GPFS
Programming, Configuration and Performance Perspectives

Raymond L. Paden, Ph.D.
HPC Technical Architect
Deep Computing

6-7 Apr 2004

raypaden@us.ibm.com
713-940-1084

Special Notices from IBM Legal

There are a lot of ways to do disk I/O.

Since we are interested in HPC I/O in general and parallel I/O in particular, lets first define parallel disk I/O and then briefly survey different paradigms of disk I/O used on HPC environment.

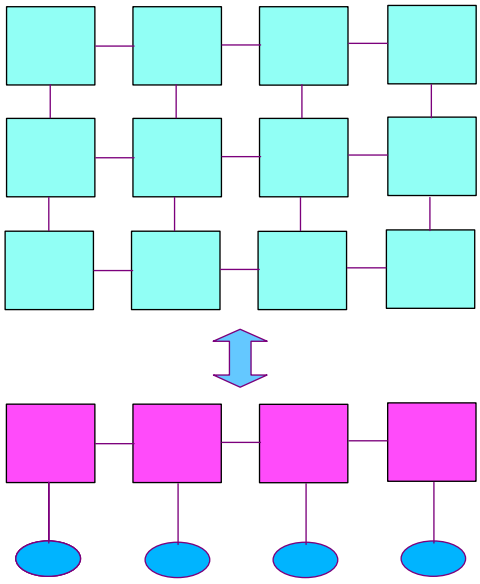
Assume that term "I/O" in this presentation refers to disk I/O unless stated otherwise.

Definition of Parallel I/O

Definition by R. Lusk of ANL:

- Multiple processes (possibly on multiple nodes) participate in the I/O
- Application level parallelism
- “File” is stored on multiple disks on a parallel file system
- Additional Interfaces for I/O (can impact portability)

What is Parallel I/O?



COMMENT:
As HPC systems have matured, the models for storage I/O have become more varied and complex.

adapted from presentation by A. Patra

What is Parallel I/O?

Parallel I/O should *safely* support

- application level I/O parallelism across multiple computational nodes
- physical parallelism over multiple disks and servers
- parallelism in file system overhead operations

A parallel file system must support

- Parallel I/O
- consistent global name space across all nodes of the cluster
  - this includes maintaining a consistent view across all nodes for the same file
- programming model allowing programs to access file data
  - distributed over multiple nodes
  - from multiple tasks running on multiple nodes
- physical distribution of data across disks and network entities
- eliminates bottlenecks both at the disk interface and the network, providing more effective bandwidth to the I/O resources

What is Parallel I/O?

The promise of parallel I/O is increased performance and robustness. And it is more natural in multi-node HPC system.

The challenge of parallel I/O is that it is a more complex model of I/O to use and manage.

Storage Architecture Taxonomy

The following pages examine an architectural taxonomy of storage I/O architectures commonly used in HPC systems. They support varying degrees of parallel I/O and do not represent mutually exclusive choices.

- **Conventional I/O**
- **Asynchronous I/O**
- **Network File Systems**
- **Basic Parallel I/O**
- **Centralized Metadata Server with SAN Attached Disk**
- **High Level Parallel I/O**
- **Recent Developments**

Conventional I/O

- Local file systems
- Basic, "no frills, out of the box" file system
- Journal, extent based semantics
  - *journaling:* to log information about operations performed on the file system meta-data as atomic transactions. In the event of a system failure, a file system is restored to a consistent state by replaying the log and applying log records for the appropriate transactions.
  - *extent:* a sequence of contiguous blocks allocated to a file as a unit and is described by a triple consisting of <logical offset, length, physical>
- If they are a native FS, they are integrated into the OS (*e.g.*, caching done via VMM)
- Intra-node process parallelism
- Disk level parallelism possible via striping
- Not truly a parallel file system
- Examples: Ext3, JFS, XFS

Asynchronous I/O

- Abstractions allowing multiple threads/tasks to *safely* and simultaneously access a common file
- Built on top of a base file system
- Parallelism available if its supported in the base file system
- Part of POSIX 4, but not supported on all unix based file systems (e.g., Linux)
- AIX, Irix, Solaris support AIO, but Linux does not *yet* support AIO

Networked File System (NFS)

- Disk access from remote nodes via network access (*e.g.*, TCP/IP over Ethernet)
- NFS is ubiquitous and the most common example
  - it is not truly parallel
    - old versions are not cache coherent (is V3 truly safe?)
    - write requires O_SYNC and -noac options to be safe
  - poorer performance for I/O intensive HPC jobs
    - write: only 90 MB/s on system capable of 400 MB/s (4 tasks)
    - read: only 381 MB/s on system capable of 740 MB/s (16 tasks)
  - uses POSIX I/O API, but not its semantics
- useful for on-line interactive access to smaller files
- while NFS is not designed for general parallel file access on an HPC system, by placing restrictions on an application's storage I/O model, some customers get "good enough" performance from it

Basic Parallel I/O

- Satisfies basic definition of parallel I/O
- Parallelizes file, metadata and control operations
- POSIX I/O model with extensions
  - byte stream using API with read(), write(), open(), close(), lseek(), stat(), etc.
  - extends POSIX model to support *safe* parallel data access semantics
    - these options guarantee portability to other POSIX based file systems for applications using the POSIX I/O API
  - generally has API extensions, but these compromise portability
- Good performance for large volume, I/O intensive jobs
- Works best for large block, sequential access patterns, but vendors can add optimizations for other patterns
- Example: GPFS (IBM...best of class), GFS (Sistina/Redhat)
  - GPFS has very flexible cluster/disk connectivity allowing large node count scaling

Centralized Metadata Servers with SAN Attached Disk

- Parallel user data semantics, but non-parallel metadata semantics
  - Support POSIX API, but with parallel data access semantics
- Metadata maintained and accessed from a single common server
  - Failover features allow a backup metadata server to takeover if the primary fails
  - Uses Ethernet (100 MbE or 1 GbE) for metadata access
  - Potential scaling bottleneck??
- All "disks" connected to all client nodes via the SAN
  - file data accessed via the SAN, not the node network
    - removes need for expensive node network (*e.g.*, Myrinet)
  - inhibits scaling due to cost of FC Switch Tree (*i.e.*, SAN)
- Ideal for smaller numbers of nodes
  - SNFS *advertises* up to 50 clients (is this reality?)
  - CXFS scales only to 10-12 servers from some users
- Example: CXFS (SGI), SNFS (ADIC), SANFS (IBM)
  - These may be called "parallel like" file systems since metadata processing is not parallel

Higher Level Parallel I/O

- High level abstraction layer providing parallel model
- Built on top of a base file system (conventional or parallel)
- MPI-I/O is the ubiquitous model
  - parallel disk I/O extension to MPI in the MPI-2 standard
  - semantically richer API
  - portable
- Requires significant source code modification for use in legacy codes, but it has the adavantages of being a standard (e.g., syntactic portability)

Recent Developments

Lustre and Panasas, are 2 recently developed HPC style parallel file systems which began "from a clean sheet of paper" in their design that distinguishes them from other file systems in this taxonomy. They have a number of architectural similarities.

- **3 component architecture**
  - storage clients, storage servers, metadata servers (e.g., metadata)
  - file data access over the node network between storage clients and servers (e.g., GbE, Myrinet)
- **Object oriented architecture**
  - object oriented disks are not generally available yet, so the current implementation is in SW and not fully generalized
  - OO design is blind to the application (i.e., uses POSIX API with parallel semantics)
- **Designed to fascilitate storage management (e.g., virtualization for SanFS)**
- **Focus on Linux/COTS environments**

With this background, lets look closer at GPFS.

- What is GPFS?
- Why is GPFS needed?
- What applications is GPFS good for?
- How can GPFS be configured?
- What are GPFS's basic features?

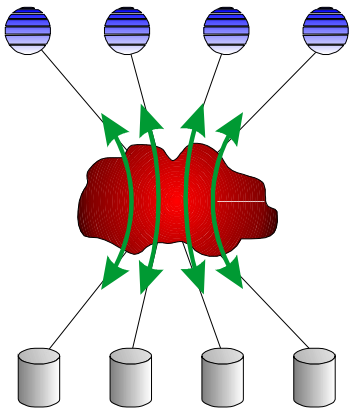
First consider these questions from a high-level perspective, then re-visit them in greater detail later.

What is GPFS?

GPFS = General Parallel File System

IBM's shared disk, parallel file system for AIX, Linux clusters

- *Cluster*: 768 nodes today, fast reliable communication, common admin domain
- *Shared disk*: all data and metadata on disk accessible from any node through disk I/O interface (i.e., "any to any" connectivity)
- *Parallel*: data and metadata flows from all of the nodes to all of the disks in parallel
- *RAS*: reliability, accessibility, serviceability
- *General*: supports wide range of HPC application needs over a wide range of configurations



Why is GPFS needed?

Clustered applications impose new requirements on the file system

- "Any to any" access
  - any node in the cluster must access to any data in the cluster
- Parallel applications need access to the same data from multiple nodes
- Serial applications dynamically assigned to processors based on load
  - need high-performance access to their data from wherever they run
- Require both good availability of data and normal file system semantics

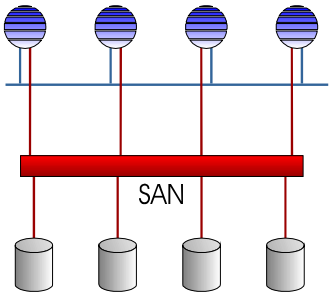
GPFS supports this via:

- Uniform access – single-system image across cluster
- Conventional Posix API – no program modification
- High capacity – large files, 100TB + file system
- High throughput – wide striping, large blocks, many GB/sec to one file
- Parallel data and metadata access – shared disk and distributed locking
- Reliability and fault-tolerance - node and disk failures
- Online system management – dynamic configuration and monitoring

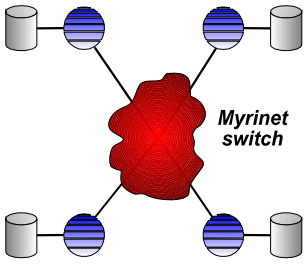
Example GPFS Applications

- Customer applications that require fast, scalable access to large amounts of file data. These applications may be serial or parallel, reading or writing.
  - Applications that serve data to visualization engines
  - Seismic data acquisition processing for serial or parallel reading/writing of files
- Environments with very large data, especially when single file servers (such as NFS) reach capacity limits
  - Digital library file serving
  - Access to large CAD/CAM file sets
  - Data mining applications
  - Data cleansing applications preparing data for data warehouses
- Applications requiring data rates which exceed what can be delivered by other file systems
  - Large aggregate scratch space for commercial or scientific applications
  - Internet serving of content to users with balanced performance
- Applications with high availability (HA) file system requirements

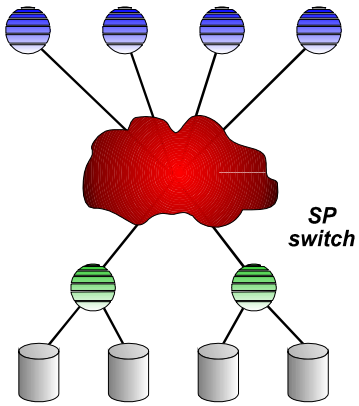
Example GPFS Configurations



Storage area network



Symmetric cluster



Cluster with dedicated storage nodes

These are some of the basic possible configurations... there are many others!

GPFS Features

1. General Parallel File System
   - mature IBM product generally available for 6 years
2. Clustered, shared disk, parallel file system for AIX and Linux
3. Adaptable to many customer environmnets by supporting a wide range of basic configurations and disk technologies
4. Provides safe, high BW access using the POSIX I/O API
5. Provides non-POSIX advanced features (*e.g.*, data-shipping, hints)
   - used by MPI-IO, but are also directly available to the application program
6. Provides good performance for large volume, I/O intensive jobs
7. Works best for large record, sequential access patterns, but also has optimizations for other patterns (*e.g.*, strided, backward)
8. Strong RAS features (reliability, accessibility, serviceability)
9. Converting to GPFS does not require application code changes provided the code works in a POSIX compatible environment

GPFS Features

GPFS Performance Features

1. striping
2. large blocks (with support for sub-blocks)
3. byte range locking (rather than file or extent locking)
4. access pattern optimizations
5. file caching (*i.e.*, pagepool)
6. prefetch, write-behind
7. multi-threading
8. distributed management functions (*e.g.*, metadata, tokens)
9. multi-pathing (i.e., multiple, independent paths to the same file data from anywhere in the cluster)

GPFS Features

GPFS provides many of its own RAS features and exploits RAS features provided by various subsystems

1. If a node fails providing GPFS management functions, an alternative node assumes responsibility preventing loss of file system.
2. In VSD/NSD mode, each disk has a primary and secondary server. In RDP or HACMP mode, all disks are seen by all nodes. Therefore loosing a node or an adapter does not result in loss of access to data.
3. In SAN environment, failover prevents loss of access to data.
4. GPFS on RAID architectures protects against loss of data and loss of data *access*.
5. Online and dynamic system management
   e.g., mmadddisk, mmaddnode, mmchconfig, mmchfs

GPFS Features

Other Features

1. Disk scaling allowing large, single instantiation global file systems (100's of TB now, PB in near future)
2. Node scaling (100's of nodes now, 1000's of nodes in near future) allowing large clusters and high BW... many GB/s
3. Journaling (logging) File System - logs information about operations performed on the file system meta-data as atomic transactions that can be replayed
4. Data Management API (DMAPI) - Industry-standard interface allows third-party applications (e.g. TSM) to implement hierarchical storage management

Tested Scaling Limits

- **AIX**
  - maximum file system size
    - 200 TB
  - maximum nodeset size
    - VSD: 128 nodes (or 512 nodes with system assurance agreement)
    - SAN: 128 nodes
- **Linux**
  - maximum file system size
    - 75 TB
  - maximum nodeset size
    - SAN or NSD model: 768 nodes

COMMENT

These are testing statements, not architectural limits... larger configurations are possible and will be a reality in near future.

GPFS is Available for AIX and Linux

- GPFS has been designed so that its architecture, commands, programming APIs and other GPFS specific entities are nearly identical for both AIX and Linux.
- Same source code except for a small "portability layer"
- The differences that do exist are in the OS substrate

Why is I/O Important in an HPC System?

A system designed to perform TFLOP calculations must be able to access TB of data per second.

Comment: Do not get hung "hung up" on details here.... this is an attention getting statement made for effect. The point is that an HPC system crunching "lots" of numbers must produce/consume "lots" of data.

Example: A HPC system capable of burst computation rates of 200 GFLOPS can produce/consume disk data at rates > 13 GB/s.

Why is Parallel I/O Important?

- Parallel I/O *safely* provides "any to any" connectivity to parallel jobs within a single "system image"
- The work of parallel I/O can naturally be distributed over multiple nodes; this provides good performance scaling in a multi-node HPC system

Why is Parallel I/O Important?

An Example Analysis Using Amdahl's Law

Speedup = 1 / (f + F/n) where
- F = fraction time that can utilize parallelism
- f = fraction of time that can NOT utilize parallelism (n.b., f = 1 - F)
- n = ideal speedup

Parallel Efficiency is then
- Efficiency = 100 * Speedup / n

I/O *can, but need not* be a large contributor to f in MPP systems. I call the inefficiency represented by the term f in Amdahl's law as "Amdahl inefficiency" or "Amdahl overhead".

Consider a job on a 32 node/64 CPU Linux Cluster (LC). This job, when executed on a single node accessing a local scratch disk, devotes 10% of its job time writing to a file. By contrast, the LC writes via NFS to a single file server preventing parallel I/O operation. Assume the following...
- the file server is the same "out of the box" Linux system used for the sequential test
- the Ethernet connection rate used for NFS exceeds the sequential job's write rate
- number crunching and file reading phases of the job runs perfectly parallel (i.e., are small enough to be ignored)

In other words, the writes are sequentialized.

What are the speedup and efficiency values for this job?

Number of Tasks	Speedup	Efficiency
8	4.71	58.9%
16	6.40	40.0%
32	7.85	24.5%
64	9.86	15.4%

Lets examine a simple disk I/O program and modify it to do parallel disk I/O so that we can better appreciate the tasks that a parallel file system *must do* and that GPFS *does* to allow a programmer to do parallel I/O *safely*.

Simple I/O Program

```
int main()
{
   int  fd, k, nrec = 1024, bsz = 16384;
   char *fid_out = "myfile", buf[bsz];
   offset_t soff;      /* 64 bit seek offset */

   fd = open(fid_out, O_WRONLY | O_CREAT | O_TRUNC, 0777);
   for (k = 0; k < nrec; k++)
   {
      do_something(buf, bsz);
      soff = (offset_t)k * (offset_t)bsz;
      llseek(fd, soff, SEEK_SET);
      write(fd, buf, bsz);
   }
   close(fd);

   return 0;
}
```

What do we need to do to parallelize disk I/O?

1. mapping function (i.e., locating proper data across multiple disks over different nodes)
2. message passing (i.e., shipping data between client node task and disk located on remote server)
3. caching system (e.g., coherence, aging, swapping, "data-shipping", etc.)
4. parallel programming model (e.g., data striping, data decomposition, node to disk access patterns)
5. critical section programming
6. performance tuning
7. maintain state information
8. provide an API

What do we need to do to parallelize disk I/O?

1. mapping function (i.e., locating proper data across multiple disks over different nodes)
2. message passing (i.e., shipping data between client node task and disk located on remote server)
3. caching system (e.g., coherence, aging, swapping, "data-shipping", etc.)
4. parallel programming model (e.g., data striping, data decomposition, node to disk access patterns)
5. critical section programming
6. performance tuning
7. maintain state information
8. provide an API

That's a lot of work! (several 100 KLOC in GPFS)

That's a Lot Work!

```
p    = pio_init(ndim,reclen)
stat = pio_set_alloc_inc(p,nrec)
stat = pio_set_bf(p,traces)
stat = pio_set_decomp(p,dtype)
stat = pio_set_map_cache_nblock(p,nblock)
stat = pio_set_map_update_policy(p,utype)
stat = pio_set_name(p,idim,name)
stat = pio_set_nmap(p,mapopt)
stat = pio_set_preclear(p,option)
stat = pio_set_ws(p,ws)
stat = pio_add_seq(p,idim,first,last,inc)
stat = pio_put_namespace(p,ns)
stat = pio_create_file(p,file,mode,id,nnode,inode,nclone,iclone...
p    = pio_open_file(file,accesstype,id,nnode,inode,nclone...
stat = pio_update(p,flags)
stat = pio_close(p)
stat = pio_delete(p)
mrbf = pio_get(p,keys)
```

18 lines from a 4500 line utility to do parallel I/O (much simpler utility than GPFS)

That's a Lot Work!

```
slot = pio_slot(p,keys);
if (-1 == slot) return NULL;
if (p->decomp != PIO_DECOMP_NONE)
   if (!pio_is_mine(p,slot)) return NULL;
mr = pio_find_memrec(p,slot);
if (!mr)
{
   if (!(mr = pio_lru_swapout(p))) return NULL;
   mr->slot = slot;
   if (-1 == pio_swapin(p,mr)) return NULL;
}
if (++p->tick < 0)
   if (-1 == pio_tick_reset(p)) return NULL;
mr->lru = p->tick;

return mr->data;
```

Another 18 lines from a 4500 line utility to do parallel I/O

GPFS Design Goal

Provide a parallel I/O system conforming to the POSIX API standard...

GPFS Design Goal

Provide a parallel I/O system conforming to the POSIX API standard...

therefore, you can write an application code to access one file without worrying (too much) about what the other tasks are doing.

GPFS Design Goal

Provide a parallel I/O system conforming to the POSIX API standard...

therefore, you can write an application code to access one file without worrying (too much) about waht the other tasks are doing.

You can't be blind, but you can focus on application needs without worrying too much about system issues. If the code is sequential, you can get the full benefits of a parallel file system without worrying at all about it!!

A Simple Parallel I/O Program

```
int main()
{
   int  fd, k, nrec = 1024, bsz = 16384, ntask = 2, tid;
   char *fid_out = "myfile", buf[bsz];
   offset_t soff;     /* 64 bit seek offset */

   tid = spawn_task(ntask);

   fd = open(fid_out, O_WRONLY | O_CREAT | O_TRUNC, 0777);
   for (k = tid; k < nrec; k+=ntask)
   {
      do_something(buf, bsz);
      soff = (offset_t)k * (offset_t)bsz;
      llseek(fd, soff, SEEK_SET);
      write(fd, buf, bsz);
   }
   close(fd);

   return 0;
}
```

In reality you will need more than this, but it will be application oriented ONLY!

You Do Not Have to Worry About...

- Which disk/file to write to (there is only one file seen by all tasks)
- If some other task/job has opened the file
- If somebody else is writing to the file right now
- Cache coherence
- If its portable ... its POSIX compliant!

You Do Not Have to Worry About... (errr... too much)

- Which disk/file to write to (there is only one file seen by all tasks)
- If some other task/job has opened the file
- If somebody else is writing to the file right now
- Cache coherence
- If its portable ... its POSIX compliant!

(provided you do not use the extensions)

Now Consider This:

What Can Go Wrong if the FS is not Parallel?

```
int main()
{
   int  fd, k, nrec = 1024, bsz = 16384, ntask = 2, tid;
   char *fid_out = "myfile", buf[bsz];
   offset_t soff;     /* 64 bit seek offset */

   tid = spawn_task(ntask);

   fd = open(fid_out, O_RDWR | O_CREAT | O_TRUNC, 0777);

   while ((soff = find_record()))    /* assume soff%bsz == 0 */
   {
      critical section begin
         llseek(fd, soff, SEEK_SET);
         read(fd, buf, bsz);
         for (k = ntask; k < bsz; k += ntask)
            buf[k] = do_something(...);
         llseek(fd, soff, SEEK_SET);
         write(fd, buf, bsz);
      critical section end
   }
   close(ofd);

   return 0;
}
```

Now Consider This:
What Can Go Wrong if the FS is not Parallel?

Task 0, node J acquires lock
Task 0, node J reads record N from disk
Task 0, node J modifies buf[] at indicies 0, 2, 4, 6, 8, ...
Task 0, node J writes record N to local cache
Task 0, node J releases lock
Task 1, node K acquires lock
Task 1, node K reads record N from disk he does not know its in node J's cache
Task 1, node K modifies buf[] at indicies 1, 3, 5, 7, 9, ...
Node J flushes cache
Task 1, node K writes record N to local cache
Task 1, node K releases lock
Node K flushes cache and clobbers Task 0's modifications!

Now Consider This:
What Can Go Wrong if the FS is not Parallel?

This sad scenario is quite possible under NFS since it is not cache coherent (after all, its not parallel!).

GPFS maintains cache coherence (among many other parallel tasks) making parallel access to a common file safe (by taking the usual concurrency precautions such as using locks or semaphores).

Now Consider This:
What Can Go Wrong if the FS is not Parallel?

This sad scenario is quite possible under NFS since it is not cache coherent (after all, its not parallel!).

GPFS maintains cache coherence (among many other parallel tasks) making parallel access to a common file safe (by taking the usual concurrency precautions such as using locks or semaphores).

Note: NFS V3 has cleaned much of this up. By using -noac option opening the file with the O_SYNC flag, parallel writes can be done more safely, though this contributes to Amdahl inefficiency by sequentializing parallel writes. However, this is not fool proof. Some customer codes fail under NFS using these options where they run safely without error under GPFS.

Parallel Access from Multiple Nodes

Earlier I said that GPFS...

provides a parallel I/O system conforming to the POSIX standard; therefore, you can write an application code to access one file without worrying (too much) about what the other tasks are doing.

Well there are 2 things to worry about:

1. Normal precautions against RAW, WAR, WAW errors
2. Performance issues
   - overlapping records sequentializes file access and contributes to "Amdahl inefficiency"

RAW = Read After Write
WAR = Write After Read
WAW = Write After Write

Let's survey the GPFS architecture.

This survey will provide a conceptual model for GPFS that will help

- applications and systems programmers more effectively utilize GPFS
- system administrators and architects more effectively design and maintain a GPFS infrastructure

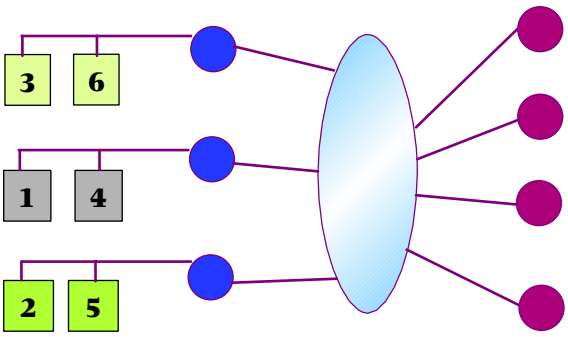
To make a parallel file system work efficiently, it must

- do something many times at once
- do it on big things

This ties together 5 components of GPFS architecture and organization

1. striping
2. blocks and sub-blocks
3. mapping function
4. file caching (page pool)
5. multithreading

Data Striping



Disk Pool | Server Nodes | SP Switch | Application Nodes

- GPFS *stripes* successive blocks of each file across successive disks
- Disk I/O for sequential reads and writes is done in parallel (prefetch, write behind)
- Make no assumptions about the striping pattern
- Block size is configured when file system is configured, and is *not* programmable
- choices are 16, 64, 256, 512, 1024 KB (most common choice = 256 KB)
- transparent to programmer

increasing file offset ---->

Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	..... one file

3 I/Os executed in parallel

Job reads at 30 MB/s
Each disk reads at 10 MB/s

GPFS Blocks and Sub-blocks

- **A GPFS file system can be configured with one of the following block sizes**
  - 1024 KB
  - 512 KB
  - 256 KB
  - 64 KB
  - 16 KB
- **Large block sizes optimize performance when large record accesses are common (by reducing the number of IOPs)**
- **Small block sizes optimize storage when small record accesses are common**
- **In this presentation, assume block size is 256 KB unless stated otherwise.**
- **Blocks can be divided into 32 sub-blocks**
  - a block is the largest chunk of contiguous data that can be accessed
  - a sub-block is the smallest chunk of contiguous data that can be accessed
  - files smaller than a block can be stored in fragments of 1 or more sub-blocks (common in life sciences applications)
  - if sizeof(record) <= sizeof(subblock), then only a sub-block is accessed, but if 2 or more sub-blocks are accessed in block, then the entire block is accessed

Mapping Function Consideration

- **Striping complicates the GPFS mapping function**
- **The GPFS mapping function must**
  - identify which disk contains the blocks of a record
  - determine if a record spans multiple blocks
  - identify which node the disk is served by (under VSD or NSD)
  - determine the striping pattern
  - resolve sub-blocks for small files
  - account for GPFS replicas and failure group restrictions
  - parse the various control files and data structures that GPFS uses to distribute the file data
- **Striping applies to both the data and metadata though the metadata files can have restrictions placed upon them**

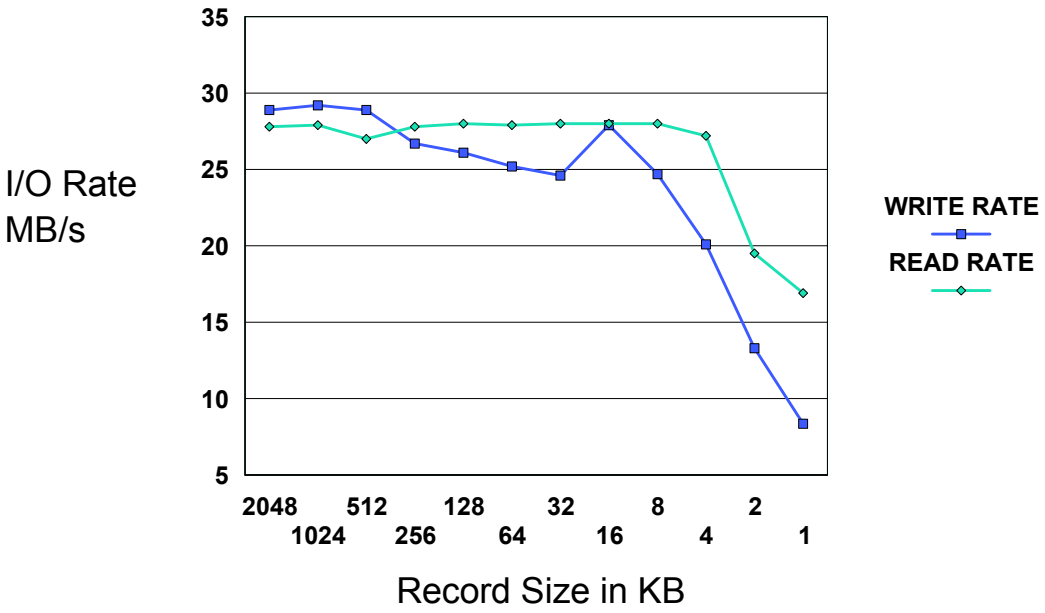
We will look at these issues more carefully by way of an example a little later.

GPFS Block vs. Record Size WRT Performance

- Large records yield excellent performance
  - e.g., 2 MB record yields 8 simultaneous accesses for 256 KB blocks
- Consider small record (e.g., 10 KB)
  - in a 256 KB block, it touches 2 sub-blocks forcing entire block to be accessed
  - only use small fraction of 256KB block (more on this in later slides)
  - two nodes may write to the same 256 KB block forcing serialization
  - "small trickle down a big pipe"
- Consider an even smaller record (e.g., 2 KB)
  - will only access a sub-block
  - while more efficient than reading entire block, it still devotes an entire IOP to the operation

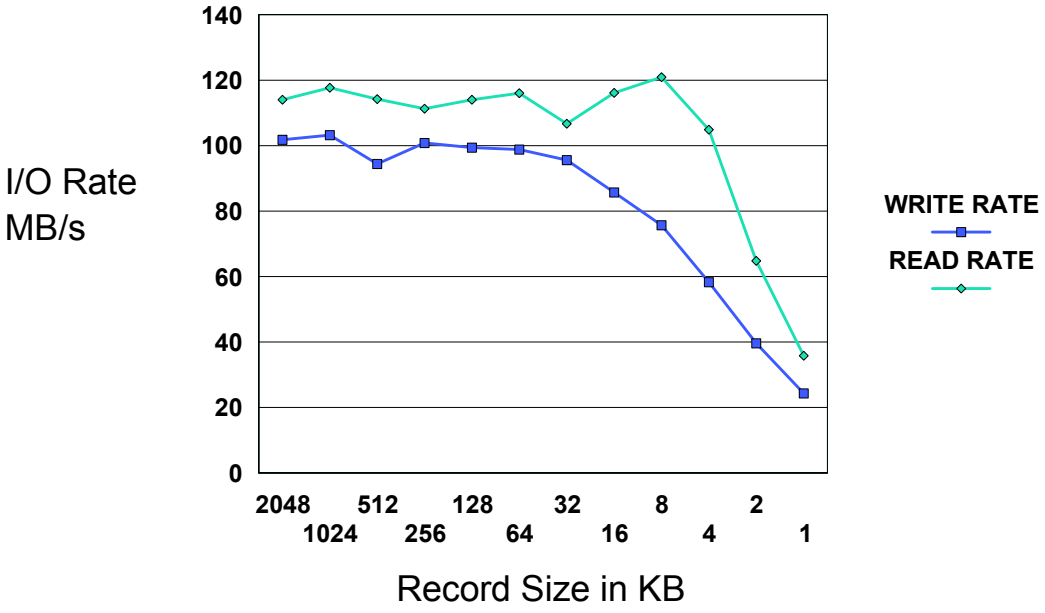
Rule: re-write applications to use large records where possible

GPFS Block Size vs. Record Size



GPFS 1.4 on an HACMP Cluster of 2, 43p260 nodes
Sequential Access Pattern
4.5 GB SSA Disks, Stripe Size = 256KB, Pagepool = 80MB

GPFS Block Size vs. Record Size



GPFS 1.3 on a WH2
1 Task, Sequential Access Pattern
9 GB, SSA Disks, Stripe Size = 256KB, Pagepool = 80MB

The Pagepool - GPFS File Cache

- GPFS cache is called the pagepool
  - 4MB to 512MB for 32 bit OS kernel
  - up to 8 GB for a 64 bit kernel (*n.b.*, if set too large mmfsd will not start)
- GPFS caches data in node memory; there are 2 kinds
  - pinned memory used for user data
  - unpinned memory used for open and recently opened files
    - full i-node cache
    - stat cache (used by stat() system call)
- Optimum sizes (rules of thumb)
  - once max performance is achieved, using larger pagepools yields diminishing returns
  - formula
    - enough to hold the next 2 application operations for each LUN
    - number of LUNs * sizeof(block) * 2
  - observations from selected benchmark tests
    - 80 MB for 4 GB RAM on a 4-way node
    - 320 - 512 MB for 32 GB RAM on a 16-way node
    - 8GB (remains to be tested)
- Large pagepools most helpful when
  - writes can overlap computation
  - heavy re-use of file records (*n.b.*, good temporal locality)
  - semi-random access patterns with acceptable temporal locality (hypothesis to be tested for 8GB)

The Pagepool - GPFS File Cache

- Read cache policy ("prefetch")
- Write cache policy ("write behind")
  - write back (write to cache only)
  - write allocate (allocate cache block before writing)
- GPFS cache is distributed over all client nodes
- Cache coherence protocols make it safe to view it as a single entity from a programming perspective (though performance may suffer if used poorly)
- The caching system (along with striping) creates a great deal of implicit asynchronous operation within GPFS
- Because cache is coherent and writes are atomic, you can avoid reading partially written records or corrupting records by having 2 tasks write to it at the same time
- vmtune does not affect GPFS cache (common misunderstanding for AIX/JFS users)
- Cache management is done via the kernel (since it uses pinned memory)

Multithreaded Architecture

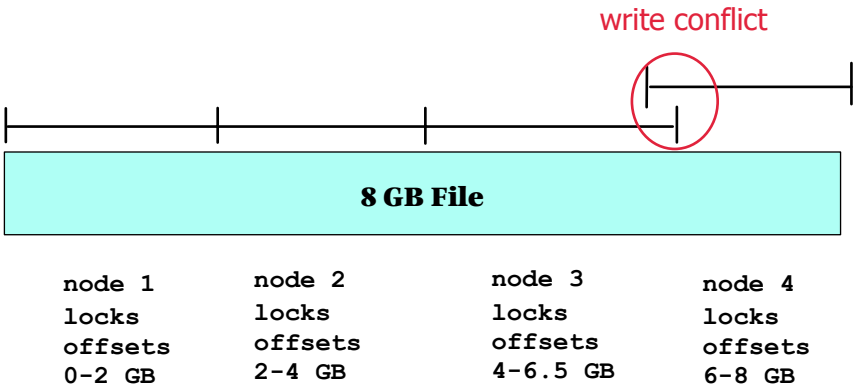
- **GPFS can spawn up to 550 threads/node simultaneously where each thread is a single IOP**
  - there is one thread per block, so large records may require multiple threads
- **The key to GPFS performance is "deep prefetch" which due its multithreaded architecture and is facilitated by**
  - GPFS pagepool
  - striping (which allows multiple disks to spin simultaneously)
  - "anticipatory" algorithms for sequential and strided access or the explicit use of hints (more on this later)
- **Because of the deep prefetch and other performance algorithms in GPFS, use the open flag O_DIRECT with caution.**

Many file systems allow safe concurrent access to a file, but will allow only one task at a time to access a file when the file is being modified. This is inefficient.

GPFS provides a finer grained approach to this allowing multiple tasks to read and write to a file at the same time using "byte range locking".

Parallel Access from Multiple Nodes
Byte Range Locks

- GPFS allows parallel applications on multiple nodes to access non-overlapping ranges of file without conflict or performance loss
- But byte range locks serialize access to overlapping ranges of a file



- byte range locks preserve data integrity
- byte range locks are transparent to the user
- byte range lock patterns can be much more intricate

Parallel Access from Multiple Nodes
Byte Range Locks and Tokens

Byte range locking uses tokens

- a task can access a byte range within a file (read or write) iff it holds a token (via a token client) for that byte range

Token management is distributed between 2 components

1. token management server (*i.e*, "token server")
   - there is only one token server per file system; it resides on a single node
   - it manages tokens on behalf of a particular file system
     - distributes tokens to requesting token clients
     - distributes lists of nodes to token clients requesting conflicting tokens
   - are processed via the kernel
2. token client
   - there is one token client per node per file system running on behalf of all application tasks running on that node
   - it requests/holds/releases tokens on behalf of the task accessing the file

Parallel Access from Multiple Nodes
Byte Range Locks and Tokens

- The process
  - Designed to offload as much operations from the token manager as possible
  - Token semantics
    - tokens are designated as having either read or write access to data or metadata within an associated byte range
    - token manager responsibility
      - "coordinates" access to files
      - distributes tokens or a list of nodes holding conflicting tokens to requesting token clients
    - token client responsibility
      - token clients act on behalf of their tasks (*n.b.*, token operations are blind to the application programmer)
      - a task can access a given byte range without further access to the token manager once it has received a token until another task attempts to access the same byte range
      - the task requests other tasks holding conflicting tokens to release their tokens
      - the task releases a token to the token manager at the request of another task, but it will not do this until it has released the byte range lock for the file (this may include waiting for an I/O operation to complete)
- Comments
  - Accessing overlapping byte ranges where a file is being modified will sequentialize file operations (*n.b.*, this contributes to Amdahl inefficiency)
  - GPFS write operations are atomic

In general, GPFS is designed to perform the same functions on each node and the functions performed on behalf of an application are executed on the node where it is generated.

However, there are 3 specialized management operations which are performed globally that affect the operation of the other nodes in the nodeset.

We shall now take a look at these management functions.

GPFS Management Functions

There are 3 classes of management functions in GPFS.

- **Configuration Manager**
- **File System Manager**
  - File system configuration
  - Manage disk space allocation
  - Token management
  - Quota management
  - Security services
- **Metanode**

GPFS Management Functions
The Configuration Manager and Quorum Rule

- **There is one configuration manager per nodeset**
- **The "oldest continuously operating node" is automatically selected as the configuration manager. (Should it fail, the 2nd oldest one is selected.)**
- **Functions**
  - selects the file system manager node
    - choice can be overridden using **mmchmgr** or **mmchconfig** commands
      - a sysadm may delegate this function and other "overhead" functions (*e.g.*, tape server, backup) to a designated management node and not allow production on this node
      - a sysadm may delegate different nodes as the file system manager when there are multiple file systems within a nodeset
    - its an open question whether production should be allowed to operate on the file system manager... run a benchmark and see what works best!
  - determines whether a quorum exists
    - a quorum is the minimum number of nodes in a nodeset that can be running for the GPFS daemon (i.e., **mmfsd**) to operate
    - quorum = 1 + sizeof(nodeset)/2
    - guarantees that a valid single token management domain exists
      - if communication were lost between nodes without this rule, the nodeset would become partitioned and the partition without a token manager would launch a 2nd token manager and corrupt data

GPFS Management Functions
The File System Manager

- **There is exactly one file system manager per file system**
- **There are 5 file system management functions**
  1. file system configuration
     - adding disks
     - changing disk availability
     - repairing the file system
     - mount/umount processing (this is also done the node requesting the operation)
  2. disk space allocation management
     - controls which regions of each disk are allocated to each node (striping management)
  3. token management
     - discussed above
  4. quota management
     - enforces quotas if it has been enabled (see **mmcrfs** and **mmchfs** commands)
     - allocates disk blocks to nodes writing to the file system
     - generally more disk blocks are allocated than requested to reduce need for frequent requests
  5. security services
     - see manual for details
     - some differences *appear* to exist between AIX and Linux based systems

Comment: disks in this context are LUNs

GPFS Management Functions
The Metanode

- **There is one metanode per open file**
- **The node chosen to be the metanode is**
  - the one which has had the file open for the longest continuous period of time
  - can migrate to any node to meet application requirements
- **Functions**
  - maintains file metadata integrity
  - updates the metadata
    - this is done *only* by the metanode

Quorum

- **GPFS quorum defines the number of nodes in the nodeset which must be functional for GPFS to remain active. If the available nodes fall below the quorum level, GPFS performs recovery in an attempt to achieve quorum again.**
- **Default**
  - one plus half of the *potential* quorum nodes in the GPFS nodeset
  - potential quorum nodes are by default all of the nodes in the nodeset,
    - a subset of the nodeset may be designated as the quorum nodes (this allows scaling beyond RPD limits)
- **Example**
  - consider system with 3 nodes
  - failure of one node allows recovery and continued operation on the two remaining nodes
- **Single node quorum**
  - allows continue functioning of GPFS in the event of the failure of the other node in a 2 node node-set
  - especially useful for p690 ("regatta") class systems

HPC applications often have characteristic access patterns.

GPFS includes algorithms that exploit some of these patterns.

GPFS Access Pattern Optimizations

GPFS uses cache to improve the performance of reading and writing for these access patterns:

- sequential (both forward and backward)
- strided (both forward and backward)

If the pattern is recognized, then the relevant records can be asynchronously pre-loaded into cache.

If the access pattern is not recognized by GPFS, then hints can be provided informing GPFS which records can be pre-loaded into cache.

Random Access Patterns Are Slow

Can not anticipate seek arm movement in advance

Can not prefetch records for reading or preallocate cache blocks for writing

Random Access Patterns with Small Records Are Even Slower

For small records, one must amortize time to deliver block over the amount of data in a record

Assumption: GPFS has no overhead.
Let R = rate to deliver 256KB block; i.e., 256KB / (latency + delivery time)
Let k = record size / 256KB and assume k < 1; i.e., record size < block size
Let r = rate to deliver a random record.
Then r = k * R and r < R

EX: let sizeof(record) = 16 KB and R = 128 MB/s
then k = 1/16 and r = 8 MB/s

Comment: Empirically, iostat will show much higher data rates (close to R) since it is measuring blocks where the application will show the much lower rates based on the actual data accessed (close to r).

Comment: GPFS compensates when a small record sequential access pattern is used since blocks can be meaningfully pre-fetched.

Strided Access Patterns

When GPFS detects a strided order, it prefetches along the stride thus improving performance.

8 tasks @ 1 per node, record size = 16 KB, file size = 5 GB
WH2 with 14 clients and 2 VSD servers, using 36 GB, 10 Krpm, SSA drives

	Write	Read
strides under GPFS 1.2	less than 1 MB/s*	11.1 MB/s*
strides under GPFS 1.3	17 MB/s	58 MB/s

* The strided rate under GPFS 1.2 is the same as the random (without hints) rate under GPFS 1.3.

Improving Strided Access Rates

But notice that increasing the record size from 16KB to 1024 KB, the rates increase.

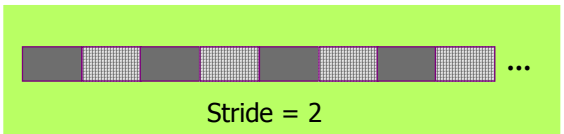
8 tasks @ 1 per node, file size >= 5 GB
WH2 with 14 clients and 2 VSD servers, using 36 GB, 10 Krpm, SSA drives

	Write	Read
record size = 1024 KB	172 MB/s	211 MB/s
record size = 16 KB	17 MB/s	58 MB/s

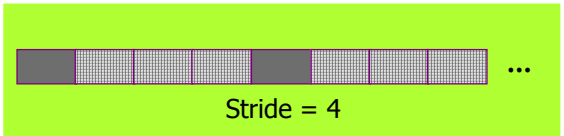
Strided Access Rates

Assume GPFS has no overhead and record size is less than block size
Let s = strided data rate and n = stride factor > 0
Let R = rate to deliver 256KB block; i.e., 256KB / (latency + delivery time)*
Let k = record size / 256KB and assume k < 1
Let record size = 32KB and block size = 256KB
Then s = k*R/n where n*32 < 256, otherwise s = k*R

* If (under GPFS 1.3 and higher) the block is already in cache, then (latency + delivery time) will be much shorter and R much greater.

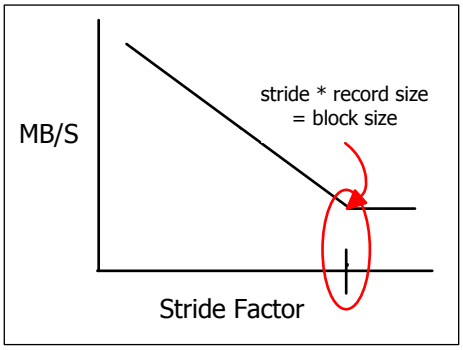


using *half* of the data in the block

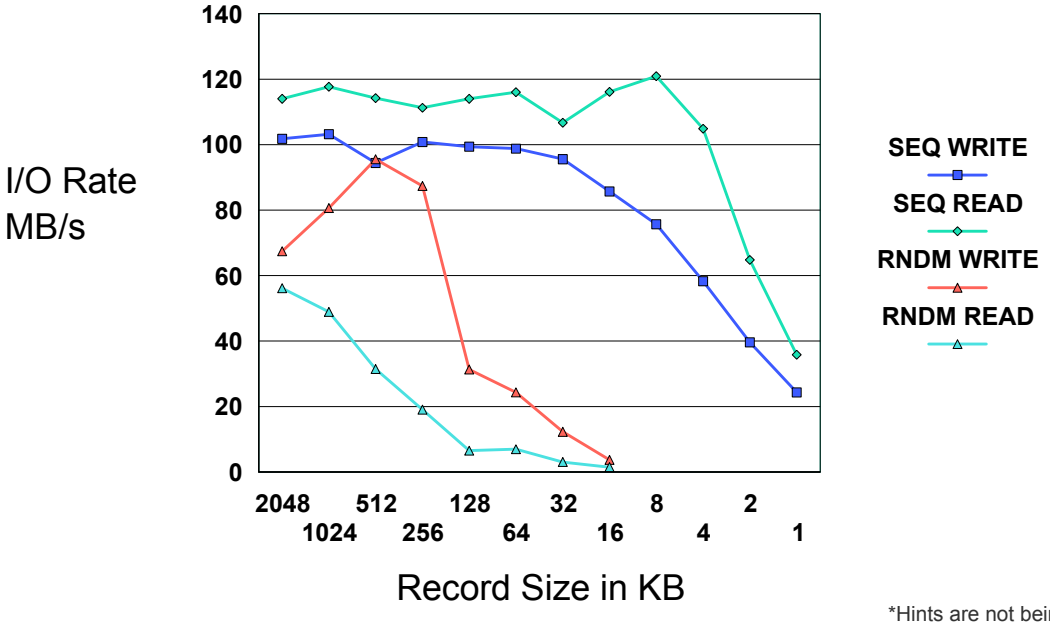


using a *quarter* of the data in the block

Theoretical Strided Data Rate



PUTTING IT ALL TOGETHER:
Block Size vs. Record Size vs. Caching



GPFS 1.3 on a WH2
1 Task, Sequential Access Pattern
9 GB, SSA Disks, Stripe Size = 256KB, Pagepool = 80MB

*Hints are not being used for Random Patterns, so no prefetches occur when recordsize < stripe size.

POSIX I/O is a simple standard covering the basics. Early versions of GPFS stuck closely to this standard. But because of its shortcomings in many environments, IBM has added API extensions to GPFS that go beyond the POSIX I/O API.

These extensions are a mixed blessing. While they improve performance and facilitate important semantics not part of POSIX I/O, they are generally not portable.

Extensions to GPFS

Examine selected non-POSIX GPFS APIs

- multiple access hint (improving random I/O)
- data shipping (trading disk latency for switch latency)

NOTE: There are others under development to be released in the near term, but an NDA is required to discuss them.

GPFS Multiple Access Hint:
Improving Random I/O

- Some applications are intrinsically based on small records
  - sorting jobs
  - dmostack in seismic processing
- The multiple access hint in GPFS allows the programmer to post future accesses and then prefetches them asynchronously.
- Reads are improved substantially for small records, but writes are still slow.

	write rate	read rate
without hints	0.921 MB/s	11.1 MB/s
using hints	3.57 MB/s	62.3 MB/s

8 tasks @ 1 per node, record size = 16 KB, file size = 5 GB
WH2 with 14 clients and 2 VSD servers, using 36 GB, 10 Krpm, SSA drives

- The multiple access hint interface is tedious to use, but a simple to use interface can be crafted.

GPFS Multiple Access Hint:
An Example of a Simple Interface

A simple GPFS multiple access hint interface can be designed by the user (hiding the low level tedium) making it easier for high level applications to use hints.

For example...

```
public:
   int pio_init_hint(struct pio *p, int maxbsz, int maxhint);
   int pio_post_hint(struct pio *p, offset_t soff, int nbytes, int nth, int isWrite);
   int pio_declare_1st_hint(struct pio *p);
   int pio_xfer(struct pio* p, char* buf, int nth);
private:
   int pio_gen_blk(struct pio *p, int nth, int isWrite);
   int pio_issue_hint(struct pio*p, int nth);
   int pio_cancel_hint(int fd);
```

GPFS Multiple Access Hint:
An Example

```
p->fd = open(fid_in, O_RDONLY, 0);
pio_init_hint(p, bsz, MAXHINT);
for (k = 0; k < ntr; k += nhints)
{
   /*-----------------------------------------------------*/
   /* generate next set of seek offsets and issue hints */
   /*-----------------------------------------------------*/
   for (i = 0; i < nhints; i++)
   {
      soff = which_record();              /* build hint */
      pio_post_hint(p, soff, bsz, i, 0);  /* structure  */
   }
   pio_declare_1st_hint(p);               /* issue 1st hint */
   /*-----------------------------------------------------*/
   /* read and process data corresponding to previous hints */
   /*-----------------------------------------------------*/
   for (i = 0; i < nhints; i++)
   {
      pio_xfer(p, buf, i);     /* issue next hint with each */
      do_something(buf);       /* read or write             */
   }
}
```

GPFS Data Shipping:
Improving the Cache Write Miss Penalty

Suppose several nodes are writing to a common block.

Before the next node can write to the cache block, the node that just "dirtied" the cache block must first write it to disk, and the next node must fetch it from disk.

The data shipping mode allows the cache block to be sent to the next node over the SP switch, without going to disk first.

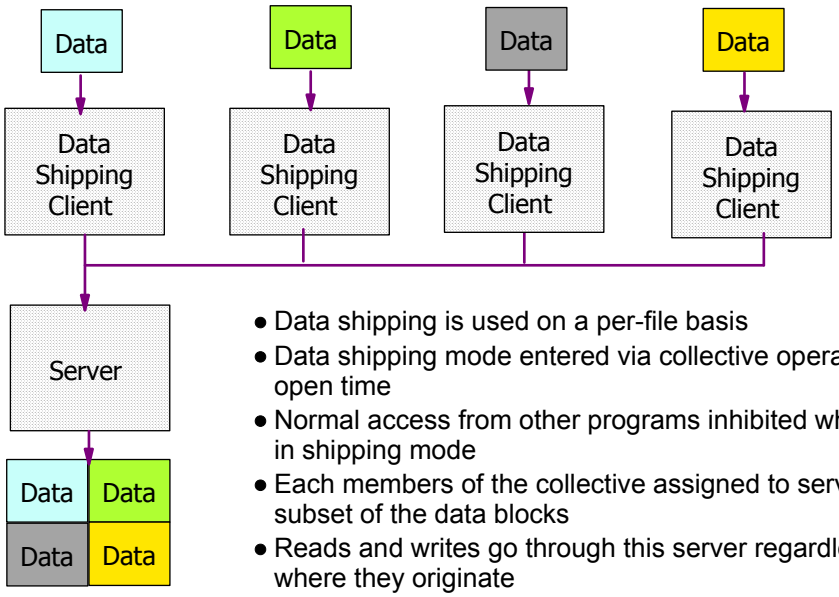
Useful only when multiple nodes frequently access common blocks (must amortize overhead over several accesses).

Drawbacks

- must maintain GPFS cache state information
- significant overhead involved
- deviates from the POSIX standard

Its utility is very application dependent. Experimentation is needed to determine if it will be useful.

GPFS Data Shipping:
Improving the Cache Write Miss Penalty



- Data shipping is used on a per-file basis
- Data shipping mode entered via collective operation at file open time
- Normal access from other programs inhibited while file is in shipping mode
- Each members of the collective assigned to serve a subset of the data blocks
- Reads and writes go through this server regardless of where they originate
- Data blocks only buffered at their server, not other clients

GPFS does not exist in isolation; it must be integrated with other components when designing an overall solution.

The following pages look at selected hardware components commonly used with GPFS file systems.

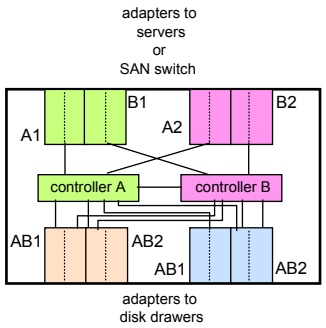
Supported Disk Products

- AIX
  - FAStT 500, 600, 700, 900
  - Enterprise Storage System (Shark)
  - EMC Symmetrix
  - SSA
  - FC disk (*e.g.*, FAStT) must connect via SAN
- Linux
  - FAStT 200, 500, 600, 700, 900
  - FC disk can connect via SAN or to the CEC

COMMENT:
These are testing statements. For example, customers commonly use SCSI.

Disk Controller
FAStT900

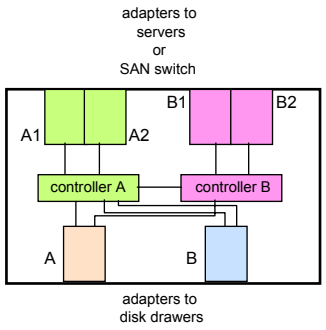
FAStT900



- 2 RAID controllers per FAStT900
- Host side mini-hubs
  - 2 min-hubs per RAID controller, 4 total
    - 2 Gbit/s (256 MB/s) per mini-hub wire speed
    - 150 MB/s sustained per mini-hub (realistic rate)
  - 2 ports per mini-hub, 8 total, but
    - the BW per mini-hub remains constant
- Drive side mini-hubs
  - same part, wire speeds as drive side mini-hubs
  - forms FC-AL loop with disk drawers
    - therefore use only 1 GBIC per mini-hub
- Aggregate controller BW: 390 MB/s
  - in actual operation with GPFS, sustained peak rates can be as high as 370 MB/s for reads and 360 MB/s for writes, depending on the configuration, OS and application
- Supported expansion units (*i.e.*, "disk drawers")
  - EXP700 (FC disk)
  - EXP100 (SATA)
- Maximum disk capacity (224 disks... 16 drawers)
- Notes
  - GPFS support for FAStT900 on AIX/pSeries or Linux/xSeries
  - AIX/pSeries systems require connectivity via a FC switch

Disk Controller
FAStT600

FAStT600



FAStT600 comes in 2 varieties

- Turbo
  - 1 GB RAM per RAID controller
  - SAN Share License
  - "good" IOP performance
    - (45% of FAStT900 IOP rate)
- Standard
  - 256 MB RAM per RAID controller
  - "good" streaming performance
    - (25% of FAStT900 IOP rate)

Rules of thumb

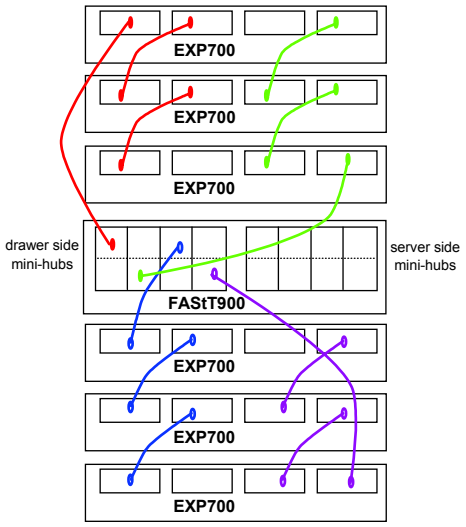
- 1 FAStT600 Turbo = 0.5 FAStT900

- 2 RAID controllers per FAStT600
- Host side ports
  - 2 ports per RAID controller, 4 total
    - 2 Gbit/s (256 MB/s) per port pair
    - 150 MB/s sustained per port pair (realistic rate)
  - a single GBIC can harvest the full BW of a port pair; using 2 GBICs in a port pair will not increase BW
- Drive side ports
  - single port per RAID controller
  - forms FC-AL loop with disk drawers
- Aggregate controller BW
  - in actual operation with GPFS, sustained peak rates can be as high as 320 MB/s for reads and 304 MB/s for writes, depending on the configuration, OS and application
    - using "hacked" version of GPFS with LTG = 1MB
  - more typical rates will be <= 200 MB/s
- Up to 14 disks come in a FAStT600 exclosure
- Supported expansion units (*i.e.*, "disk drawers")
  - EXP700 (FC disk)
  - EXP100 (SATA)
- Maximum disk capacity
  - Turbo - 112 drives
  - Standard - 56 drives

Disk Drawer with Controller

EXP700 and FAStT900 with Typical Cabling

FAStT900 with EXP700
Cascaded Cabling

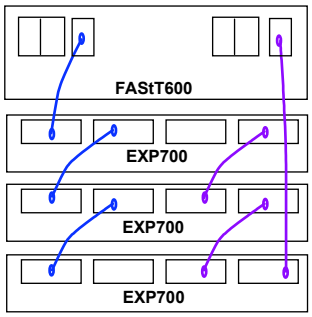


- FC-AL configuration with cascaded cabling
  - each cable has a bi-directional pair of fibers
  - use only 1 adapter per drawer side mini-hub
  - loop is closed when last drawer is not cascaded
    - do **not** connect cable from last drawer in cascade to the open slot in the controller mini-hub
  - physical max per FAStT900: 224 disks
    - 16 EXP700 drawers
  - 2 loops per RAID controller, 4 total
- Number of adapter slots and adapter rates:
  - EXP700: 4 @ 2 Gb/s (sustained ~150 MB/s)
- FC disk drive alternatives:
  - "dual-loop, hot-swappable, slim-profile"
  - 10 Krpm: 36 GB, 73 GB, 146 GB
  - 15 Krpm: 18 GB, 36 GB, 73 GB.
- Benchmark study: FAStT900/EXP700

Disk Drawer with Controller

EXP700 and FAStT600 with Typical Cabling

FAStT600 with EXP700
Cascaded Cabling



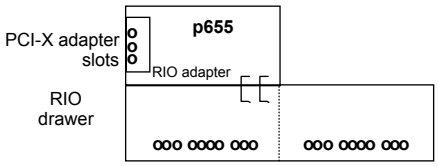
- FC-AL configuration with cascaded cabling
  - each cable has a bi-directional pair of fibers
  - use only 1 adapter per drawer side mini-hub
  - loop is closed when last drawer is not cascaded
    - do **not** connect cable from last drawer in cascade to the open slot in the controller mini-hub
  - physical max per FAStT600: 224 disks
    - 16 EXP700 drawers
  - 2 loops per RAID controller, 4 total
- Number of adapter slots and adapter rates:
  - EXP700: 4 @ 2 Gb/s (sustained ~150 MB/s)
- FC disk drive alternatives:
  - "dual-loop, hot-swappable, slim-profile"
  - 10 Krpm: 36 GB, 73 GB, 146 GB
  - 15 Krpm: 18 GB, 36 GB, 73 GB.
- Benchmark study: FAStT900/EXP700
  - access "raw device"
    - dd -if /dev/zero -of /dev/rhdisk<n> ...
  - 8+P RAID 5 Group using 15 Krpm drives:
    - read: ~= 350 MB/s
    - write: ~=280 MB/s
  - rates much lower when aggregating under FS

Miscellaneous FAStT Comments

- **Notes**
  - GPFS support for FAStT900 on AIX/pSeries or Linux/xSeries/pSeries
  - AIX/pSeries system requires connectivity via FC switch
- **Legacy Systems**
  - FAStT700
    - 2 Gbit/s FC adapters and improved
    - improved cache performance
  - FAStT500
    - 1 Gbit/s FC adapters
  - FAStT200
    - 1 Gbit/s
    - low cost of entry
  - EXP500
    - 1 Gbit/s FC-AL adapters
  - EXP200
    - 1 Gbit/s FC-AL adapters
    - low cost of entry

File Server

p655 Server Node and RIO Drawer



p655 vs p690

The are many significant differences between a p655 and a p690, but both use the same

- POWER4 chips and MCM technology
- RIO drawer.

By contrast, the p690 can have

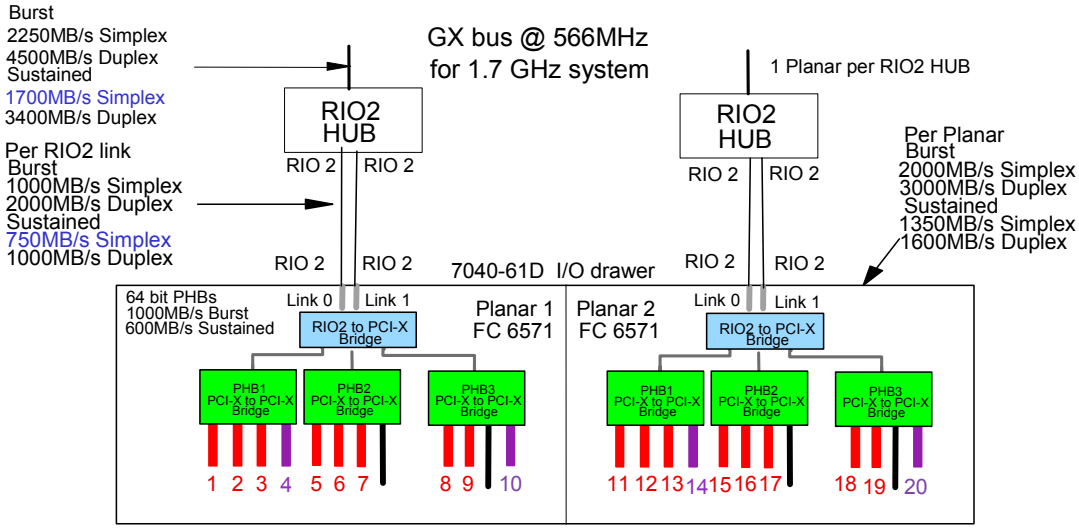
- 32 CPUs
- at least 8 LPARs
- at most 12 RIO drawers
- 8 HPS links per node

For purposes of this presentation, the p690 LPARs act as independent n-way POWER4 nodes.

- p655 node
  - 4-way or 8-way
    - 4-way... single core MCM at 1.7 GHz
    - 8-way... double core MCM at 1.5 GHz
  - 1 to 4 memory cards per node up to 32 GB
    - 4 cards = 15 GB/s for DAXPY, STREAM, DDOT
    - 2 cards = 10 GB/s for DAXPY, STREAM, 15 GB/s for DDOT
    - 1 card = 5 GB/s for DAXPY, STREAM, 7.5 GB/s for DDOT
  - 1 or 2 LPARs
- 3 PCI-X slots per node
- RIO drawer
  - 7040-61D... "Bonnie & Clyde-X"
  - 20 hot-plug/blind-swap PCI-X slots per drawer
    - 2 planars with 10 slots each
    - plugging rules (see manual SA38-0538-15)
  - 1350 MB/s simplex operation per planar
  - 1650 MB/s duplex operation per planar
- SP Switch2 PCI-X adapter (230 MB/s)
  - single adapter port, single plane
  - up to 2 switch adapters per node (1 per LPAR)
- HPS (High Performance Switch... "federation")
  - 2 links per node (8 CPUs)
  - 1.2 GB/s for IP to 1.5 GB/s for user space
    - user space comes "free" with MPI
  - attaches to GX bus on the CEC

Close-up

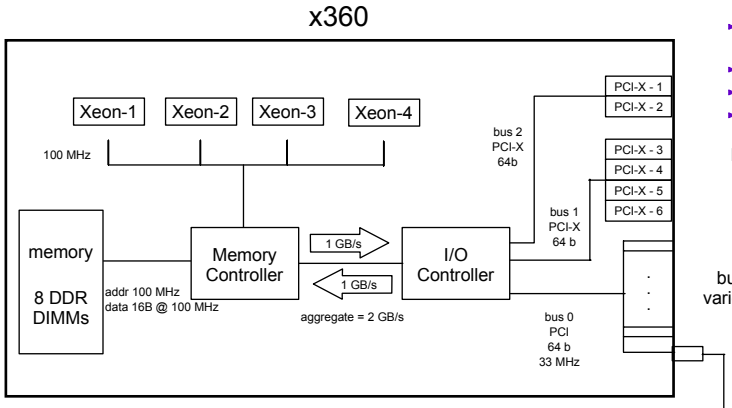
7040-61D RIO Drawer ("Bonnie & Clyde")



COMMENT: Used with the p655 and p690

File Server

x360 Server Node and RIO Drawer



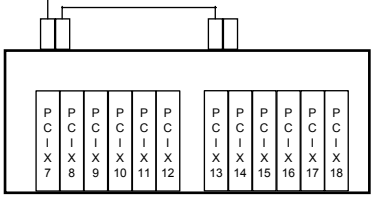
bus 2 adapter options
- 1 PCI-X @ 133 MHz
- 2 PCI-X @ 100 MHz
- 2 PCI-X @ 66 MHz
- 2 PCI @ 66 MHz
- 2 PCI @ 33 MHz

bus 1 adapter options
- 4 PCI-X @ 66 MHz
- 2 PCI @ 66 MHz
- 4 PCI @ 33 MHz

bus 0 connects to various ports/devices

Comment:
The x360's I/O design is optimized for file system access (common in scientific and engineering applications) as opposed to database access (common in commercial applications).

RXE-100

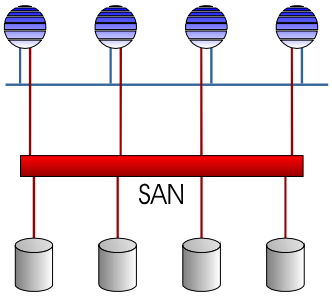


NOTES:
- both units are 3u
- bus 0 = 260 MB/s
- bus 1 = 520 MB/s
- bus 2 = 1040 MB/s

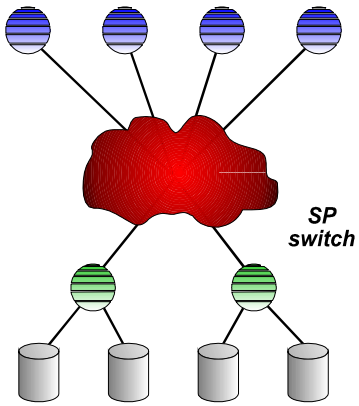
The next slides look at a variety of abstract configurations. These are intended to illustrate GPFS configuration alternatives... they are not an exhaustive list of the possibilities.

Actual configurations with benchmark results will be presented later.

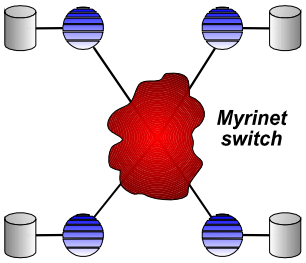
Example GPFS Configurations Quick Overview



Storage area network

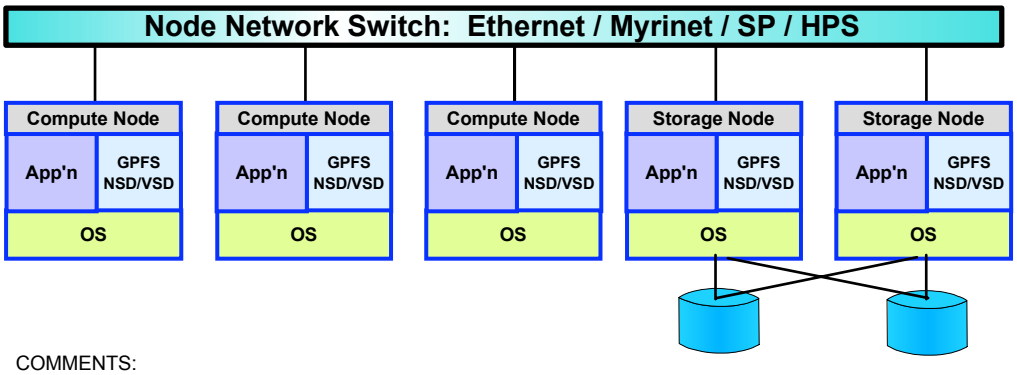


Cluster with dedicated storage nodes



Symmetric cluster

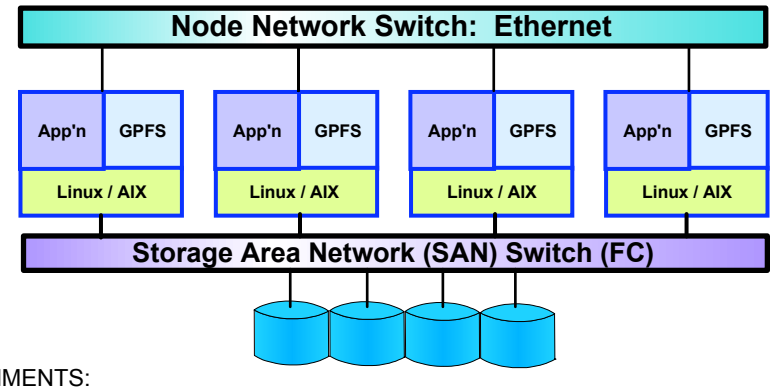
Network Shared Disk (NSD) Server Model



COMMENTS:

1. Assume that OS is identical for each node in the cluster (i.e., AIX xor Linux).
2. If the OS is AIX
   - if the node network is the SP or HPS switch
     - use rpd cluster type with VSD (n.b., do not use NSD)
     - use lc cluster type with NSD "on top of" VSD
   - if the node network is Ethernet, use lc cluster type with NSD
3. If the OS is Linux, always use NSD (n.b., VSD is an AIX product only).
4. NSD is a component of GPFS; VSD is a seperate SW product.
5. Disk data (both user and meta data) as well as GPFS overhead traffic (*e.g.*, tokens) traverse the node network switch.
6. For synchronous applications (implicit or explicit), a job's tasks should *not* run on both the storage and compute nodes.
7. The disks can be mounted over a FC switch, but GPFS will not operate in SAN mode.
8. A common bottle neck in this configuration is the adapters for the node network switch of the storage nodes.

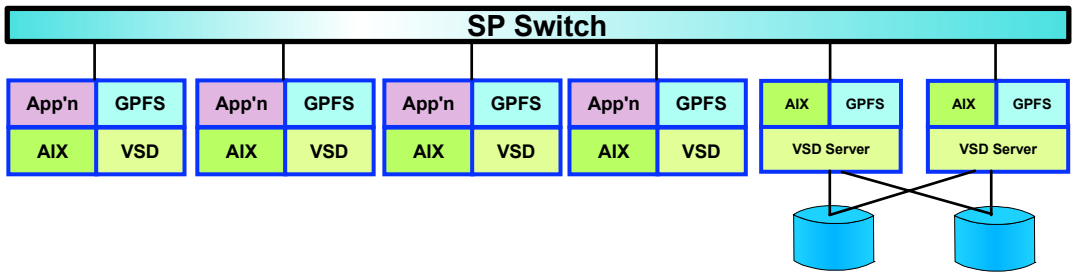
Storage Area Network (SAN) Model



COMMENTS:

1. Assume that OS is identical for each node in the cluster (i.e., AIX xor Linux).
2. If the OS is AIX
   - use rpd cluster type with LVs
   - use lc cluster type with NSD "on top of" LV
3. If the OS is Linux, use lc cluster type with NSD
4. All disks are mounted on all nodes
   - if the cluster type is lc, specify primary and secondary NSD servers
5. Disk data (both user and meta data) traverses the SAN switch. GPFS overhead traffic (*e.g.*, tokens) traverses the node network switch.
6. GbE node network is sufficient for GPFS, though other higher speed IP networks will work
7. Common performance bottle necks in this configuration include
   - insufficient number of FC adapters
   - insufficient number of disk controllers

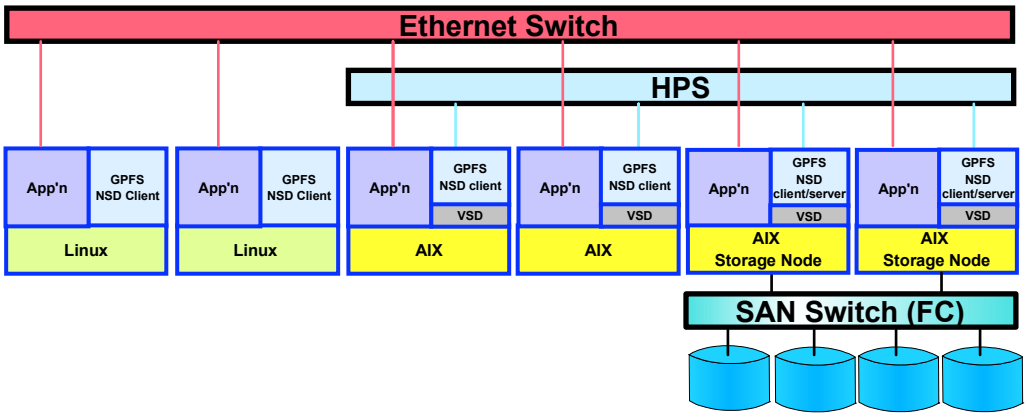
Virtual Shared Disk (VSD) Server Model



COMMENTS:

1. Disk data (both user and meta data) as well as GPFS overhead traffic (*e.g.*, tokens) traverse the node network switch.
2. Use sp cluster type with VSD
3. Generally configured with dedicated VSD servers (*i.e.*, they do not run applications); but some customers using SSA disk configure all nodes to both be a VSD server and compute client.
4. For synchronous applications (implicit or explicit), a job's tasks should *not* run on both the storage and compute nodes.
5. A common bottle neck in this configuration is the adapters for the node network switch of the storage nodes.

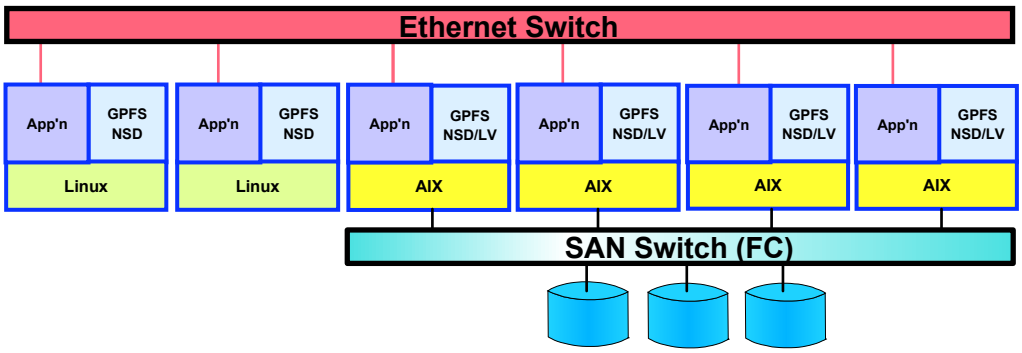
Mixed AIX/Linux Cluster #1



COMMENTS:

1. The cluster type must be lc
2. For AIX nodes, NSD must be built on top of VSD to enable using the HPS switch.
   - For AIX nodes, Disk data traverses the HPS Switch; GPFS overhead traffic traveses Ethernet switch.
   - For Linux nodes, Disk data and GPFS overhead traffic traverses the Ethernet switch.
3. While the disks are mounted over a SAN, GPFS is *not* operating in SAN mode (n.b., disks are mounted on designated nodes with disk data traversing the switches).
4. If the application is implicitly/explicity synchronous, storage nodes should be NSD servers only.

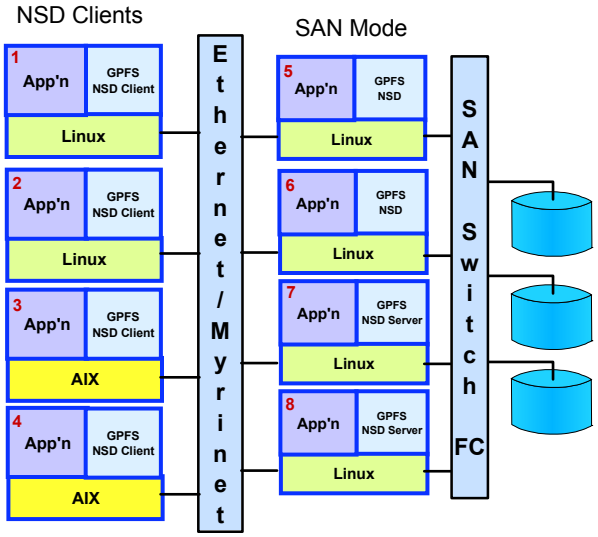
Mixed AIX/Linux Cluster #2



COMMENTS:

1. The cluster type must be lc
2. GPFS operates in SAN mode on the AIX nodes
3. If the OS is AIX
   - NSDs built on top of LVs (can also use hdisks or VSDs)
     - if Linux nodes are attached to the SAN, the NSDs *must* be built on top of hdisks
   - Disk data traverses the SAN; GPFS overhead traffic traveses Ethernet switch
4. If the OS is Linux
   - Disk data and GPFS overhead traffic traverses the Ethernet switch

GPFS SAN and NSD Client/Server



COMMENTS:

1. Must use lc cluster type
2. Nodes 1-4 are NSD client nodes. All disk data and GPFS overhead data traverse the node network switch to the NSD servers.
3. Nodes 5-8 are part of the GPFS SAN. Disk data traverses the SAN switch while GPFS overhead data traverses the node network.
4. Only nodes 7-8 are the NSD server nodes. The NSD client nodes access the GPFS file system only through the NSD server nodes via the node network.

GPFS Configuration Alternatives

All Supported Combinations

Cluster Type	OS	Clustering Technology	Disk Subsystem	Disk Interconnect	Network Interconnect
lc	AIX 5.2	RSCT Peer Domain (RPD)	NSD "on top of" hdisk, LV, VSD	NSD server xor SAN attached	IP Network • 100Mb/sec at a minimum • 1Gb/sec or better is suggested
	Linux/xSeries Linux/pSeries		NSD	NSD server xor SAN attached	
	Mixed AIX/Linux		**appropriate OS/NSD combination**	Combination SAN/NSD	
rpd	AIX 5.1 or 5.2	RSCT Peer Domain (RPD)	VSD	VSD servers	IP Network • 100Mb/sec at a minimum • 1Gb/sec or better is suggested
			LV	SAN attached	
hacmp	AIX 5.1 or 5.2	HACMP	LV	SAN attached	IP Network • 100Mb/sec at a minimum • 1Gb/sec or better is suggested
sp	AIX 5.1 or 5.2	PSSP	VSD	VSD servers	SP Switch SP Switch 2 (colony) *n.b.*, PSSP does not support HPS (federation)

COMMENTS

1. Specify the cluster type (*e.g.*, lc, rpd, hacmp, sp) using the **mmcrcluster** command
2. Use the appropriate "mm create" command to configure the disk subsystem (*i.e.*, **mmcrlv**, **mmcrnsd**, **mmcrvsd**)
3. Valid disk subsystems include NSD (Network Shared Disk), VSD (Virtual Shared Disk), LV (Logical Volume); hdisk<n> is the AIX physical device created after doing cfgmgr (*i.e.*, lsdev -Cc disk)
4. "SAN attached" means all disks are physically attached to all nodes via the SAN
5. Disks can be attached via a SAN to NSD or VSD servers
6. When using NSD or VSD servers, file records are accessed via the network interconnect
7. Many of these configurations exist to support legacy systems. Furture releases of GPFS will simply this.

Which Configuration is Best?

It is application/customer dependent!

Each configuration has its limitations and its strong points. And each one is commonly used.

The following pages illustrate specific GPFS configurations.

The next slides look at a sample of actual configurations with benchmark results. These are intended to be examples to illustrate and suggest GPFS configuration alternatives. They illustrate how systems *have been* configured, not necessarily how they *should be* configured.

Disclaimers:

- These slides are not intended to be "wiring diagrams"; rather, they are to illustrate basic concepts when integrating various components into an overall solution.
- "Feeds and speeds" are based on conservative upper bound estimates under ideal benchmarking conditions. Actual performance may vary "according to actual driving conditions".

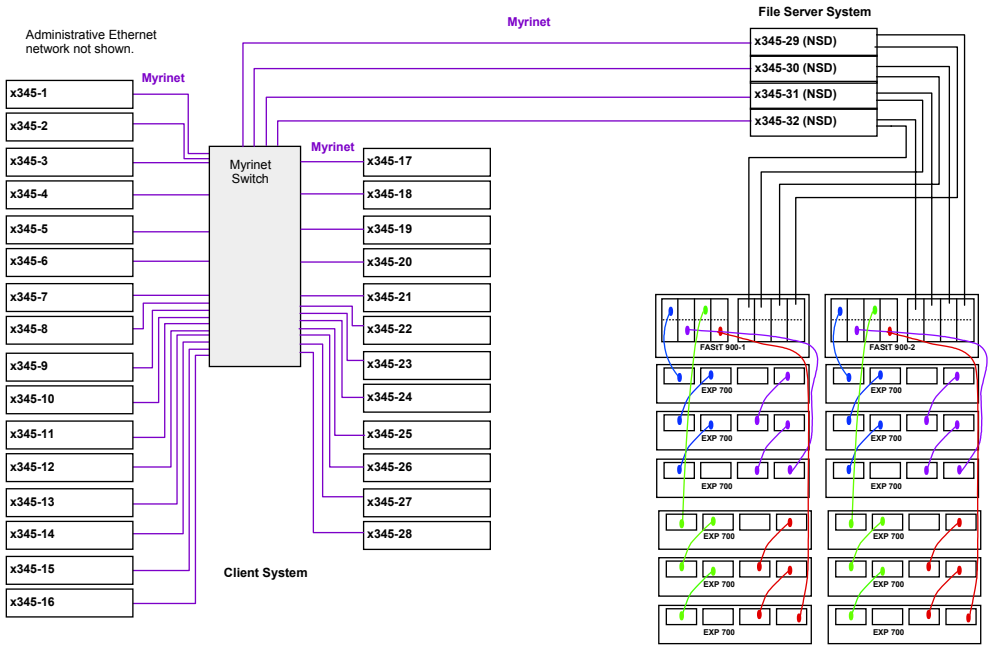
I/O Subsystem Design Goal

Ideally, an I/O subsystem should be balanced. For example, there is no point in making one component of an I/O subsystem fast while another is slow. Moreover, overtaxing some components of the I/O subsystem may disproportionately degrade performance (*e.g.*, HBAs).

However, this goal can not always be perfectly achieved. For example, a common imbalance is when data volumes are more important than bandwidth; then the aggregate bandwidth based on the number of spinning disks and disk controllers may exceed the aggregate bandwidth of the HBAs.

"Performance is inversely proportional to capacity."
-- Todd Virnosch

Configuration #1
Linux, Cluster Type = lc, NSD mode



Configuration #1
Linux, Cluster Type = lc, NSD mode

Client Nodes
- 28 nodes
  - x335 or x345 (dual CPU, 2.8 GHz, 4 GB RAM)
  - Linux (Redhat)
  - GPFS
    - pagepool = 128 MB
    - blocksize = 256 KB
  - PVM
- GPFS network
  - 28 Myrinet adapters (1 adapter per node)

Server Nodes
- 4 nodes
  - x345 (2-way) or x360 (4-way)
  - Linux (Redhat)
  - GPFS
  - PVM
- GPFS Network
  - 4 Myrinet adapters (1 adapter per node)

Storage Infrastructure
- FC adapters (2 Gbit)
- 2 FAStT900 disk controllers
- 12 EXP700 disk drawers (fully populated with 36 GB or larger drives)
  - full BW could have been acheived with 3 drawers
- 8 HBA FC adapters (2 per server node)
- 8 server side FC adapters (4 per FAStT900)
- Key FAStT900 parameters
  - RAID level: 4+P
  - segment size: 16 KB
  - cache block size: 16 KB
  - read caching: disabled
  - write caching: disabled
  - read-ahead multiplier = 0

Application
- Access pattern: large record sequential
- Number of tasks: see chart

	CPU usage <= 93%	Natural Aggregate		Harmonic Aggregate		Harmonic Mean	
Number of nodes	Number of tasks per node	Write Rate MB/s	Read Rate MB/s	Write Rate MB/s	Read Rate MB/s	Write	Read
1	1	159.75	184.63				
2	1	273.85	324.94	288.82	341.13	144.41	170.56
4	1	320.76	394.52	420.34	483.16	105.09	120.79
8	1	382.95	536.70	385.03	559.73	48.13	69.97
16	1	388.43	516.92	396.57	584.77	24.79	36.55
28	1	380.49	560.52	385.31	563.83	13.76	20.14

Alternatives on Configuration #1

Linux, Cluster Type = lc, NSD mode

COMMENT: The Myrinet switch is relatively expensive, so unless its bandwidth/latency features are required, customers generally use GbE or 100 MbE instead. This is especially true for large clusters (*e.g.*, 1000's of nodes)

Benchmark results Using GbE

	CPU usage <= 86%	Natural Aggregate		Harmonic Aggregate		Harmonic Mean	
Number of client nodes	Number of tasks per client node	Write Rate MB/s	Read Rate MB/s	Write Rate MB/s	Read Rate MB/s	Write	Read
1	1	92.26	111.27				
2	1	178.02	209.94	178.03	220.60	89.02	110.03
4	1	292.54	320.46	296.06	337.92	74.02	84.48
8	1	383.69	338.26	383.86	360.21	47.98	45.03
16	1	315.12	347.32	335.75	357.07	20.98	22.32

Benchmark results using 100 MbE to clients and GbE to servers

		Natural Aggregate		Harmonic Aggregate		Harmonic Mean	
Number of client nodes	Number of tasks per client node	Write Rate MB/s	Read Rate MB/s	Write Rate MB/s	Read Rate MB/s	Write	Read
1	1	0.141	7.87				
2	1	0.209	15.44	0.209	15.49	0.105	7.74
4	1	0.296	30.59	0.307	30.74	0.077	7.68
8	1	0.357	56.27	0.393	60.22	0.049	7.53
16	1	0.429	111.89	0.498	116.41	0.031	7.28

Configuration #2

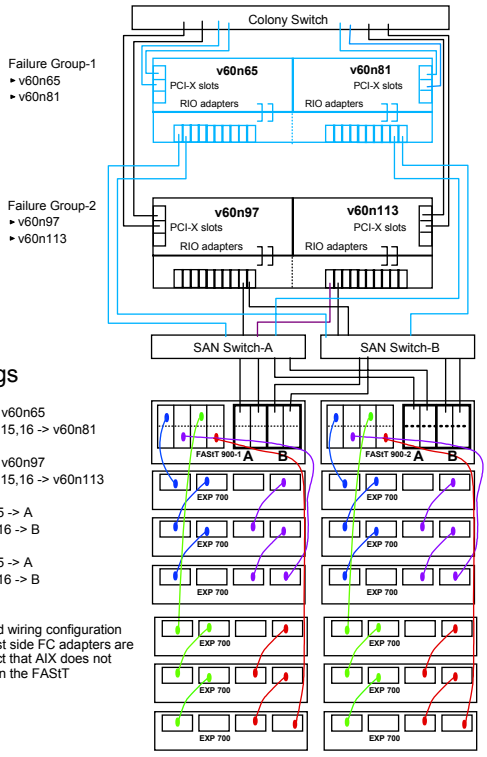
AIX, Cluster Type = rpd, VSD mode

System Components

- 4 nodes
  - 8-way p655
  - AIX 5.1
  - GPFS 2.1
- Storage Area Network
  - 2 Gbit FC adapters
  - 8 HBA FC adapters
  - 8 server side FC adapters
  - 16 FC adapters for switch
- 2 FAStT900 disk controllers
  - 6 EXP700 drawers per controller
  - 14 pdisks per drawer
  - 73 GB/pdisk @ 10 Krpm
- 2 PCI-X colony adapters per node
- 2 RIO adapters per node
- Key GPFS parameters
  - pagepool: 512 MB
  - Blocksize: 256 KB
- Key FAStT900 parameters
  - RAID level: 4+P
  - segment size: 32 KB
  - cache block size: 16 KB
  - read caching: enabled
  - write caching: enabled
  - read-ahead multiplier = 0

Peak sustained performance

- read rate: 746 MB/s
- write rate: 724 MB/s
- Access pattern - large record seq
- 16 Tasks, 4 tasks/node



LUN Mappings

- Failure Group-1
  - 1,2,3,4,5,6,7,8 -> v60n65
  - 9,10,11,12,13,14,15,16 -> v60n81
- Failure Group-2
  - 1,2,3,4,5,6,7,8 -> v60n97
  - 9,10,11,12,13,14,15,16 -> v60n113
- FAStT900-1
  - 1,3,5,7,9,11,13,15 -> A
  - 2,4,6,8,10,12,14,16 -> B
- FAStT900-2
  - 1,3,5,7,9,11,13,15 -> A
  - 2,4,6,8,10,12,14,16 -> B

COMMENT:
The LUN mappings and wiring configuration guarantee that all 4 host side FC adapters are active in spite of the fact that AIX does not support multi-pathing on the FAStT controllers.

Alternative on Configuration #2

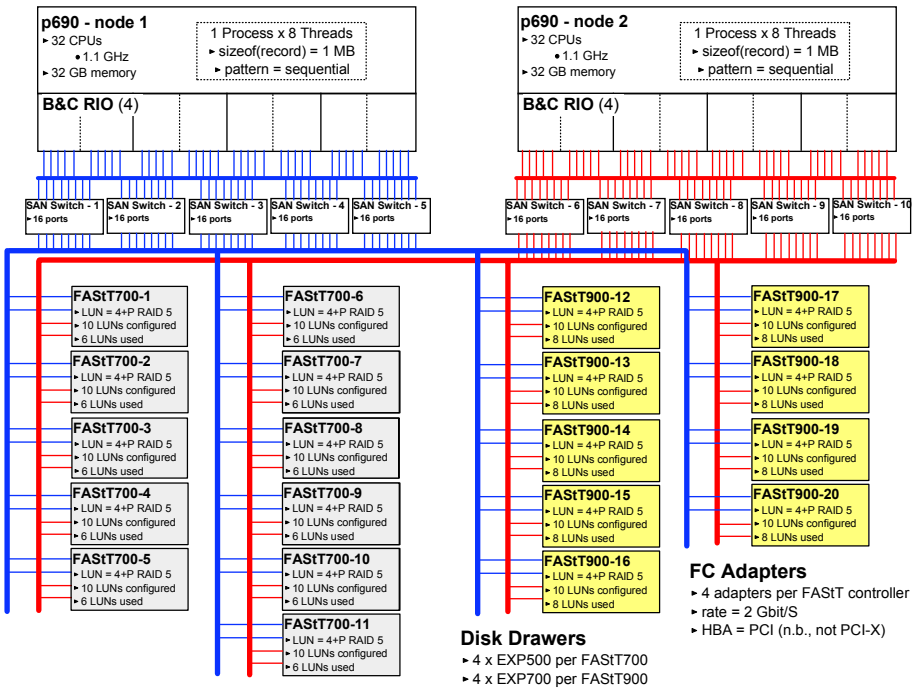
AIX, Cluster Type = rpd, VSD mode

The following is a customer system now in production.

- Compute Nodes (92)
  - 8-way p655+
  - AIX 5.2, GPFS 2.1
- Switch
  - Federation
- Storage Nodes (4)
  - servers (4)
    - 8-way p655+
    - 4 PCI-X FC HBA per node
  - SAN switch
    - Brocade 2109 F32 (2)
  - disk controllers (8)
    - FAStT900
    - 4 Host port adapters per FAStT900
  - disk drawers (32)
    - EXP700 (4 per FAStT900)
    - 14 pdisks per EXP700 (73 GB, 10 Krpm)
  - observation
    - Peak performance on this system is *measured* at 1.2 GB/s, but the FAStT900's are capable of delivering 2.8 GB/s. The problem is that this configuration is not balanced. 8 HBAs per server are needed.

Configuration #3

AIX, Cluster Type = rpd, LV mode



Configuration #3
AIX, Cluster Type = rpd, LV mode

Benchmark
- large record sequential (1 MB/record)
- 8 threads per process, 1 process per node
- source code: /usr/lpp/mmfs/samples/perf/gpfsperf

p690
- 32 CPUs (1.1 GHz)
- 44 FC HBA per node (only 40 were used)
  - PCI (n.b., not PCI-X)
- AIX 5.1, GPFS 2.1
- Key GPFS parameters
  - SAN configuration
  - blocksize = 1 MB
  - pagepool = 400 MB

Disk
- RAID 5, 4+P
- Drive speed = 10 Krpm
- 12 x FAStT700 (but only 11 were used)
  - 4 EXP500 drawers per FAStT700
    - 10 LUNs configured, 6 LUNs used
- 10 x FAStT900 (but only 9 were used)
  - 4 EXP700 drawers per FAStT900
    - 10 LUNs configured, 8 LUNs used
- 4 host port FC adapters per FAStT, 2 adapters from each node

FAStT parameters
- segment size = 16 KB
- cache block size = 16 KB
- read and write caching disabled
- read ahead multiplier = 0

Performance Results
- Read (1 node)
  - 2.5 GB/s
- Read (2 nodes)
  - 3.7 GB/s
- Write (1 node)
  - 1.9 GB/s to write/create file
  - 2.3 GB/s to re-write existing file
- Write (2 node)
  - 2.9 GB/s to write/create file

COMMENT: This is configuration is in SAN mode... all LUNs are seen by both nodes.

Alternative on Configuration #3
AIX, Cluster Type = rpd, LV mode

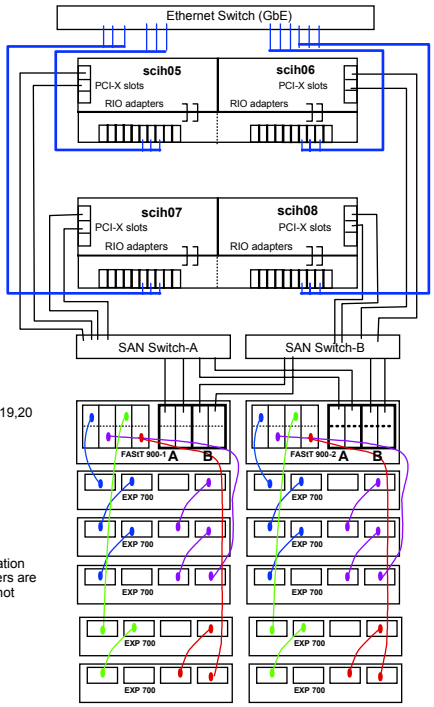
System Components
- 4 nodes
  - 4-way p655
  - AIX 5.2
  - GPFS 2.1
- Storage Area Network
  - 2 Gbit FC adapters
  - 8 HBA FC adapters
  - 8 host port FC adapters
  - 16 FC adapters for switch
- 2 FAStT900 disk controllers
  - 5 EXP700 drawers per controller
  - 14 pdisks per drawer
  - 73 GB/pdisk @ 10 Krpm
  - 13 LUNs configured per FAStT900
  - 8 LUNs used per FAStT900
- Ethernet
  - Jumbo frame
  - 3 GbE adapters/node
  - Bonded
- 2 RIO adapters per node
- Key GPFS parameters
  - pagepool: 256 MB
  - Blocksize: 256 KB
- Key FAStT900 parameters
  - RAID level: 4+ P
  - segment size: 32 KB
  - cache block size: 16 KB
  - read caching: enabled
  - write caching: enabled
  - read-ahead multiplier = 0

Peak sustained performance
- read rate: 719 MB/s
- write rate: 522 MB/s
- Access pattern - large record seq
- 16 Tasks, 4 tasks/node

LUN Mappings
- Mounted LUNs
  0,1,2,3,4,5,6,7,13,14,15,16,17,18,19,20
- FAStT900-1
  0,2,4,6,8,10,12 -> A
  1,3,5,7,9,11,13 -> B
- FAStT900-2
  14,16,18,20,22,24 -> A
  15,17,19,21,23,25 -> B

COMMENT:
The LUN mappings and wiring configuration guarantee that all 4 host side FC adapters are active in spite of the fact that AIX does not support multi-pathing on the FAStT controllers.

Cf. configuration #2.
This configuration is a SAN; all LUNs are seen by all nodes and disk data does ***not*** traverse the ether.



Alternative on Configuration #3

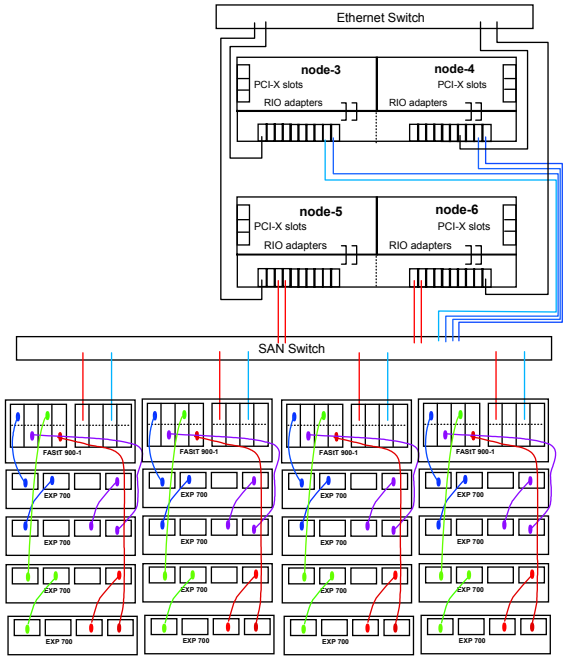
AIX, Cluster Type = rpd, LV mode

System Components

- 4 nodes
  - 8-way p655+
  - AIX 5.2
  - GPFS 2.1
- Storage Area Network
  - 2 Gbit FC adapters
  - 8 HBA FC adapters
  - 8 server side FC adapters
  - 16 FC adapters for switch
- 4 FAStT900 disk controllers
  - 4 EXP700 drawers per controller
  - 14 pdisks per drawer
  - 73 GB/pdisk @ 10 Krpm
  - 11 LUNs per FAStT900 (all are used)
- Assume default Ethernet configuration
- 2 PCI-X colony adapters per node
- 2 RIO adapters per node
- Key GPFS parameters
  - pagepool: 512 MB
  - Blocksize: 256 KB
- Key FAStT900 parameters
  - RAID level: 4+ P
  - segment size: 32 KB
  - cache block size: 16 KB
  - read caching: disabled
  - write caching: disabled
  - read-ahead multiplier = 0

Peak sustained performance

- read rate: 740 MB/s
- write rate: 526 MB/s
- Access pattern - large record sequential
- 16 Tasks, 4 tasks/node

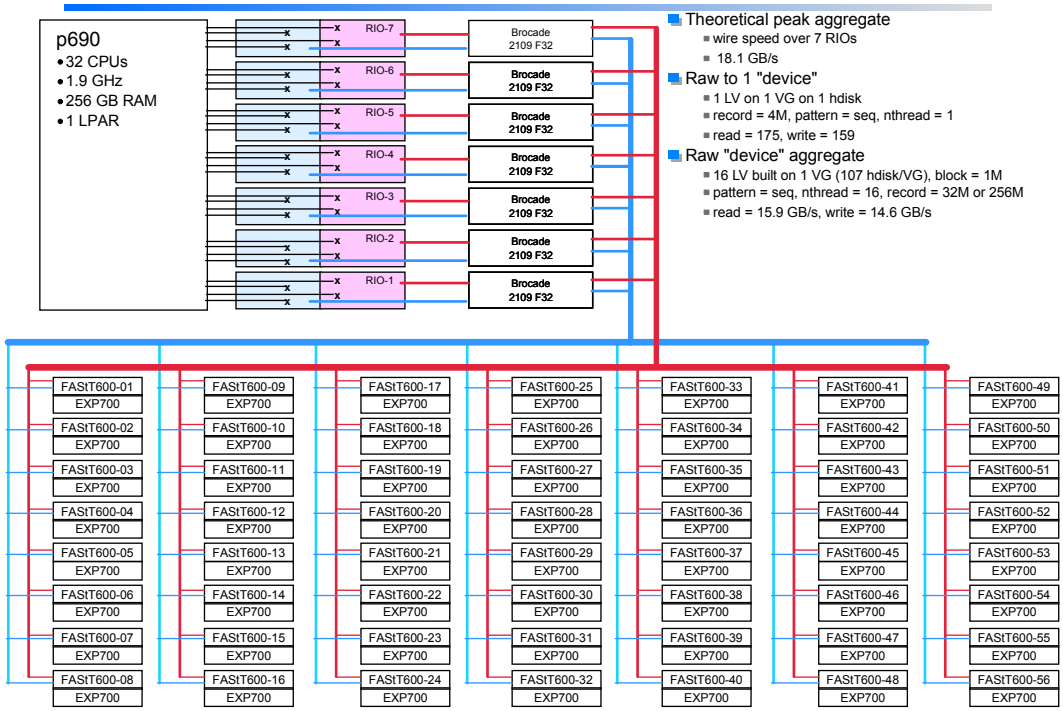


COMMENT: This is an actual customer configuration. Its performance is gated by LUN assignments and the use of only 2 FC adapters per FAStT900. However, by doubling the number of host port adapters per FAStT900 and changing the LUN assignments, the rate per FAStT900 can be doubled (cf. previous slide).

Configuration #4

AIX, Cluster Type = rpd, VSD mode

p690
- 32 CPUs
- 1.9 GHz
- 256 GB RAM
- 1 LPAR

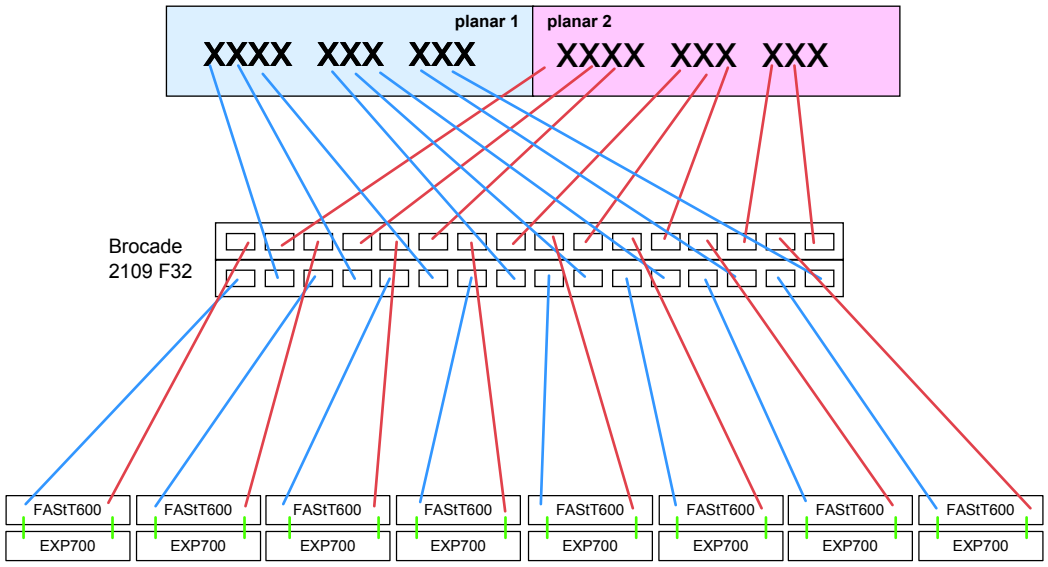


- Theoretical peak aggregate
  - wire speed over 7 RIOs
  - 18.1 GB/s
- Raw to 1 "device"
  - 1 LV on 1 VG on 1 hdisk
  - record = 4M, pattern = seq, nthread = 1
  - read = 175, write = 159
- Raw "device" aggregate
  - 16 LV built on 1 VG (107 hdisk/VG), block = 1M
  - pattern = seq, nthread = 16, record = 32M or 256M
  - read = 15.9 GB/s, write = 14.6 GB/s

For each FAStT600,EXP700 pair there is...
2 GBICs, 28 pdisks, 2 hdisks where each hdisk is an 8+P RAID group
LTG = 1M, segment = 128K, cache block = 16K, read/write cache = disabled

COMMENT:
alternative config: cluster type = lc, LV mode

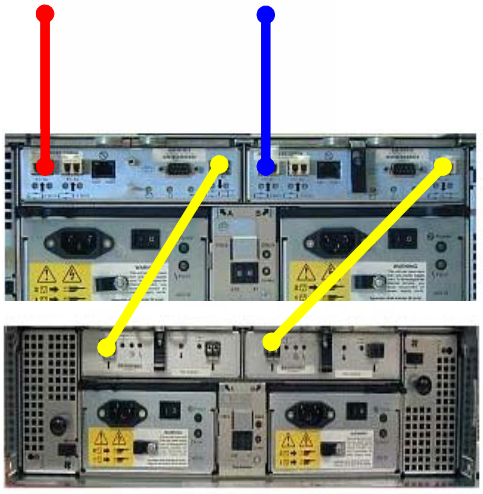
Configuration #4
FAStT600 to Brocade to RIO Drawer



Configuration #4
FAStT600 and EXP700

RIO2 Drawer Planar 1

RIO2 Drawer Planar 2



FAStT600 Turbo Enclosure

(With 14 drives 36 GB, 15K RPM)

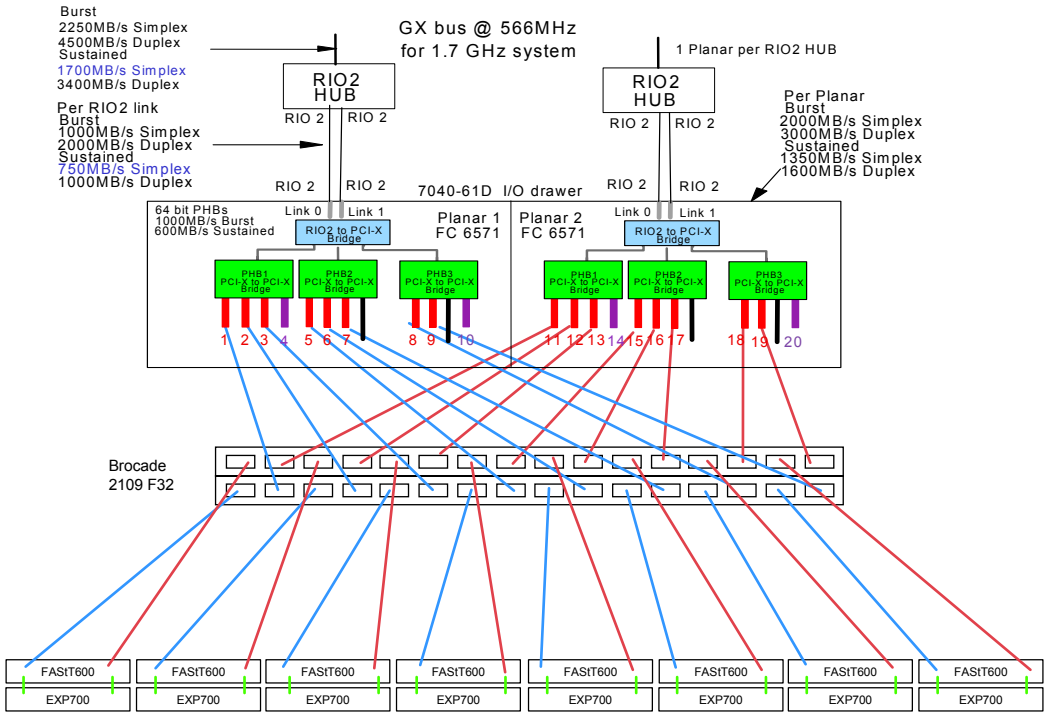
EXP700

(With 14 drives 36 GB, 15K RPM)

NOTE: Using 18 out of 28 disks, configured as 2 RAID groups (8+P)

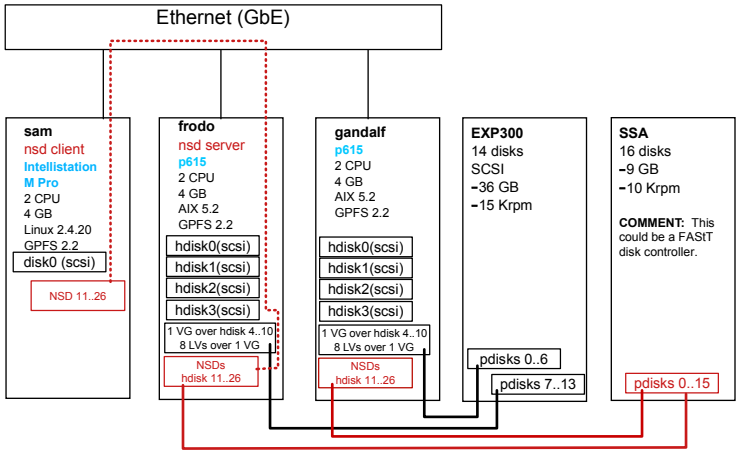
Configuration #4

FAStT600 to Brocade to RIO Drawer



Configuration #5

AIX/Linux, Cluster Type = lc, NSD mode



To build the file system, do the following on gandalf...

1. create rpd domain
2. mmcrcluster
3. mmconfig
4. mmstartup
5. mmcrnsd
   - specify primary, secondary, client, server nodes in disk descriptor file
6. mmcrfs
7. mount /<FS name>

preprpnode
mkrpdomain
startrpdomain

Benchmark Results

- TBD

Let's take a look at some selected sysadm details. Much of this this information is also relevant to system programmers.

This discussion is not intended to be exhaustive; rather it is intended to provide general guidance and examples (i.e., "give me an example and I will figure out how to do it").

The emphasis is on content, not syntax. Nor are all options are explanined. See manuals for further details.

Where to Find Things in GPFS

- **Some useful GPFS directories**
  - /usr/lpp/mmfs
    - /bin... commands (binary and scripts)
      - most GPFS commands begin with "mm"
    - /gpfsdocs... pdf and html versions of basic GPFS documents
    - /include... include files for GPFS specific APIs, etc.
    - /lib... GPFS libraries (*e.g.*, libgpfs.a, libdmapi.a)
    - /samples... sample scripts, benchmark codes, etc.
  - /var/adm/ras
    - error logs
      - files... mmfs.log.<time stamp>.<hostname> (new log every time GPFS restarted)
      - links... mmfs.log.latest, mmfs.log.previous
  - /tmp/mmfs
    - used for GPFS dumps
    - sysadm must create this directory
    - see **mmconfig** and **mmchconfig**
  - same directory structure for both AIX and Linux systems
- **Today's trivia question...**

  Question: What does mmfs stand for?

  Answer: Multi-Media File System... predecessor to GPFS in the research lab

Comments on Selected GPFS Manuals in

- GPFS Documentation
  - *Concepts, Planning and Installation Guide*
    - One of the most helpful manuals on GPFS... it provides an excellent conceptual overview of GPFS.
    - If this were a university class, this manual would be your assigned reading. :->
  - *Administration and Programming Reference*
    - Documents GPFS related administrative procedures and commands as well as an API guide for GPFS extensions to the POSIX API. The command reference is identical to man pages.
  - *Problem Determination Guide*
    - Many times, GPFS error messages in the mmfs.log files have an error number. You can generally find these referenced in this guide with a brief explanation regarding the cause of the message. They will often point to likely earlier error messages helping you to find the cause of the problem as opposed to its symptom.
  - *DMAPI Guide*
  - Documentation available online at...
    - http://publib.boulder.ibm.com/clresctr/windows/public/gpfsbooks.html#aix_rsctpd22wo
  - Available with the GPFS SW distribution in /usr/lpp/mmfs/gpfsdocs
    - note to sysadm's... Be sure to install this directory! Also install the man pages!
- IBM Redbooks and Redpapers
  - http://w3.itso.ibm.com/... do a search on GPFS

GPFS provides a number of commands to list parameter settings, configuration components and other things.

I call these the "mmls" or "mm list" commands.

COMMENT:

By default, nearly all of the mm commands require root authority to execute. However, many sysadm's reset the permissions on mmls commands to allow programmers and others to execute them as they are very useful for the purposes of problem determination and debugging.

Selected "mmls" Commands

- **mmlsfs <device name>**
  - Without specifying any options, it lists all file system attributes

```
gandalf(ROOT):/usr/lpp/mmfs/bin> mmlsfs gpfs1    <-- you could also type "mmlsfs /dev/gpfs1"
flag value          description
---- -------------- -----------------------------------------------------
 -s  roundRobin     Stripe method
 -f  8192           Minimum fragment size in bytes
 -i  512            Inode size in bytes
 -I  16384          Indirect block size in bytes
 -m  1              Default number of metadata replicas
 -M  1              Maximum number of metadata replicas
 -r  1              Default number of data replicas
 -R  1              Maximum number of data replicas
 -a  1048576        Estimated average file size
 -n  32             Estimated number of nodes that will mount file system
 -B  262144         Block size
 -Q  none           Quotas enforced
 -F  139264         Maximum number of inodes
 -V  4              File system version. Highest supported version: 4
 -z  no             Is DMAPI enabled?
 -d  gpfslv0;gpfslv1;gpfslv2;gpfslv3;gpfslv4;gpfslv5;gpfslv6;gpfslv7;gpfslv8;gpf
slv9;gpfslv10;gpfslv11;gpfslv12;gpfslv13;gpfslv14;gpfslv15  Disks in file system

 -A  no             Automatic mount option
 -C  set1           GPFS nodeset identifier
 -E  no             Exact mtime default mount option
 -S  no             Suppress atime default mount option
```

Selected "mmls" Commands

- **mmlsconfig**
  - Without specifying any options, it lists all current nodeset configuration info

```
node42(ROOT):/usr/lpp/mmfs/bin> mmlsconfig
Configuration data for nodeset set1:
------------------------------------
pagepool 256M
dataStructureDump /log/mmfs
multinode yes
autoload yes
useSingleNodeQuorum no
clusterType rpd
group Gpfs.set1
recgroup GpfsRec.set1
useDiskLease yes

File systems in nodeset set1:
-----------------------------
/dev/gpfs1
```

Other Selected "mmls" Commands

- **mmlsattr <file name>**
  - query file attributes
- **mmlscluster**
  - display current configuration information for a GPFS cluster
- **mmlsdisk /dev/<FS name> -d "<disk name list>"**
  - display current configuration and state of the disks in a file system
- **mmlsgpfsdisk -> mmlsnsd**
- **mmlsmgr -C .**
  - display which node is the file system manager for the specified file systems
- **mmlsnode**
  - -a display all nodesets
  - -C . determine the nodeset of the node you are running on
- **mmlsnsd**
  - display current NSD information in the GPFS cluster
- **NOTES**
  - See documentation for other parameters and options.

GPFS provides a number of commands needed to create the file system.

These commands of necessity require root authority to execute.

Selected "mm" Commands

- **mmcrcluster**
  - create a GPFS cluster from an existing RSCT peer domain
  - parameters and options
    - -n specifies a file with a list of node descriptors - NodeName:NodeDesignations
      - NodeName is the IP address or hostname
      - NodeDesignators specifies client or server, quorum or non-quorum
    - -p primary GPFS cluster data server node
    - -s secondary GPFS cluster data server node
    - -t cluster type (lc, rpd, hacmp, sp)
      - lc means "loose cluster", *not* Linux cluster
      - under GPFS 2.2 and higher one must create an RPD domain under "-lc" option

Selected "mm" Commands

- **mmconfig**
  - Create a GPFS nodeset and configure GPFS prior to creating file system
  - Execute this command *before* executing **mmstartup**
    - -C identifier for the new nodeset
    - -a adds all of the nodes in the GPFS cluster to the new nodeset
    - -n node file name
    - -A do we start GPFS daemons when node comes up (default = no)
    - -D path for GPFS dumps (default is /tmp/mmfs ... be sure to create it!)
    - -M number of i-nodes to cache for recently used files that have been closed
    - -p pagepool size (4M to 512M, default = 20M)
    - -U do we start mmfsd if only one node is up in a 2 node nodeset

Selected "mm" Commands

- **mmstartup and mmshutdown**
  - startup and shutdown the **mmfsd** daemons
  - if necessary, mount file system after running **mmstartup**
  - umount file system before running **mmshutdown**
  - properly configured, mmfsd will startup automatically (n.b., no need to run **mmstartup**); if it can not start for some reason, you will see **runmmfs** running and a lot of messages in /var/adm/ras/mmfs.log.latest

Selected "mm" Commands

- **mmcrnsd**
  - Creates and globally names Network Shared Disks for use by GPFS.
  - mmfsd daemon must be running to execute mmcrnsd (i.e., do mmstartup first)
  - mmcrnsd -F disk.lst
    - disk.lst is a "disk descriptor file whose entries are in the format
      - DiskName:PrimaryServer:BackupServer:DiskUsage:FailureGroup:DesiredName
        - **DiskName**: The disk name as it appears in /dev
        - **PrimaryServer**: The name of the primary NSD server node.
        - **BackupServer**: The name of backup NSD server node. Specifying primary and secondary servers is recommended even in SAN mode.
        - **DiskUsage**: dataAndMetadata (default) or dataOnly or metadataOnly
        - **FailureGroup**: GPFS uses this information during data and metadata placement to assure that no two replicas of the same block are written in such a way as to become unavailable due to a single failure. All disks that are attached to the same adapter or NSD server should be placed in the same failure group. Applies only to GPFS in non-SAN mode.
        - **DesiredName**: Specify the name you desire for the NSD to be created. Default format... **gpfs**<integer>**nsd**
    - dsk.lst is modified for use as the input file to the mmcrfs command
- **Notes**
  - mmcrlv and mmcrvsd are very similar
  - mmcrlv or mmcrvsd can be used with mmcrnsd
    - output file from mmcrlv or mmcrvsd forms input file for mmcrnsd
    - mmcrvsd/mmcrnsd pair allows GPFS to use the SP or HPS switch in NSD mode
    - mmcrlv is being maintained for legacy/migration reasons
    - use these commands prior to using mmcrnsd

Selected "mm" Commands
Disk Descriptor Files

```
> cat disk.lst
hdisk56::::
hdisk57::::
hdisk58::::
hdisk59::::
> mmcrnsd -F disk.lst
> cat disk.list
# hdisk56::::
gpfs364nsd:::dataAndMetadata:-1
# hdisk57::::
gpfs365nsd:::dataAndMetadata:-1
# hdisk58::::
gpfs366nsd:::dataAndMetadata:-1
# hdisk59::::
gpfs367nsd:::dataAndMetadata:-1
>
```

NOTES

- This is the results from a single node system in SAN mode with 112 hdisks
- Using disk descriptor defaults.
- The integer in nsd disk names is based on a counter. If you delete and re-create the file system, the counter is not generally re-initialized.

Selected "mm" Commands

- **mmcrfs <mountpoint> <device name> <options>**
  - Create a GPFS file system
  - Execute this command *after* executing **mmconfig**
    - -F specifies a file containing a list of disk descriptors (one per line)
      - this is the output file from **mmcrnsd** xor **mmcrvsd** xor **mmcrlv** depending the choice for the "**-t** " parameter in the **mmcrcluster** command
    - -C nodeset identifier (created by **mmconfig** command)
      - using "-C ." implies the file system belongs to the nodeset from which the mmcrfs command was issued
    - -A do we mount file system when starting mmfsd (default = yes)
    - -B block size (16K, 64K, 256K (default), 512K, 1024K)
      - If you choose a block size larger than 256 KB, you must run mmchconfig to change the value of maxblocksize to a value at least as large as BlockSize.
    - -E specifies whether or not to report exact mtime values
    - -m default number of copies (1 or 2) of i-nodes and indirect blocks for a file
    - -M default max number of copies of inodes, directories, indirect blocks for a file
    - -n estimated number of nodes that will mount the file system
    - -N max number of files in the file system (default = sizeof(file system)/1M
    - -Q activate quotas when the file system is mounted (default = NO)
    - -r default number of copies of each data block for a file
    - -R default maximum number of copies of data blocks for a file
    - -S suppress the periodic updating of the value of atime
    - -v verify that specified disks do not belong to an existing file system
    - -z enable or disable DMAPI on the file system (default = no)
- **Typical example**
  - mmcrfs /fs fs -F disk.lst -A yes -B 1024k -v no

GPFS provides a number of commands to change configuration and file system parameters after being initially set.

I call these the "mmch" or "mm change" commands.

There are some GPFS parameters which are initially set only by default; the only way to modify their value is using the appropriate mmch command.

n.b., There are restrictions regarding changes that can be made to many of these parameters; be sure to consult the *Concepts, Planning and Installation Guide* for tables outlining what parameters can be changed and under which conditions they can be changed. See the *Administration and Programming Reference* manual for further paramter details.

Selected "mmch" Commands

- **mmchconfig**
  - change GPFS configuration attributes originally set (explicitly or implicitly) by mmconfig
    - relative to **mmconfig**, parameter IDs are different
  - parameters and options
    - -C nodeset ID (as specified by **mmconfig** command)
      - using "-C ." changes the configuration parameters for the nodeset from which the mmchconfig command was issued
    - list of node names (default is all nodes in the nodeset)
      - can not be used for all options
    - autoload (same as -a)
    - dataStructureDump (same as -D)
    - designation (can a node be selected as a client, manger, quorum, or nonquorum)
    - maxblocksize
    - maxMBpS (data rate estimate (MB/s) on how much data can be transferred in or out of 1 node)
      - The value is used in calculating the amount of IO that can be done to effectively prefetch data for readers and write-behind data from writers. By lowering this value, you can artificially limit how much IO one node can put on all of the disk servers. This is useful in environments in which a large number of nodes can overrun a few virtual shared disk servers.
      - The default is 150 MB/s which can severally limity performance on HPS ("federation") based systems.
    - maxFilesToCache (same as -M)
    - maxStatCache (specifies number of i-nodes to keep in statcache)
    - pagepool (same as -p)
    - singleNodeQuorum (same as -U)
    - following options apply only to dataStructureDump, maxblocksize, pagepool
      - -i changes are immediate and permanent
      - -I changes are immediate, but do not persist after **mmfsd** daemon is restarted

Selected "mmch" Commands

- **mmchfs <device name> <options>**
  - Change an existing GPFS file system
    - The file system must be unmounted on all nodes prior to issuing the command.
  - Options
    - -C nodeset identifier (created by **mmconfig** command)... can use "-C ."
    - -A do we automatically mount file system when starting mmfsd (default = yes)
    - -F points to a file containing a list of disk descriptors (one per line)
      - this is the output file from the mmcrnsd command for example
    - -m default number of copies (1 or 2) of i-nodes and indirect blocks for a file
    - -Q activate quotas when the file system is mounted (default = NO)
    - -r default number of copies of each data block for a file
    - -S suppress the periodic updating of the value of atime
    - -T change mountpoint of file system starting at the next mount of the file system.
    - -V change the file system format to the latest format supported by this version
  - COMMENT
    - the "-b" option (blocksize) is not included in this list; to change the blocksize requires rebuilding the OS

A concise "cookbook" example to build a GPFS 2.2 file system consistant with the Configuration #5 example (repeated on the following page) is to be added to this presentation very soon.

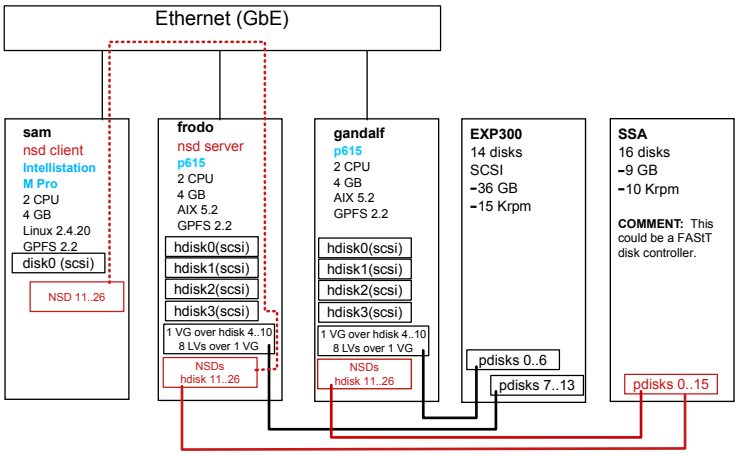
Also refer to the following Redpaper for a complete Linux example.

Linux Cluster with CSM & GPFS

It can be found at

http://w3.itso.ibm.com/ (do a search on GPFS)

Configuration #5
(repeated here in context)



To build the file system, do the following on gandalf...

1. create rpd domain
2. mmcrcluster
3. mmconfig
4. mmstartup
5. mmcrnsd
   - specify primary, secondary, client, server nodes in disk descriptor file
6. mmcrfs
7. mount /<FS name>

preprpnode
mkrpdomain
startrpdomain

Benchmark Results

- TBD

Consider an eclectic assortment of GPFS tips and warnings... some collective wisdom.

These items are based on things I have observed working with customers which have had a significant impact on performance.

While they are simple, common sense things, they are easy to overlook, especially when you are working with legacy codes developed under different conditions and assumptions.

Tips and Warnings

- performance hierarchy
- better way to sort
- random read vs. random write
- use caution mixing NFS and GPFS
- read: write ratio for scientific and technical HPC environments
- pick the correct file server
- avoid gold plating

Programming Tips and Warnings:
Performance Hierarchy

1. large record sequential order
2. large record strided order or small record sequential order
3. large records in random order or small records in strided order
4. issue hints when reading in small records in random order
5. small record random order without hints

NOTE: large records are >= 256KB (e.g., 1MB)
small records are <= 256KB (e.g., 16KB)

Programming Tips and Warnings:
Performance Hierarchy

Example illustrating how code can be rewritten to eliminate small record accesses.

- **Suppose you are sorting directly a set of *small,* randomly or semi-randomly distributed records.**
- **Because records are small, GPFS will perform poorly.**
- **Rewrite code sort as follows:**
  - divide file into N subsets and assign each subset to a node
  - choose the subset size so that it can fit entirely within RAM
  - sort each subset
    - depending on file size, a node may need to sort several subsets
  - merge all of the subsets together
- **This improves performance by**
  - performing all small record references in memory
  - sequentially accessing records on disk in the merge step
- **This is a variation of the mergesort algorithm**

Programming Tips and Warnings:
Random Reads are Preferable over Random Writes

Without hints (8 SP2 nodes, 16KB records)

- read rate = 3 to 4 MB/s
- write rate = less than 1 MB/s

With hints (8 SP2 nodes, 16KB records)

- read rate = 15 MB/s
- write rate = 1 MB/s

Therefore, write sequentially and read random

Sysadm Tips and Warnings:
Use caution mixing GPFS and NFS

GPFS mounted file systems can effectively be accessed via NFS (often as a cost reducing measure for Linux clusters), but…

If the file system is accessed directly by jobs using GPFS and remotely by jobs running NFS, then a slow NFS operation (e.g., NFS write) can hold a GPFS token with GPFS performance being seriously degraded.

Sysadm Tips and Warnings:
Read:Write Ratio for HPC Processing

General rule of thumb in CS textbooks

- read: 90%
- write: 10%

But this generalization is more typical of commercial applications than technical HPC applications.

For example, the ratio for many scientific applications is

- read: 60% to 70%
- write: 40% to 30%

Therefore, plan accordingly.

Sysadm Tips and Warnings:
File Server and Database Server Profiles are Different

Select the proper server!

File cache is main memory. L1, L2, L3, L4 caches are main memory caches designed to compensate for Moore's law. Database and file systems utilize file cache (and hence main memory) very differently.

Database servers are characterized by many small record accesses and cache friendly transactions resulting in good memory temporal locality (i.e., file records can fit in the memory cache and stay there a long time). Servers optimized with large memory caches and having low latency/high bandwidth memory cache access are preferred.

File servers, especially GPFS for HPC workloads, tend to *stream* through large volumes of data resulting large spatial memory locality. Therefore servers with low latency/high bandwidth main memory access are preferred.

e.g., x440 is optimized for database, x360 is optimized for file systems

Sysadm Tips and Warnings:
Avoid "Gold Plating"

- **Rule of thumb**
  - Configure a file system to handle peak performance up to 3 or 4 standard deviations above the mean to avoid "gold plating". (John Watts, IBM)
  - Programmers worried about performance will often over architect a system
- **When sizing the disk I/O subsystem, programmers...**
  - should *not* ask
    - What is the peak load?
  - should ask
    - What are the highest mean loads?
    - What are their standard deviations?
    - What loads are you prepared to pay to meet? Once a month? Once a year?

Conclusion and Observation

- GPFS is a good product (n.b., best of class) with good features, but it is not a "silver bullet"
- Without careful design, I/O can seriously degrade parallel efficiency according to Amdahl's law
- Good I/O performance requires hard work, careful design and the intelligent use of GPFS's performance features
- I/O is not the entire picture; improving I/O performance will uncover other bottle necks