Redbooks Paper

Gautam Shah
James Wang

GPFS Sequential Input/Output Performance on IBM pSeries 690

Abstract

The purpose of this high performance I/O study is 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 is capable of sustaining 18.5 gigabytes per second (GB/s) sequential I/O. When performing disk I/O using a *modified* GPFS, the server can sustain 15.8 GB/s aggregate sequential read or 14.5 GB/s aggregate write performance.

In this paper we discuss the capabilities of the newest IBM pSeries 690 processor and its I/O subsystem. We provide GPFS configuration details and discuss GPFS options for attaining the best performance. We also include details about the selection of the disk subsystem and disk adapters. Finally, we provide guidance on selecting the proper RAID configuration, as well as other performance and tuning tips.

Introduction

There are many applications today, in both industry and in the public sector, that 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 industries, 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 high performance processing capability and a high capacity input/output infrastructure in order to provide fast access to 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 fully loaded 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 file system 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 many customer environments today 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 seven IBM 2109 Fibre Channel switches.

With this test system, a series of performance tests were executed to provide I/O workload sizing information and to examine the following areas:

- 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
- The sequential throughput capacity of the system-wide striped logical volumes
- The sequential throughput capacity of single file and the aggregate file system throughput using GPFS
- The aggregate application read-behind-write performance using GPFS

IBM pSeries 690 system architecture

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

The 1.9GHz 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 pSeries690, doubling the capacity previously available.

The new pSeries 690 also includes faster 633MHz features, which allow pSeries 690 systems to take full advantage of the performance of 1.9GHz 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. The maximum I/O configuration is seven 7040-61D I/O drawers.

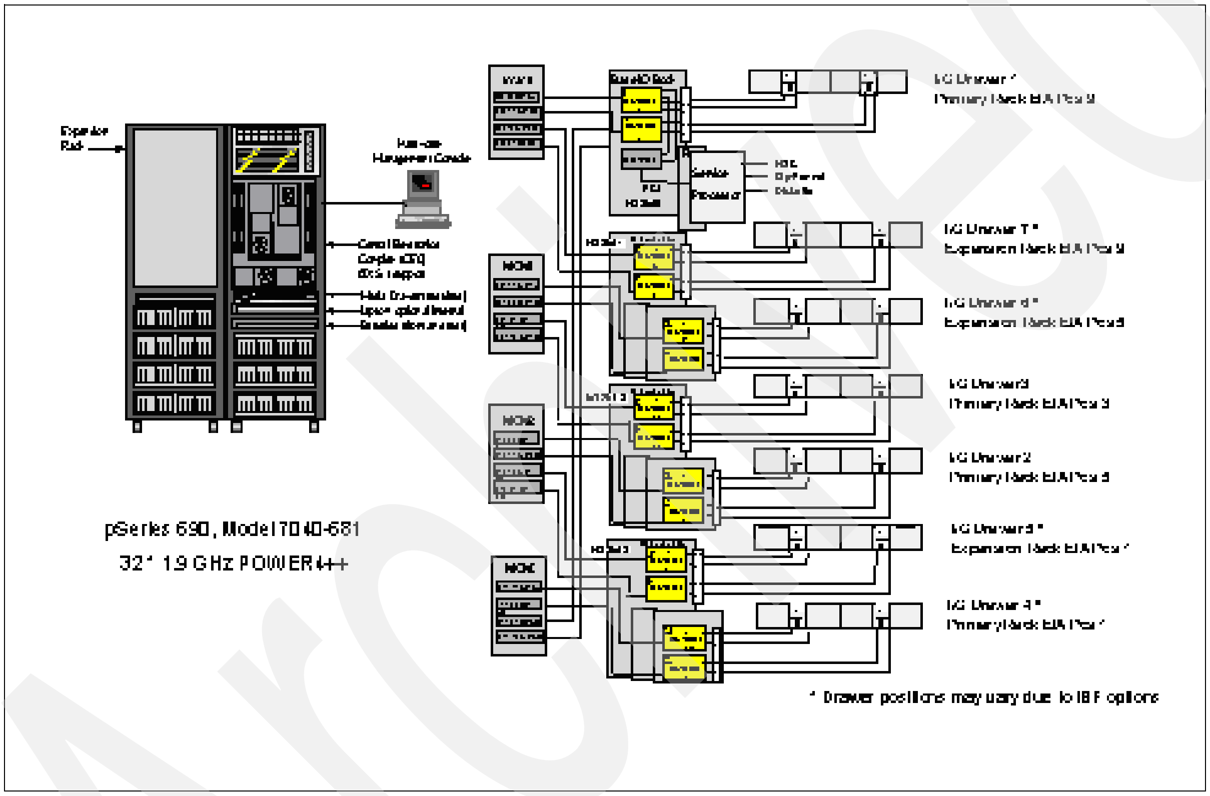

Figure 1 pSeries 690 I/O subsystem default cabling, maximum bandwidth per PCI slot, all drawers double loop, 7 drawers maximum

The 7040-61D I/O drawer has two I/O planar options: a high performance RIO2 to PCI-X I/O planar (Feature Code 6571), and the RIO to PCI I/O planar used on the pSeries 690 (Feature Code 6563). The 7040-61D I/O drawer with FC 6571, represented in Figure 2 (the 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 they 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 a RIO2 port on an I/O book in the CEC. The other RIO2 can be connected to a RIO2 port on an I/O book in the CEC (Figure 2), or it can be connected to the 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 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.

Large pages support

The POWER4+ processor and the POWER4 processor in the IBM pSeries 690 system support 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 the benchmark, is discussed in "AIX support of large pages" on page 10.

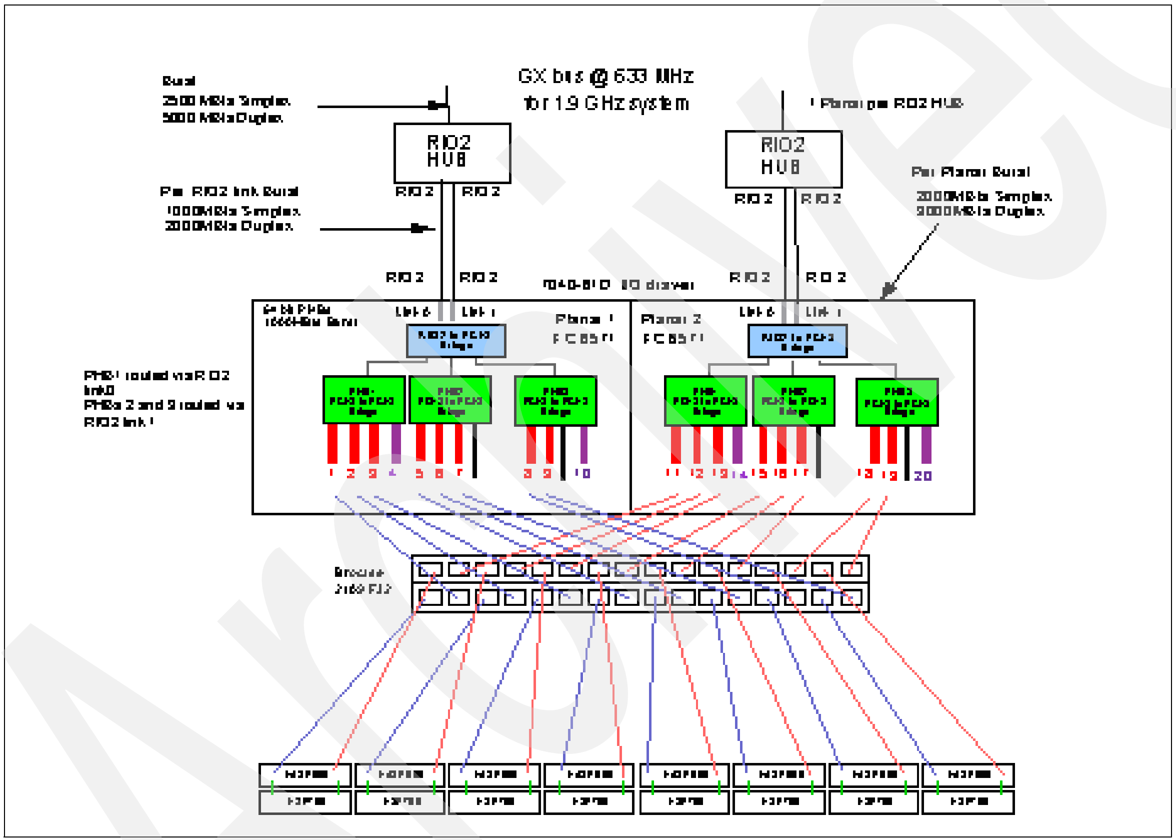

Figure 2 pSeries 690 I/O subsystem, I/O planars with default dual RIO2 loop cabling

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.

- One pSeries 690 (IBM model 7040-681). The machine was equipped with four 8-way POWER4+ Turbo 1.9Ghz 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 HMC build level 20040113.1. The HMC 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, 15 K RPM).
- 56 drive enclosures, EXP 700 with 14 drives (36 GB, 15 K RPM).
- 112 one port 2 Gbps Fibre Channel HBA (Host Bus Adapter) – 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 IBM 2109 F32 32 port switches.

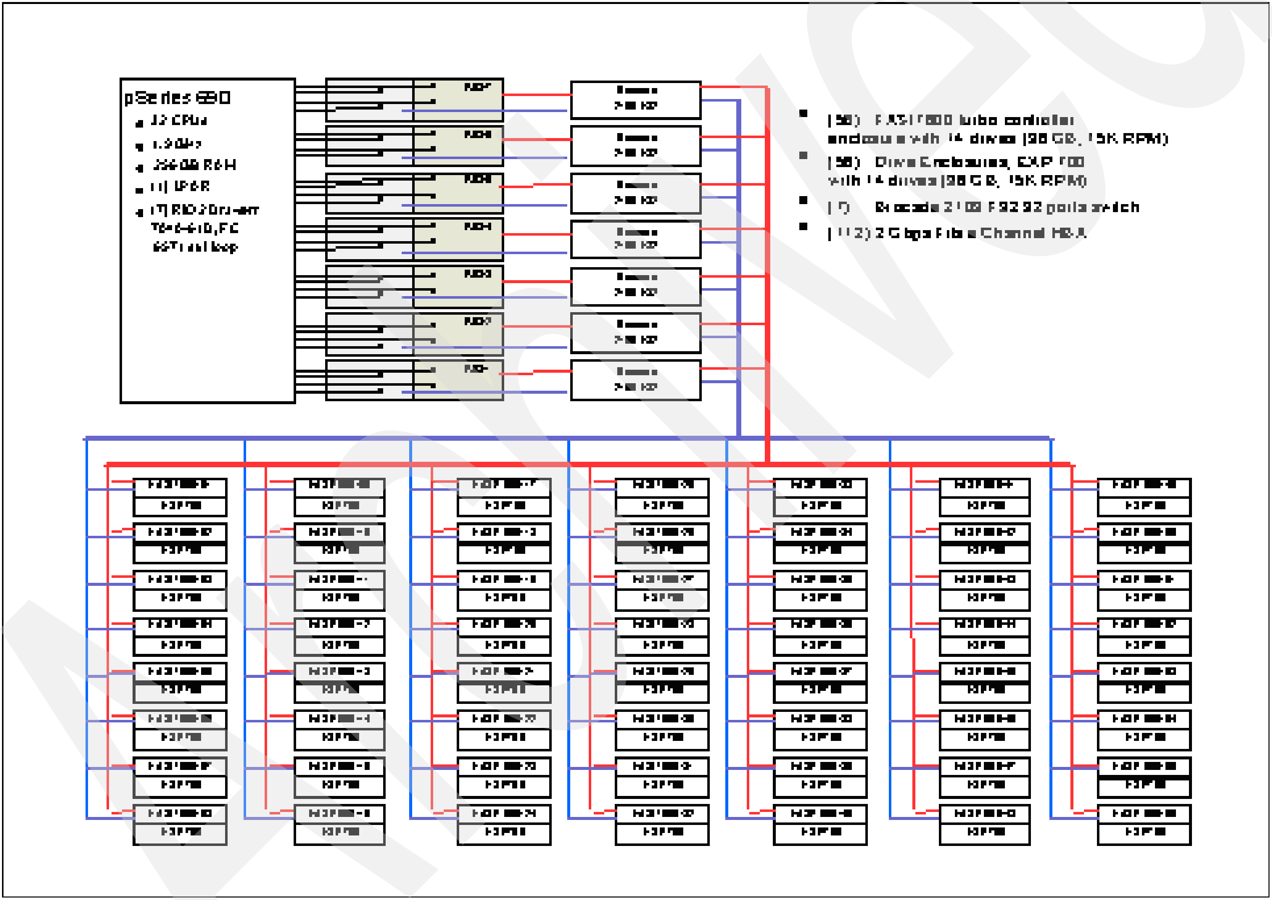
pSeries 690
32 CPUs
1.9GHz
256GB RAM
(1) LPAR
(7) RIO2 Drawer 7040-61D, FC 6571 dual loop
(56) FAStT600 turbo controller enclosure with 14 drives (36 GB, 15K RPM)
(56) Drive Enclosures, EXP 700 with 14 drives (36 GB, 15K RPM)
(7) Brocade 2109 F32 32 ports switch
(112) 2 Gbps Fibre Channel HBA

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 the 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 meant 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 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 slots 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 at slots 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 slots 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 customers.

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. In the following sections, we describe these products in more detail.

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.

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, and the system configuration is shown in Figure 3.

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 choosing 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 aggregate sequential read and write bandwidth with GPFS, 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 defined 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 FAStT600 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 an application called Xdd. Xdd is described in detail in “The Xdd I/O application” on page 12. We use an Xdd I/O application with two threads, with each thread reading or writing four MB application I/O blocks against a RAID5 array. The results are listed in Table 1.

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, 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 the 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 this section, we describe the environment we set up for this benchmark.

Configure the pSeries 690 as one LPAR for highest performance

For a heavy I/O workload, it is recommended that the pSeries 690 be configured in full system LPAR mode. This is because of a difference in how AIX interacts with the hardware I/O translation control entry (TCE) regions.

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 busses in the system due to operating system usage of memory within the TCEBAR region. This possibility, 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 forth.

Operating system AIX 5.2 ML2

The pSeries 690 was built with 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 must 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, 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, as follows:

```
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

The following FAStT settings were used 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.

The 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 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 shmget/shmat case, we set the v_pinshm VMM tuning parameter to 1 by using the command vmo –p –o v_pinshm=1.

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 is 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 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 all the adapters and disk subsystems (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.

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 the 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. 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.

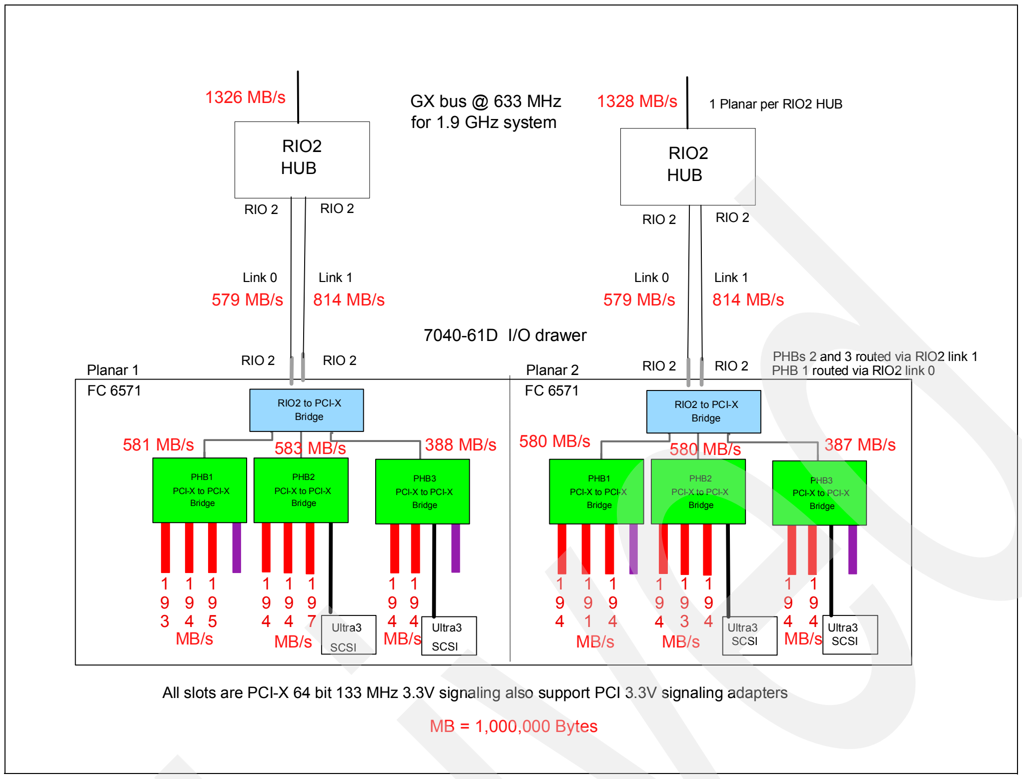

Figure 4 pSeries 690 RIO2 internal simplex bandwidth profile

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

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 “pSeries 690 RIO2 drawer internal bandwidth profiling” on page 13, we profile 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” GPFS 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 section, 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 GPFS 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 GPFS concurrent read or write behavior—is the closest approximation. The intent of studying the raw striped LV performance was to reveal the GPFS with DIO overhead over raw devices.

The *striping* mechanism, also known as RAID0, 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 reassembled into the actual data.

Note the following points:

- From the high availability point of the view, the striping function does not provide redundancy, 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 after. 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 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 KB segments 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 written. 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.

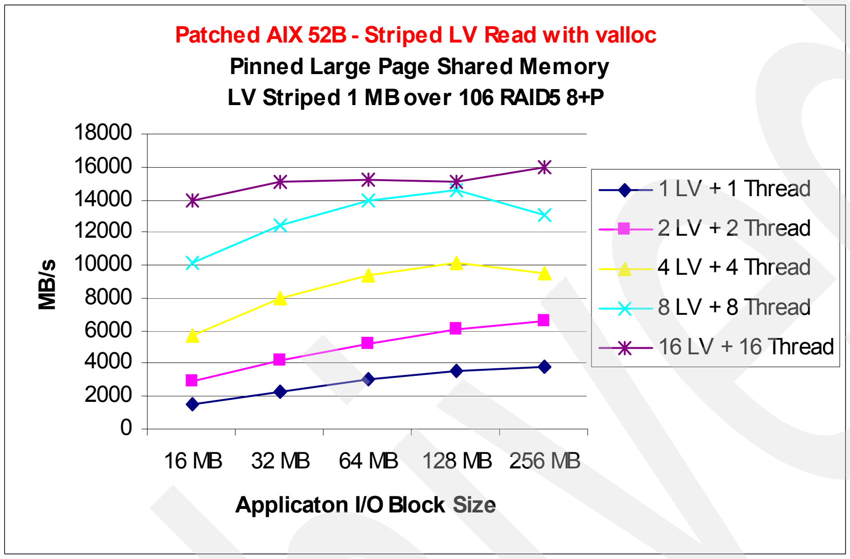

Figure 5 Striped logical volume read performance

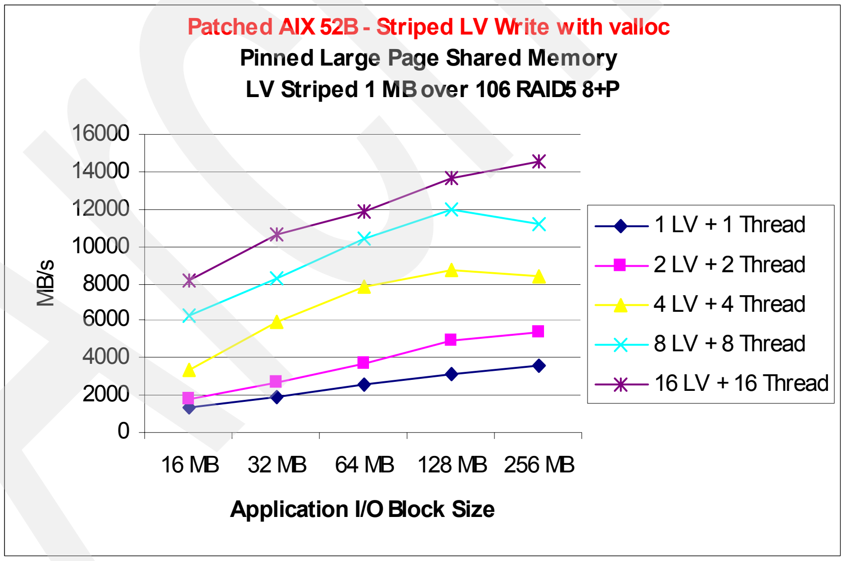

Figure 6 Striped logical volume write performance

The GPFS file system

The General Parallel File System (GPFS) is the IBM high performance cluster file system, which has been available on AIX clusters since 1998 and on Linux® clusters since 2001.

GPFS allows file data access from all nodes in the cluster by providing a global name space for files. Applications can efficiently access files using standard UNIX® file system interfaces. GPFS supports 32-bit as well as 64-bit applications. It also has been tested up to a 100 TB file system size. The file system size can be dynamically increased or decreased by the addition or deletion of logical disks.

GPFS provides standard POSIX application interfaces. It supports buffered I/O, synchronous I/O (file is opened with O_SYNC or O_DSYNC flags), kernel asynchronous I/O (through the use of the Asynchronous I/O system calls) and Direct I/O (non-buffered I/O). Direct I/O can be done on a per file basis if the file is opened with O_DIRECT, or it can be done on a per file system basis when the file system is mounted with the "dio" mount option.

The use of Direct I/O allows GPFS to bypass the page pool (file system cache) to perform I/O. This saves CPU cycles because the copy from the page pool to the application buffer is avoided. Direct I/O prevents GPFS from using pre-fetch algorithms, but the performance can be compensated by applications issuing larger-sized I/Os, use of asynchronous I/O, and/or use of multiple threads.

GPFS distinguishes itself from other cluster file systems by providing applications with concurrent high speed file access in a multiple node AIX and/or Linux cluster. It provides excellent performance, especially for large sequential record access patterns. Although GPFS is typically targeted for a cluster with multiple nodes, it can also provide high performance benefit for a single node.

GPFS allows AIX and Linux nodes in a heterogeneous cluster

GPFS 2.2 allows AIX and Linux nodes to coexist in the same heterogeneous cluster. GPFS for AIX and GPFS for Linux are derived from the same source code base and differ only in adapting to different hardware and operating system environments.

GPFS uses Shared Disk Model

GPFS assumes a shared disk model. The shared disk model can be implemented in many forms:

- Via a hardware SAN, where all nodes in the cluster have physical access to all storage
- Via a separate software device driver called Virtual Shared Disk (VSD), that presents a shared disk layer to a cluster of nodes, which schedules I/O to the appropriate storage server
- Via a software layer within GPFS called Network Shared Disk (NSD), which schedules I/O to the appropriate storage server

To enhance the GPFS performance in VSD or NSD environments, it is recommended that the data to be transferred via high-speed interconnects from the storage server to the application node.

The keys to high GPFS performance

The following are some of the key GPFS features that help to achieve high performance I/O:

- Striping data across multiple disks and multiple nodes.
- Efficient client-side data caching.
- Allowing administrators to configure large block size on a file system basis to meet application characteristics.
- Using read-ahead and write-behind functions for "regular" access patterns.

- Using byte-range locking which is based on a sophisticated token management to provide data consistency. This enables multiple application nodes to have concurrent access to files.

GPFS metadata

GPFS handles metadata on *all* nodes of the cluster. This key and unique feature distinguishes the GPFS architecture and design from other cluster file systems, which typically require a centralized metadata server to handle fixed regions of a file system. A potential downside of the centralized metadata server approach is that it could become a performance bottleneck under metadata-intensive operations. It may also be a single point of failure if a backup server is not configured.

Highly Available GPFS

GPFS is a highly available file system. It can be configured to allow uninterrupted data access even under the failure of compute nodes, I/O server nodes, or disk attachments. The metadata is organized by GPFS in a fashion that lends itself to efficient parallel access and maintenance. It can be configured with multiple copies (replication) to allow continuous operation even if the disk path or the disk itself is inaccessible.

In addition, GPFS can be used with RAID or other hardware redundancy capabilities to provide business continuity even under media failures. To accomplish this, the disks should be multi-tailed-attached to multiple I/O servers. When an I/O server fails, I/O requests can be served by a backup server that provides an alternate path to the same disk.

The loss of connectivity to disks from one node does not affect the other nodes in the direct-attached SAN environment. GPFS uses the IBM RSCT function to continuously monitor the health of the various file system components. When any failure is detected, the appropriate recovery actions will be taken automatically. GPFS also provides extensive logging and recovery capabilities, which maintain metadata consistency across the failure of application nodes holding locks or performing services for other nodes.

Single node GPFS

Since GPFS is often targeted as a cluster file system, a question is sometimes raised about the relevance of a single node GPFS configuration. There are two points to consider:

- GPFS is a well-proven, scalable cluster file system. For a given I/O configuration, typically multiple nodes are required to saturate the aggregate file system performance capability. If the aggregate performance of the I/O subsystem is the bottleneck, then GPFS can help achieve the aggregate performance even on a single node.
- GPFS is a highly available file system. Therefore, customers who are interested in single-node GPFS often end up deploying a multi-node GPFS cluster to ensure availability.

The product documentation, which provides more detailed information about all aspects of the GPFS file system, is available at the following site:

`http://publib.boulder.ibm.com/clresctr/windows/public/gpfsbooks.html`

GPFS sequential read or write performance considerations

In setting up a GPFS file system for our benchmark, there were several configuration and tuning decisions to be considered. In this section, we discuss the GPFS configuration options as well as the application design considerations that can influence performance.

Note: Some of the GPFS configuration options used in the benchmark are modified features that are not available in the generally available (GA) version of GPFS.

Configuring disks for GPFS

GPFS can be configured using either VSD or AIX hdisks. Since the VSD option has been available for a long time, it is currently the preferred choice in large clusters. This is especially true in systems where you want to take advantage of the high performance switch.

Since a VSD is built on a logical volume, its I/O size to the disk device driver is determined by the volume group logical track group (LTG) size. To study the LTG performance impact on GPFS, we enhanced this *modified* GPFS to allow the exploitation of 1 MB LTG size with GPFS commands. In comparison, GPFS 2.2 only allows a default track size of 128 KB.

In this benchmark, we only studied the GPFS performance of 1 MB LTG. However, it is our belief that the performance impact of using a smaller LTG size is expected to be small because the device driver can coalesce multiple contiguous requests into a single large request to the disk.

Data striping choices

To achieve high sequential I/O GPFS performance under AIX, there are three methods to be considered in configuring the available RAID5 arrays:

a. GPFS striping, where each RAID5 array constitutes a GPFS logical disk. This is the most typical way to configure GPFS. It can take full advantage of the GPFS pre-fetch/write-behind capability for large sequential accesses.
b. AIX Logical Volume Manager (LVM) striping, where a large striped logical volume can be built with a subset or all of the RAID5 arrays. In this case, each logical volume would constitute a GPFS logical disk.
c. Using a combination of GPFS striping and AIX LVM striping.

We typically prefer method a) because it enables GPFS to perform “deep” pre-fetching. However, in a single node GPFS configuration with a large number of disks, we may potentially derive some benefit from methods b) or c). This is because they may enable the concurrent use of disks even when GPFS or application does not have multiple outstanding requests.

In this benchmark, using method b) or c) requires GPFS to support a larger than 1 TB logical disk. This is because the aggregate size of all the RAID5 arrays used to build a striped logical volume can exceed 1 TB. While the current GPFS 2.2 can only support logical disks up to 1 TB, our *modified* GPFS allowed us to configure logical disks larger than 1 TB.

Note: A future GPFS release will add support for a logical GPFS disk to be larger than 1TB.

GPFS block size

Another important decision to make when configuring GPFS for large sequential I/O is the choice of a file system block size. GPFS 2.2 supports block sizes up to 1 MB. Since we anticipated the use of a larger GPFS block size could improve performance, we enhanced the *modified* GPFS to allow block sizes up to 32 MB. Some of the GPFS performance measurements using a 32 MB GPFS block size confirm our expectation.

Note: There is no plan to support very large block size in the GA product. This is because there are other factors to be considered, such as the amount of memory used for GPFS pagepool or the fragmentation impacts of sub-block allocation.

Contiguous block allocation

Another option available when creating the GPFS file system is the ability to allocate blocks contiguously on an individual logical disk. This was also a new function available in this *modified* GPFS. In contrast, GPFS 2.2 allocates blocks randomly on each logical disk. We used contiguous block allocation in our single node GPFS because we expect a sequential I/O application would benefit from contiguous block placement on disk.

GPFS tunables

In addition to the configuration options, there are also GPFS tunables that can impact performance. Some of the GPFS tunables used in this benchmark are:

- pagepool: this is the amount of space allocated for GPFS buffering.
- prefetchThreads: this indicates the number of threads that the GPFS daemon should use for read or write operations. This parameter sets the maximum number of concurrent I/O requests used by GPFS.
- maxMBpS: this dynamically adjusts the number of prefetchThreads used based on I/O response time (with an intent to sustain a particular I/O rate for the client).
- maxBuddyBufferSize: this sets the maximum VSD communication message size, and dictates the VSD I/O granularity.

Application considerations

There are also application-level considerations to keep in mind when exploiting high performance GPFS. Some I/O-intensive applications perform large sequential I/Os using a single thread to read or write. In many cases the simple use of a read or write system call works well with GPFS because GPFS uses multiple threads to schedule I/O simultaneously to multiple disks. This method involves staging data through the GPFS file system cache. In the case of a single node GPFS with an intensive I/O workload, the aggregate CPU cycles to stage data can potentially saturate all the CPU capability.

To improve I/O performance in cases where the CPU is a bottleneck, applications can use direct I/O to avoid the data copy between the file system cache and application buffers. The saved CPU cycles can be used to drive additional I/O throughput. Although the use of direct I/O avoids the data copy overhead, it requires the application buffers to be pinned. The pinning is required to ensure that the application buffers corresponding to the I/O request do not get paged out.

Since pinning is an expensive operation, if the application buffer pinning can be done a priori outside the I/O critical path, it can save CPU cycles. But this requires the file system to have a mechanism to recognize that the application buffers have already been pinned, so it does not have to pin the same buffers again before scheduling I/O.

GPFS 2.2 does not have the capability to recognize that the application buffers have already been pinned. But this new feature was incorporated into the *modified* GPFS. Applications can inform GPFS that its application buffers used for Direct I/O have been pinned, so GPFS does not have to pin again. The application can pin its buffers using appropriate options on shared memory with large pages, as described in “The Xdd I/O application” on page 12.

Note: The intent of Direct I/O is to read or write data directly into or out of the application buffer. Although it prevents GPFS from pre-fetching, performance can be compensated by using asynchronous I/O and/or multiple threads to ensure sufficient concurrency.

GPFS sequential read/write performance measurements

In this section we describe the performance results of using some of the configuration options described in the previous section. Our intent is to show how we progressively improve the GPFS performance by using the tunables and some of the modified features previously described.

"Out-of-box" performance using GPFS Striping (5 - 7 GB/s, 1 application thread)

A GPFS file system was created with 109 VSDs over 109 RAID5 8+P arrays. Each VSD was created on a logical volume using 1 MB LTG size, which is the largest size possible under AIX 5.2. Further, as described in "Hardware configuration and sizing" on page 4, the segment size of the RAID5 8+P array was set to 128 KB. This is to ensure that the RAID5 8+P array with stripe width of 1 MB (8 disks * 128 KB segment size) matched the 1 MB LTG size. The optimal setting avoids the RAID5 read-modify-write penalty.

The GPFS file system was created with a block size of 4 MB. The file system was configured for contiguous block allocation on a per logical disk basis. In this stage of the performance study, the Xdd application used only one single thread to do large sequential reads or writes without using either Direct I/O or shared memory. With a pagepool size of 2 GB, prefetchThreads of 400, maxMBpS of 5000, and maxBuddyBufferSize of 1MB, we were able to measure GPFS performance in the range of 5 – 7GB/s. Note that the VSD tunable maxBuddyBufferSize was set to 1 MB to match the logical track size. If the maxBuddyBufferSize did not match the volume group LTG, we observed considerable CPU overhead that reduced overall GPFS performance.

Tuning the GPFS maxMBpS (7 – 8 GB/s, 1 application thread)

In our quest to understand why GPFS performance was limited, we discovered that only about 120 threads out of the 400 pre-fetch threads configured for the GPFS daemon were used by the GPFS daemon. To further improve GPFS performance, we increased the maxMBpS to 15000. This new setting is only available in this *modified* GPFS. In comparison, GPFS 2.2 only allows the maxMBpS to have a maximum value of 5000.

Using the new 15000 maxMBpS setting, we were able to measure 7 – 8 GB/s sequential I/O performance with 80% to 90% CPU utilization. The high CPU utilization suggests that the sequential I/O performance obtained by GPFS pre-fetching may be gated by the CPU cycles used to copy data between the application buffer and the file system cache. This finding suggested that the application needed to use Direct I/O for more efficient sequential I/O operations.

In this stage, we measured 7 – 8 GB/s sequential I/O performance using the Xdd I/O application described earlier. We also obtained similar single file performance using "gpfsperf" (a program distributed with GPFS).

We expect that with a GPFS 2.2 PTF to allow larger maxMBpS, applications doing large sequential I/O on a similarly configured system will be able to obtain similar performance as was achieved by the *modified* GPFS.

Using Direct I/O with application I/O buffer pinning (12 - 12.5 GB/s, 128 application threads)

When the Xdd application was rerun using Direct I/O, we only observed 2 GB/s sequential I/O throughput using 16 application threads. Further investigation revealed that the lower I/O performance occurred because:

- GPFS “pins” the application buffers used for direct I/O.
- For a given application request, GPFS did not issue concurrent I/Os to disks. GPFS breaks down a large application I/O request into multiple disk requests based on the GPFS block size. For a given application I/O request, GPFS waited for each disk request to complete before issuing the next disk request.

The pinning of application buffer by GPFS during the I/O path results in higher CPU utilization. As mentioned previously, the application can pin buffers outside the I/O critical path using SHM_LGPAGE and SHM_PIN flags when allocating buffers in shared memory. We added the feature to “inform” the *modified* GPFS that the application buffer has been pinned in order to avoid the GPFS pinning overhead in the I/O critical path. The resulting measurements using 128 application threads and Direct I/O showed performance of 12 - 12.5 GB/s sequential read or write performance.

Enabling GPFS concurrency for a single large application Direct I/O request (12 - 12.5 GB/s, 16 application threads)

To enhance sequential I/O performance, applications often issue large I/O requests to the file system. Each large I/O request is converted by GPFS internally into multiple smaller requests to different disks. In GPFS 2.2, these multiple requests are handled serially. To compensate for this serial behavior within GPFS, we had to use a large number of application threads to ensure that disks could be used concurrently.

To further enhance the GPFS performance, we updated the *modified* GPFS to allow multiple GPFS internal disk requests generated from a large application I/O request to proceed in parallel. These changes were made only to the read and write path--and not to the create path.

With this enhancement, we were able to duplicate the 12 – 12.5 GB/s sequential I/O performance with just 16 application thread. The GPFS block size was 4 MB and 109 hdisks were used. This enhancement allows a single application thread to obtain much higher sequential I/O performance using Direct I/O.

A different approach - Single Striped LV GPFS (14.5 – 15.8 GB/s, 16 application threads)

The performance enhancement discussed in the last paragraph was based on a GPFS file system that only used GPFS striping. In this sub-section, we describe our testing of GPFS performance with AIX LVM striping.

For this part of the analysis, a single striped LV with 20 TB was created from 106 RAID5 8+P arrays available at that time. A GPFS file system was then created using this 20 TB striped LV as a single logical GPFS disk. With this configuration, we measured the best GPFS performance of this benchmark. With 32 MB GPFS block size and 16 application threads with each thread making 256 MB size I/O requests, we measured 15.8 GB/s sequential I/O read performance (as shown in Figure 7 and Figure 8), and 14.5 GB/s sequential write performance (as shown in Figure 10, Figure 11, and Figure 12).

The CPU utilization of either reads or writes was about 40% (as shown in Figure 9 and Figure 10). The low CPU utilization is the result of Direct I/O usage and pinning of application buffers outside the I/O critical path. As noted earlier, there is no plan to support 32 MB large block size because of other implications.

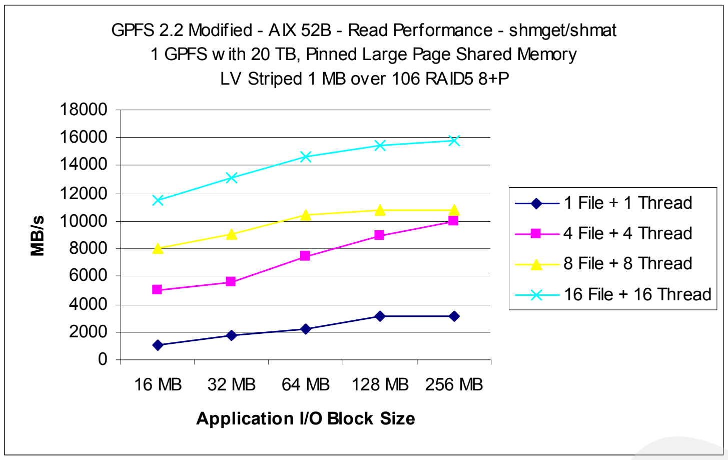

Figure 7 GPFS read performance using Direct I/O to large page pinned shared memory

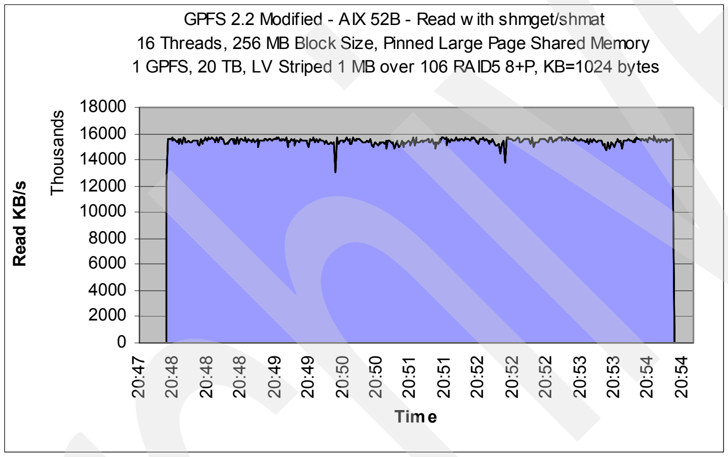

Figure 8 GPFS Read performance using Direct I/O to large page pinned shared memory with 256 MB I/O size and 16 threads

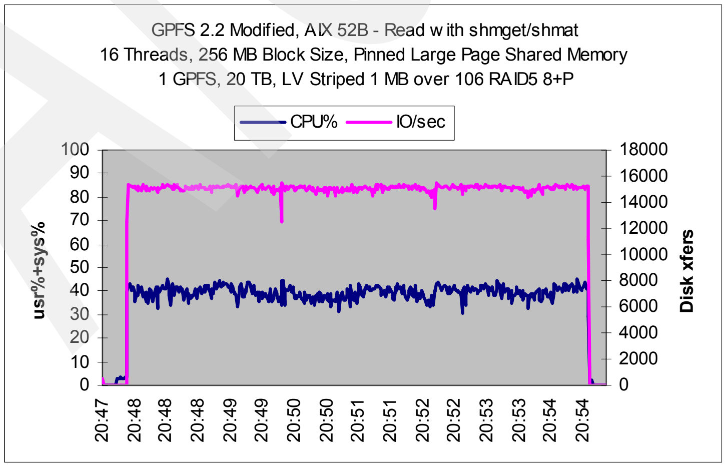

Figure 9 GPFS CPU utilization and IOs/sec during reads with direct I/O to large page pinned shared memory with 256 MB I/O size and 16 threads

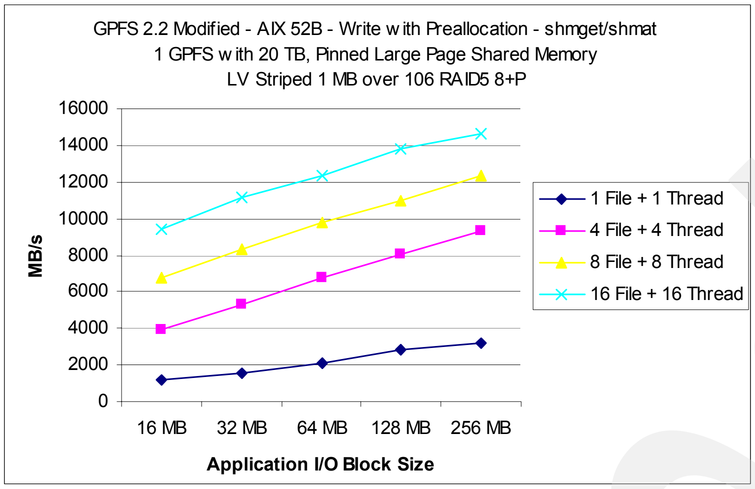

Figure 10 GPFS write performance using Direct I/O to large page pinned shared memory

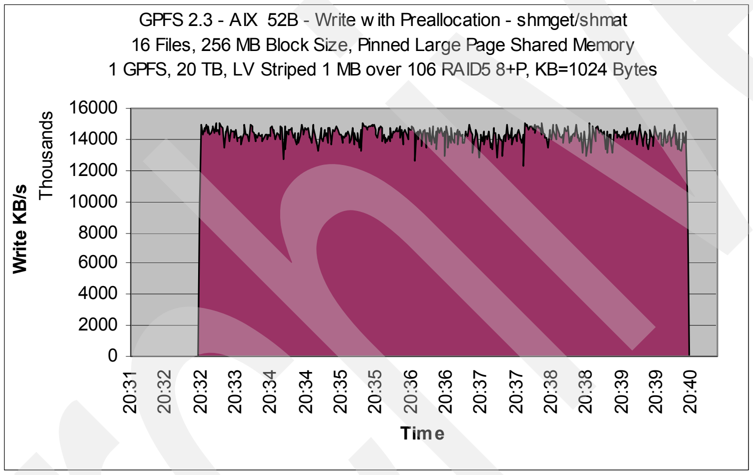

Figure 11 GPFS write performance using Direct I/O to large page pinned shared memory with 256 MB I/O size and 16 threads

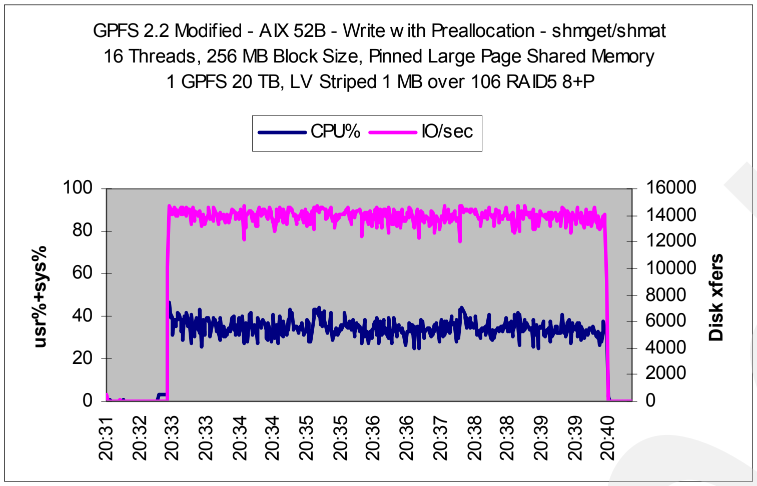

Figure 12 GPFS CPU utilization and IOs/sec during writes with Direct I/O to large page pinned shared memory with 256 MB I/O size and 16 threads

GPFS read-behind-write performance

Read-behind-write is a technique used by some high-end customers to lower latency and improve performance. The read-behind-write technique 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. This concept is beneficial on machines with slow I/O performance. For a high I/O throughput machine such as pSeries 690, it may be worth considering first writing the entire file out in parallel and then reading the data back in parallel.

There are many ways that read-behind-write can be implemented. In the scheme 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 one does not care about how many records that a reader lags behind the writer, then one 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 are written. The reader can then pull the global variable to find out how many records it has to read.

Each pair of read-behind-writes 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 GPFS I/O throughput is 6.35 GB/s for writes and 6.35 GB/s for reads delivering an aggregate I/O throughput of 12.7 GB/s.

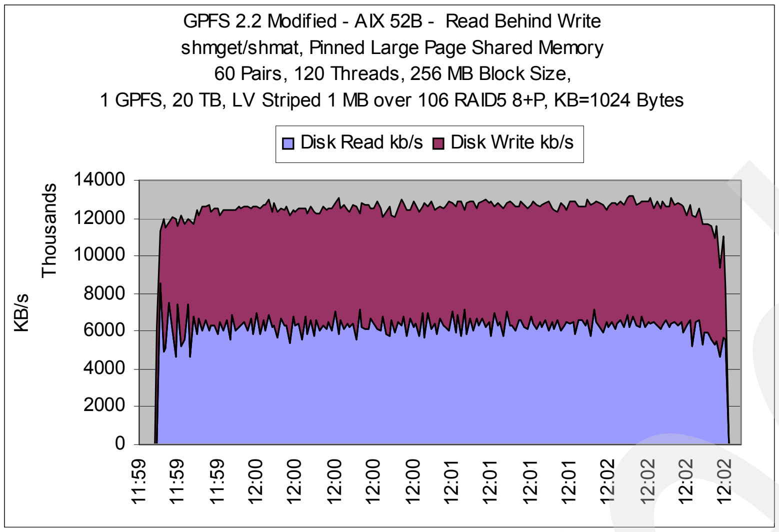

Figure 13 GPFS read-behind-write performance using Direct I/O to large page pinned shared memory with 256 MB I/O size and 60 pairs of threads

As shown in Figure 14, the performance is gated by the CPU utilization. The high CPU utilization of the read-behind-write scenario is contributed by the locking required to synchronize the reader and writer. In contrast, the CPU utilization was only about 40% for the aggregated 15.8 GB/s sequential read performance and 14.5 GB/s sequential write performance reported in the last section.

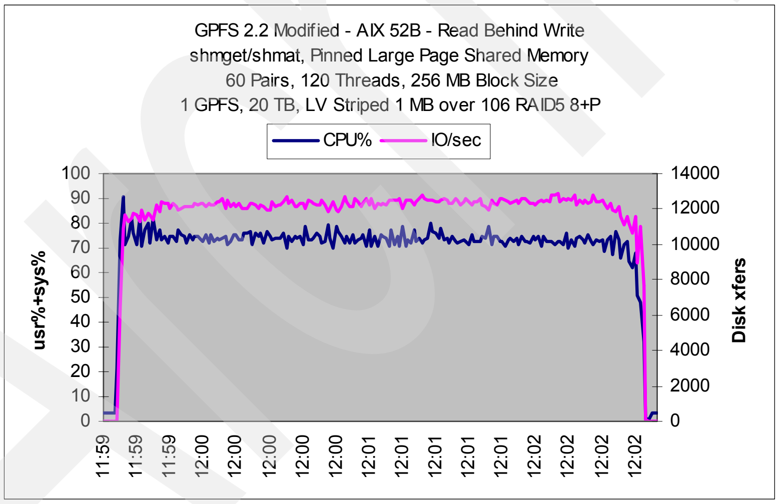

Figure 14 GPFS CPU utilization and IOs/sec during read-behind-writes with Direct I/O to large page pinned shared memory with 256 MB I/O size and 60 pairs of threads

Conclusions

The 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 very good performance using a high performance GPFS with very reasonable CPU overhead.

The scalability to “near raw” hardware performance gives the option of using GPFS to meet almost any high sequential I/O requirement even on a single node. *The delivered* “raw” performance of the I/O subsystem matches exactly the *designed* theoretical maximum throughput.

The GPFS file system on the pSeries server has proven itself in numerous HPC and other customer applications—even using first-generation RIO drawers. GPFS performance has been well established in large cluster environments. In this benchmark, we have shown exceptional performance using GPFS even on a single-node IBM pSeries 690.

Allowing users to configure GPFS using either GPFS striping or AIX LVM striping provides configuration flexibility while delivering a significant proportion of the available I/O bandwidth of the server. Configuration and tuning suggestions to obtain high performance are discussed in “GPFS sequential read or write performance considerations” on page 19 and “GPFS read-behind-write performance” on page 26. The performance discussion in this paper may also provide guidance to other GPFS environments. Furthermore, the performance of a single IBM pSeries 690 server is comparable to the aggregate throughput observed on some large clusters.

References

- *IBM @server® pSeries® 690 Configuring for Performance*, Harry Mathis, John D. McCalpin, Jacob Thomas, May 6, 2003
- *GPFS Primer for AIX Clusters*, available at:

  `http://www-.ibm.com/servers/eserver/pseries/software/whitepapers/gpfs_primer.html`
- *General Parallel File System (BPFS) 1.4 for AIX - Architecture and Performance,* Heger, D. and Shah, G.
- *GPFS on AIX Clusters: High Performance File System Administration Simplified*, SG24-6035-00, Abbas Farazdel, Robert Curran, Astrid Jaehde, Gordon McPheeters, Raymond Paden, Ralph Wescott, available at:

  `http://www.redbooks.ibm.com`
- *The Complete Partitioning Guide forIBM @server® pSeries® Servers*, SG24-7039-01, Kelgo Matsubara, Nicolas Guerlin, Stefan Reimbold, Tomoyuki NiiJima, available at:

  `http://www.redbooks.ibm.com`
- *IBM Total Storage: FAStT600/900 and Storage Manager 8.4*, SG24-7010-00, Bertrand Dufrasne, Bernd Baeuml, Carlos De Nobrega, Ales Leskosek, Stephen Manthorpe, Jonathan Wright, available at:

  `http://www.redbooks.ibm.com`

Acknowledgements

Thanks to the following people for their contribution to this project:

- Pat Buckland – TCEM for I/O on iSeries™ and pSeries, IBM
- Alan Byrd - System Integration, IBM
- Thomas E. Engelsiepen – General Parallel File System, Almaden Research Center, IBM
- 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.
- Daniel L McNabb – General Parallel File System, Almaden Research Center, IBM

- Raymond L. Paden – HPC Technical Architect, IBM
- 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
- James C. Wyllie – General Parallel File System, Almaden Research Center, IBM

The team that wrote this Redpaper

Gautam Shah is a Senior Engineer with the Scalable I/O (GPFS) team in the Systems and Technology group. His areas of expertise include High Performance Computing, File Systems, Communication Protocols, Parallel & Distributed systems, Programming Models with a focus on architecture and performance.

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 10 years of database experience in pSeries and AIX. He is certified in UDB 8.1, UDB for clusters, Oracle9i, SAP UNIX/Oracle, AIX, HACMP™, Business Intelligence and Grid Computing Technical Sales. His areas of expertise include database system design, implementation, tuning and high availability.

Notices

Trademarks