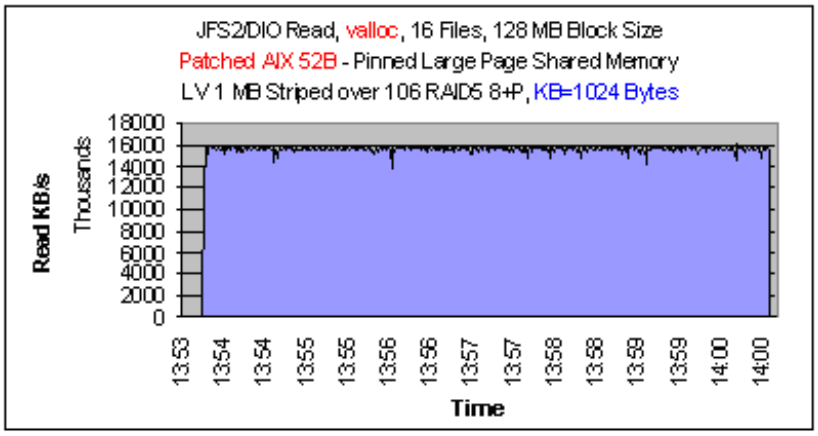

Figure 8 JFS2/ DIO 16 files concurrent read sustained at 15.9 GB/s

Figure 9 on page 23 shows that even under 16 GB/s system throughput, the CPU system time is as low as 14.5%. This allows the application to use most of the CPU cycles to do other tasks, such as computation, thereby demonstrating that pSeries 690 with JFS2/DIO provides a very balanced system environment.

Without using the large page pinned shared memory, the CPU system time can be as high as 70%.

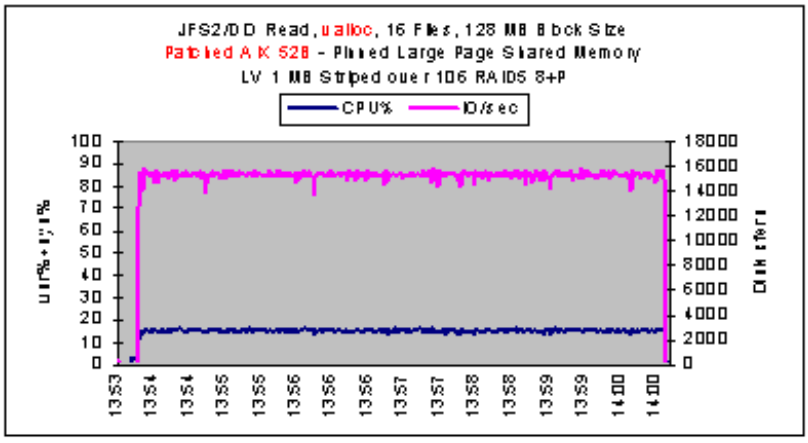

Figure 9 JFS2/DIO 16 files concurrent read used only 14.5% CPU system time

JFS2/DIO 1 MB striped write performance

The write performance of JFS2 with the DIO mount option is shown in Figure 10. Its single file write results are slightly lower than the corresponding read measurements, and its throughput varies from 1.4 GB/s to 4.3 GB/s. As the application block size gets larger, the write performance also improves.

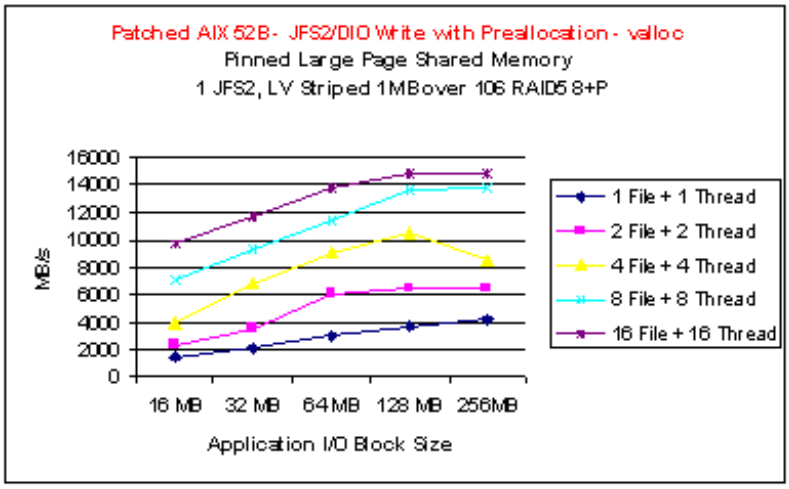

Figure 10 JFS2/DIO 1 MB striped write performance

The “sweet spot” of the aggregate write performance again occurs when Xdd uses a 128 MB application I/O block size.

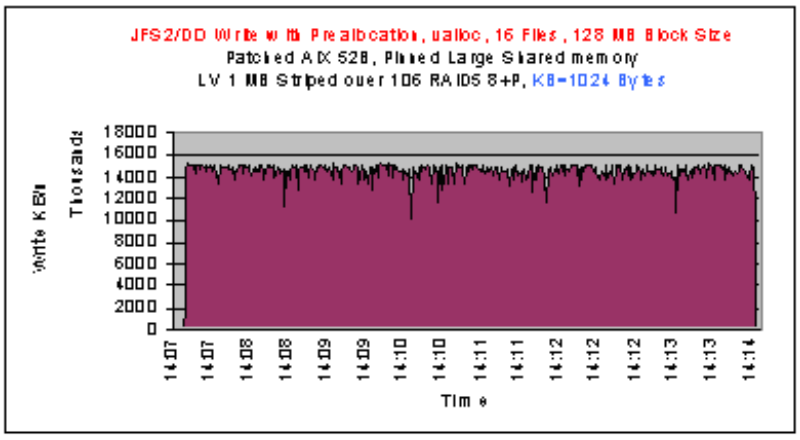

Figure 11 JFS2/DIO 16 files concurrent write sustained at 14.9 GB/s

Its “nmon64” throughput monitoring results, displayed in Figure 11, show that the sequential write with pre-allocation can sustain 14.8 GB/s with 12.9% CPU system time; see Figure 12.

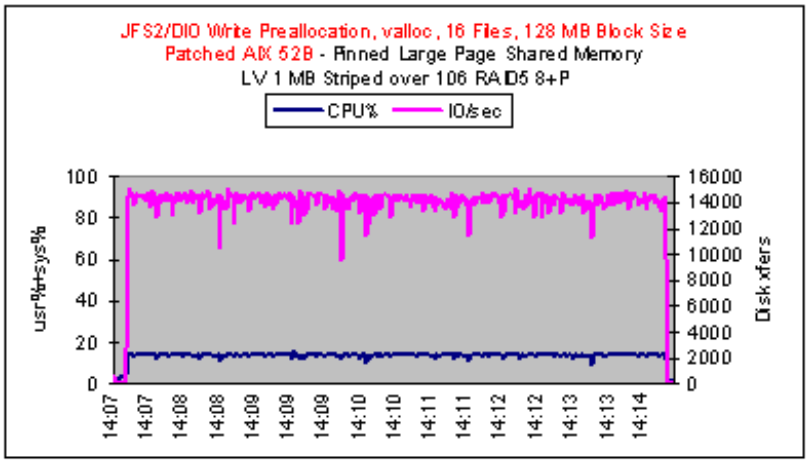

Figure 12 JFS2/DIO 16 files concurrent write uses 12.9% CPU system time

FS2 with DIO read-behind-write performance

Read-behind-write is one of the schemes used by high-end customers to shorten the write-to-read time. The term “read-behind-write” means that once the writer starts to write, the reader will immediately trail behind to read. The idea is to overlap the write time with read time.

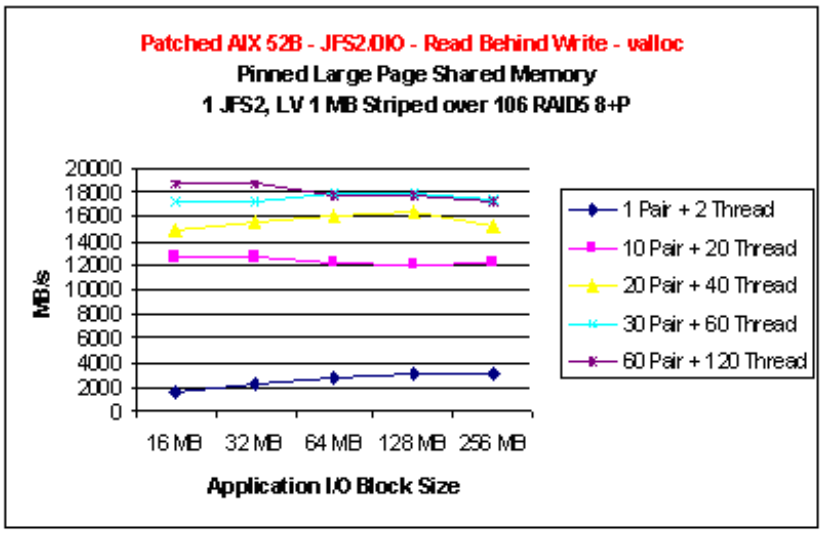

Figure 13 JFS2/DIO read-behind-write performance; reader reads one record after writer writes one record

This concept can be quite useful in overcoming the limitations of a slow I/O machine. For a high I/O throughput machine such as pSeries 690, however, it may be worth considering first parallel writing the entire file, and then reading the data in parallel.

There are many ways that this read-behind-write can be implemented. In the scheme that has been implemented by Xdd, after the writer writes one record, it will wait for the reader to read that record before the writer can proceed. Although this scheme keeps the writer and reader in sync just one record apart, it takes system time to do the locking and synchronization between writer and reader.

If the customer does not care how many records lag behind the writer, then you can implement a scheme for the writer to stream down the writes as fast as possible. The writer can update a global variable after a certain number of records have been written. The reader can then pull the global variable to determine the number of records written by the writer and play “catch up”.

Since both writer and reader cannot stream without a barrier, the performance shown in Figure 13 does not represent the true pSeries 690 duplex bandwidth. Each pair of read-behind-write has only one target or one file. Since both writer and reader need to write to and read from the same file, Xdd will launch two threads for each pair of read-behind-write.

As shown in Figure 13, with 60 pairs and using 120 threads, the highest duplex pSeries 690 I/O throughput is 18.7 GB/s with 9.35 GB/s write and 9.35 GB/s read.

Conclusions

IBM delivers a high performance native filesystem with AIX. IBM pSeries 690, using the current RIO2 drawers, more than doubles the I/O performance of the first generation pSeries 690 using RIO drawers. The pSeries 690 is able to sustain the maximum performance of an AIX filesystem, with very low system CPU overhead. The scalability to “near raw” hardware performance gives you the option of using this native filesystem to meet almost any high

sequential I/O requirement. The *delivered* "raw" performance of the I/O subsystem matches exactly the *designed* theoretical maximum throughput.

We expect that it is possible to achieve even greater throughput by using a larger than 1 MB Logical Volume Track Group (LTG) size and a larger than 1 MB LVM stripe size (we would set this to whatever the maximum size FAStT supports, which is currently 2 MB). The current LVM limit for LTG and stripe size is 1 MB, but the limits are much higher in AIX 5.3 (available in 2004).

The JFS2 filesystem on the pSeries server has proven itself in numerous customer data warehouse and other I/O-intensive applications—even using first generation RIO drawers. Defining large "striped" filesystems (as in this benchmark), or using more and smaller filesystems (under a terabyte) as preferred by many database applications, affords you flexibility of layout while delivering all the available I/O bandwidth of the server.

References

- IBM White paper *IBM eServer pSeries 690 Configuring for Performance*, by Harry Mathis, John D. McCalpin, Jacob Thomas, May 6, 2003
- IBM Redbook *The Complete Partitioning Guide for IBM eServer pSeries Servers*, SG24-7039, Kelgo Matsubara, Nicolas Guerlin, Stefan Reimbold, Tomoyuki NiiJima
- IBM Redbook *IBM Total Storage: FAStT600/900 and Storage Manager 8.4*, SG24-7010, Bertrand Dufrasne, Bernd Baeuml, Carlos De Nobrega, Ales Leskosek, Stephen Manthorpe, Jonathan Wright

Acknowledgements

We would like to thank the following people for their contribution to this project:

Alan Byrd
System Integration, IBM

Patrick A. Buckland
TCEM for IO on iSeries and pSeries

Dawn Hamilton
Design Center of e-business on demand, IBM

David Hepkin
AIX Kernel Development, IBM

Randy Kreiser
Storage Architect, LSI Logic Storage Systems, Inc.

Dwayne Perrin
pSeries Federal Technical Consultant, IBM

Sheryl Qualters
pSeries Benchmark Center, IBM

Tom Ruwart
Consultant, I/O Performance Inc.

Kurt Sulzbach
pSeries Benchmark Center, IBM

Tina Tarquinio
pSeries Benchmark Center, IBM

Appendix A: Array controller concepts

Storage array controllers, like the FAStT controllers used in this study, provide a significant enhancement in storage technology over the use of direct-connected disk. The array controller takes over the functions of managing the disks and disk space, and can thereby relieve the system administrator of some of the complexities of managing disk storage. The storage array also offloads functions from the server, allowing more processing power to be dedicated to actual application functions.

Array controllers

The primary component of a storage array controller is the processing unit or controller function. To the server, the storage controller emulates a SCSI-attached disk or disks. By using a standard SCSI protocol over a standards-based connection, the storage controller can be implemented into an existing environment with very little effort.

The emulated SCSI disk is usually described as a logical volume or LUN. LUN stands for Logical Unit Number and represents the number a host uses to access the logical drive over the SCSI protocol. As data is sent to the logical volume, the storage controller is responsible for storing that data onto the disks attached to the array controller. Various RAID technologies are used to safely store the data and guard against data loss due to failures in the disk subsystem. To provide high availability, most array controllers have dual controllers which work together to ensure that the data is accessible to the servers even if one of the controller units fails.

The intelligence of the array controller and its ability to emulate a standard SCSI disk device allows the implementation of many advanced functions within the disk subsystem that are transparent to the servers attached. These advanced functions include the capability of doing point-in-time copies of the data, and the capability of creating a mirrored copy of the data to further ensure the availability of the data stored on the array.

Disk arrays

In order to store data within an array controller there have to be physical disks attached to the controller. These disks are logically grouped into constructs known as “arrays”. The number of disks in each array can vary and depending on the specific implementation used by the manufacturer of the array controller, this number is either predetermined, or can be chosen by the user during the setup of the array controller.

In order to guarantee that the data stored on the system is available even if a drive fails, the array controller uses RAID technologies across the arrays. RAID is an industry standard solution for using groups of independent disks (arrays) to store the data in a format that allows the data to be retrieved even if one of the disks in that array fails.

There are multiple RAID standards defined to provide this availability; they differ in the way the data is written and in the actual amount of usable space that is available in the array after the array is formatted. The most common RAID implementations are discussed in the following sections.

Another improvement available in most storage arrays is the ability to define one or more disks in the system as “hot spares”. These drives are available to the array controller to be used in place of drives that have failed. By predefining these spare drives, the array controller can immediately start to recover the data that was stored on the failed drive without human

intervention. The system can then return to normal operation and the failed drive can be replaced at a later time.

RAID levels

RAID0: For performance

RAID0 is also known as “data striping”. It is well-suited for program libraries requiring rapid loading of large tables, or more generally, for applications requiring fast access to read-only data or fast writing.

RAID0 is only designed to increase performance. There is no redundancy, so disk failures require reloading from backups.

Select RAID0 for applications that would benefit from the increased performance capabilities of this RAID level. Never use this level for critical applications that require high availability.

RAID1: For availability/good read response time

RAID1 is also known as “disk mirroring”. It is most suited to applications that require high data availability and good read response times, and where cost is a secondary issue. Because there are two copies of the data on separate disks, when the application reads from disk, the disk controller can get the data from whichever copy is ready first, giving enhanced performance on a workload with a high proportion of reads. The response time for writes can be somewhat slower than for a single disk, as the disk controller does not consider the write complete until it has written both copies.

The writes can either be executed in parallel for speed, or serially for safety. By using the capability of the FAStT to cache writes, performance of writes in a RAID1 array is greatly improved.

Select RAID1 for applications with a high percentage of read operations and where cost is not the major concern. The other consideration to keep in mind with RAID1 is that since the data is mirrored, the capacity of the logical drive when assigned RAID level 1 is only 50% of the array capacity, thus greatly increasing the number of disks required and the cost of the system.

RAID3: Sequential access to large files

RAID3 is a parallel process array mechanism, where all drives in the array operate in unison. Similar to data striping, information to be written to disk is split into chunks (a fixed amount of data), and each chunk is written out to the same physical position on separate disks (in parallel). This architecture requires parity information to be written for each stripe of data.

Performance is very good for large amounts of data, but poor for small requests because every drive is always involved, and there can be no overlapped or independent operation.

RAID3 is well-suited for large data objects such as CAD/CAM or image files, or for applications requiring sequential access to large data files. Select RAID3 for applications that process large blocks of data. It provides redundancy without the high overhead incurred by mirroring in RAID1.

RAID5: High availability and fewer writes than reads

RAID5 stripes data and parity across all of the drives in the array, offering both data protection and increased throughput. When you assign RAID5 to an array, the capacity of the array is reduced by the capacity of one drive (for data-parity storage). RAID5 gives you higher capacity than RAID1, but RAID level 1 offers better performance.

RAID5 is best used in environments requiring high availability and fewer writes than reads. RAID5 is good for multi-user environments, such as database or file system storage, where the typical I/O size is small, and there is a high proportion of read activity. Applications with a low read percentage (write-intensive) do not perform as well on RAID5 logical drives because of the way a controller writes data and redundancy data to the drives in a RAID5 array.

The use of write caching on RAID5 arrays removes much of the write overhead associated with RAID5, and RAID5 arrays with caching can give as good performance as any other RAID level in most environments. With some workloads, the striping can give better performance than RAID1 at a significantly lower price.

RAID10: Striping and mirroring

RAID10, also known in as RAID 0+1, implements block interleave data striping and mirroring. In RAID10, data is striped across multiple disk drives, and then those drives are mirrored to another set of drives.

The performance of RAID10 is approximately the same as RAID0 for sequential I/Os. RAID10 provides an enhanced feature for disk mirroring that stripes data and copies the data across all the drives of the array. The first stripe is the data stripe; the second stripe is the mirror (copy) of the first data stripe, but it is shifted over one drive. Because the data is mirrored, the capacity of the logical drive is 50% of the physical capacity of the hard disk drives in the array.

RAID10 is an enhancement of RAID1 and should be considered in place of RAID1 when available on the storage controller. Environments with high read percentages and where performance is more important than cost are good candidates for the use of RAID10.

Logical drives

After an array is created within a storage array controller, the space available on that array can be used to store data from the servers attached. A “logical drive” is the structure created on a storage subsystem to map the usable space on attached arrays to the virtual device definitions that are emulated to the servers attached. The logical drive appears to the server as a single physical drive attached via SCSI protocol.

The actual details of how a specific array controller maps the space on the attached arrays to logical drives differs based on the design of that controller, but the generic concepts are similar.

Appendix B: FAStT setup concepts

In this appendix, we explain FAStT setup concepts, including configuration options and considerations for choosing RAID levels and array sizes.

FAStT configuration options

The FAStT is extremely flexible regarding how arrays are defined. Arrays can be defined using RAID levels 0, 1, 3, 5, and 10, and the user can also choose the number of drives in the array. The minimum number of drives in an array is determined by the specific RAID level used; for example RAID1 requires a minimum of two drives, and RAID5 requires a minimum of three drives. The maximum number of drives in an array is thirty. Methods of determining the optimal setup for your environment are discussed later in this appendix.

After the arrays are defined, logical drives are defined and mapped to the different servers attached to the controller. The space used for each logical drive must be located within a single array; therefore, the maximum size of a logical drive is limited to the size of the largest array defined on the system. However, each array can be subdivided into multiple logical drives, if desired.

Choosing RAID levels and array sizes

Before you can start using the physical disk space, you must configure it by dividing the disk drives into arrays and then creating one or more logical drives inside each array. Choosing the appropriate configuration will help you to optimize the performance of your system.

Choosing the RAID level for your system

One of the first decisions that you need to make is which RAID level to implement on your disk arrays. Choosing the RAID level will have an impact on the overall performance of your system and the number of physical disks required to provide the amount of usable space available to your servers.

Choosing the RAID level to use for your environment is a compromise between cost, performance, and availability. Refer to “Choosing RAID levels and array sizes” on page 31 for a discussion of some of the considerations for each of the RAID levels and their relative strengths and weaknesses in different workload environments.

One advantage of the FAStT is the ability to change the RAID level of an array dynamically, without loss of access to the data. If you find that performance for a specific environment might benefit from using a different RAID technology, the FAStT product allows you to remedy that situation without major impact to your data.

RAID5 or RAID10

The two most common RAID implementations in use today are RAID5 and RAID10. Each RAID implementation offers different advantages in terms of performance, availability and cost.

- When writing a block of data, RAID10 requires two writes whereas RAID5 requires two reads (read original data and parity) and two writes. However, newer controllers with cache and advanced cache management can aggregate writes in cache and write the data and parity in a single stripe with one write operation, thereby giving RAID5 an

advantage over RAID10 in a large block sequential workload where block sizes and array sizes are properly matched.

- If a real disk in a RAID array fails, RAID10 rebuilds it by copying all the data on the mirrored disk to a spare—but in order for RAID5 to rebuild a failed disk, it must read the contents of the surviving disks in an array and, using the parity information, then write the result to a spare.

RAID10 is considered overall the best fault-tolerant solution in terms of protection and performance, but it comes at a cost since you must purchase nearly twice the number of disks as needed with RAID5. With newer and faster array controllers, and with larger amounts of cache available on those controllers, many of the disadvantages of RAID5 can be overcome. In most workloads RAID5 provides excellent performance, so that the additional cost of using RAID10 is not required.

Number of disks in the array

Choosing the number of disks in an array is a balance between cost and performance, with multiple considerations. Using larger arrays can provide benefits in several ways. Using more disks can improve performance because the data on an array is striped across multiple disks in that array, and each disk is only required to handle a small part of the operation. Having more disks available also improves the possibility of getting multiple reads from the same array where the data is located on different sets of disks, so that both reads can be done concurrently without contention on the individual disks.

From a capacity standpoint, if you are using RAID5 arrays, the percentage of usable space is increased significantly in larger arrays, as only one parity disk is allocated in each array. For example, two arrays with 5 disks provide the same usable space that is available using only 9 drives in a single array.

However, having larger arrays can also have negative effects on performance. For RAID5 arrays, the larger an array is, the longer it takes the system to rebuild the array onto a spare drive if a drive failure occurs, and performance on all arrays can be affected. Drive contention can also be an issue if you have multiple competing workloads trying to access the same physical disks concurrently.

Segment size

Choosing the segment size of a logical drive can affect the performance of that LUN. When data is written to an array, the array controller breaks the data into blocks of equal size that are striped across the disks in the array. The FAStT supports segment sizes defined at the individual logical drive range from 4 KB to 256 KB. The choice of segment size can have a major influence on performance in both IOPS and throughput.

Large segment sizes increase the request rate (IOPS) by allowing multiple disk drives to respond to multiple requests. Small segment sizes increase the data transfer rate (Mbps) by allowing multiple disk drives to participate in one I/O request. Use a small segment size relative to the I/O size to increase sequential performance.

If the typical I/O size is larger than the segment size, increasing the segment size will minimize the number of drives needed to satisfy an I/O request. This is especially helpful in a multi-user, database, or file system storage environment where using fewer drives for a single request leaves other drives available to simultaneously service other requests.

If you are using the logical drive in a single user, large I/O environment (such as for multimedia application storage), performance is optimized when a single I/O request can be

serviced with a single data stripe (the segment size multiplied by the number of drives in the array that are used for I/O). In this case, multiple disks are used for the same request, but each disk is only accessed once. Generally, small segment sizes are used for databases, normal sizes are used for file servers, and large segment sizes are used for multimedia applications.

Choosing cache options

A useful feature of FAStT disk systems is the cache memory that is installed on the RAID controller. This cache, ranging in size from 256 MB to 2 GB per system, is used to store data as it is read from or written to disk. The use of cache allows the movement of data to and from the disks to be done asynchronously from the reads and writes done by the application.

The use of cache for most environments will enhance the overall performance of the disk subsystem. The FAStT configuration tools allow the system administrator to customize the usage of the cache in the system to provide optimum benefit in performance. When a logical drive is configured, the administrator can tell the disk subsystem what cache parameters to use for I/O to that logical drive. These parameters can be dynamically changed and the settings can be used to tune the performance of the system.

Write caching

Write caching can be turned on or off on each logical drive. When write caching in enabled, it can increase the performance of write operations. The data is only written to the cache initially, and it is the responsibility of the cache controller to eventually flush the unwritten cache entries to the disk drives.

If write caching is disabled, it means that writing operations do not use cache at all. The data is always going to be written directly to the disk drive, and the application will wait for an acknowledgement until the data is written.

Write-back mode (write cache enabled) appears to be faster than write-through mode (write-cache disabled), because from the application perspective it increases the performance of a write as data is written to memory and the application does not have to wait for the disk write operation to complete.

Most workloads benefit when write cache is enabled, but since cache is shared between read and write operations, some workloads may actually perform better with cache disabled. When you disable write cache, more cache is available for read operations. In an environment with a high percentage of reads versus writes, the application might perform better without write caching.

Some heavy workloads also perform better without the use of write cache enabled. If the data rate is so heavy that as soon as the data is written to the cache, it has to be flushed to the disks in order to make room for new data arriving into cache, the controller might perform faster if the data went directly to the disks instead of writing the data to the cache and then using resources on the controller to immediately write the data to disk. The use of cache in this environment is an extra step that can decrease throughput.

Write cache mirroring

When write cache is used, the responsibility of moving the data from cache to disk is assumed by the FAStT controller. If something happens to the controller prior to the data

being flushed to cache, then data corruption occurs and the data on that volume is lost or will need to be recovered from another source.

To remove this exposure to data loss, the FAStT controller provides write cache mirroring. When write cache mirroring is enabled, each controller reserves half of its cache space to store a mirror copy of data in the other controller's cache. When a controller receives data to be written into cache, a copy of that data is sent via the drive-side FC loop to be stored in the second controller's cache. Only after a copy of the data is stored in both controllers' caches will the controller acknowledge the write and allow the application to proceed with its work. The cache in both controllers is protected from loss due to a power failure by a battery which will maintain power to the cache cards until power can be restored and the system can write the data to the appropriate disks.

The use of cache mirroring is very useful from a high availability perspective, but it creates overhead that can limit the throughput of the subsystem. It is recommended that you keep the controller write cache mirroring enabled for data integrity reasons. However, if you have an environment where the data being stored on a volume consists of temporary files, data that can be easily recreated, or environments that can tolerate data inconsistencies, then turning off cache mirroring for those volumes will provide better performance.

Cache management

Cache is a finite and shared resource on the FAStT controller that is used for multiple, competing processes. In order for a controller to do a read or write operation to cache, it must first have space in cache available. If the cache is full of other data, the controller must first clear some space before accepting the new data. The method of clearing out space is called "cache management", and there are parameters in the FAStT controller that allow you to optimize cache management for your specific environment.

The use of write cache is enabled or disabled at the logical drive level, allowing the flexibility to choose which volumes or workloads will be able to take advantage of the cache. The use of read cache can also be enabled or disabled at the volume level, providing even further granularity in determining what data is in cache. One method of improving the performance of a specific application is to disable cache usage (for either read or write) for lower-priority applications using volumes on the controller, thus leaving more cache available for higher-priority data access.

Writing the unwritten cache entries to the disk drives is called "flushing". Two parameters (Start Flushing and Stop Flushing) can be configured within the FAStT controller to affect how much cache is available for write operations, and how soon data must be flushed to disk. These parameters have a global impact across *all* volumes in the controller, and they are both expressed as percentages of the entire cache capacity.

Using these parameters, any cache data older than 20 seconds is first flushed automatically. When the number of unwritten cache entries reaches the percentage given by the Start Flushing value, the controller begins to write data to the disk drives using a first-in, first-out algorithm. When the amount of unwritten data falls below the Stop Flushing value, the controller stops flushing and gives higher priority to handling data.

A typical Start Flushing level is 80%. This means the cache controller does not allow more than 80% of the entire cache size to be used as a write-back cache. If you want to give higher priority and reserve more cache for read activity, you should lower this value to limit the amount of write cache and therefore allow more cache to be used for read operations.

Performance tests have shown that it is a good idea to use a value for Stop Flushing that is close to the Start Flushing value. If the stop level value is set significantly lower than the start

value, it will cause a high amount of disk traffic when flushing the cache. If the values are similar, the controller only flushes the amount needed to stay within the limits.

Cache block size

Cache memory is allocated in blocks of either 4 K or 16 K. This value is a configuration option specified by the administrator. Selecting the proper value for your particular situation can significantly improve the caching efficiency and performance of the system.

For example, if applications mostly access the data in small blocks up to 8 K, but you use 16 K for the cache block size, each cache entry block is only partially populated; you always occupy 16 K in cache to store 8 K (or less) of data. This means only up to 50% of the cache capacity is effectively used to store the data. You can expect lower performance. For random workloads and small data transfer sizes, 4 K is better.

On the other hand, if the workload is sequential, and you use large segment sizes, it is a good idea to use a larger cache block size of 16 K. A larger block size means a lower number of cache blocks and reduced cache overhead delays. In addition, a larger cache block size requires fewer cache data transfers to handle the same amount of data.

AIX and FAStT disk subsystems

In the following section we explain the AIX structures used for the FAStT disk subsystems, and discuss logical drives and controller ownership.

AIX structures used for FAStT disk subsystems

AIX uses the RDAC driver to provide support for the FAStT. Because it is a failover driver, each disk that is available to the server appears as a single disk and the RDAC driver is responsible for determining the active path to that disk.

After a logical disk volume is defined in the FAStT subsystem, the administrator defines mapping tables (called “partitions” in Storage Manager) that define which specific host adapters will be able to access those LUNs. When the AIX server discovers the LUNs that are defined and mapped for its use, AIX builds a single hdisk object to represent that disk. The exact hierarchy is as follows:

dar This object is created by the RDAC driver to represent a controller subsystem. In order to have failover capabilities, the system needs to have discovered two controllers within the controller subsystem. Each controller is designated by a “dac”. The dar is the parent device for the hdisks.

dac This object is created to represent a specific controller within a subsystem. It defines the path (fcs, fcscsi) and the physical controller. Each controller subsystem should have two of these.

Each dac should have a corresponding dar defined, and a dac cannot be deleted from the system if its corresponding dar has not been removed.

A primary restriction in the AIX implementation of the RDAC driver is that the system should be allowed to see exactly *one* path to a dac. This determines the zoning requirements for the SAN.

hdisk This is the standard object created by AIX to represent a SCSI disk image, either SCSI-attached, or using the SCSI protocol over fiber channel.

The net result of this implementation is that the object used in the filesystem definitions is the hdisk, and each hdisk represents a unique logical disk.

Logical drives and controller ownership

The logical drive appears to the server as a single physical drive attached via the SCSI protocol. It is the responsibility of the RDAC device driver to determine the path to the logical drive.

Controller ownership

The FAStT subsystem has two controllers, which are normally used in an active-active failover configuration. That means that both controllers are active and transferring data under normal circumstances and if one of the controllers fails, the surviving controller will take over the work that was being done by the failed controller. When the controller is repaired or replaced, it can be brought back online and the workload can be distributed across both controllers again.

The workload of a specific FAStT subsystem is shared between the controllers by designating a preferred owner for each logical drive. This means that every LUN is owned by only one controller, and all data operations to that LUN are handled by the owning controller. If a controller fails, all of the LUNs that it owned are moved to the surviving controller and that controller becomes the owner until action is taken to return ownership to the preferred owner.

The result of this ownership concept is that at the system level, it is the user's responsibility to assign LUN ownership across the controllers in order to balance the workload among controllers. In order to use both controllers in a FAStT subsystem, you must define at least two LUNs, one being owned by controller A and the other owned by controller B, as shown in Figure 14.

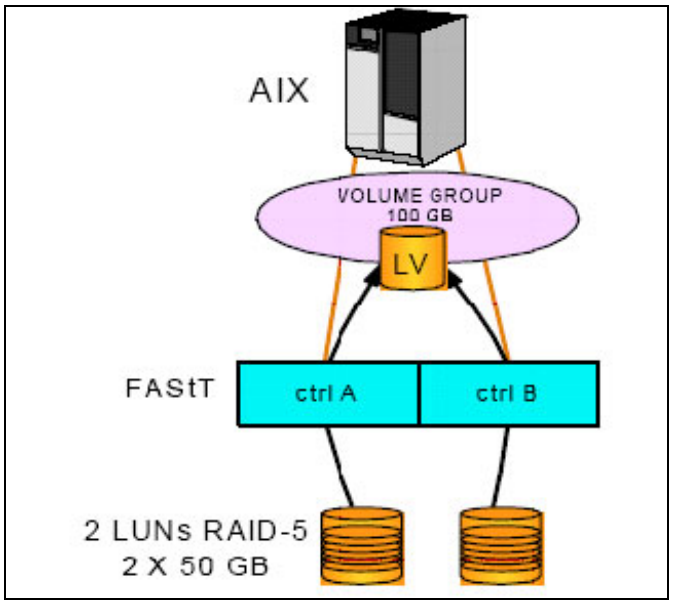

Figure 14 Logical view of FAStT disks in AIX

Unfortunately, balancing traffic not always a trivial task. For example, if an application requires a large disk space to be located and accessed in one chunk, it becomes harder to balance traffic by spreading the smaller volumes among controllers. Also, the load across controllers and logical drives typically changes constantly; the logical drives and data accessed at any given time depend on which applications and users are active during that time period, hence the importance of monitoring the system. The preferred owner for a logical drive is initially selected by the controller when the logical drive is created, but can be dynamically changed from the management tools.

Zoning requirements

The zoning requirements for the FAStT subsystem in the AIX environment are driven by the following restrictions.

An AIX system with two HBAs connected to a single FAStT subsystem would have the following:

- Two HBAs available to the AIX system (LPAR)
- Two connections from the FAStT subsystem:
  - One connected to controller A
  - One connected to controller B
- One logical connection from server HBA 1 to controller A, either direct-connected or enforced by zoning rules
- One logical connection from server HBA 2 to controller B, either direct-connected or enforced by zoning rules
- One partition definition (logical host mapping definition) on the FAStT with the WWPNs of the two HBAs on the server

An AIX system with four HBAs connected to a single FAStT subsystem would have the following:

- Four HBAs available to the AIX system (LPAR)
- Four connections from the FAStT subsystem:
  - Two connections to controller A
  - Two connections to controller B
- One logical connection from server HBA 1 to the first connection to controller A, either direct-connected or enforced by zoning rules
- One logical connection from server HBA 2 to the first connection to controller B, either direct-connected or enforced by zoning rules
- One logical connection from server HBA 3 to the second connection to controller A, either direct-connected or enforced by zoning rules
- One logical connection from server HBA 4 to the second connection to controller B, either direct-connected or enforced by zoning rules
- Two partition definitions (logical host mapping definition) on the FAStT:
  - One partition definition contains the WWPNs of HBA1 and HBA2, which were logically connected to the first connections to controller A and controller B via zoning
  - One partition definition contains the WWPNs of HBA3 and HBA4, which were logically connected to the second connections to controller A and controller B via zoning

Appendix C: Considerations for choosing storage

An important factor in achieving the desired results in our benchmark involved creating a disk environment that would support the required data throughput. Today there are many choices available for storing data in the pSeries environment; selecting the right configuration will enhance your chance of success, while choosing the wrong configuration may make it difficult to reach your goals. In this appendix, we discuss some of the options we considered using for this benchmark, and then explain our reasons for choosing the final configuration.

Direct SCSI-attached

Direct SCSI-attached disk provides a very effective disk subsystem for many smaller environments. With newer adapters and disk technologies, it is possible to achieve a very high performance system—and the cost of SCSI disk is relatively low. However, if you attempt to scale the amount of disk storage required in a system, there are some major inhibitors in the SCSI technologies that make it impractical:

- A limitation on the bus/cable length between disk devices and the adapters forces the disk devices and server to be in close proximity and may actually limit the total amount of storage that you can attach.
- A limitation on the number of devices that attach to a single SCSI bus forces the number of adapters and cables required to grow to the point that management is difficult.
- SCSI is by definition a single controller bus, and the adapter becomes a single point of failure. While this exposure can be overcome by the effective use of RAID technologies, it makes system design more complicated.

Because of these inhibitors and the management issues, we eliminated SCSI disk as a serious option for our benchmark environment.

SSA disk

Until very recently, SSA would have been the disk subsystem of choice for a high performance pSeries benchmark. The SSA architecture provides a very scalable, multi-path disk subsystem which has proven many times to be capable of extremely high bandwidth and exceptional throughput.

SSA uses the SCSI data format, but replaces the parallel bus architecture of native SCSI with a high speed serial protocol. By using this high speed serial architecture, SSA was able to overcome the limitations of distance and connectivity imposed by the parallel SCSI bus architecture. SSA supports distances many times those supported by SCSI—and that distance can be further extended by using fiber connections between components. SSA is defined with a loop architecture and the ability to attach multiple controllers into a loop, so that there are fewer single points of failure and it is easier to build a redundant and reliable system compared to SCSI disks.

However, recent advancements in Storage Area Network (SAN)-attached disk subsystems have provided alternatives that are equally scalable and that afford high performance using fiber connections. The high level of acceptance and proliferation of these devices has driven prices on these SAN disk systems to new lows. In addition, advancements in speed and raw processing capabilities of the components in these SAN-connected disks have made these subsystems easier to use and less complex to install and maintain.

These factors have begun to make the SSA disk solutions less attractive from a pricing and management perspective. Because of the desire to provide a solution for the customer that

not only met the throughput requirements of the benchmark but also provided the most cost-effective solution, we eliminated SSA from consideration as our disk subsystem.

SAN-attached disk

SAN-attached disks utilize the SCSI protocol over a fiber-based network. The use of fiber provides higher speeds and extended distances compared to copper-based parallel SCSI or even serial SSA connectivity.

In addition, most SAN-attached disk subsystems include highly intelligent disk controllers that provide enhanced RAID implementation with onboard cache that allows the use of lower-cost components without affecting the performance or reliability of the systems. These controllers also provide the ability to implement new functions such as point-in-time copies of your data and remote mirror solutions to provide better availability of the customer's data.

IBM provides several options for SAN-attached disk. Choosing which option is best for your environment involves understanding not only your requirements, but also the different capabilities and strengths of each of the IBM offerings.

IBM Enterprise Storage Subsystem (ESS)

The ESS has gained acceptance as a premier disk subsystem within the industry. It provides exceptional performance and excels in providing service to large numbers of heterogeneous servers. It is also recognized as having superior availability and reliability because of its design.

The ESS design includes two high performance disk controllers based on the IBM high performance pSeries server architecture. The ESS provides a wide range of connectivity for servers, including SCSI and Fibre Channel for servers in the open environment, and FICON® or ESCON® in the mainframe environment. A total of up to sixteen adapters is available.

Another differentiator for the ESS versus many of the smaller, modular disk solutions available is that it provides significantly more scalable cache options, starting at the 8 GB minimum configuration and scaling to 64 GB. In some environments, the additional cache provides an additional boost in performance. From a raw throughput perspective, the 2105-800 would provide a good base for a high speed solution.

The major argument against using an ESS configuration in our target environment is that the ESS requires significantly more floor space to install. Even though our solution would not require a large amount of disk in each controller, the ESS base frame is sized to support a much larger amount of disk, and so much of the space in the cabinet would be empty. This means that an ESS solution would take significantly more floor space and require more power and cooling to implement than the more modular disk solutions like the FAStT. It could be argued that the use of the ESS might make a better technical solution for our project, and that the ESS solution would be easier to manage and control.

However, the final decision to not use the ESS was made because it was felt that a useful solution could be created using FAStT technology that would be less expensive and require less floor space while still meeting all of our requirements for performance, scalability, reliability, and manageability.

Fiber Array Storage Technology (FAStT) disk

The IBM TotalStorage® FAStT family of storage products offers an ideal set of building blocks for your Fibre Channel-based storage area network. Each subsystem consists of a controller drawer and one or more disk enclosures, allowing a subsystem to scale from 36 GB

to over 32 TB of capacity. It is easy to install and manage utilizing the FAStT Storage Manager that is included with every FAStT Storage Server.

The FAStT family of products offers various controller models at different performance and price points to best match your SAN needs. Multiple FAStT Storage Servers can be centrally managed from a common FAStT management console across your existing LAN infrastructure—and with the FAStT advanced copy service functions such as FlashCopy® and Remote Mirror, your data protection and disaster recovery needs can be addressed.

The high availability models of the FAStT family contain two controllers to provide access to your data. The controllers run in an active/active configuration with failover capability to allow access to the data if one of the controllers fails. The controllers are connected with Fibre Channel connections to the servers. Connectivity to the disks in the system is provided by dual redundant Fibre Channel loops to ensure availability of the data.

The FAStT family provides a multiplatform storage environment, supporting a wide variety of servers, operating systems and cluster technologies. By using a modular approach and a common management tool, the FAStT family allows customers to create a subsystem that meets their capacity and performance requirements.

Choosing the FAStT controller model and configuration

The controller models of the FAStT family are shown in Figure 15. Now we take a closer look at both the Model 900 and the Model 600.

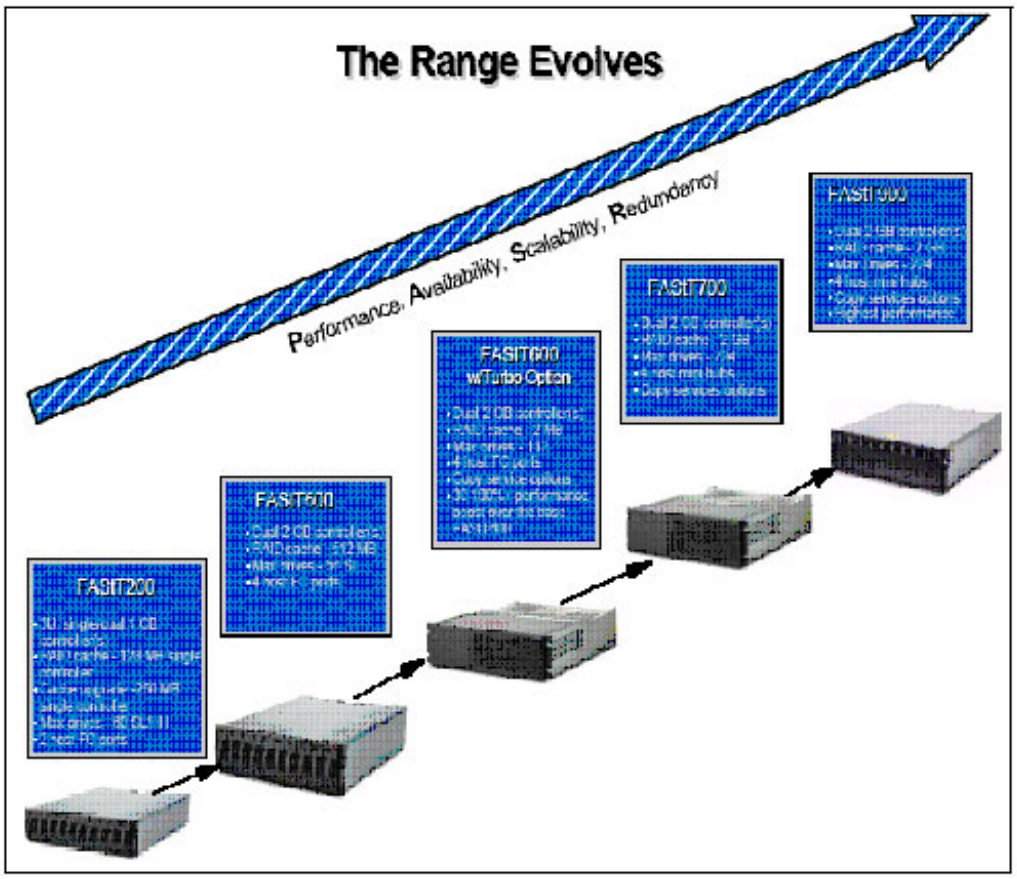

Figure 15 FastT Family overview

The highest performance model of the family is the Model 900. With four 200 MB/sec fiber connections, the Model 900 can sustain sequential read throughput of 750 MB/sec or more; this is nearly double the throughput capability of the Model 600/600 Turbo or the Model 700. This makes the Model 900 appear to be the obvious choice for our test. However, when you look at the Model 600 and consider that it can support more than 370 MB/sec in a controller using two fiber connections, the actual throughput over each fiber is nearly the same.

Another advantage of the Model 900 is higher scalability in terms of storage that can be attached—but in this environment, we are maximizing the number of controllers to provide raw performance and the amount of disk behind each controller is supported equally well by both the Model 600 and the Model 900.

Another major difference between the Model 900 and the Model 600 is that the Model 600 has room in the controller drawer for up to 14 disks, whereas the Model 900 must have at least one disk expansion drawer in addition to the controller drawer, to hold the same amount of disk space. This means that a solution using two Model 600 controllers, each with two fiber connections, actually requires less rack space compared to a solution using one Model 900 controller with the equivalent amount of disk. In addition, due to the pricing structure of the FAStT family, the initial cost of the Model 600-based solution might be less than the equivalent Model 900-based configuration.

In the final analysis, we concluded that both the FAStT Model 900 and the FAStT Model 600 are excellent choices to support a high performance workload because of their modularity and scalability. Although we chose the Model 600 solution in this specific case due to a slightly lower initial cost for an equivalent performance solution, it would be wise to evaluate the overall TCO of both solutions in your specific environment before deciding which one to use.

The SAN infrastructure

Fibre Channel-attached disks can either be directly connected to a server, or they can be connected through a SAN infrastructure consisting of one or more SAN switches. In most environments, the use of a SAN reduces the cost of connecting servers and storage, as fiber connections to a disk can be shared by multiple servers—and conversely, a server can connect to multiple devices over the same fiber.

IBM works with several leading SAN equipment manufacturers to provide customers with options for creating their SAN. Each of these vendors (Brocade, McData, CNT (Inrange), Cisco) provide the same basic service of connecting devices together, but they offer different strengths and capabilities that you need to consider when selecting the components of your SAN.

For this project, we chose to implement a SAN using the IBM 2109-F32 switches manufactured by Brocade. However, it is worthwhile to evaluate options from each of the manufacturers before choosing which components to use in your configuration. A discussion of the considerations involved in choosing one SAN switch over another is beyond the scope of this Redpaper, but it is important that you understand the architecture of the product that you choose and what impact, if any, specific designs within the SAN might have on transferring data through the SAN at a rate sufficient to meet your goals.

SAN design considerations

When designing a network of any kind, consideration of how much total traffic is supported through the network connections at any time is critical. In most networks, it is not expected that every device in the network will be operating at full capacity at the same time.

As an example, in telephone networks today large numbers of phones are attached to telephone switches, but the expectation is that not all of them will be used at the same time. If everybody in one area tried to use their phones at the same time, only a portion of the phone calls would be connected while the rest of the subscribers would have to wait for service.

A switch that is designed with enough internal bandwidth to allow *all* ports to be active at full speed simultaneously is called “non-blocking”, while a switch that provides only enough bandwidth for some percentage of its ports to be transferring data at full speed is known as a “blocking” switch. As the number of ports on a switch increases, the cost of providing a non-blocking switch increases dramatically.

Since not all environments require all ports to be transmitting at full speed simultaneously, some SAN switches provide a less expensive SAN solution that does not provide a fully non-blocking design.

The 2109-F32 has the capability of allowing all ports to communicate at full speed simultaneously. However, we discovered when we first connected our devices to the 2109-F32 that we were not able to achieve our total throughput goals. In fact, we were only getting half of the throughput per adapter path that we needed.

By examining the internal architecture of the 2109-F32, we quickly understood why we were getting these results and we made some simple changes in our cabling scheme that allowed us to double the throughput through the switch, as we will explain later in this section.

When we first designed the cabling for our test environment, we chose a basic and simple design that would make it easy to connect the cables as we set up the server and the storage. We used ports zero through 15 to connect all of the HBAs in one of the I/O drawers on the pSeries server, connecting them in the same physical order as the HBA locations in the I/O drawer. We then used ports 16 through 32 to connect to the FAStT storage controllers to be used by those adapters; this is shown in Figure 16 on page 43.

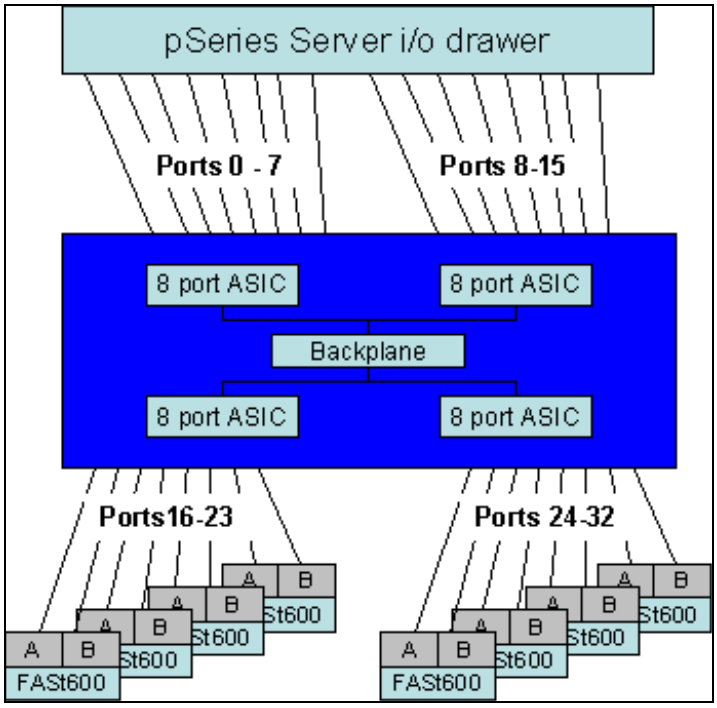

Figure 16 Worst case design for 2109 SAN Switch - switch becomes a bottleneck

To demonstrate why this configuration was a problem we need to explain the architecture of the 2109-F32. As shown in Figure 17 on page 44, the switch is logically divided into four independent ASIC chips, each of which is supporting eight ports for external device connectivity. Specifically, ports 0 through 7 are on the first ASIC, ports 8 through 15 are on the second ASIC, ports 16 through 23 are on the third ASIC, and ports 24 through 31 are on the fourth ASIC.

When two devices are connected to the same ASIC and are sending data to each other, the data does not need to leave that ASIC; within each ASIC there is enough bandwidth capacity to support 800 MB/sec or four full-speed conversations between the adapters on that ASIC. To communicate between ASICs, each ASIC is connected to a backplane via internal ports on that ASIC.

When a device is communicating to a device on *another* ASIC, then the data must leave the ASIC containing the originating port, travel through the backplane, and then enter the ASIC containing the destination port.

When the 2109-F32 was designed, it was determined that the probability of all ports needing to communicate through the backplane at full speed was very small. As a result, the backplane and the internal connections from each ASIC were scaled back to reduce cost while still providing more than enough capacity to meet most workload requirements. The internal bandwidth of the backplane was designed to support 1600 MB/sec, eight full-speed conversations between ASICs out of 16 possible conversations on the switch.

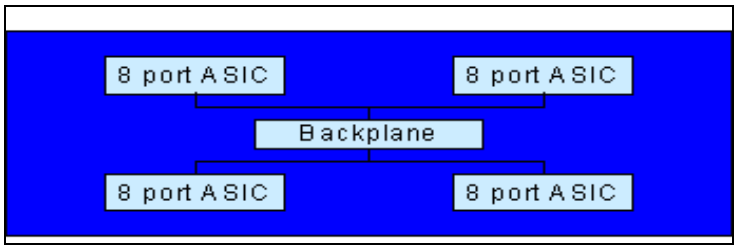

Figure 17 Internal design of 2109 F32 - logical view

In the configuration shown in Figure 16 on page 43, the server HBA on port 0 is zoned to communicate with the storage port on port 16, and the HBA on port 1 is zoned to communicate with port 17, and so on. This means that all traffic on the switch is being routed through the backplane, and we would need 3200 MB/sec of throughput to support the full bandwidth of all 32 ports (16 connections).

As mentioned, the backplane and the internal connections from each ASIC to the backplane only support 1600 MB/sec of sustained bandwidth—and in our configuration, this becomes a bottleneck.

The solution was to reconfigure the cabling into the switch so that most of the connections between the devices remained on the same ASIC. Each ASIC had enough bandwidth to support the throughput we required if the connections stay within the ASIC and do not cross the backplane.

We accomplished this by switching the cables so that each server HBA was plugged adjacent to a port connecting to a storage device. The zoning configuration was also changed so that the server HBA was zoned to communicate with the storage device adapter plugged into the adjacent port. This new physical cabling setup and zoning configuration provided the desired effect of keeping traffic isolated onto each ASIC, and thus eliminated the backplane as a bottleneck.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Performance is based on measurements and projections in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

This document created or updated on October 5, 2004.

Send us your comments in one of the following ways:

- Use the online **Contact us** review redbook form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

@server®	FlashCopy®	POWER4+™
@server®	FICON®	pSeries®
AIX®	HACMP™	Redbooks™
AIX 5L™	IBM®	Redbooks (logo)™
e-business on demand™	POWER™	Redbooks (logo) ™
ESCON®	POWER4™	TotalStorage®

The following terms are trademarks of other companies:

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.