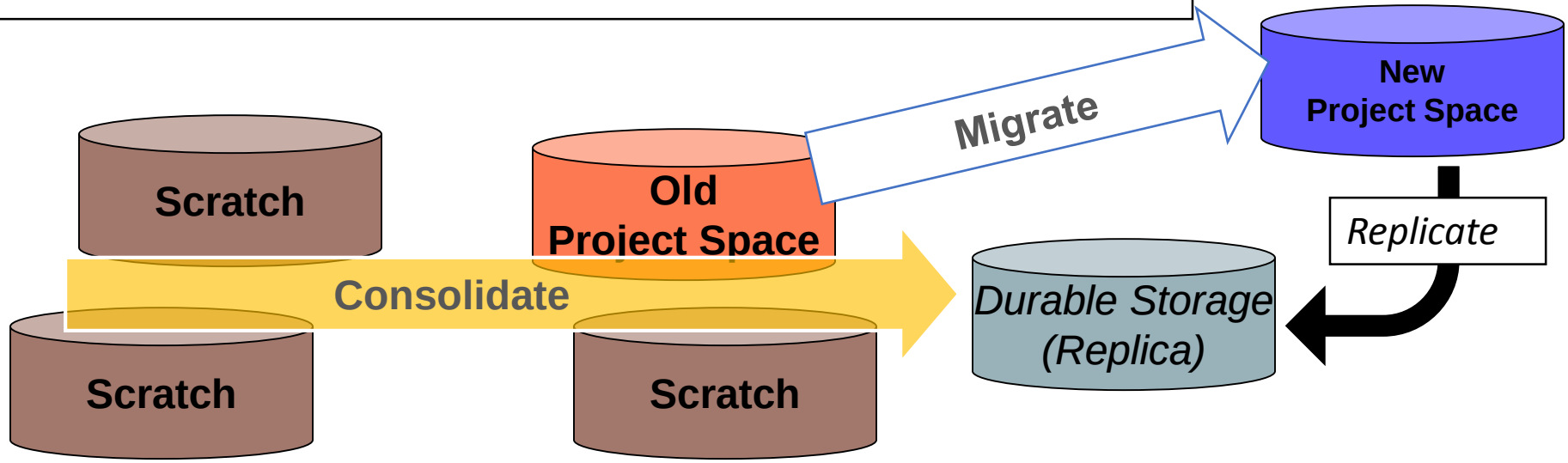
# Lustre Data Mover

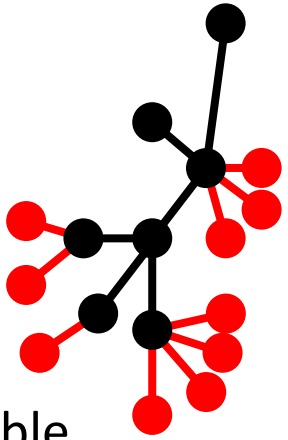
Dmitry Mishin Rick Wagner

# Motivation (From LUG 2015)

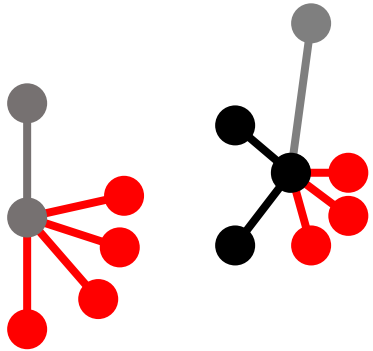
- Consolidating Data Oasis Gen I servers into Durable Storage (local disaster recovery replica)
- Preserve data when migrating Project storage to new Gen II servers
- You can't `rsync` a petabyte



# Goals, Drives, Solution



- #1 Go Review Marc Stearman's LUG 2013 presentation Sequoia Data Migration Experiences
  - TL;DR You can't `rsync` a petabyte
- Assume file system structure (files and directories) is unknowable
  - Alternative is to presort: Why not just copy during sort?
- Need a scalable (preferably dynamically) tool
- Our solution
  - **Recursive worker service**
- Treat file systems as rooted directed acyclic graphs
- Have each worker handle a directory or group of files



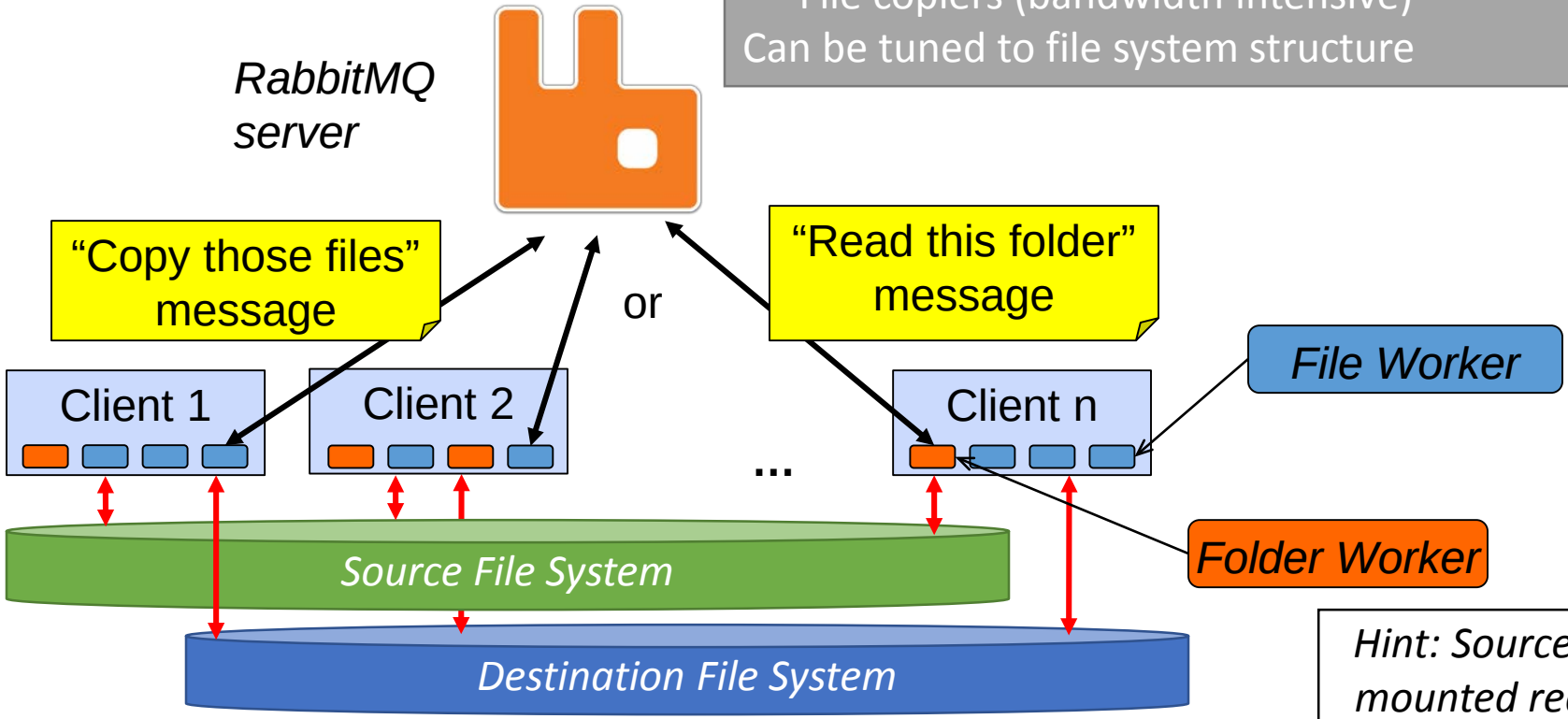
# Building Blocks

- Some Lustre clients
  - We use compute nodes for big transfers
  - Dedicate pool of 4 lightweight nodes for continuous replication
- A RabbitMQ (AMQP) server
- Celery (Python) scripts
  - Worker service
- Bash scripts to start, stop, pause workers

# Architecture

From experience, best to have 2 types of workers:
- Folder scanners (metadata intensive)
- File copiers (bandwidth intensive)

Can be tuned to file system structure

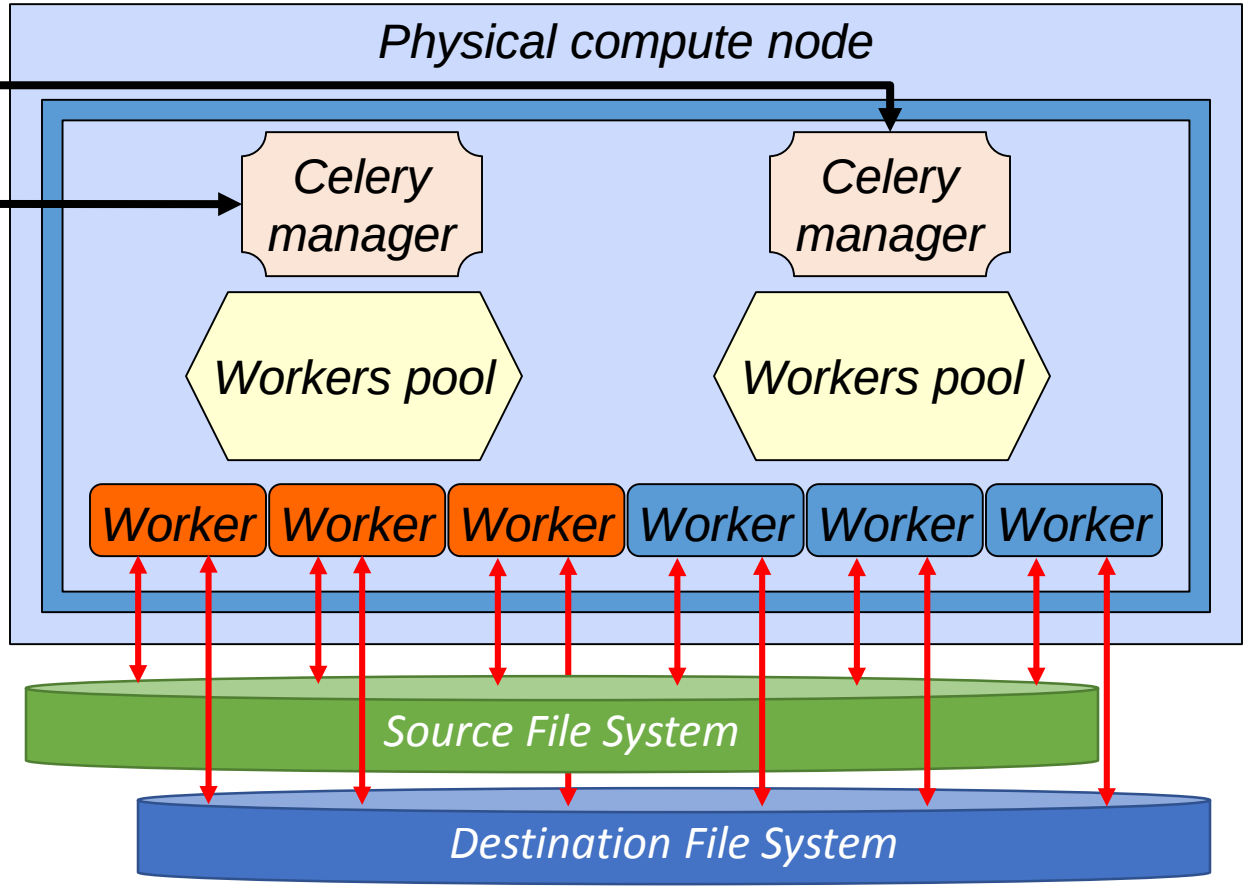



*Hint: Source can be mounted read-only*

# Celery Workers

Files queue

Folders queue

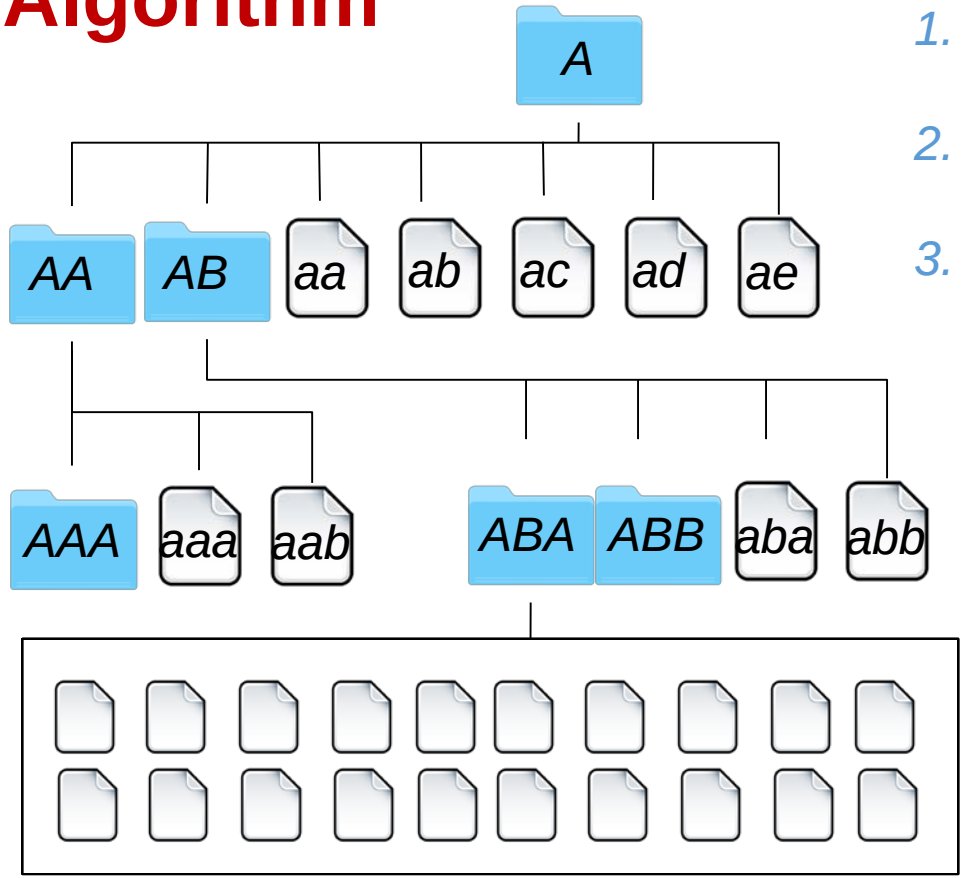



Workers pool base on billiard - fork of the Python 2.7 'multiprocessing' package

# Basic Replication Process

- 2 Phases:
  - Background (copy data from live source to destination)
  - Cut over (take a downtime and quiesce the file systems)
- Each phase has 2 or 3 passes
  - **Copy** files created after lass copy pass (trivially first pass copies all files)
  - **Delete** (remove destination files & folders not on source)
  - **Fix** `mtime` (match destination folder `mtime` to source)
- Background phase never fully consistent
  - Data copied will get get modified, moved, deleted
  - Can get pretty close with repeated passes
- During downtime run all 3 passes to fully sync

# Algorithm




**Scan A**

1. A exists? Create
2. Scan files
   cp aa - ae
3. Scan dirs
   ls AA
   ls AB

**Scan AA**

1. AA exists? Create
2. Scan files
   cp aaa, aab
3. Scan dirs
   ls AAA

**Scan AB**

1. AB exists? Create
2. Scan files
   cp aba, abb
3. Scan dirs
   ls ABA
   ls ABB

**Scan ABA**

1. ABA exists? Create
2. Scan files
   cp abaa-abae
   cp abaf-abam
   cp aban-abas
   cp abat-abay
3. Scan dirs

# 1st Pass (Copy): `ls <dir>`

- Does the folder exist on destination? Not the mount point?
  - Create if needed
    - If first level, optionally put on one of 2 MDS for balancing those
      - `lfs setdirstripe -i`
    - `chmod, chown, setstripe`
- List folder contents
  - `lfs find <dir> -maxdepth 1 ! -type d`
    - For each ≦1000 files submit message: “copy <list of files>”
  - `lfs find <dir> -maxdepth 1 -type d`
    - For each subdir submit message: “scan <subdir>”

# 1st Pass (Copy): cp <files list>

- Check file `atime`, `mtime`, skip if needed
- If regular file
  - If destination exists
    - If destination's size || mtime || uid || gid || mode is different
      - Delete destination
    - Else skip – no need to copy
  - Create file with 'setstripe' with source file stripe value
  - Copy file contents
- If symlink
  - Create symlink

# 2nd Pass (Delete): ls <destination dir>

- Does the folder exist on source?
  - If not, recursively delete destination folder
- Else: List destination folder contents
  - `lfs find <dir> -maxdepth 1 ! -type d`
    - For each ≦1000 files submit message: “check <list of files>”
  - `lfs find <dir> -maxdepth 1 -type d`
    - For each subdir submit message: “scan <subdir>”

# 2nd Pass (Delete): Check <files list>

- If source does not exist (! lexists – don't follow symlinks)
  - Delete destination
- Else skip

# 3rd Pass (Fix Folders `mtime`)

- Change `mtime` of destination folder to one of source
- Scan for folders inside of current one
  - `lfs find <dir> -maxdepth 1 -type d`
    - For each subdir submit message: “scan <subdir>”

# Additional Parameter

- *Exceptions list* – match files and folders by beginning of string, skip if matches
- Skip files with `atime` older than N - unused
- Skip files with `mtime` newer than N – likely to change soon
- If “permission denied” on folder read
  - Root squashed - resend message to special queue, process on non root squashed worker
- Report statistics each N minutes

# Data Moving Performance

- During massive replication (~1 PB to move)
- Peaks to 15 GB/s on 24 nodes
- Submits messages to RabbitMQ – up to 8K/sec (in-memory queue)

# Performance: Copy Pass

4 nodes, single 10GbE
Peaks to

- 1.5GB/s
- 2K files/s,
- 450 folders/s

~3 days for 1 PB file system

Backup process, only copying changed data

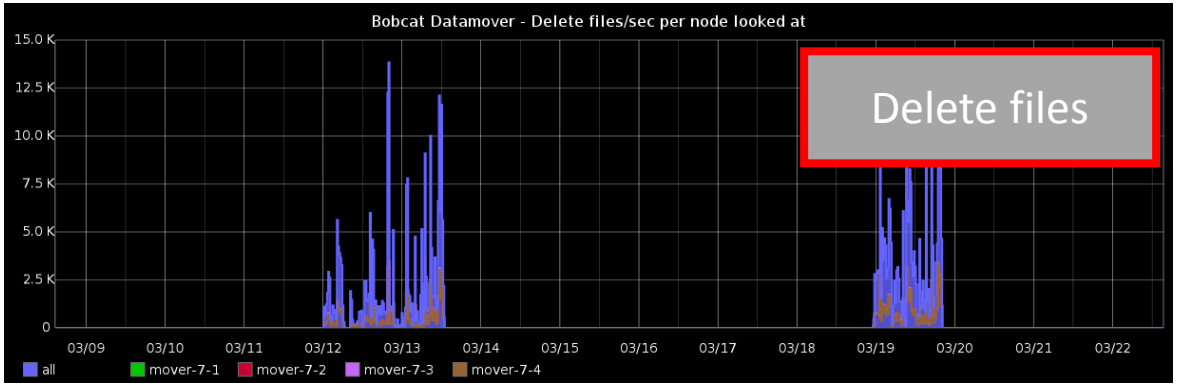
# Performance: Delete Pass

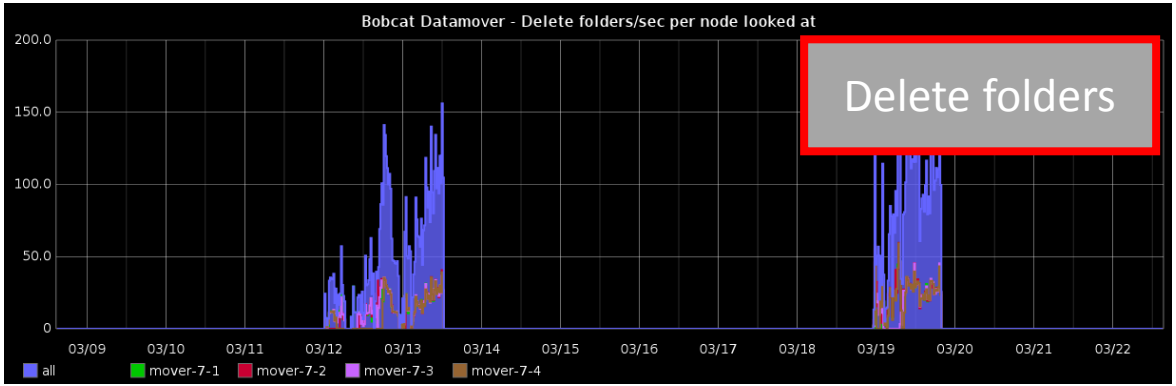
4 nodes, single 10GbE
Peaks to

- 13K files/s,
- 150 folders/s

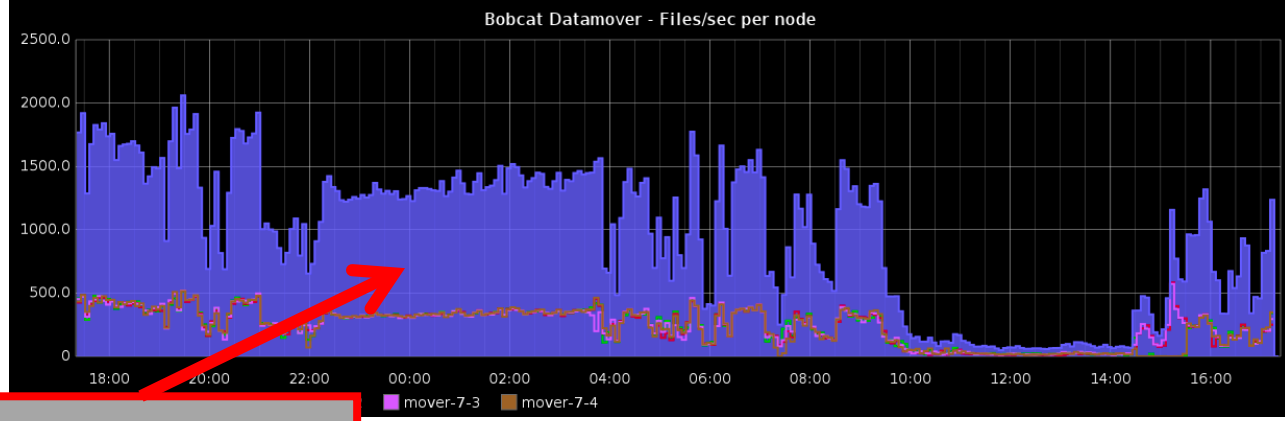
~1 day for 1 PB file system

Backup process, only copying changed data

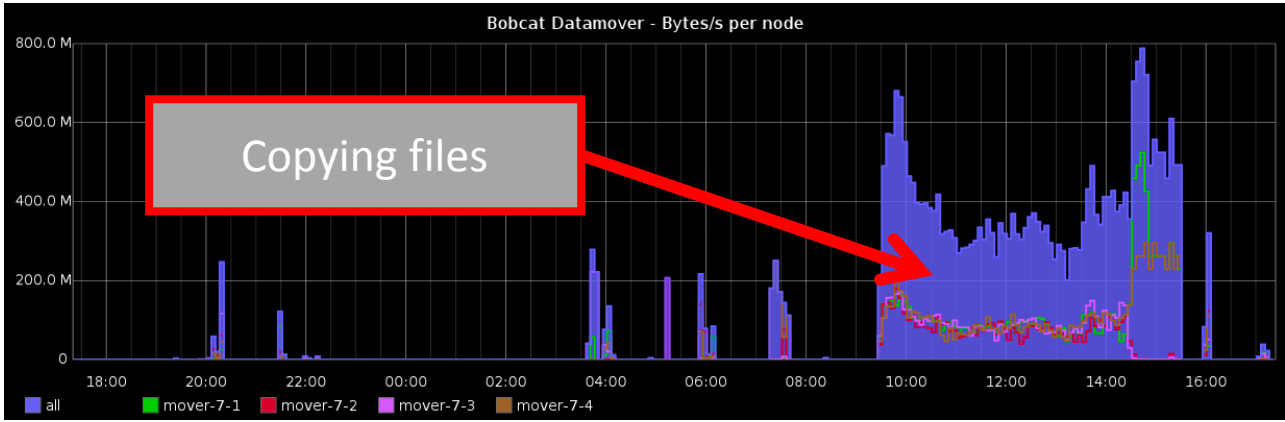


# Performance: Metadata vs. Bandwidth




Hard to balance scanning directories and files with moving data



# Possibilities, Ideas, The Future

- 4 years of production ahead
  - We're motivated to keep it up-to-date
- File system interactions (list, copy, delete, etc.) are generic
  - Could be swapped out
  - Object storage?
  - Pretty much a scalable dumb copy tool
- WAN
  - Replace cp with bbFTP, Netcat, HPN-SCP, whatever
- Checksums

# Test & Contribute!

**https://github.com/sdsc/lustre-data-mover**

- BSD License
- Needs the usual:
  - Testing
  - Documentation
  - A better name
- Please break it! We'll fix it!