Rapid Replication of Multi-Petabyte File Systems

Justin G. Sybrandt
Grove City College, NERSC
SybrandtJG1@gcc.edu

Jason Hick
NERSC
JHick@lbl.gov

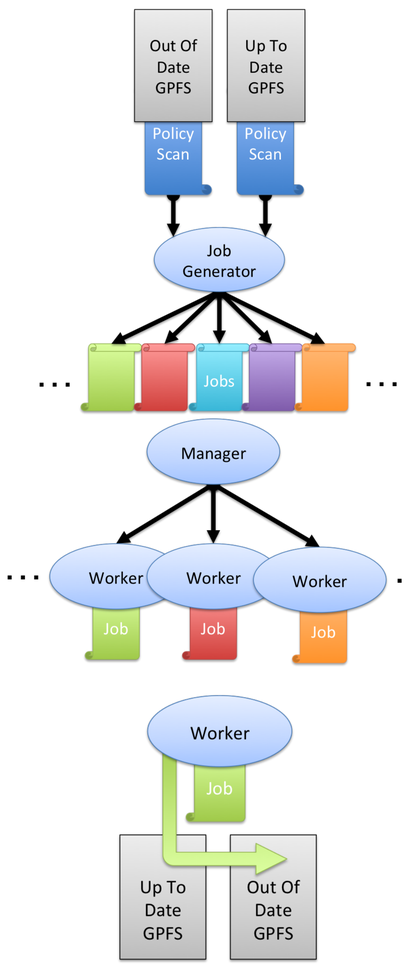
Introduction

- NERSC houses supercomputers (like Edison) which cater to scientific users.
- Users store millions of files and petabytes of data on General Parallel File Systems (GPFS) [3].
- NERSC is moving to a new location and needs to replicate their GPFSs as quickly as possible.

- Current tools are too slow for use at NERSC's scale.
  - Current replication methods can take over a week.
  - Most methods process files that are the same on both systems.
- Distsync is a new tool which can help replicate GPFSs at NERSC.
  - It uses metadata scans to quickly find changes between file systems.
  - It distributes the work across many nodes.
  - It can be used elsewhere to freshen large backups.

Architecture

Distsync is comprised of three parts, the Job Generator, the Manager, and the Worker.



Every file and directory in the GPFS is found using a policy scan.
This is the fastest way to get a list of all files.
Scans include file attributes, sorted by path.

Scans are compared to create job files.
Job files describe changes that have been made to the system.
Most job files are limited to 10,000 records and 1TB.

Types of jobs:
[Create | Remove | Modify] [Directories | Files]
Hardlinked files are handled separately.

The manager assigns job files to workers.
Job scheduling ensures that operations occur in the right order.
i.e. Directories need to exist before they can be filled with files.

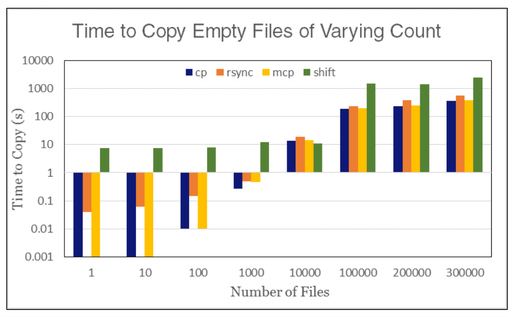
Each connected node eventually receives a job.
The worker invokes tools to act on each path in the job file.
Each worker runs simultaneously to update the stale copy.

See the code on **GitHub**

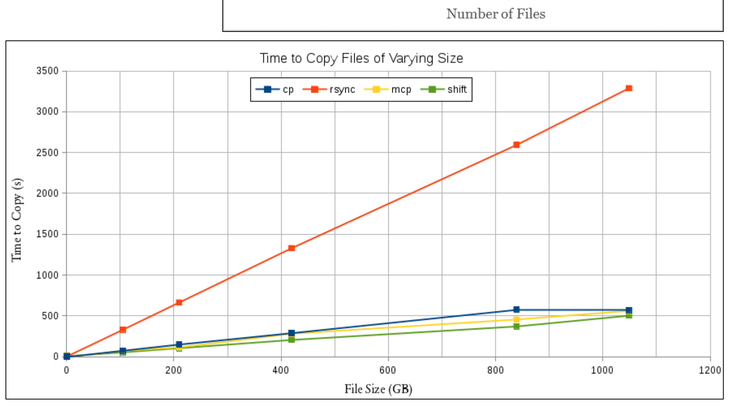
Tool Evaluation

Tests ran to determine which tools were best for workers to use.
Four tools were evaluated: cp, rsync, mcp [2], and shift [1].

- This experiment determined how long it takes each tool to copy large numbers of files, without regard to file size.
- The GPFS variable MaxStatCache was set to 50,000 which explains the jump between 10,000 and 100,000 files.



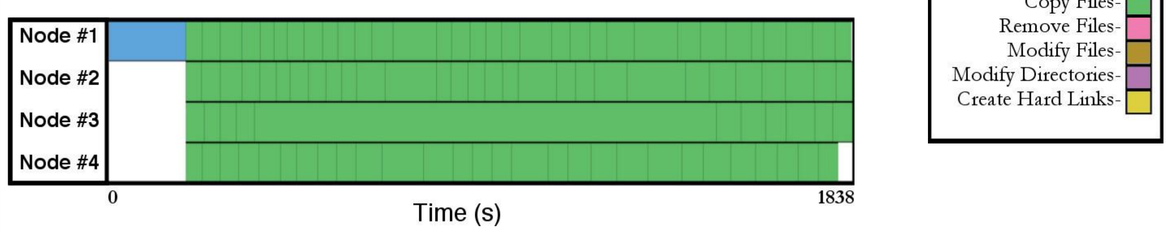
- Each tool was also timed while copying single files of varying sizes.
- Each file contained random characters, and the entire file needed to be copied.
- Default settings were used for every tool. Shift was limited to one machine.



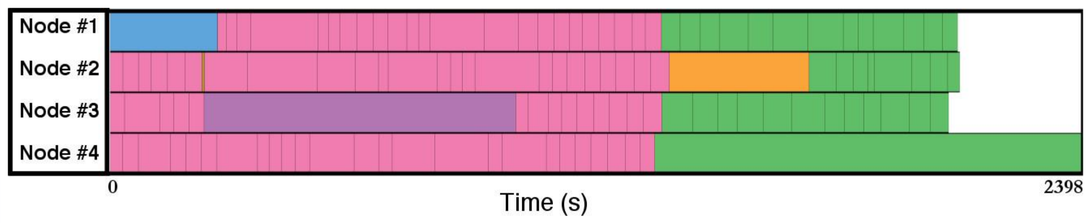
Job Type	Tool	Comments
Creating Directories	mkdir + chown + chmod	Single threaded to ensure order is followed.
Removing Directories	rm -r	Single threaded, always removing children before parents.
Copying Files	cp -pP	Many copies run in parallel. Each instance guaranteed to use only a single thread. Mcp froze in some cases, cp was more reliable.
Removing Files	rm	Many copies run in parallel.
Modifying Files	rsync - aSHAXd	Faster than *shiftc --sync-fast* in the case of small files. Many copies are run in parallel.
Modifying Directories	chown + chmod	Many copies are run in parallel.
Copying Hard Links	rsync -aSHAX	A single instance of rsync is given the entire job file. This is the only way to preserve hardlinks.

Results

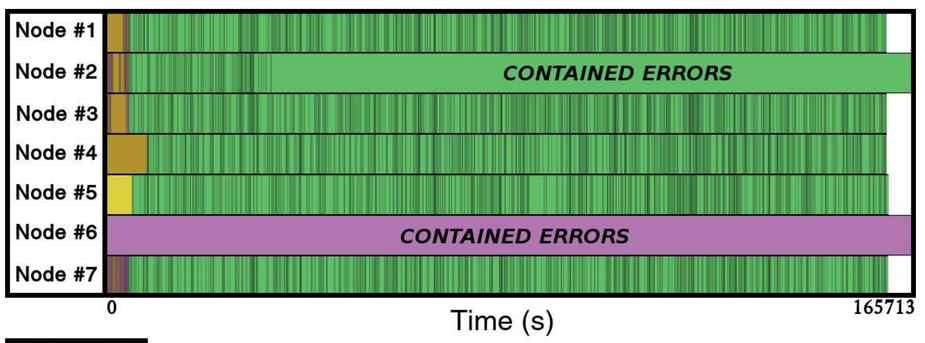
- Distsync used to copy a file system containing approximately one million files.
  - Work was split into over 100 job files.
  - Note time spent waiting for directories to be built.
  - Shift copied the same data in 6,721 seconds.

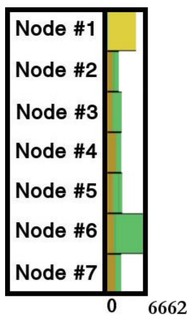


- Distsync used to synchronize a file system containing approximately two million changes.
  - Work was split into over 120 job files.
  - Note the long tail which copied 100GB all on its own.



- Distsync used to synchronize a production file system at NERSC.
  - Copied almost 350 TB.
  - Contained 3,718 job files.
    A user placed devices in his temporary directory, which was not handled properly by the workers.
    Since this run, that case has been checked for.





- Three days after the original sync, the file systems were re-synchronized.
  - The results are to scale.

Read the **Paper**

Conclusions

- The configuration of the GPFS cluster makes dramatic differences in distsync's performance.
  - Replication occurs faster when all working nodes are also GPFS managers.
  - Stat cache should be large enough so all workers can all stat files.
- Automatic configuration could be added.
  - Distsync could determine which machines have the best access to a file system.
  - Could automatically set variables based on GPFS configuration.
- Distsync is more straightforward that existing methods.
  - Before this tool, NERSC used shift, rsync, and various scripts to ensure all changes were propagated.
- Distsync scales well.
  - Scheduling diagrams suggest that long tails are a potential issue, but seem less problematic than expected.
  - Worst case is that frozen tools lead to stalled out workers.
  - It scales with regard to the number of modifications, not the number of files.
- Distsync is currently being evaluated at NERSC.
  - It is being run against larger, multi-petabyte file systems.

Hear the **Talk**

References

[1] P. Z. Kolano, "High performance reliable file transfers using automatic many-to-many parallelization," in 5th Wk- shp. on Resiliency in High Performance Computing (editor, ed.).
[2] P. Z. Kolano and R. B. Ciotti, "High performance multi- node file copies and checksums for clustered file systems," in Proc. if the 24th USENIX Large Installation System Administration Conf.
[3] F. Schmuck and R. Haskin, "Gpfs: A shared-disk file system for large computing clusters," in FAST '02 Proceedings of the 1st USENIX Conference on File and Storage Technologies (editor, ed.), no. 19, USENIX Association Berkeley, CA, USA.
[4] M. Andrews, J. Hick, J. Emmons, and K. Powell, "Parallel graph reduce algorithm for scalable file system structure determination.".

Acknowledgments

This work was supported in part by TRUST, Team for Research in Ubiquitous Secure Technology, which receives support from the National Science Foundation (NSF award number 1157075). I would also like to thank my advisor, Jason Hick, as well as the Gregory Butler and Ravinderjeet Cheema for enabling me on this project. I would like to thank the University of California Berkeley for hosting me during this project. Lastly I would like to thank the professors at Grove City College for giving me such a strong foundation in Computer Science and encouraging me to pursue research.