Optimizing Lustre Performance Using Stripe-Aware Tools

Paul Kolano
NASA Advanced Supercomputing Division
paul.kolano@nasa.gov

Introduction

- Lustre has great performance...
  - ...If you know how to use it
- Standard system tools employed by users to manipulate files do not know how to use it
  - Do not take striping into consideration
    - Files end up on too few or too many stripes
  - Not enough parallelism to keep Lustre busy
    - File operations achieve fraction of available I/O bandwidth
- Subject of this talk
  - Modify standard tools to more appropriately support Lustre
    - Stripe-aware system tools
    - High performance system tools

Stripe-Aware System Tools

Part 1/2

Lustre Stripe Counts

- Stripe count determines how many OSTs a file will be divided across
- Stripe count can significantly impact I/O performance
  - Good: more OSTs = more available bandwidth
  - Bad: more OSTs = more overhead
- Striping is set when file created and cannot be modified without copying data
  - Need to specify stripe count carefully or may be sorry later!

Specifying Lustre Stripe Counts

- Option 1: Default striping policy
  - Stripe count of newly created files will default to configured value when not explicitly set
- Problem 1: Different file sizes behave better with different stripe counts
  - High default value
    - Small files waste space on OSTs
    - Small files generate more OST traffic than desirable for things like stat operations
  - Low default value
    - Large files achieve significantly reduced performance
    - Large files result in imbalanced OST utilization

Specifying Lustre Stripe Counts (cont.)

- Option 2: Manual striping by user
  - Prestripe files and/or directories with "lfs setstripe -c"
- Problem 2: What's a stripe?
  - Users may not know what a stripe is
  - Users may not remember to set striping
  - Users may not know what the appropriate value should be for their files/directories
  - User directories typically contain mixture of small/large files
    - Same dilemma as default case

Specifying Lustre Stripe Counts (cont.)

- Option 3: Stripe-aware system tools
  - Stripe files dynamically based on size as users perform normal system activities
  - Default can be kept low for more common small files
- Problem 3: Few (if any) system tools know about Lustre striping

Specifying Lustre Stripe Counts (cont.)

- Option 3: Stripe-aware system tools
  - Stripe files dynamically based on size as users perform normal system activities
  - Default can be kept low for more common small files
- Problem 3: Few (if any) system tools know about Lustre striping
- Solution: Enhance commonly used system tools with this knowledge!

Tools Used In Typical HPC Workflow

- User remotely transfers data to file system
  - scp, sftp, rsync, bbftp, gridftp
- User prepares data for processing
  - tar -x, gunzip, bunzip2, unzip
- User processes data on compute resources
  - Unknown
    - Input: will already be striped appropriately (hopefully!)
    - Output: still based on default/user-specified striping
- User prepares results for remote transfer
  - tar -c, gzip, bzip2, zip
- User remotely retrieves results from file system
  - Not our problem!

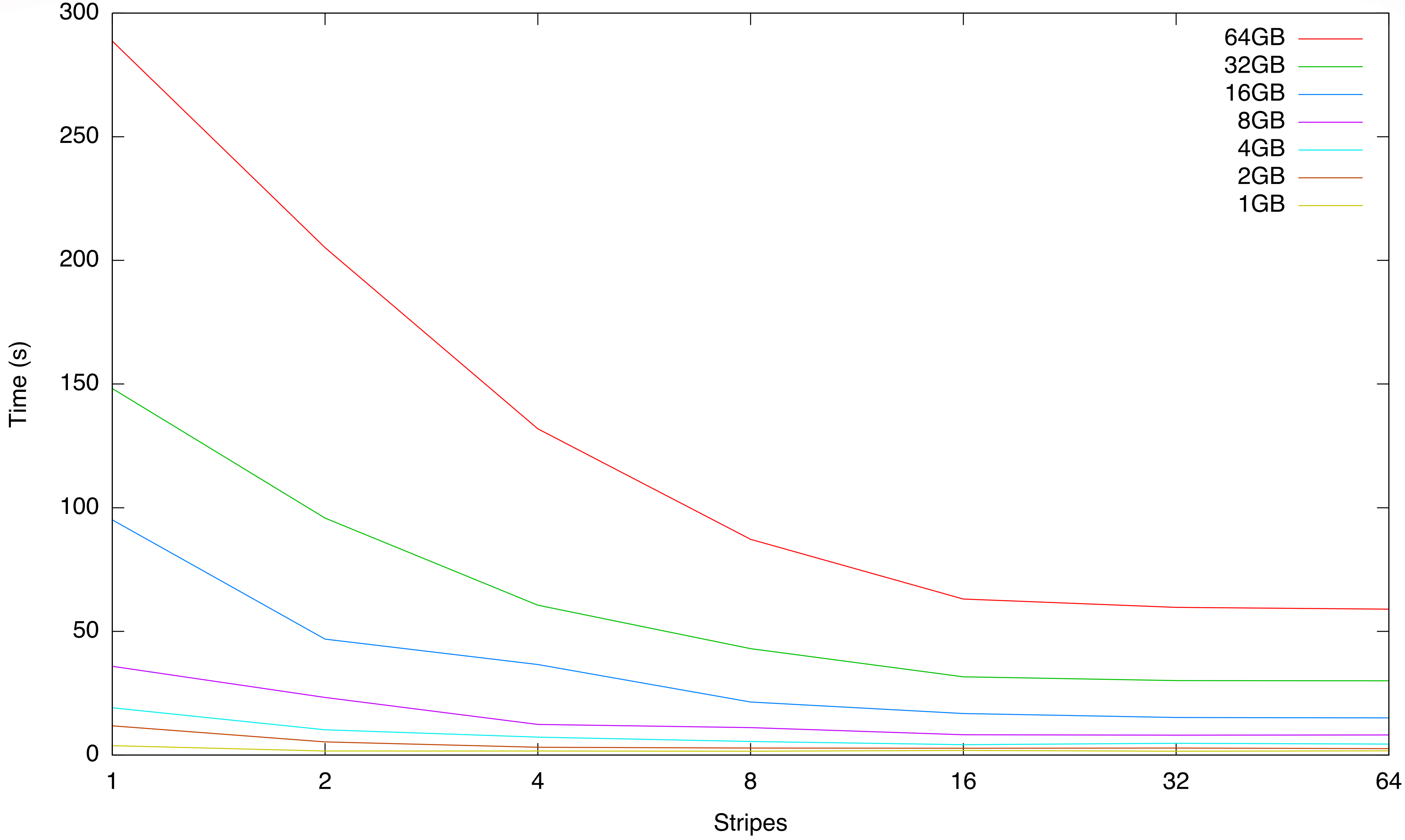
Tools Used In Other Common Activities

- Admin copies data between file systems to balance utilization
  - cp, rsync
- User copies data between file systems (e.g. home/backup directory to scratch space)
  - cp, rsync
- User retrieves data from archive systems
  - scp, sftp, rsync, bbftp, gridftp

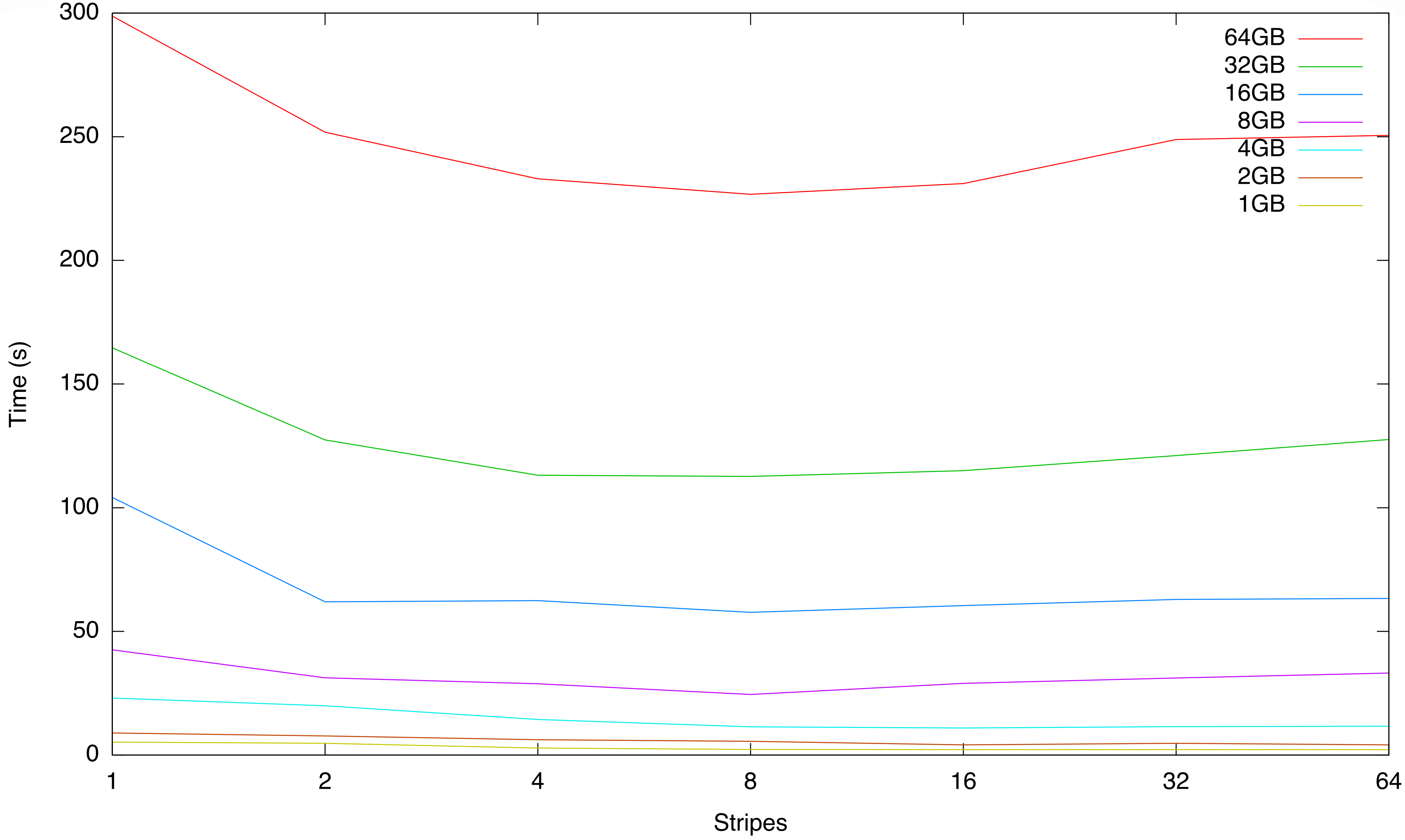
Adding Stripe-Awareness (Simple!)

- Find instances of open() using O_CREAT flag
  - Striping needs to be specified at file creation
- Determine if target file is on Lustre
  - statfs() f_type == LL_SUPER_MAGIC
- Determine projected size of target file
  - Complexity may be higher in some applications
    - e.g. Must sum over individual file sizes during tar creation
- Compute desired stripe count based on size
  - Can preserve source striping with llapi_file_get_stripe()
- Switch open() to llapi_file_open() with stripe count

4 Host Parallel dd **Write** Time (Different Offsets of Same File with Direct I/O)



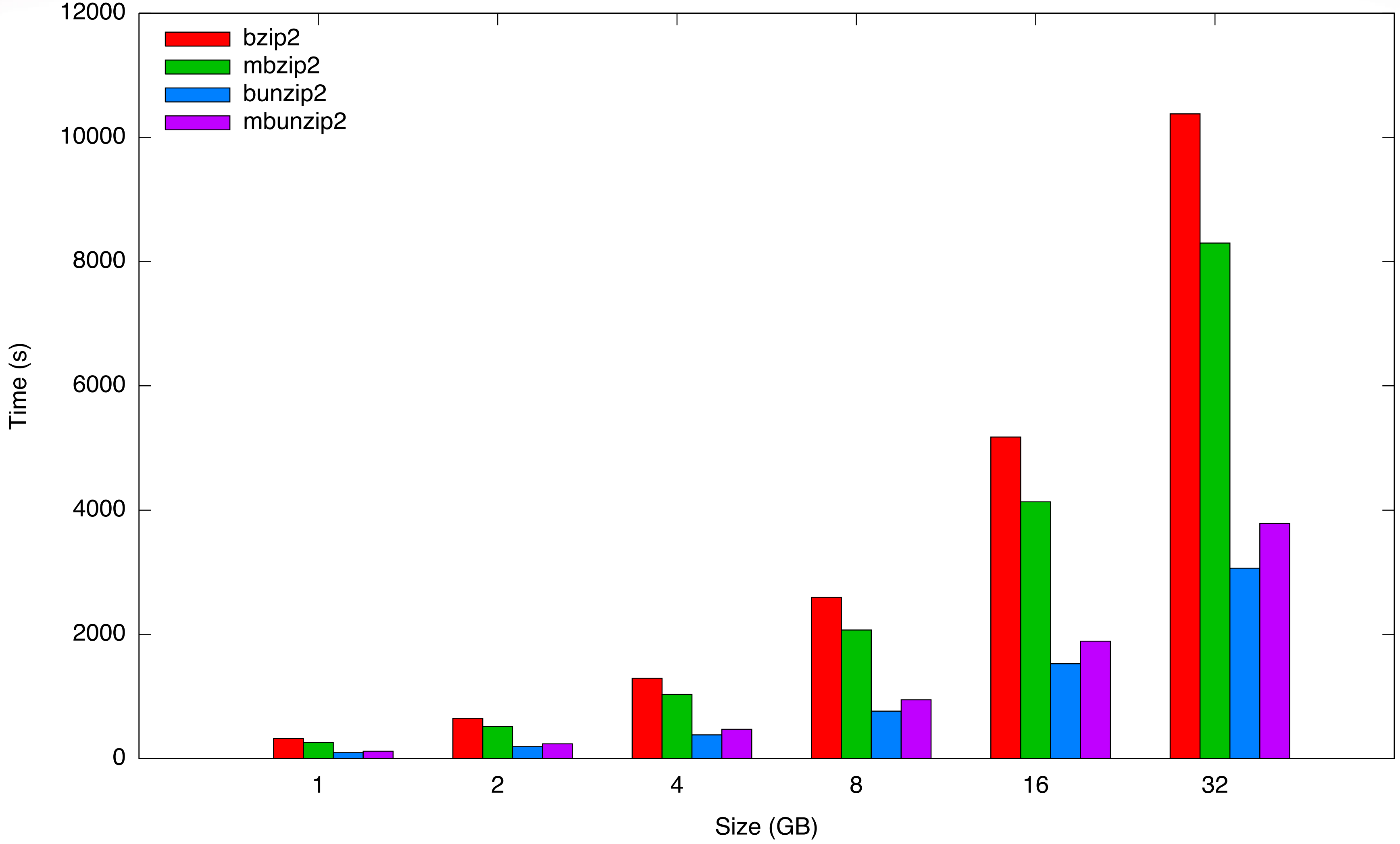
4 Host Parallel dd **Read** Time (Different Offsets of Same File with Direct I/O)



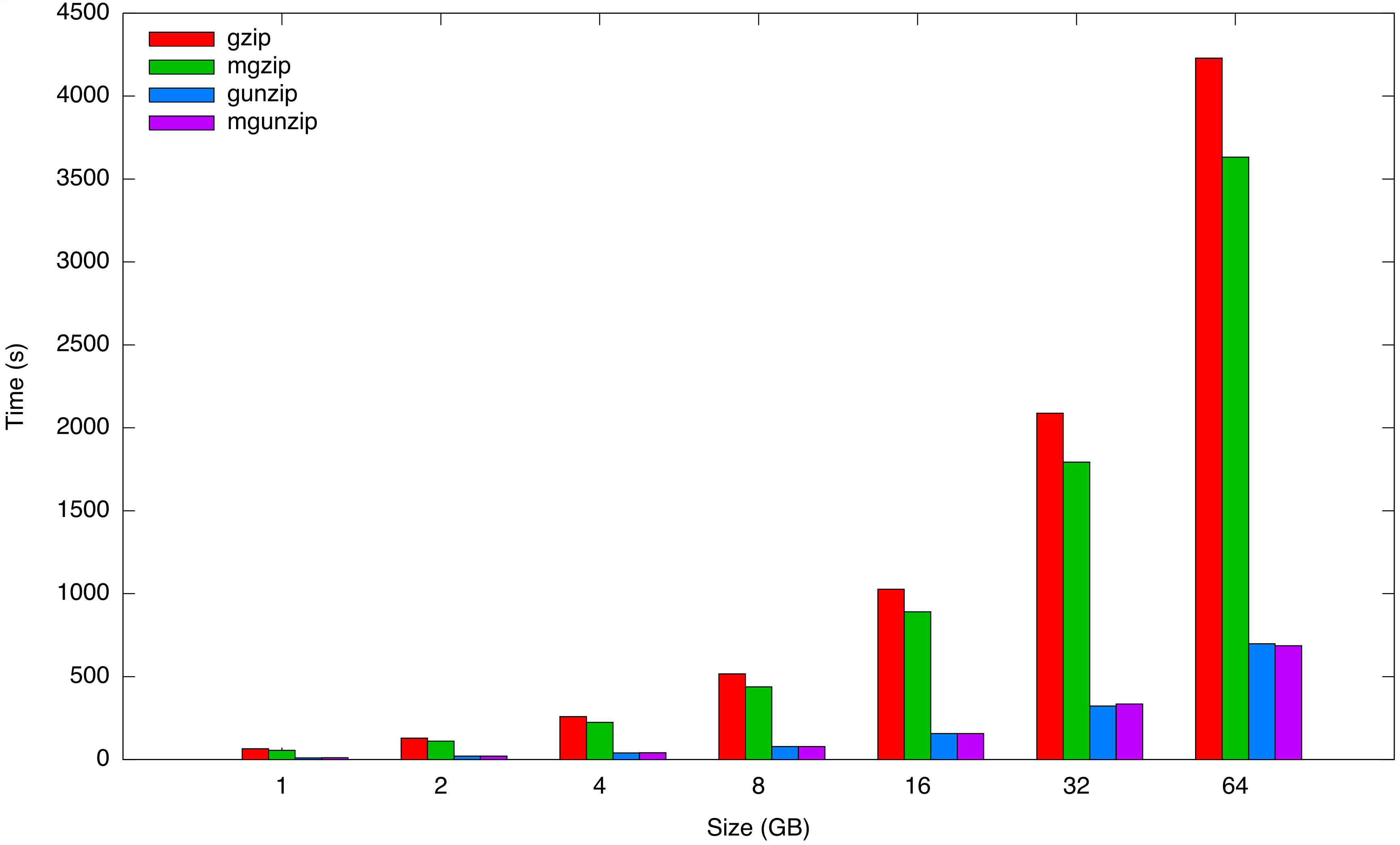
Retools: Restriping Tools for Lustre

- These particular results seem to indicate 1 stripe per 2-4 GBs may be best
  - Probably needs further analysis
- Implemented set of stripe-aware tools
  - Tools start with "m" for historical (and possibly future) purposes
  - Basic activities covered
    - Archival/Extraction: mtar
    - Compression/Decompression: mbzip2/mbunzip2, mgzip/mgunzip
    - Local transfer: mcp, mrsync
    - Remote transfer: mrsync
  - Striping policy
    - Originally set at 1 stripe per GB (graphs schmaphs!)
    - Before any analysis based on "gut feeling" of staff members

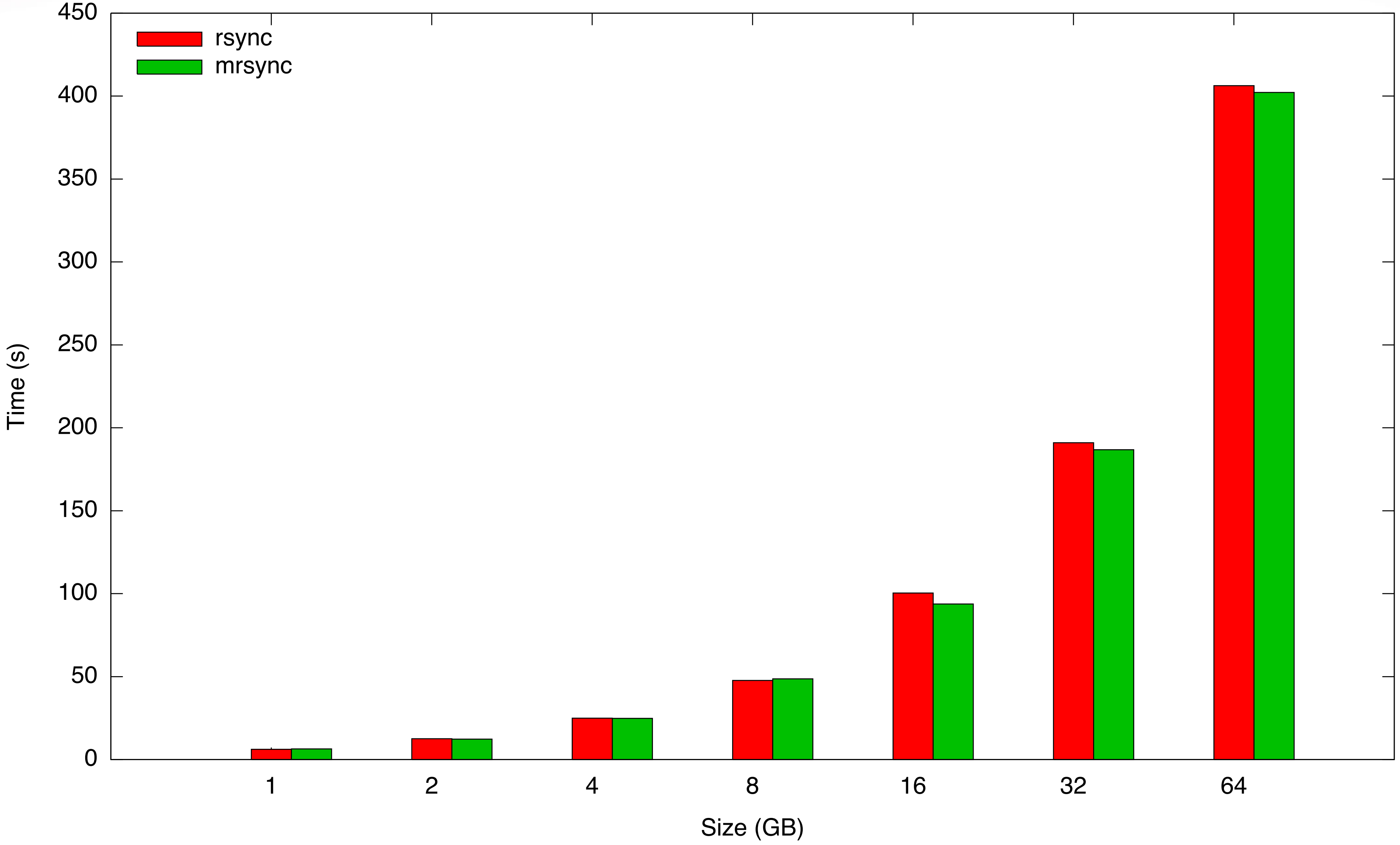
Bzip2/Bunzip2 Execution Times (1 Source File with 1 Stripe)



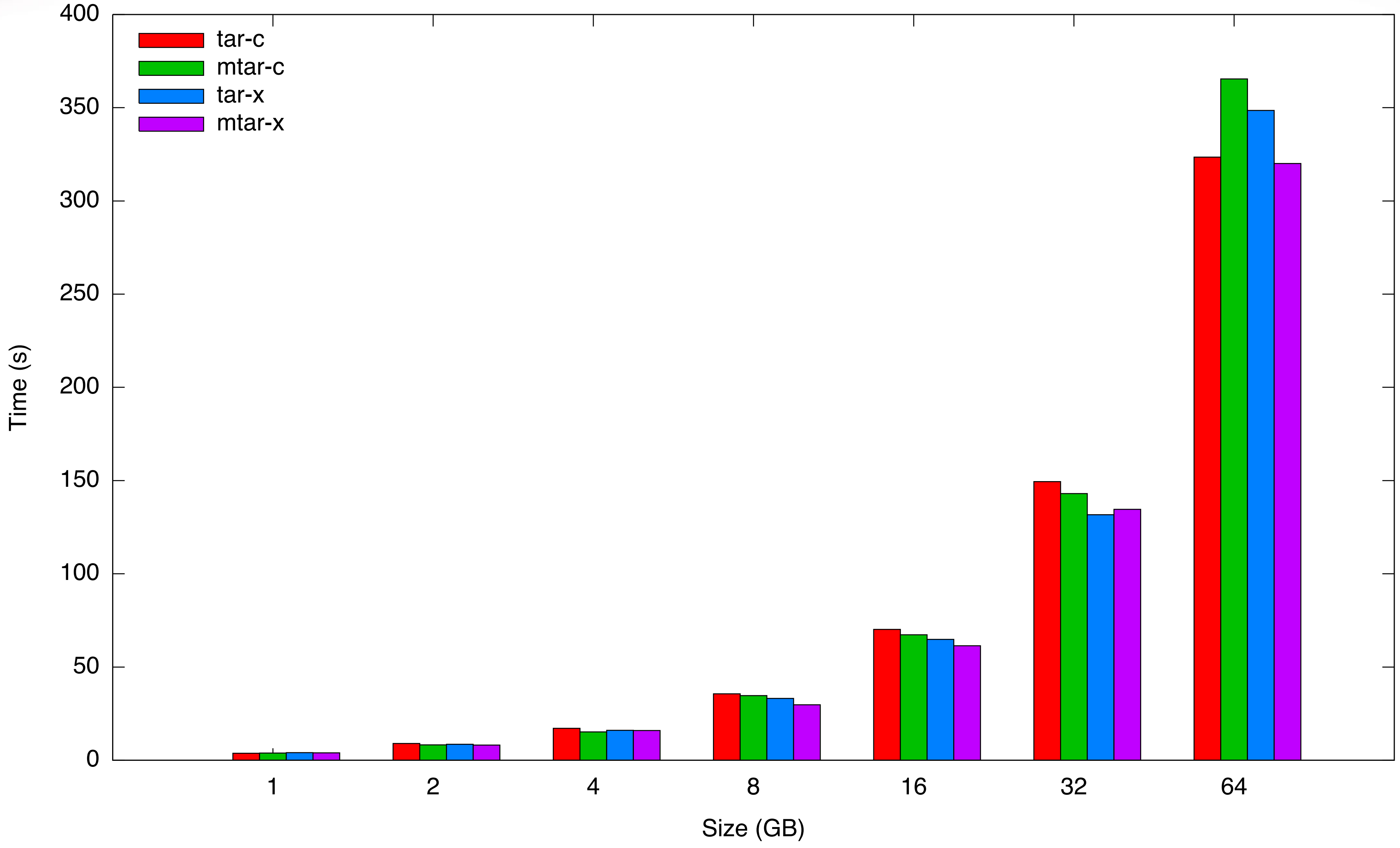
Gzip/Gunzip Execution Times (1 Source File with 1 Stripe)



Rsync Execution Times (1 Source File with 1 Stripe)



Tar Create/Extract Execution Times (1 Source File with 1 Stripe)



Stripe-Awareness: A Good First Step

- Can keep default stripe count low for more common small files
  - Reduced OST contention and wasted space
- Large files will automatically use more stripes as they are manipulated by standard system tools
  - User computations will transparently achieve higher performance
  - OST utilization will be kept in better balance
- Modest performance gains for tools themselves
- But...
  - Standard system tool performance still nowhere near raw Lustre I/O rates

High Performance System Tools

Part 2/2

High Performance Tools

- Problem: Standard system tools don't know how to take advantage of Lustre's high bandwidth
  - Use single thread of execution, which cannot keep single system I/O bandwidth fully utilized
  - Rely on operating system buffer cache, which becomes bottleneck
  - Forego parallelism in favor of simplicity by using sequential reads and writes
  - Operate on one host, where single system bottlenecks limit max performance

High Performance Tools

- Problem: Standard system tools don't know how to take advantage of Lustre's high bandwidth
  - Use single thread of execution, which cannot keep single system I/O bandwidth fully utilized
  - Rely on operating system buffer cache, which becomes bottleneck
  - Forego parallelism in favor of simplicity by using sequential reads and writes
  - Operate on one host, where single system bottlenecks limit max performance
- Solution: Enhance commonly used system tools with this knowledge!

Increasing Tool Performance Beyond Striping (Complex!)

- Use multiple threads to keep single host busy
- Use direct I/O to bypass buffer cache
- Use asynchronous I/O to overlap reads/writes
- Use multiple hosts for aggregate bandwidth
- Large files reduce effectiveness of parallelism
  - Split processing of files into parallelizable chunks

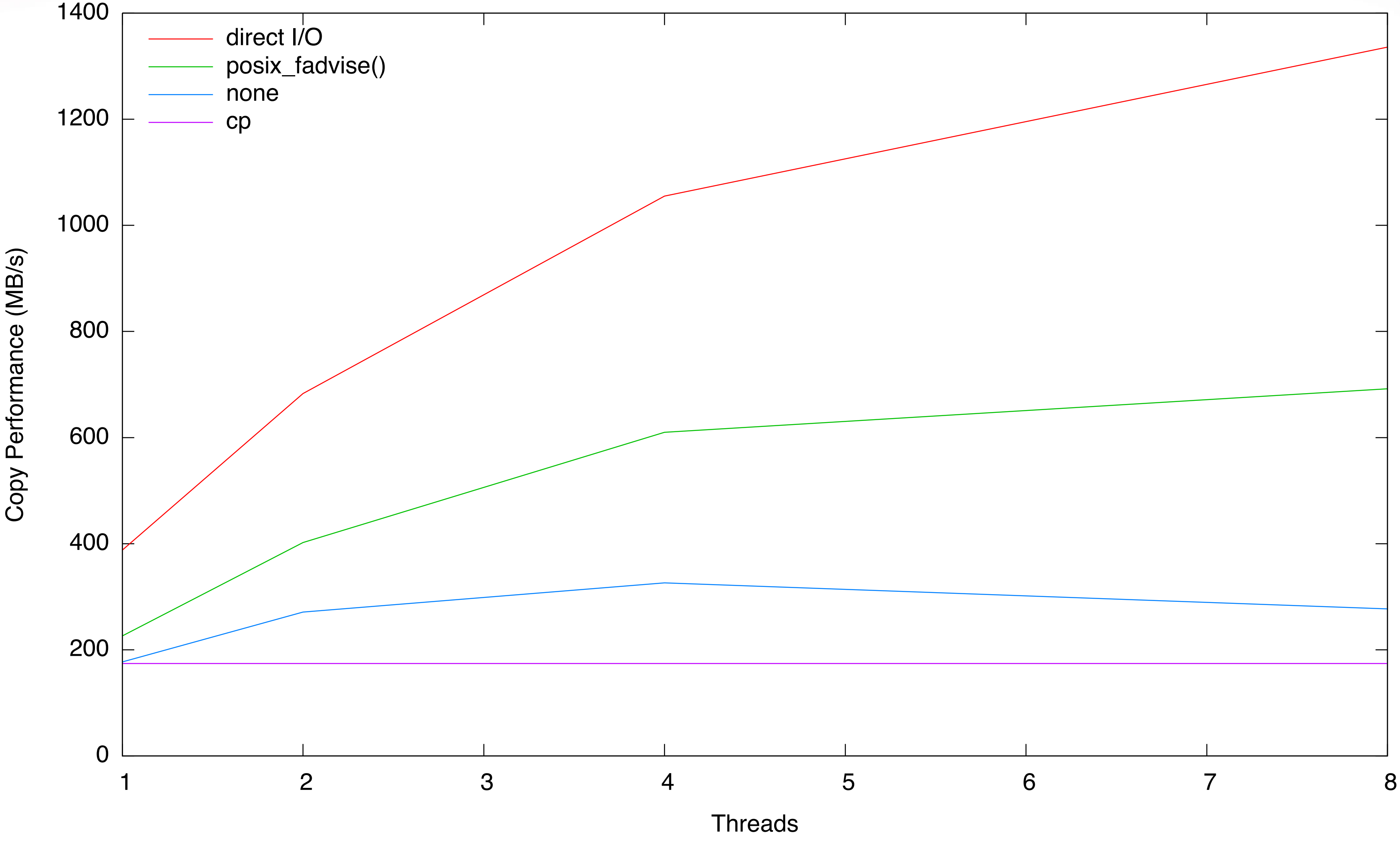
Example: High Performance Cp (The rest are left as exercises for the reader!)

- Mcp: the original (and still the best!) "m" util
  - Multi-threaded
  - Multi-node
- Original single-threaded cp behavior
  - Depth-first search
  - Directories are created with write/search permissions before contents copied
  - Directory permissions restored after subtree copied

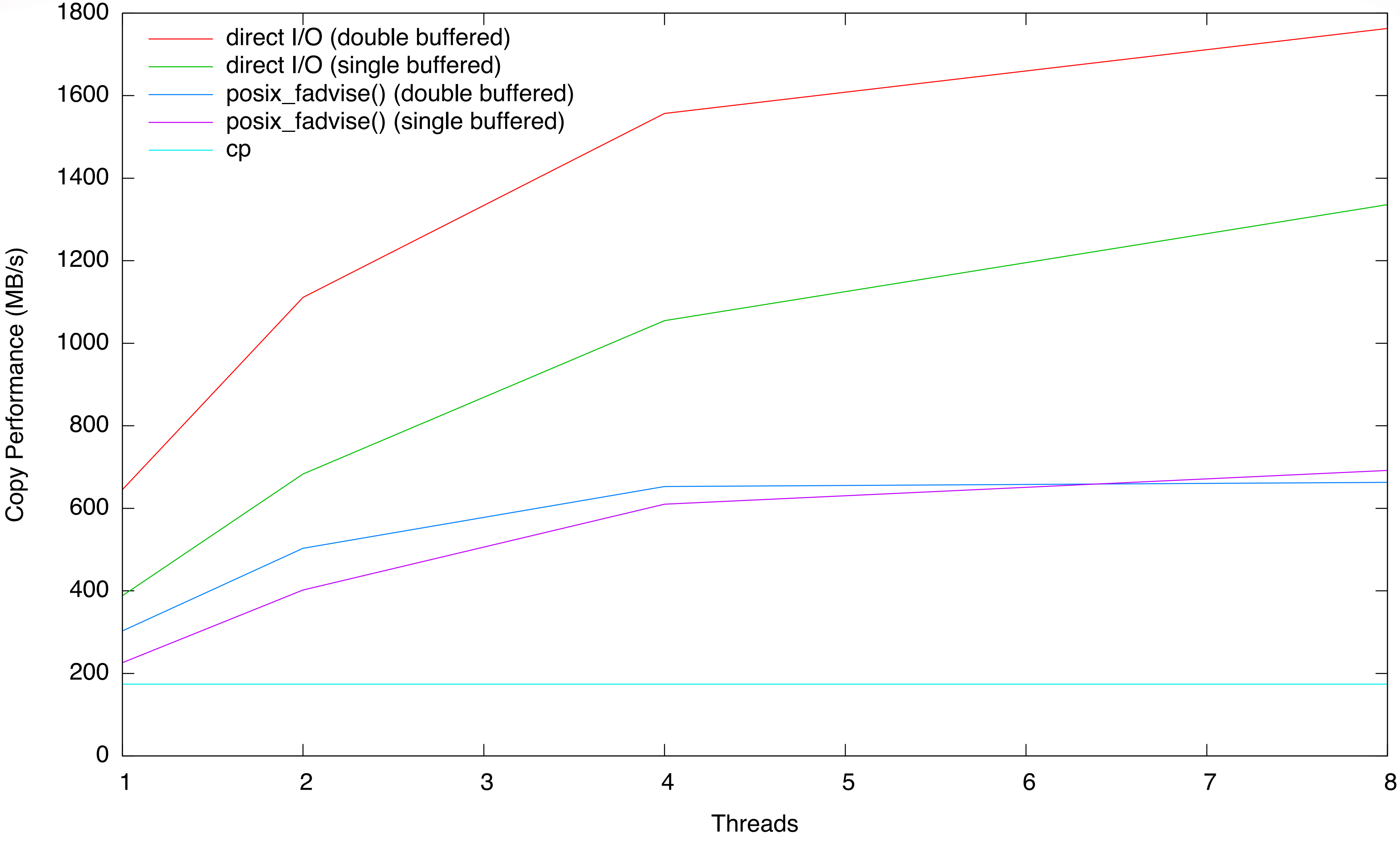
Multi-Threaded Parallelization of Cp (via OpenMP)

- Traversal thread
  - Original cp behavior except when regular file encountered
    - Create copy task and push onto semaphore-protected task queue
    - Pop open queue indicating file has been opened
    - Set permissions and ACLs
- Worker threads
  - Pop task from task queue
  - Open file and push notification onto open queue
    - Directory permissions and ACLs are irrelevant once file is opened
  - Perform copy
- Multi-node capability
  - Manager node and worker nodes with TCP or MPI threads handling distribution of tasks between traversal thread and worker threads

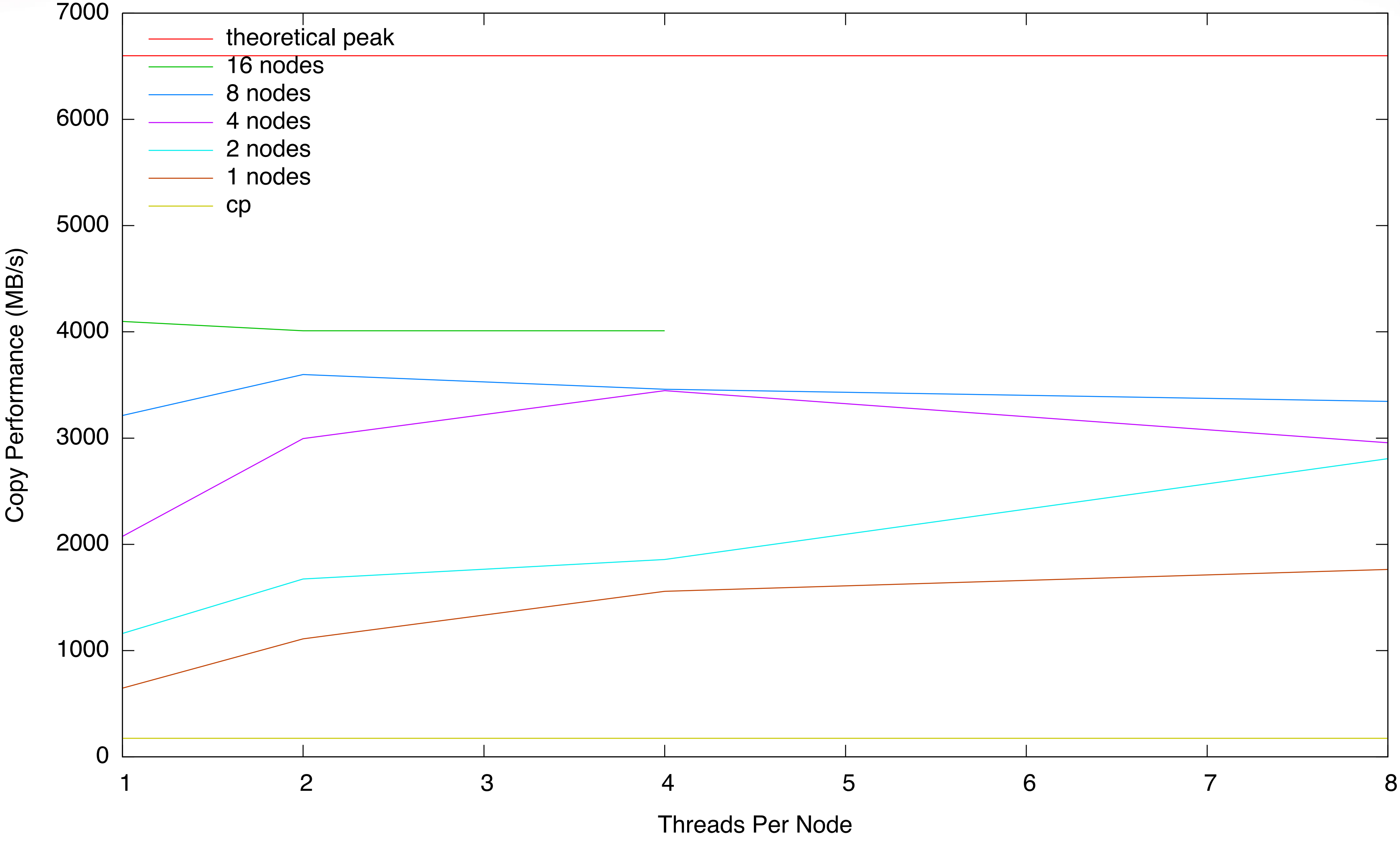
Adding Multi-Threading/Buffer Management (64x1GB)



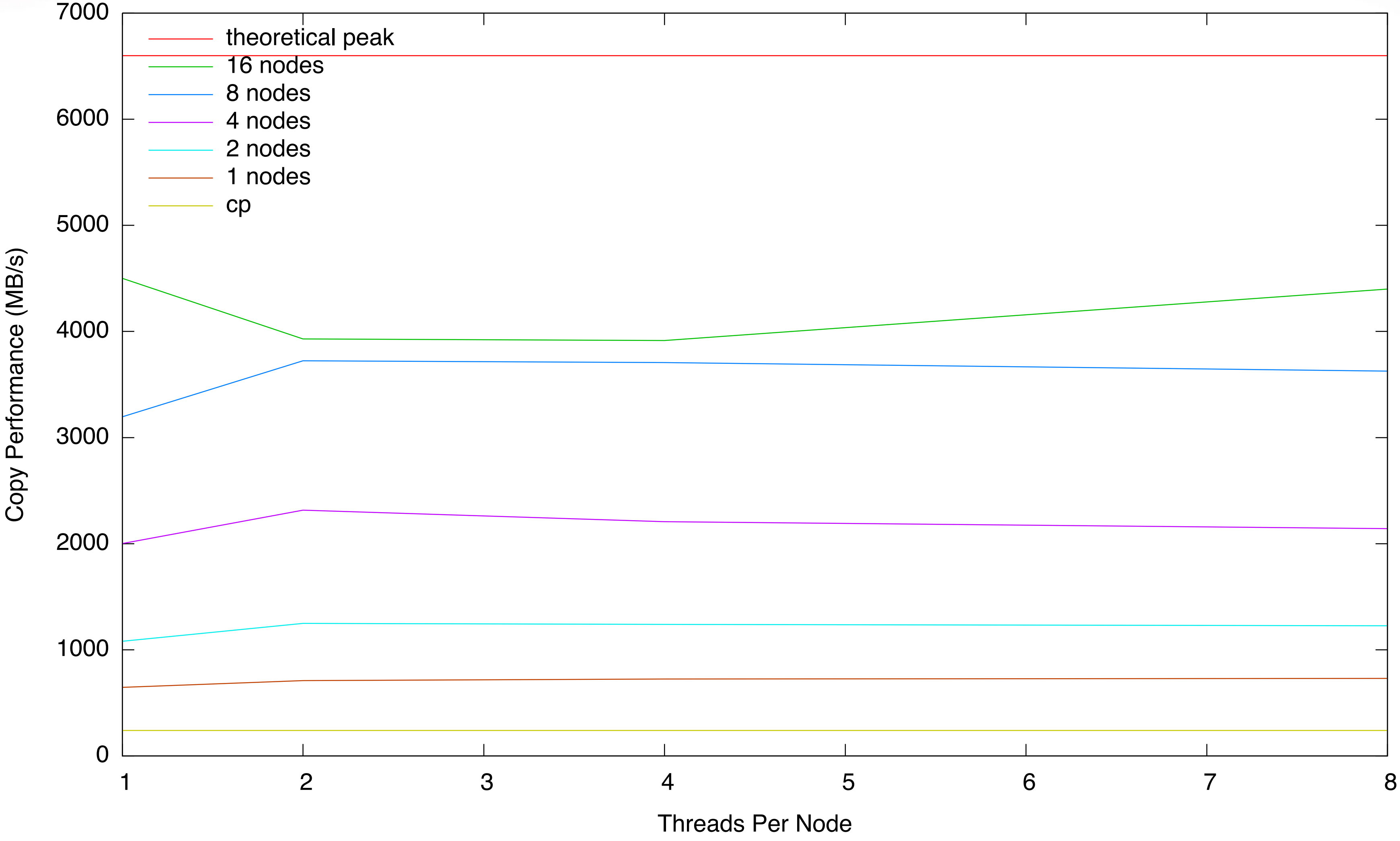
Adding Double Buffering via Asynchronous I/O (64x1GB)



Adding Multi-Node Support via TCP/MPI (64x1GB)



Adding Split-File Support (1x128GB)



Mcp Results

- Cp performance now more in line with that of Lustre
  - 10x/27x of original cp on 1/16 nodes
  - 72% of peak based on (old) 6.6 GB/s max read/write
- Side benefit: fast restriping
  - Only way to restripe files is to copy
  - Mcp does fast copies and is stripe-aware!

Conclusion

- Modified standard system tools commonly found in user workflows to better support Lustre
  - Stripe-aware tools
  - High performance tools
- Based on original source code
  - 100% compatible drop-in replacement for standard tools
    - e.g. install as "tar", not "mtar"
- Better for users
  - Transparently achieve higher performance by simply using the tools they already use
- Better for file systems
  - Reduce contention, wasted space, and imbalances on OSTs

Future Work

- Make other tools in standard workflow stripe-aware
  - Archive/compression: zip
  - Transfer: scp, sftp, bbftp, gridftp
- Make other tools high performance
  - Tar a good candidate since it is widely used and very slow
- Better analysis of optimal stripe count formula

Finally…

- Retools: mbzip2, mgzip, mrsync, and mtar
  - In process of being open sourced (takes a few months)
    - U.S. Govt.: can get right now through inter-agency release
  - Will live at http://retools.sourceforge.net when released
- Mutil: mcp and msum (high performance md5sum)
  - Already open sourced and available
  - http://mutil.sourceforge.net
- Email:
  - paul.kolano@nasa.gov

- Questions?