High Performance Multi-Node File Copies and Checksums for Clustered File Systems

Paul Kolano, Bob Ciotti

NASA Advanced Supercomputing Division

{paul.kolano,bob.ciotti}@nasa.gov

Overview

- Problem background
- Multi-threaded copies
- Optimizations
  - Split processing of files
  - Buffer cache management
  - Double buffering
- Multi-node copies
- Parallelized file hashing
- Conclusions and future work

File Copies

- Copies between local file systems are a frequent activity
  - Files moved to locations accessible by systems with different functions and/or storage limits
  - Files backed up and restored
  - Files moved due to upgraded and/or replaced hardware
- Disk capacity increasing faster than disk speed
  - Disk speed reaching limits due to platter RPMs
- File systems are becoming larger and larger
  - Users can store more and more data
- File systems becoming faster mainly via parallelization
  - Standard tools were not designed to take advantage of parallel file systems
- Copies are taking longer and longer

Existing Solutions

- GNU coreutils cp command
  - Single-threaded file copy utility that is the standard on all Unix/Linux systems
- SGI cxfscp command
  - Proprietary multi-threaded file copy utility provided with CXFS file systems
- ORNL spdcp command
  - MPI-based multi-node file copy utility for Lustre

Motivation For a New Solution

- A single reader/writer cannot utilize the full bandwidth of parallel file systems
  - Standard cp only uses a single thread of execution
- A single host cannot utilize the full bandwidth of parallel file systems
  - SGI cxfscp only operates across a single host (or single system image)
- There are many types of file systems and operating environments
  - ORNL spdcp only operates on Lustre file systems and only when MPI is available

Mcp

- Copy program designed for parallel file systems
  - Multi-threaded parallelism maximizes single system performance
  - Multi-node parallelism overcomes single system resource limitations
    - Portable TCP model
  - Compatible with many different file systems
- Drop-in replacement for standard cp
  - All options supported
  - Users can take full advantage of parallelism with minimal additional knowledge

Parallelization of File Copies

- File copies are mostly embarrassingly parallel
  - Directory creation
    - Target directory must exist when file copy begins
  - Directory permissions and ACLs
    - Target directory must be writable when file copy begins
    - Target directory must have permissions and ACLs of source directory when file copy completes

Multi-Threaded Copies

- Mcp based on cp code from GNU coreutils
  - Exact interface users are familiar with
  - Original behavior
    - Depth-first search
    - Directories are created with write/search permissions before contents copied
    - Directory permissions restored after subtree copied

Multi-Threaded Copies (cont.)

- Multi-threaded parallelization of cp using 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
  - 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
    - Optionally, push final stats onto stat queue
  - Stat (and later...hash) thread
    - Pop stats from stat queue
    - Print final stats received from worker threads

Test Environment

- Pleiades supercluster (#6 on Jun. 2010 TOP500 list)
  - 1.009 PFLOPs/s peak with 84,992 cores over 9472 nodes
  - Nodes used for testing
    - Two 3.0 GHz quad-core Xeon Harpertown
    - 1 GB DDR2 RAM per core
- Copies between Lustre file systems
  - 1 MDS, 8 OSSs, 60 OSTs each
  - IOR benchmark performance
    - Source read: 6.6 GB/s
    - Target write: 10.0 GB/s
  - Theoretical peak copy performance: 6.6 GB/s
- Performance measured with dedicated jobs on (near) idle file systems
  - Minimal interference from other activity
- Test cases, baseline performance, and stripe count

tool	stripe count	64x1 GB	1x128 GB
cp	default (4)	174	102
cp	max (60)	132	240

Multi-Threaded Copy Performance (MB/s)

tool	threads	64 x 1 GB	1 x 128 GB
cp	1	174	240
mcp	1	177	248
mcp	2	271	248
mcp	4	326	248
mcp	8	277	248

- Less than expected and diminishing returns
- No benefit in single large file case

Handling Large Files (Split Processing)

- Large files create imbalances in thread workloads
  - Some may be idle
  - Others may still be working
- Mcp supports parallel processing of different portions of the same file
  - Files are split at a configurable threshold
  - The main traversal thread adds n “split” tasks
  - Worker threads only process portion of file specified in task

Split Processing Copy Performance (MB/s)

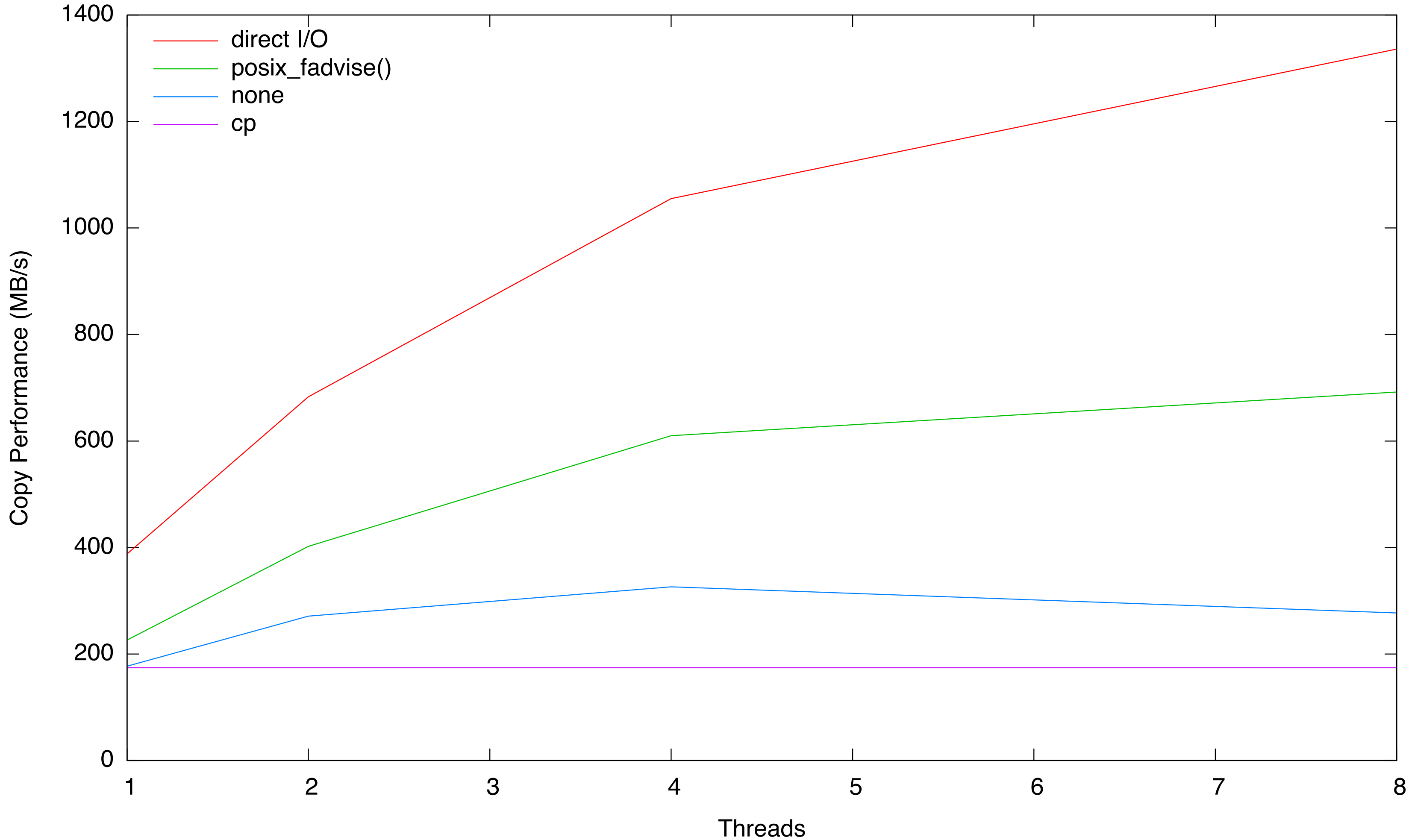
tool	threads	split size	1 x 128 GB
mcp	*	0	248
mcp	2	1 GB	286
mcp	2	16 GB	296
mcp	4	1 GB	324
mcp	4	16 GB	322
mcp	8	1 GB	336
mcp	8	16 GB	336

- Less than expected and diminishing returns
- Minimal difference in overhead
  - Will use 1 GB split size in remainder

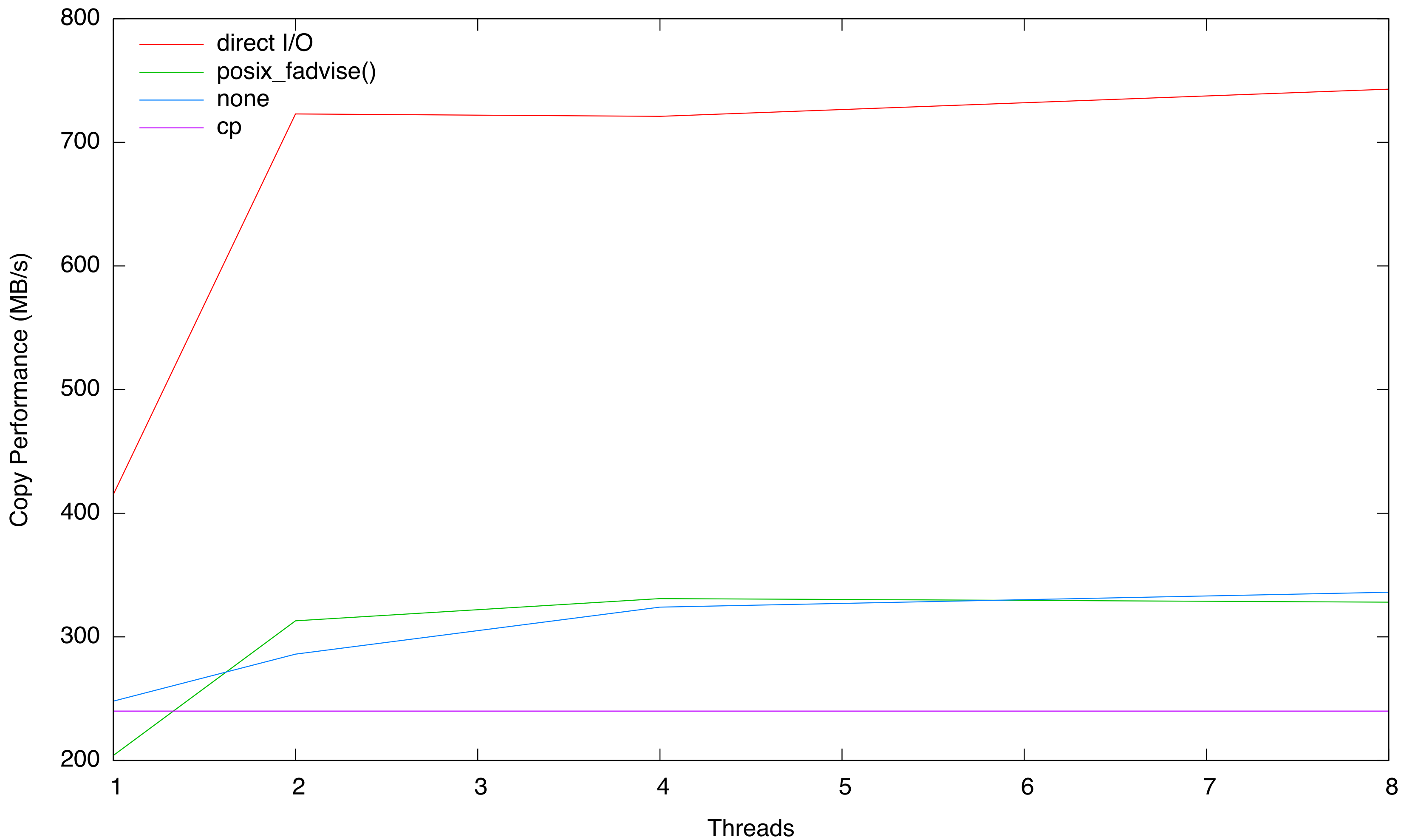
Less Than Expected Speedup (Buffer Cache Management)

- Buffer cache becomes liability during copies
  - CPU cycles wasted caching file data that is only accessed once
  - Squeezes out existing cache data that may be in use by other processes
- Mcp supports two alternate management schemes
  - posix_fadvise()
    - Use buffer cache but advise kernel that file will only be accessed once
  - Direct I/O
    - Bypass buffer cache entirely

Managed Buffer Cache Copy Performance (64x1 GB)



Managed Buffer Cache Copy Performance (1x128 GB)



We Can Still Do Better On One Node (Double Buffering)

- Read/writes of file blocks are serially processed within the same thread
  - Time:

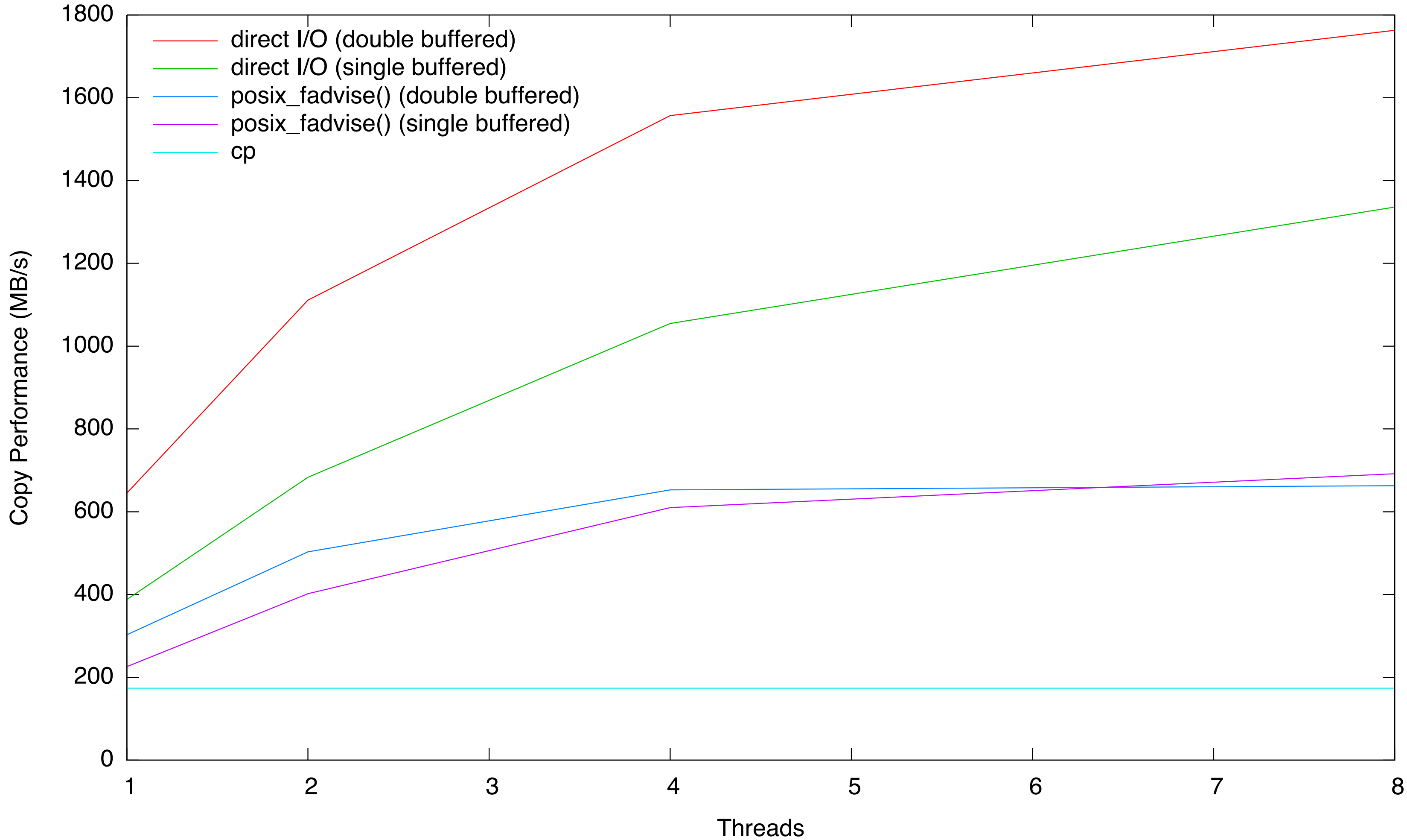
    n_blocks * (time(read) + time(write))
- Mcp uses non-blocking I/O to read next block while previous block being written
  - Time:

    time(read) +

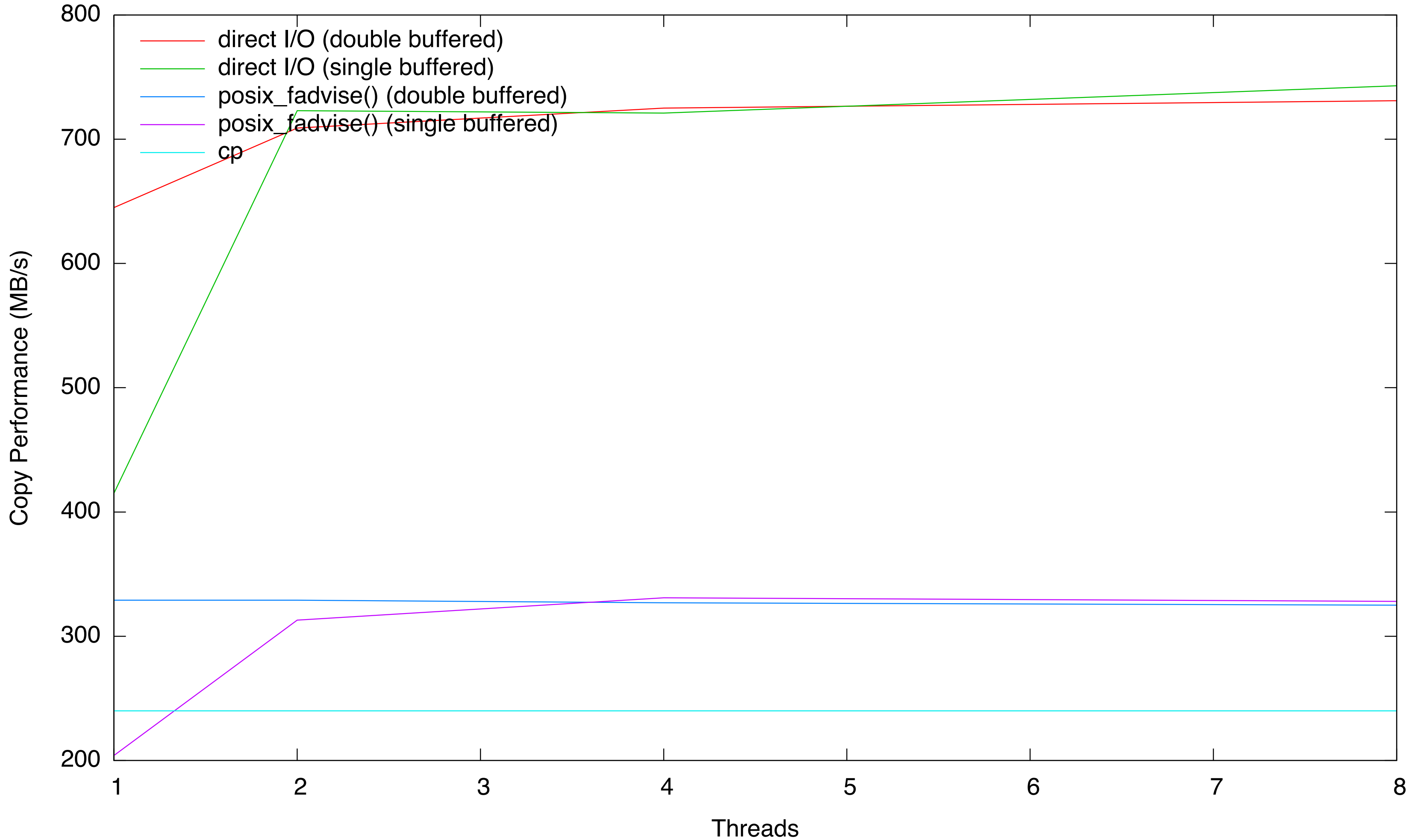
    (n_blocks-1) * max(time(read), time(write)) +

    time(write)

Double Buffered Copy Performance (64x1 GB)



Double Buffered Copy Performance (1x128 GB)



Multi-Node Copies

- Multi-threaded copies have diminishing returns due to single system bottlenecks
- Need multi-node parallelism to maximize performance
- Mcp supports both MPI and TCP models
  - Only TCP will be discussed (MPI similar)
    - Lighter weight
    - More portable
    - Ability to add/remove workers nodes dynamically
      - Can use larger set of smaller jobs instead of one large job
      - Can add workers during off hours and remove during peak

Multi-Node Copies Using TCP

- Manager node
  - Traversal thread, worker threads, and stat/hash thread
  - TCP thread
    - Listens for connections from worker nodes
      - Task request
        - Pop task queue
        - Send task to worker
      - Stat report
        - Push onto stat queue
- Worker nodes
  - Worker threads
    - Push task request onto send queue
    - Perform copy in same manner as original worker threads
    - Push stat report onto send queue instead of stat queue
  - TCP thread
    - Pop send queue
    - Send request/report to TCP thread on manager node
    - For task request, receive task and push onto task queue

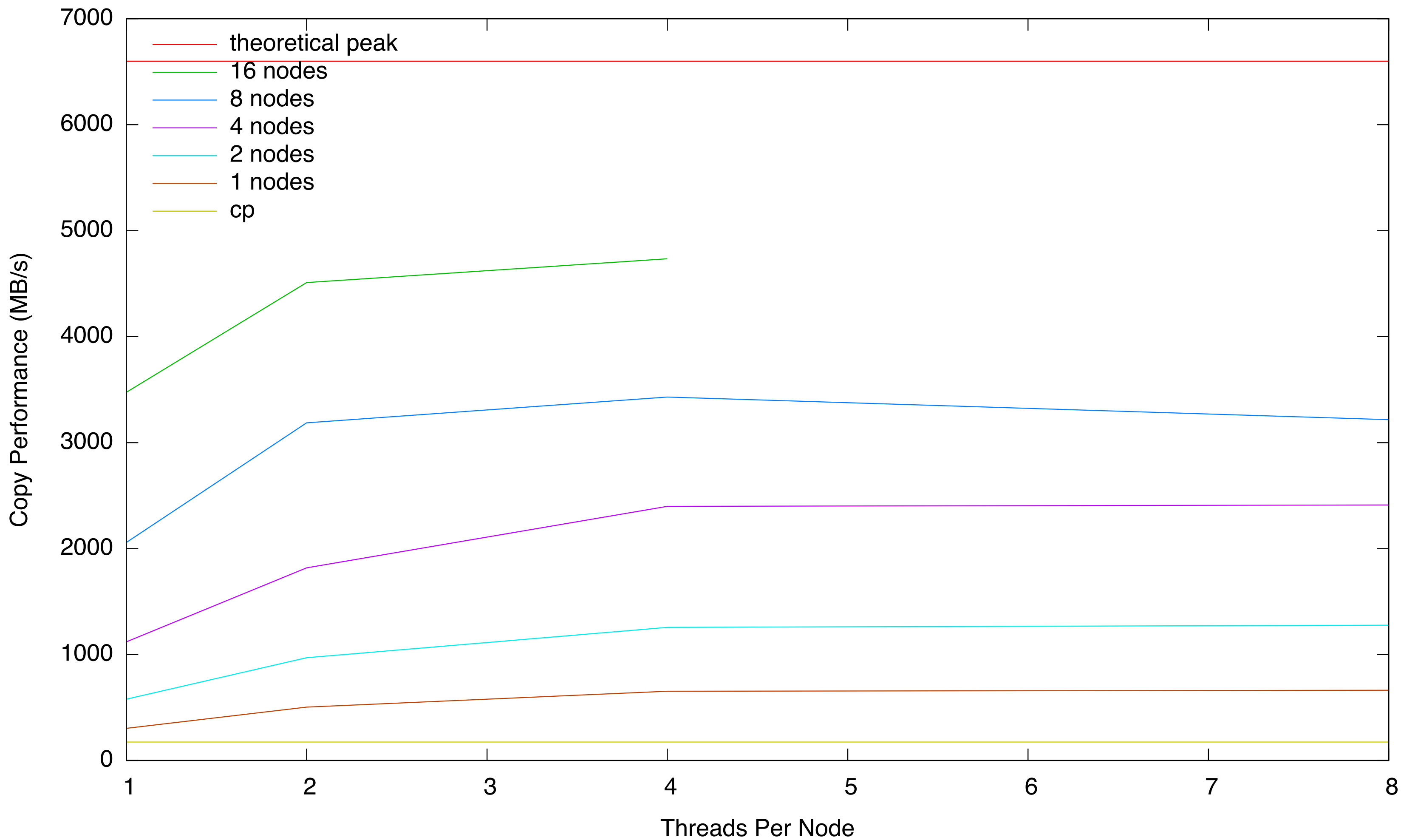
TCP Security Considerations

- Communication over TCP is vulnerable to attack (especially for root copies)
  - Integrity
    - Lost/blocked tasks
      - Files may not be updated that were supposed to be
        - e.g. cp /new/disabled/users /etc/passwd
    - Replayed tasks
      - Files may have been changed between legitimate copies
        - e.g. cp /tmp/shadow /etc/shadow
    - Modified tasks
      - Source and destination of copies
        - e.g. cp /attacker/keys /root/.ssh/authorized_keys
  - Confidentiality
    - Contents of normally unreadable directories can be revealed
      - Tasks intercepted on the network
      - Tasks falsely requested from the manager
  - Availability
    - Copies can be disrupted by falsely requesting tasks
    - Normal network denials of service (won’t discuss)

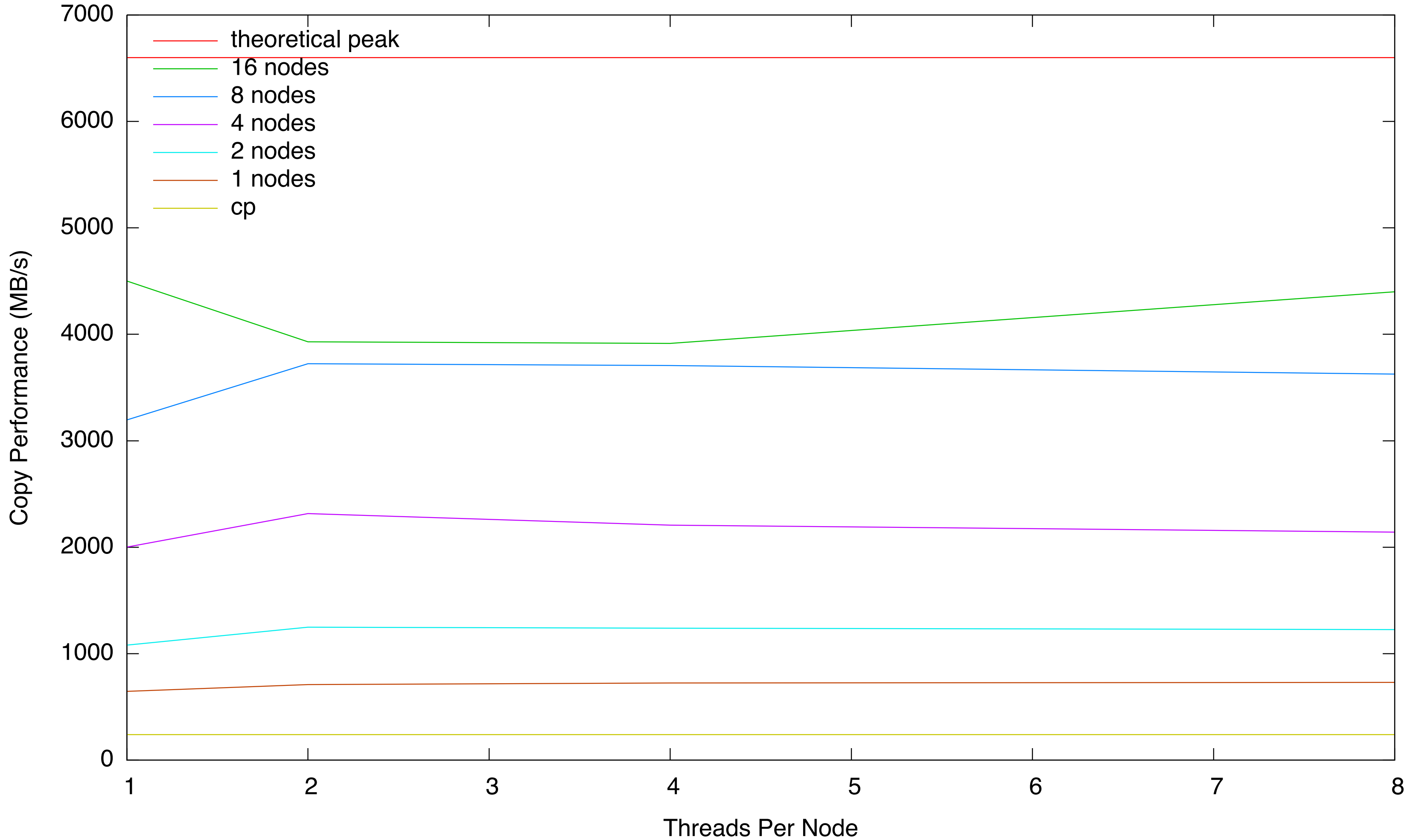
TCP Security Implementation

- Mcp secures all communication via TLS-SRP
  - Transport Layer Security (TLS)
    - Provides integrity and privacy using encryption
      - Tasks cannot be intercepted, replayed, or modified over the network
  - Secure Remote Password (SRP)
    - Provides strong mutual authentication using simple passwords
      - Workers will only perform tasks from legitimate managers
      - Manager will only reveal task details to legitimate workers

Multi-Node Copy Performance (64x1 GB w/ posix_fadvise())



Multi-Node Copy Performance (1x128 GB w/ direct I/O)



Good New and Bad News

- Good news
  - We can do fast copies
    - 10x/27x of original cp on 1/16 nodes
    - 72% of peak based on 6.6 GB/s max read/write
- Bad news
  - The more data copied, the greater the probability for corruption
    - Disk corruption, memory glitches, etc.
    - Traditional approach to verify integrity
      - Hash file at source (e.g. md5sum)
      - Hash file at destination and verify (e.g. md5sum –c)
  - Hashes are inherently serial
    - hash(ab) != hash(ba)

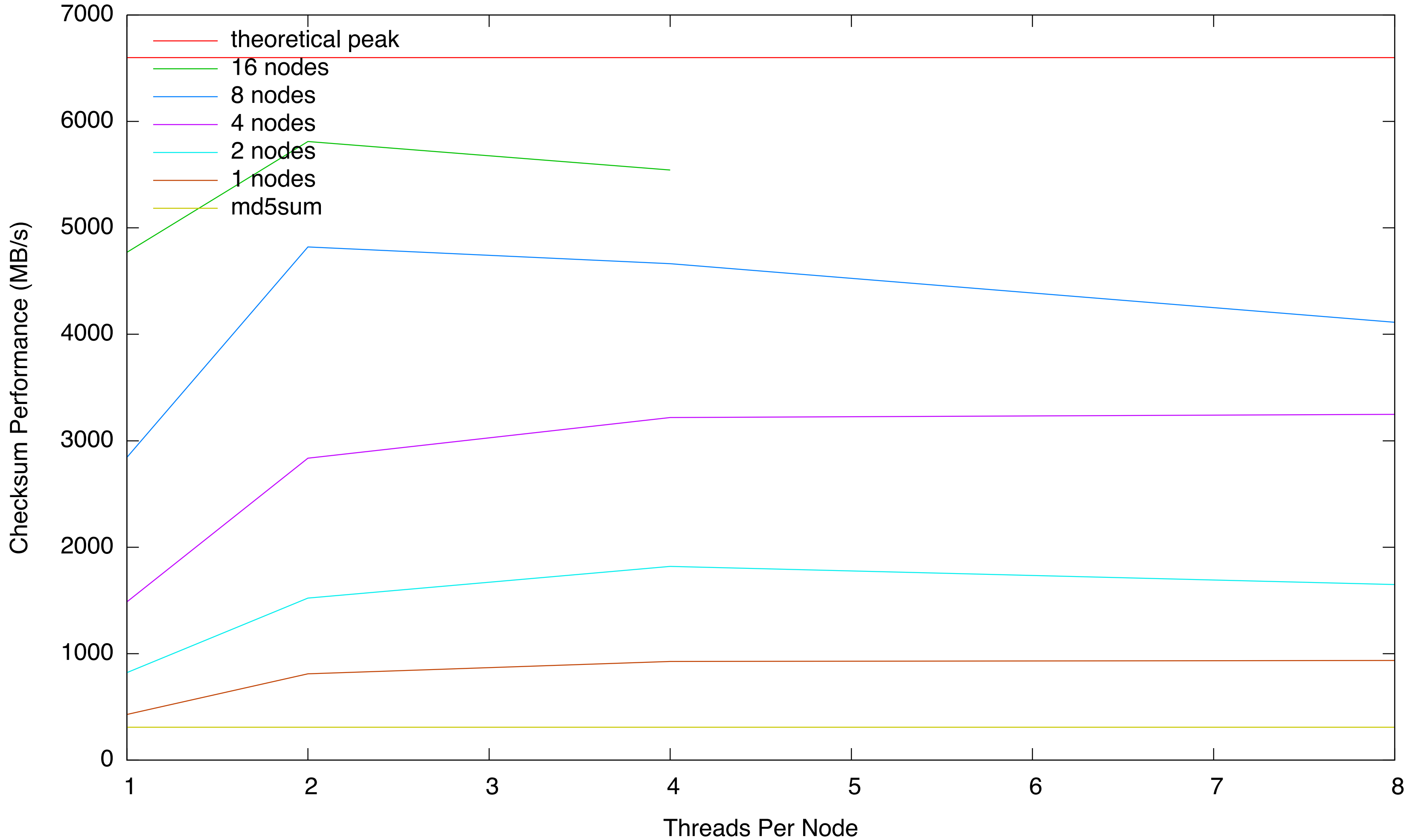
Good News About the Bad News

- Use hash trees
  - Leaf nodes are standard hashes of each subset of file at a given granularity
  - Internal nodes are hashes of concatenated child hashes
  - Root is single hash value
- Hash trees can be parallelized
  - All subtrees computed in parallel
  - Computation of remaining root of tree done serially

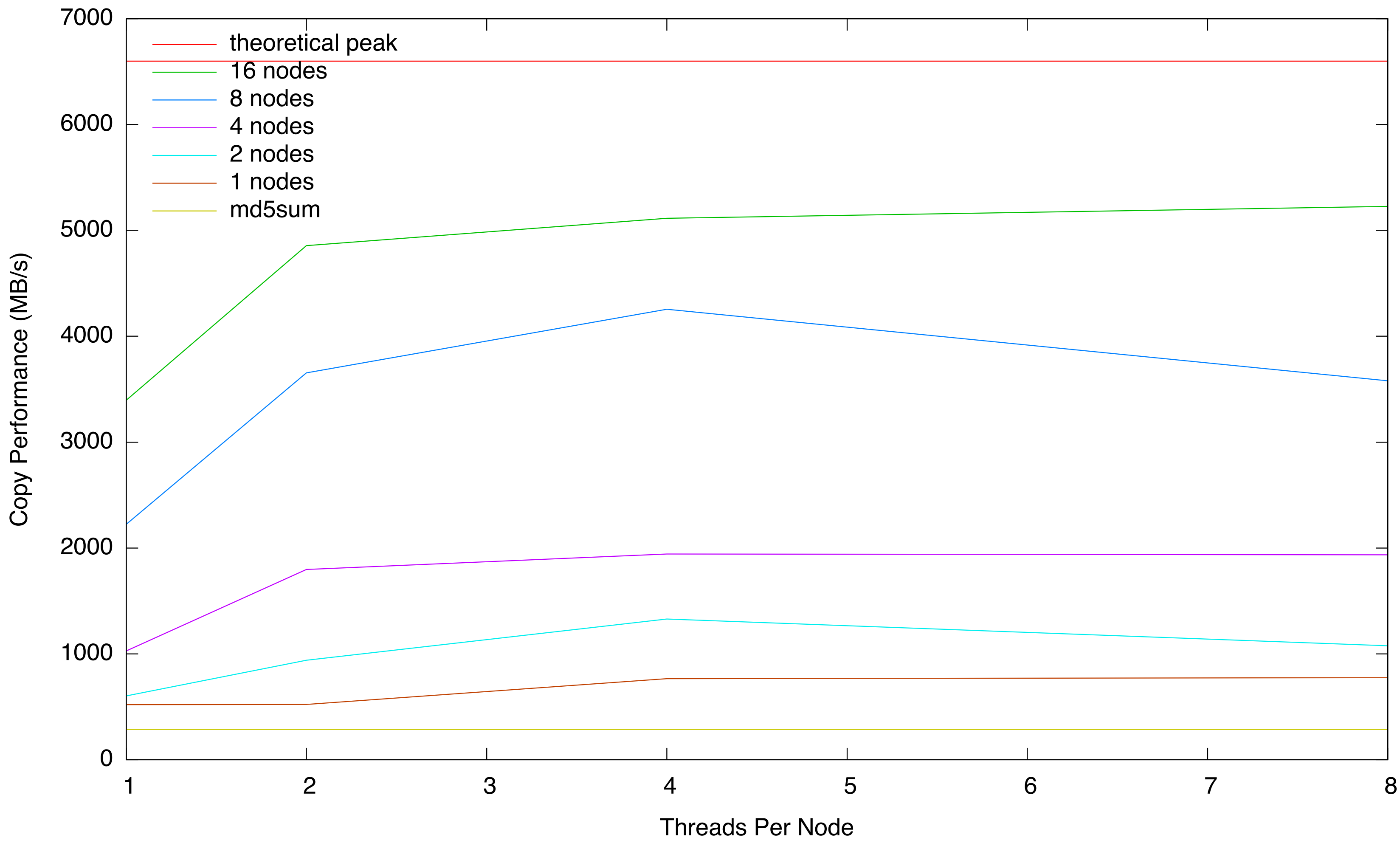
Another Utility: Msum

- Drop-in replacement for md5sum
  - Based on md5sum code from GNU coreutils
- Supports multiple hash types
- Supports all the performance enhancements of mcp
  - Multi-threading, split processing, buffer cache management, double buffering
    - Details and performance in paper
  - Multi-node support via TCP/MPI
- Works mostly the same as mcp but instead of copy tasks, there are sum tasks
  - Worker threads compute hash subtrees they are responsible for
  - Subtree roots sent to stat/hash thread on main node
  - Stat/hash thread computes remaining root of tree once all subtrees received

Multi-Node Checksum Performance (64x1 GB w/ posix_fadvise())



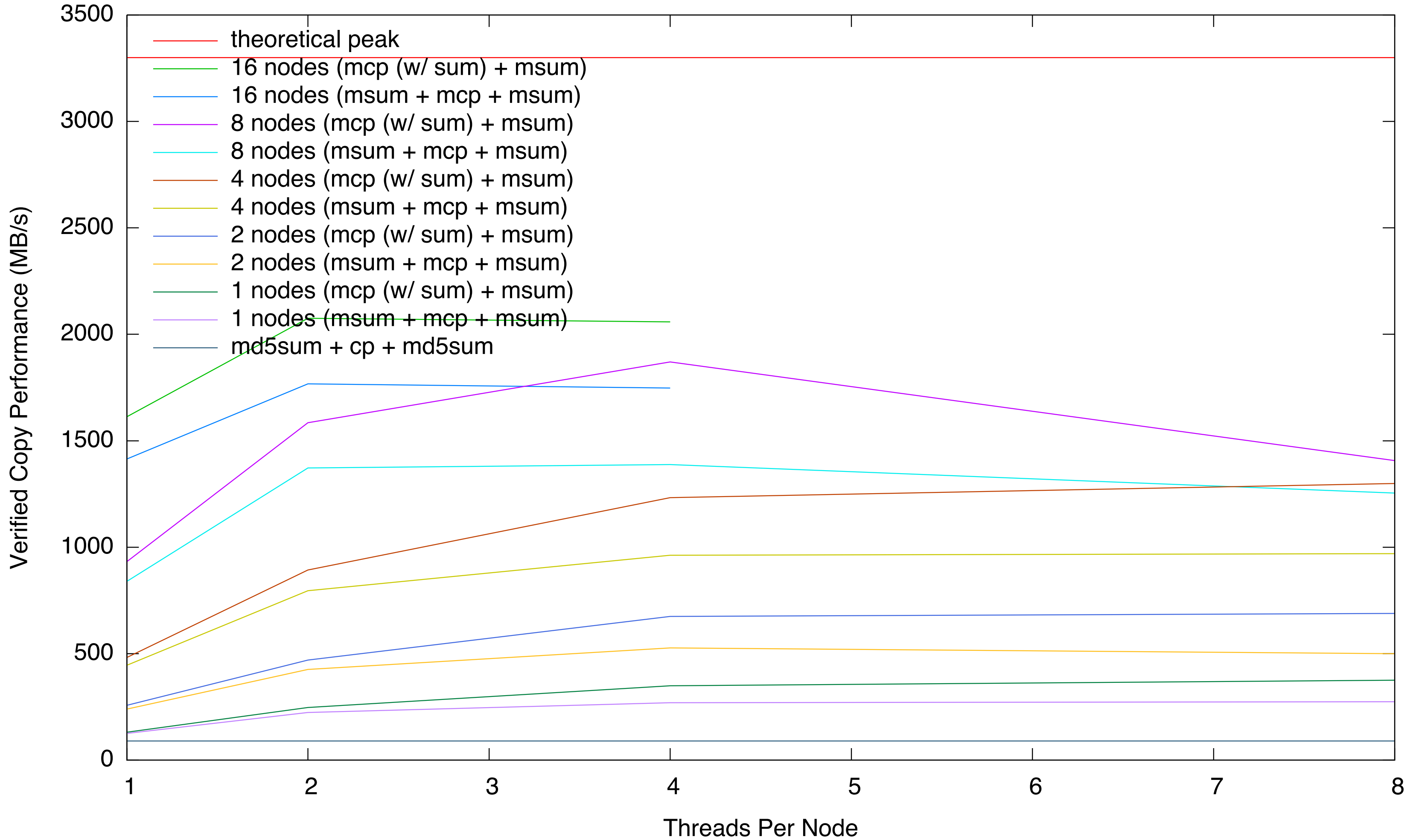
Multi-Node Checksum Performance (1x128 GB w/ direct I/O)



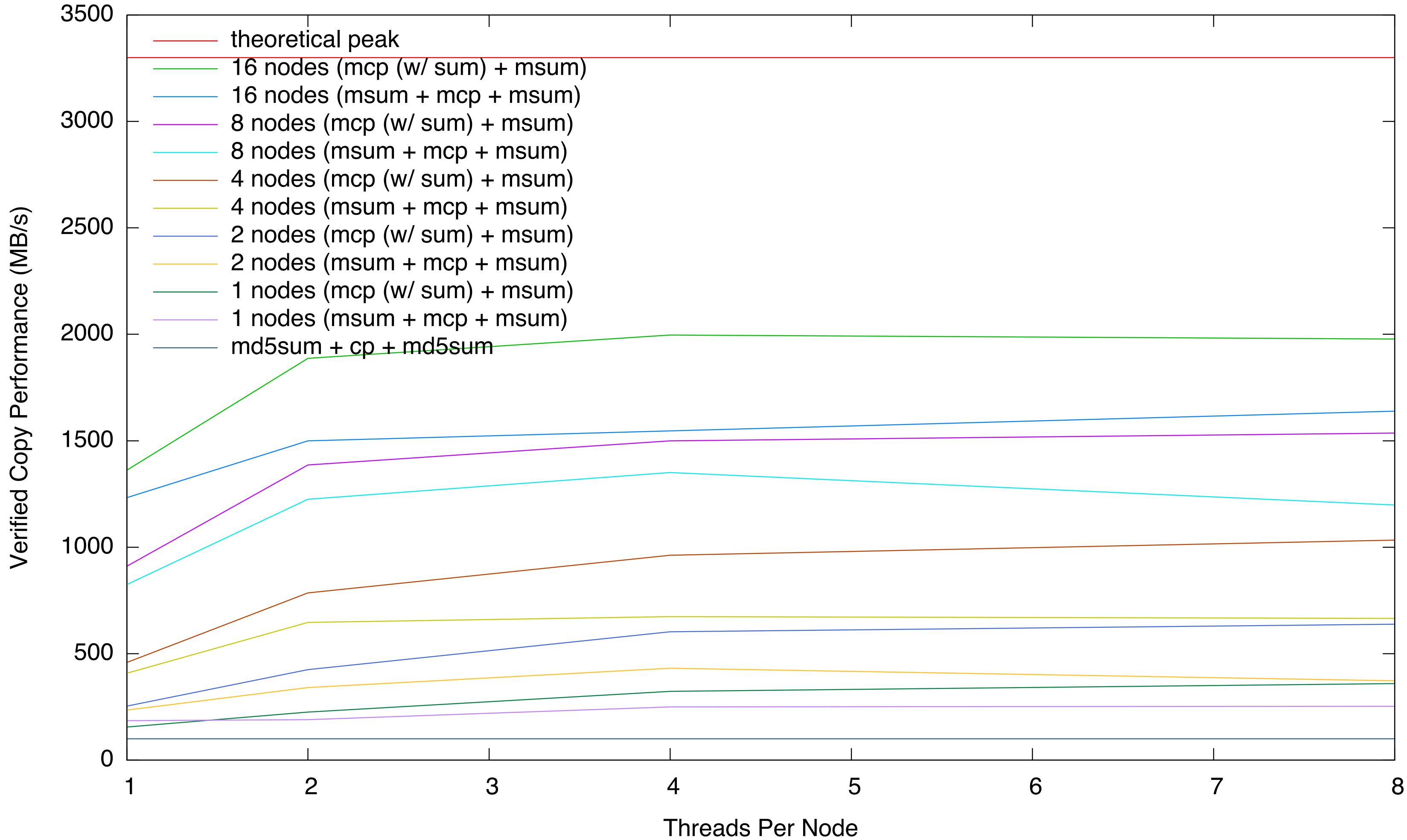
Integrity-Verified Copies

- Cost of verified copies
  - msum + mcp + msum = 3 reads + 1 write
  - Theoretical peak: 2.2 GB/s
- Mcp already has access to the source data during the copy
- Mcp includes embedded hashing functionality
  - Worker threads compute hash subtrees with data read for copy
  - Subtree roots sent to stat/hash thread on main node
  - Stat/hash thread computes remaining root of tree once all subtrees received
- Final cost of verified copies
  - mcp (w/ sum) + msum = 2 reads + 1 write
  - Theoretical peak: 3.3 GB/s

Multi-Node Verified Copy Performance (64x1 GB w/ posix_fadvise())



Multi-Node Verified Copy Performance (1x128 GB w/ direct I/O)



Conclusion

- Mcp/msum provide significant performance improvements over cp/md5sum
  - Multi-threaded parallelism to maximize single system performance
    - Buffer cache management to eliminate kernel bottlenecks
    - Double buffering to overlap reads/writes/hashes
    - Split processing to achieve single file parallelism
  - Multi-node parallelism to overcome single system resource limitations
  - Hash trees to achieve checksum parallelism

Conclusion (cont.)

- Summary of performance improvements
  - cp
    - 10x/27x on 1/16 nodes
    - 72% of peak
  - md5sum
    - 5x/19x on 1/16 nodes
    - 88% of peak
  - md5sum + cp + md5sum
    - 7x/22x on 1/16 nodes
    - 66% of peak
- Mcp and msum are drop-in replacements for cp and md5sum

Future Work

- Find bottleneck in single node single file case
- Parallelize other utilities
  - install, mv, rm, cmp
- Extend mcp to high performance remote transfer utility
  - Most of required infrastructure already exists
  - Need network bridge between read buffer and write buffer

Finally...

- Mcp and msum are open source and available for download
  - http://mutil.sourceforge.net
- Contact info
  - paul.kolano@nasa.gov
- Questions?