

---

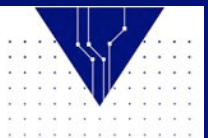
# Large Scale Parallel I/O and Checkpointing

**John Urbanic**

**Nathan Stone**

Pittsburgh Supercomputing Center

May 3 & 4, 2004





---

# Outline

- Motivation
- Solutions
  - Code Level
  - Parallel Filesystems
  - IO API's





# Motivation

Many best-in-class codes spend significant amounts of time doing file I/O. By significant I mean upwards of 20% and often approaching 40% of total run time. These are mainstream applications running on dedicated parallel computing platforms.



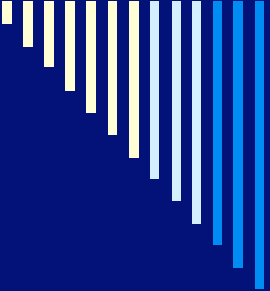


# Terminology

A few terms will be useful here:

- Start/Restart File
- Checkpoint File
- Visualization File



- 
- Start/Restart File(s): The file(s) used by the application to start or restart a run. May be about 25% of total application memory.
  - Checkpoint File(s): a periodically saved file used to restart a run which was disrupted in some way. May be exactly the same as a Start/Restart file, but may also be larger if it stores higher order terms. If it is automatically or system generated it will be 100% of app memory.
  - Visualization File(s): used to generate interim data which is usually for visualization or similar analysis. These are often only a small fraction of total app memory (5-15%) each.





## How Often Are These Generated?

- Start/Restart File: Once at startup and perhaps at completion of run.
- Checkpoint: Depends on MTBF of machine environment. **This is getting worse, and will not be better on a PFLOP system.** On order of hours.
- Visualization: Depends on data analysis requirements but can easily be several times per minute.





# Latest (Most Optimistic) Numbers

- Blue Gene/L
  - 16TB Memory
  - 40 GB/s I/O bandwidth
  - 400s to checkpoint memory
  
- ASCI Purple
  - 50TB Memory
  - 40 GB/s
  - 1250s to checkpoint memory

**Latest machine will still take on order of minutes to 10's of minutes to do any substantial IO.**





# Example Numbers

We'll use Lemieux, PSC's main machine, as most of these high-demand applications have similar requirements on other platforms, and we'll pick an application (Earthquake Modeling) that won the Gordon Bell prize this past year.





# 3000 PE Earthquake Run

- Start/Restart: 3000 files totaling 150 GB
- Checkpoint: 40 GB every 8 hours
- Visualization: 1.2 GB every 30 seconds

Although this is the largest unstructured mesh ever run, it still doesn't push the available memory limit. Many apps are closer to being memory bound.





---

## A Slight Digression: Visualization Cluster

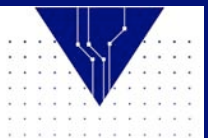
What was once a neat idea has now become a necessity. Real time volume rendering is the only way to render down these enormous data sets to a storable size.





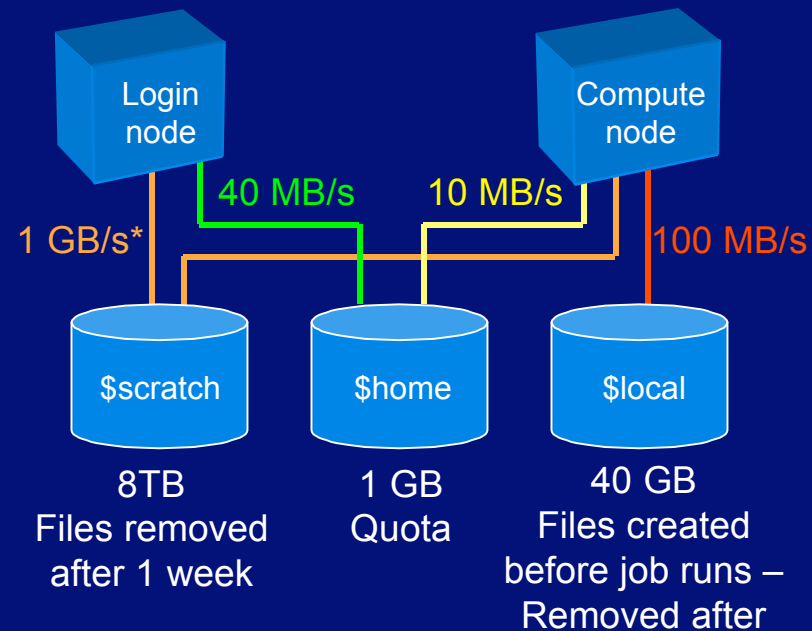
# Actual Route

- Pre-load startup data from FAR to SCRATCH (~12 hr)
- Start holding breath (no node remapping)
- Move from SCRATCH to LOCAL (4 hr)
- Run (16 hour, little IO time w/ 70GB/s path)
- Move from LOCAL to SCRATCH (6 hr)
- Release breath
- Move to FAR/offsite (~12 hr)

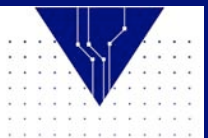


# Which filesystem?

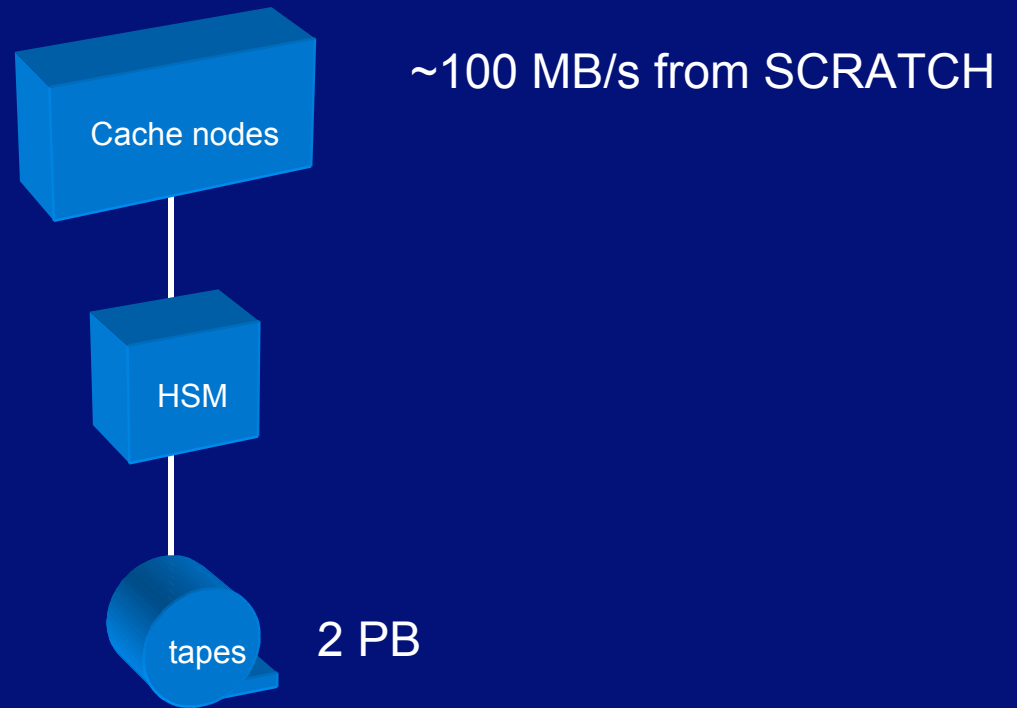
- ❑ \$LOCAL
- ❑ \$HOME
- ❑ \$SCRATCH

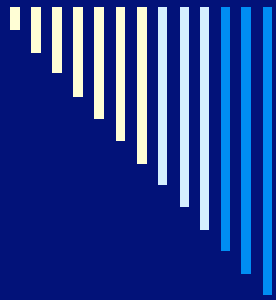


\*Now 3GB/s optimal as of 5/2/04



# Final Destination (FAR)





Bottom Line (which is always some bottleneck)

Like most of the TFLOP class machines, we have several hierarchical levels of file systems. In this case we want to leverage the local disks to keep the app humming along (which it does), but we eventually need to move the data off (and on) to these drives. The machine does not give us free cycles to do this. This pre/post run file migration is the bottleneck here.





## Skip local disk?

Only if we want to spend 70X more time during the run. Although users love a nice DFS solution, it is prohibitive for 3000 PE's writing simultaneously and frequently.





# Where's the DFS?

It's on our giant SMP \_

Just like the difficulty in creating a massive SMP revolves around contention, so does making a DFS (NFS, AFS, GPFS, etc.) that can deal with thousands of simultaneous file writes. Our SCRATCH (~ 1 GB/s) is as close as we get. It is a globally accessible filesystem. But, we still use locally attached disks when it really counts.



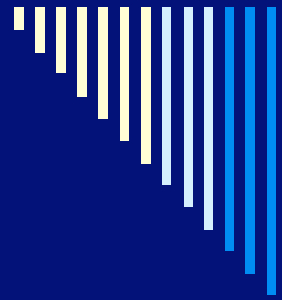


# Parallel Filesystem Test Results

Parallel filesystems were tested with a [simple mpi program](#) that reads and writes a file from each rank. These tests were run Jan, 2004 on the clusters while they were in production mode. The filesystems and clusters were not in dedicated mode, and so these results are only a snapshot.

Hosts * ppn	Approx. Size of Test File	Filesystem	Agg. Transfer rate [MB/s]
32*4	4 gigabytes	PSC/scratch	3000 (5/2/04)
110*2	5 gigabytes	SDSC /gpfs	753
128*2	5 gigabytes	NCSA / gpfs	423
32*2	2.5 gigabytes	Caltech / pvfs	99





## Data path jumps through hoops, how about the code?

Most parallel code has naturally modular, isolated I/O routines. This makes the above issue much less painful. This is very unlike computational algorithm scalability issues which often permeate a code.





## How many lines/hours?

Quake, which has thousands of lines of code, has only a few dozen lines of I/O code in several routines (startup, checkpoint, viz). To accommodate this particular mode of operation (as compared to the default “magic DFS” mode) took only a couple hours of recoding.





---

## How Portable?

This is one area where we have to forego strict portability. However, once we modify these isolated areas of code to deal with the notion of local/fragmented disk spaces, we can bend to any new environment with relative ease.





## Pseudo Code (writing to local)

```
synch
if (not subgroup #X master)
    send data to subgroup #X master
else
    openfile datafile.data.X
    for (1 to number_in_subgroup)
        receive data
        write data
```





## Pseudo Code (reading from local)

```
synch
if (not subgroup #X master)
    receive data
else
    openfile datafile.data.X
    for (1 to number_in_subgroup)
        read data
        send data
```





## Pseudo Code (writing to DFS)

```
synch  
openfile SingleGiantFile  
Setfilepointer(based on PE #)  
write data
```





---

# Platform and Run Size Issues

- ❑ Various platforms will strongly suggest different numbers or patterns of designated I/O nodes (sometimes all nodes, sometimes a very few). Simple to accommodate in code.
- ❑ Different numbers of total PE's or I/O PE's will require different distributions of data in local files. This can be done off-line.





---

# File Migration Mechanics

- ftp, scp, gridco, gridftp, etc.
- tcsio (a local solution)





# How about MPI-IO?

- ❑ Not many (any?) full MPI-2 implementations. More like some vendor/site combinations have implemented the features to accomplish the above type of customization for a particular disk arrangement. Or:
- ❑ Portable-looking, code that runs very, very slow.
- ❑ You can explore this separately via ROMIO:  
<http://www-unix.mcs.anl.gov/romio/>





---

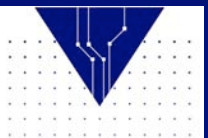
# Parallel Filesystems

## □ PVFS

- <http://www.parl.clemson.edu/pvfs/index.html>

## □ LUSTRE

- <http://www.lustre.org/>





# Current deployments

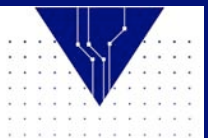
- ❑ Summer 2003 (3 of the top 8 run Linux. Lustre on all 3)
  - ❑ LLNL MCR: 1,100 node cluster
  - ❑ LLNL ALC: 950 node cluster
  - ❑ PNNL EMSL: 950 node cluster
- ❑ Installing in 2004
  - ❑ NCSA: 1,000 nodes
  - ❑ SNL/ASCI Red Storm: 8,000 nodes
  - ❑ LANL Pink: 1,000 nodes





# LUSTRE = Linux+Cluster

- Provides
  - Caching
  - Failover
  - QOS
  - Global Namespace
  - Security and Authentication
- Built on
  - Portals
  - Kernel mods

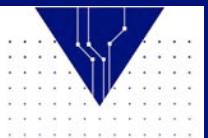


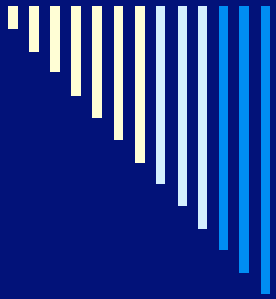


---

# Interface (for striping control)

- Shell
  - Istripe
- Code
  - ioctl



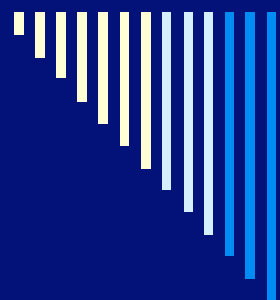


# Performance

From <http://www.lustre.org/docs/lustre-datasheet.pdf>

File I/O % of raw bandwidth:	>90%
Achieved client I/O:	>650 MB/s
Aggregate I/O 1,000 clients:	11.1 GB/s
Attribute retrieval rate: (in 10M file directory, 1,000 clients)	7500/s
Creation rate: (one directory, 1,000 clients)	5000/s





# Benchmarks

## □ FLASH

- [http://flash.uchicago.edu/~zingale/flash\\_benchmark\\_io/#intro](http://flash.uchicago.edu/~zingale/flash_benchmark_io/#intro)

## □ PFS

- <http://www.osc.edu/~djohnson/gelato/pfbs-0.0.1.tar.gz>





---

## Didn't Cover (too trivial for us)

- ❑ Formatted/Unformatted
- ❑ Floating Point Representations
- ❑ Byte Ordering
- ❑ XDF – No help for parallel performance





---

Now for that last bullet...

A Checkpointing API with Nathan Stone

