Parallel File Systems at HPC Centers: Usage, Experiences, and Recommendations

William (Bill) E. Allcock
ALCF Director of Operations

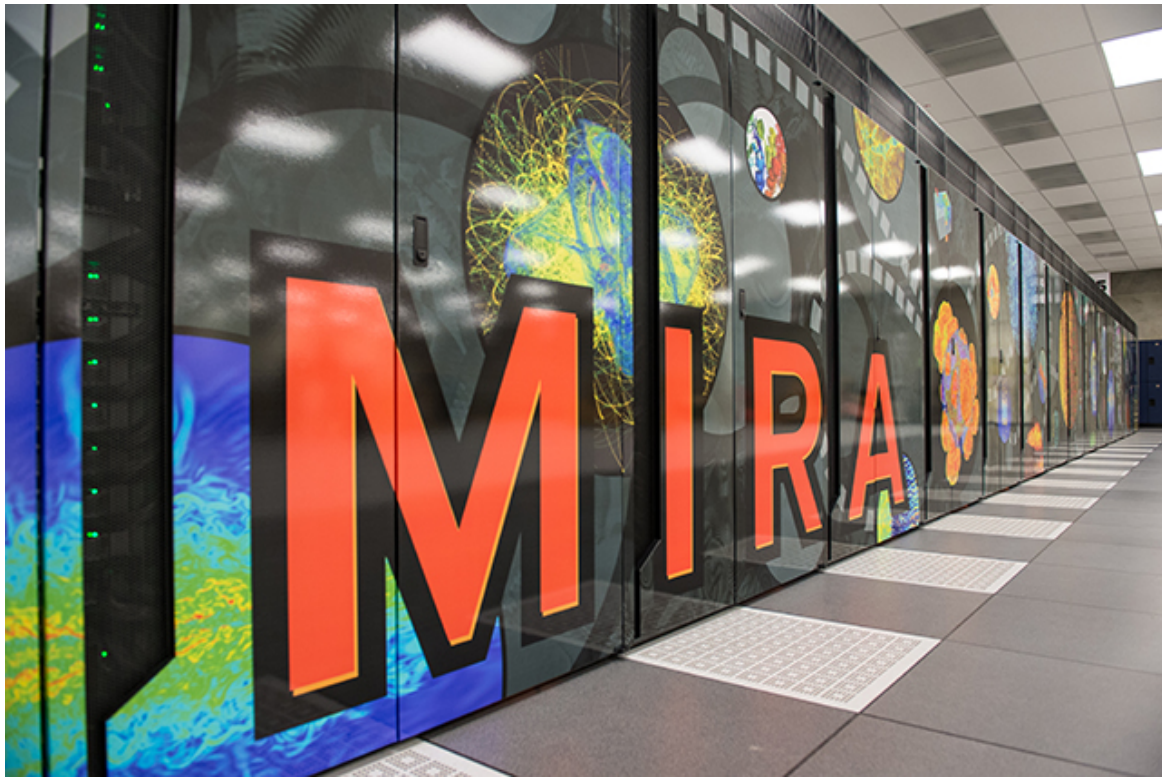
Production Systems: ALCF-2

Mira – *BG/Q system*
- 49,152 nodes / 786,432 cores
- 786 TB of memory
- Peak flop rate: 10 PF
- Linpack flop rate: 8.1 PF

Vesta - *BG/Q system*
- 2,048 nodes / 32,768 cores
- 32 TB of memory
- Peak flop rate: 419 TF

Cetus - *BG/Q system*
- 1,024 nodes / 16,384 cores
- 16 TB of memory
- Peak flop rate: 209 TF

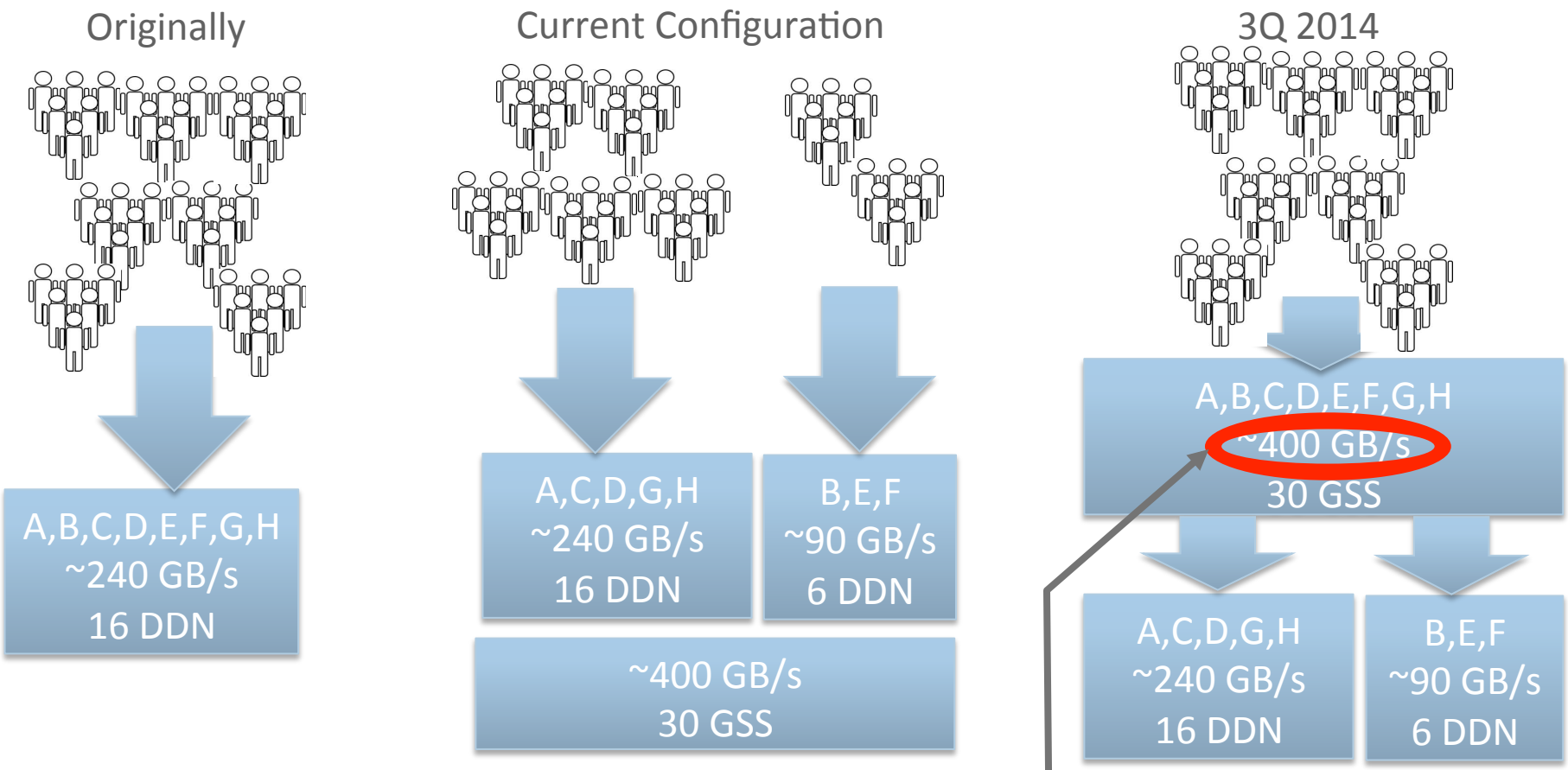
Tukey – *NVIDIA system*
- 100 nodes / 1600 x86 cores
- 200 M2070 GPUs
- 6 TB x86 memory / 1.1TB GPU memory
- Peak flop rate: 220 TF



Storage at ALCF

- We use GPFS
- How do we handle storage?
  - Home is pretty typical (stored on 3 arrays, replicated metadata and data, backed up to tape)
  - For computational output, we want one namespace, seen everywhere
  - We do not purge at all
    - On intrepid, it was the honor system and we nagged when things got full; It worked, but not well.
    - On Mira, we have quotas in effect, which forces the users to manage their disk space.
    - Weird side effect: We have more available disk capacity than tape capacity
- Originally, that one namespace was one GPFS file system for computational output
  - Very convenient for the users; Very little time spent on data management and movement.
  - However, single point of failure.
  - We ran this way for 5 years on Intrepid without problems, then had a hard power failure and found disk corruption. For a variety of reasons, it took nearly 3 weeks to fsck our 4.5PB file system and get it back online
  - Having a second file system (PVFS) saved our bacon, because we had a few users that could run from there.
- We had just configured Mira the same way when the corruption hit. You can imagine what happened next…

Our storage upgrade path



Per GPFS Development, this is the fastest GPFS filesystem in the world

- IBM GPFS Storage Server (GSS)
  - Uses GPFS "Native RAID" (Software RAID)
  - No controllers; x3650 servers with attached JBODs

“Burst buffer like” solution

- Before anyone yells “off with his head” I said burst buffer like. Depending on your world view, you may or may not consider it to be a burst buffer.
- It does, however, share some characteristics
  - Fast, but not big enough to permanently keep the data there (a cache)
  - Improves performance for most common scenarios (defensive I/O, possibly in-situ analysis)
  - Forces us to deal with how to manage moving data in and out without user intervention
- It keeps the aspects of our storage system that we like, but avoids the ones we don’t:
  - To the user, it still looks like a single namespace visible to everything
  - They will see better performance than today
  - We have three file systems so if one is down, some of our users can still run
- Makes use of GPFS Active File Management (AFM)
  - But not in the way GPFS developers planned ☺
  - They designed it primarily for WAN file systems, so performance characteristics are different, particularly the speed of migration into and out of the cache.
  - Collaborating with GPFS development team; Specific features added / accelerated to support the “burst buffer scenario”.
  - Currently in testing
    - GPFS protocol parallel migrations are working well, seeing full performance to the “home” file system
    - Working on scaling the thread count up
    - Testing failure scenarios

We would like to take this one more step…

- "Hands off data management"
- GPFS – HPSS – Interconnect (GHI)
  - Use HPSS as a hierarchical storage back end for GPFS
  - Policy driven object store
    - Make a copy of a file on disk to tape "shortly" after creation (wait a while in case they delete)
    - If the file goes some period of time (weeks or months) without being accessed delete the copy on disk
    - It still shows up in the GPFS metadata and will be automatically be migrated back to disk if accessed.
      - We have requested a change to this functionality. We would like to be able to configure it to throw an error instead to avoid accidental major migrations.
  - Gives us a "backup" of scratch, something most HPC centers (including us) do not do
  - Ideally, eliminates "mundane" human management of data
    - Policy will take care of migrating to tape and removing data from disk that has not been used
    - Entry stays in GPFS metadata and can be explicitly recalled by command or file is automatically recalled from tape if it is accessed.
    - This includes long term archival after a project ends.
    - Use GPFS filesets; New feature allows filesets to have their own inode pool, so don't have to worry about inode pool getting too large (at least we hope, proof will be in the pudding)
    - When a project ends, policy will move all the data from disk, but leave their fileset in place. If they every come back and ask for a file, it is a simple copy.

The good, the bad, and the ugly…

- Key attributes that we believe will make this work
  - A 13PB cache is big enough for several months worth of production, and the back end store is big enough for years of production.
  - We can be very relaxed on our migration policies so that once we remove data from any layer in the hierarchy, it is extremely unlikely it will ever be recalled.
- What are the possible issues we might see?
  - Accidental migration: A user might do something like a grep –r and accidentally stage in significant amounts of data.
    - This is why we want the option of making the system throw an error rather than automatically stage back in. This requires "a human in the loop". If they really do want to stage the data back in, they can issue a command to do so.
  - We waste space putting data on tape that is not needed.
    - This is a real possibility. We have to weigh this against the value of the users time spent copying data around to avoid deletion and manually archiving data.
    - We plan to aggressively educate our users about this storage system and hopefully, they will spend the time to delete unneeded data well before a migration would be scheduled to occur.