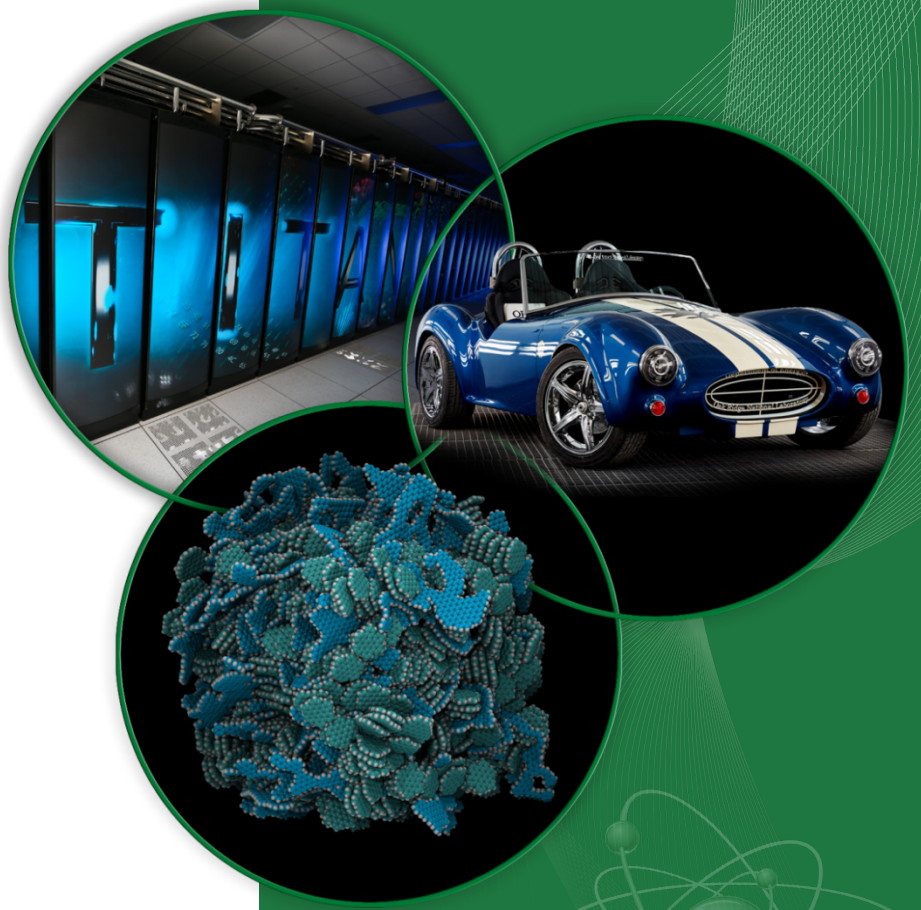
FCP: A Fast and Scalable Data Copy Tool for High Performance Parallel File Systems



Feiyi Wang (Ph.D.)

Veronica Vergara Larrea

Dustin Leverman

Sarp Oral

ORNL is managed by UT-Battelle
for the US Department of Energy

OAK RIDGE National Laboratory

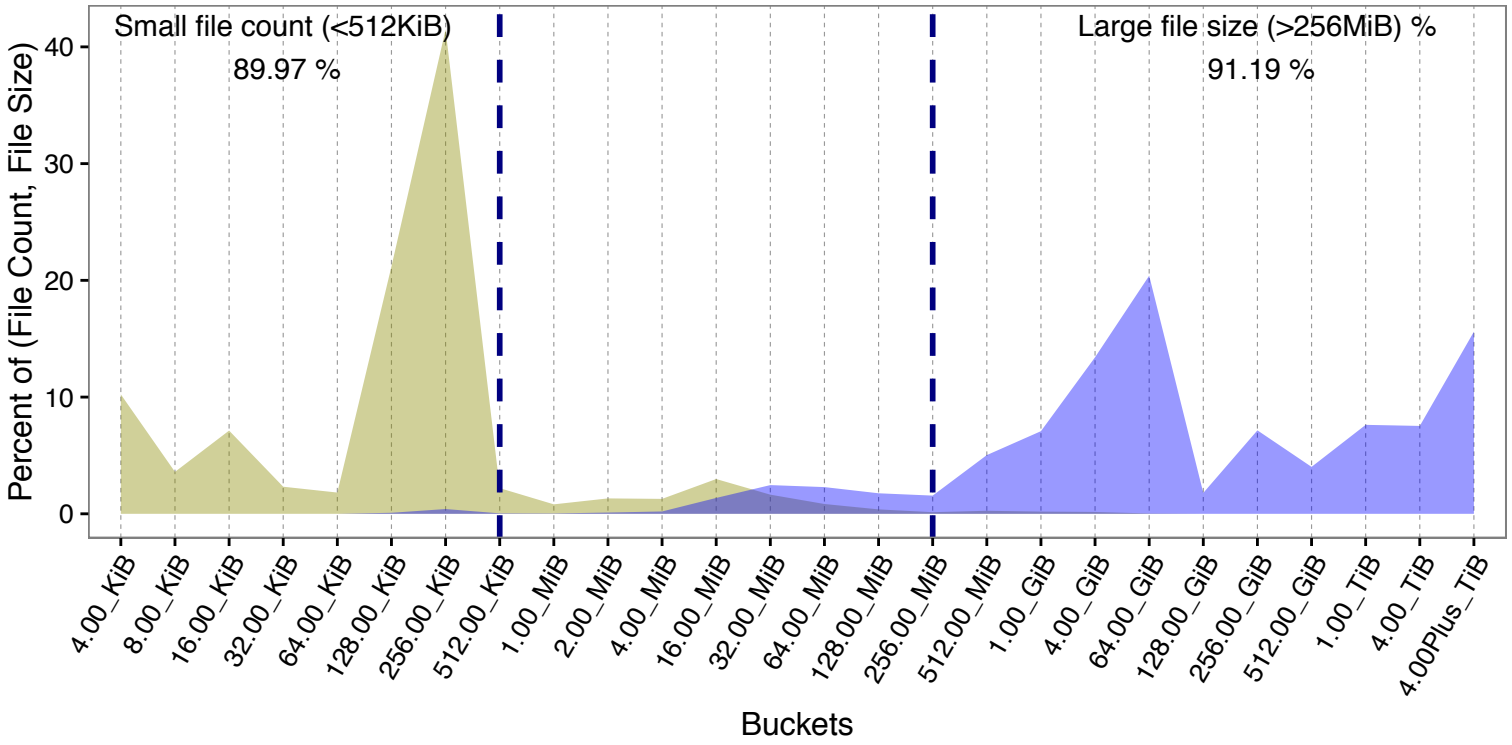
The Challenge

- Conceptually simple, everyone knows.
- Balance usability, scalability, and performance
  - Workload parallelization
  - Asynchronous Progress Report
  - Chunking
  - Checksumming
  - Checkpoint & Restart
  - Out of Core

What Tools Do We Have?

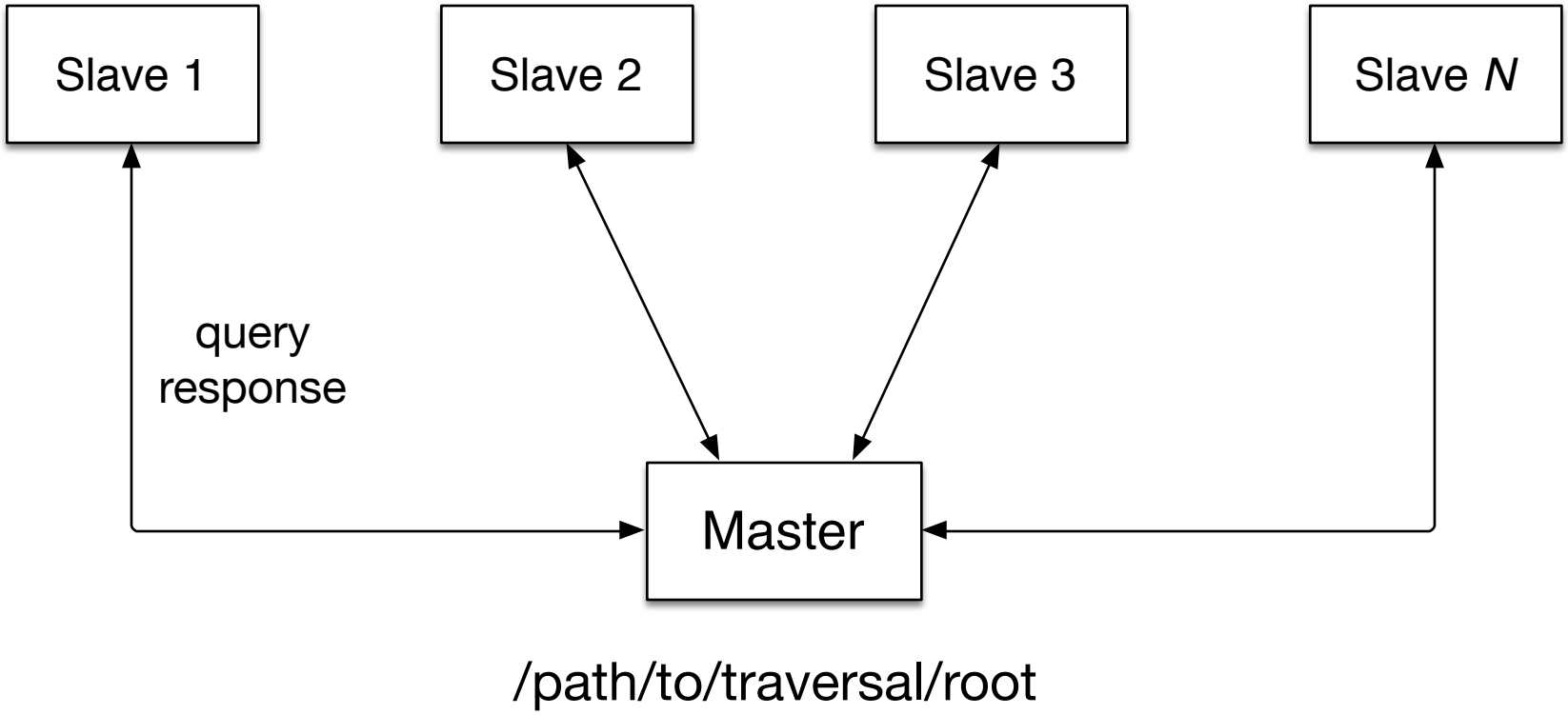
- cp
- bbcp
- grid-ftp
- dcp
- customized rsync (manual or automatic, with GNU parallel utility)

Large Scale Profiling



~ 750 million files
~ 88 million directories
~ 3.4 million files in single directory (max)
~ tens of thousands of files in TB range , 32 TB (max)

Workload Parallelization: Master Slave



Problem: (1) centralized (2) unbalanced

Workload Parallelization: Work Stealing

Key Ideas

- Each worker maintains it own work queue
- After local work queue is processed, it picks a random worker, and asks for more work items.

Key attributes

- Initial load placement doesn't matter, self-correction.
- Slow worker(s) doesn't matter, self-pacing.

Question: Without a master process, how do we know when to terminate?

Distributed Termination Detection

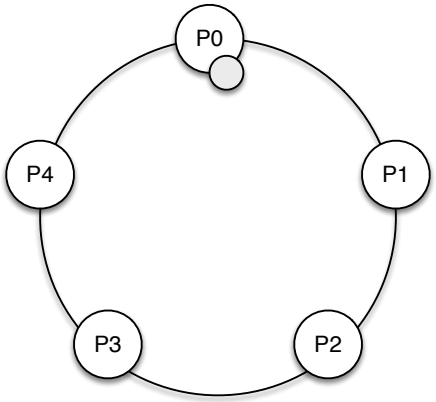
Edsger W. Dijkstra: *Derivation of a termination detection algorithm for distributed computations*. June 10, 1983

1. The system in consideration is composed of N machines, $n_0, n_1, \ldots, n_{N-1}$, logically ordered and arranged as a ring. Each machine can be either white or black. All machines are initially colored as white.
2. A **token** is passed around the ring. machine n's next stop is $n+1$. A token can be either white or black. Initially, machine n_0 is white and has the white token.
3. A machine only forwards the token when it is passive (no work)
4. Any time a machine sends work to a machine with lower rank, it colors itself as black.
5. Both initially at n_0, or upon receiving a token:
    1. if a machine is white, it sends the token unchanged.
    2. if a machine is black, it makes the token black, makes itself white, and forward the token.

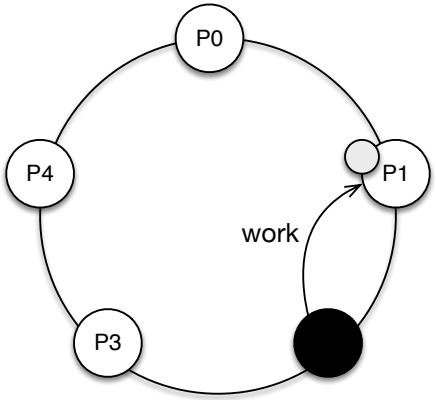
Termination condition: white n_0 receives a white token.

Understanding the Algorithm

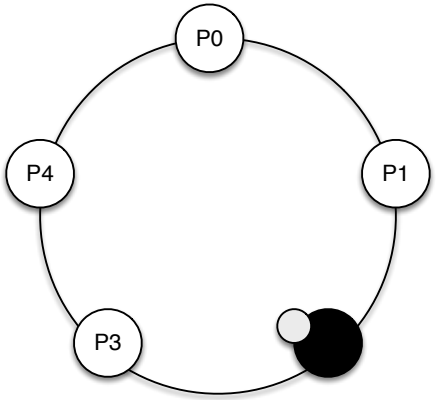
- Stable state is reached when all machines are passive.
- Edge case: a system is composed of one machine: it will send a white token to itself, thus it meets the termination condition, also it reaches the stable state.
- Even a machine becomes passive at time *t* and forward the token, it can become active again upon receiving works from others.
- When a black token returns to machine n_0 or a white token returns to a black machine n_0, a termination conition can not be met. The token forwarding continues.



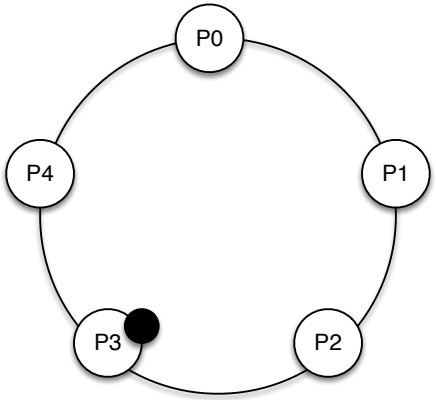
Initial State



Send work to j < i
turn black

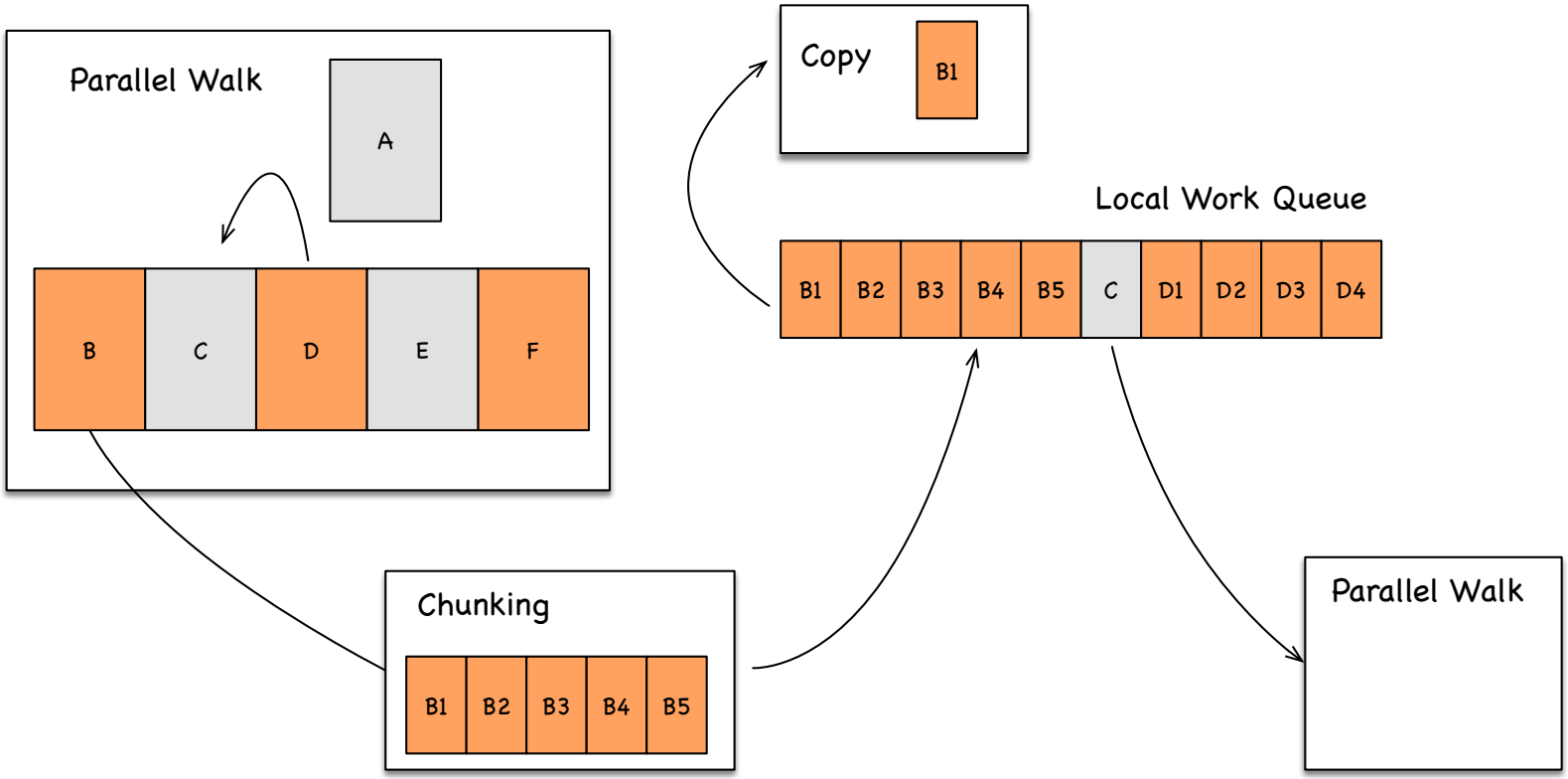


Color token black
if itself is black



P2 forwards the token,
color itself white

Workflow and Division of Labor: A Compromise



We have a problem of intermixing all three: parallel walk, chunk, and copy

Async Progress Report

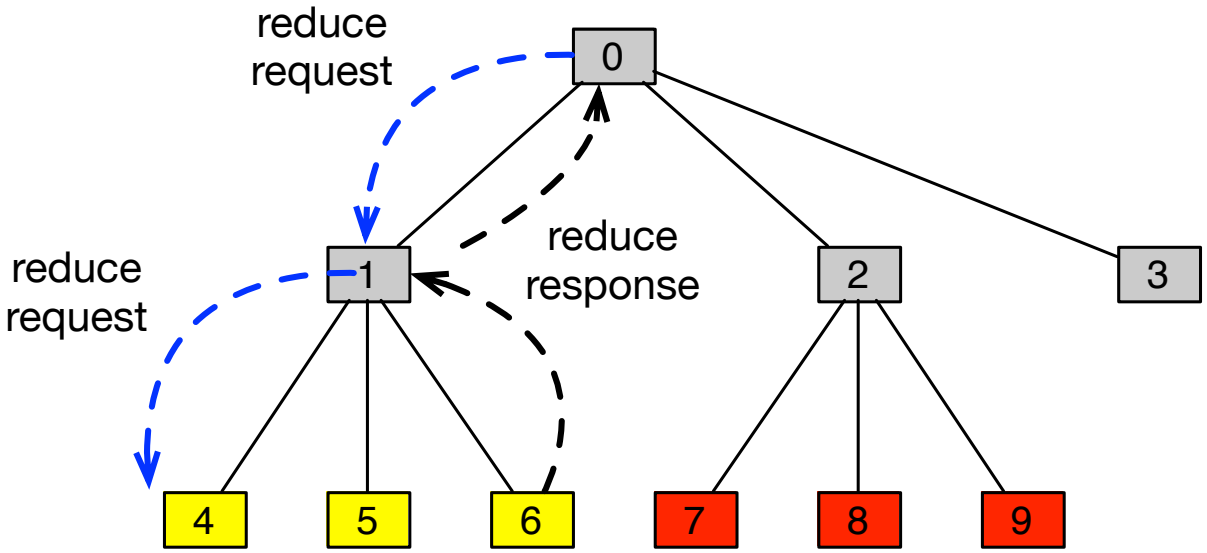
Users want to know the progress, in particular when doing a large data transfer.

Yet, this can be difficult in a fully distributed setup environment

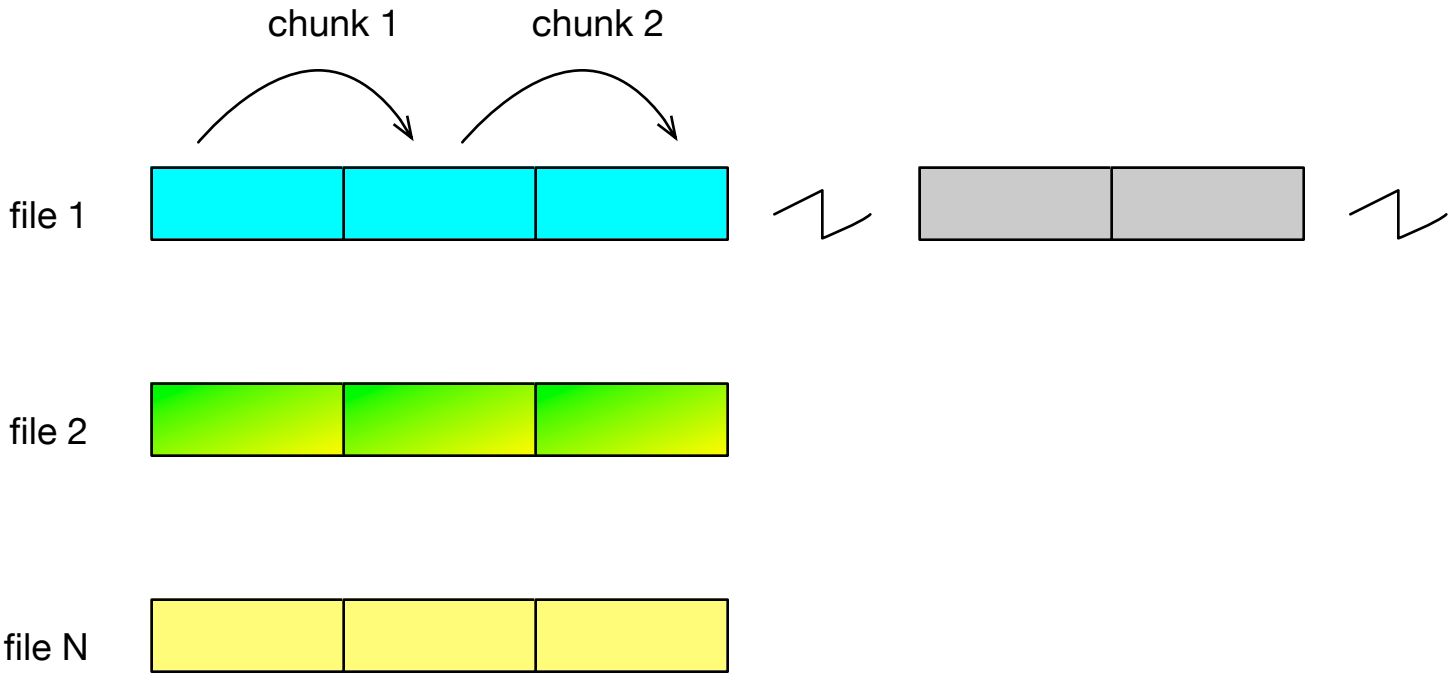
Solution

`reduce()` callback

- Hierarchical
- Async

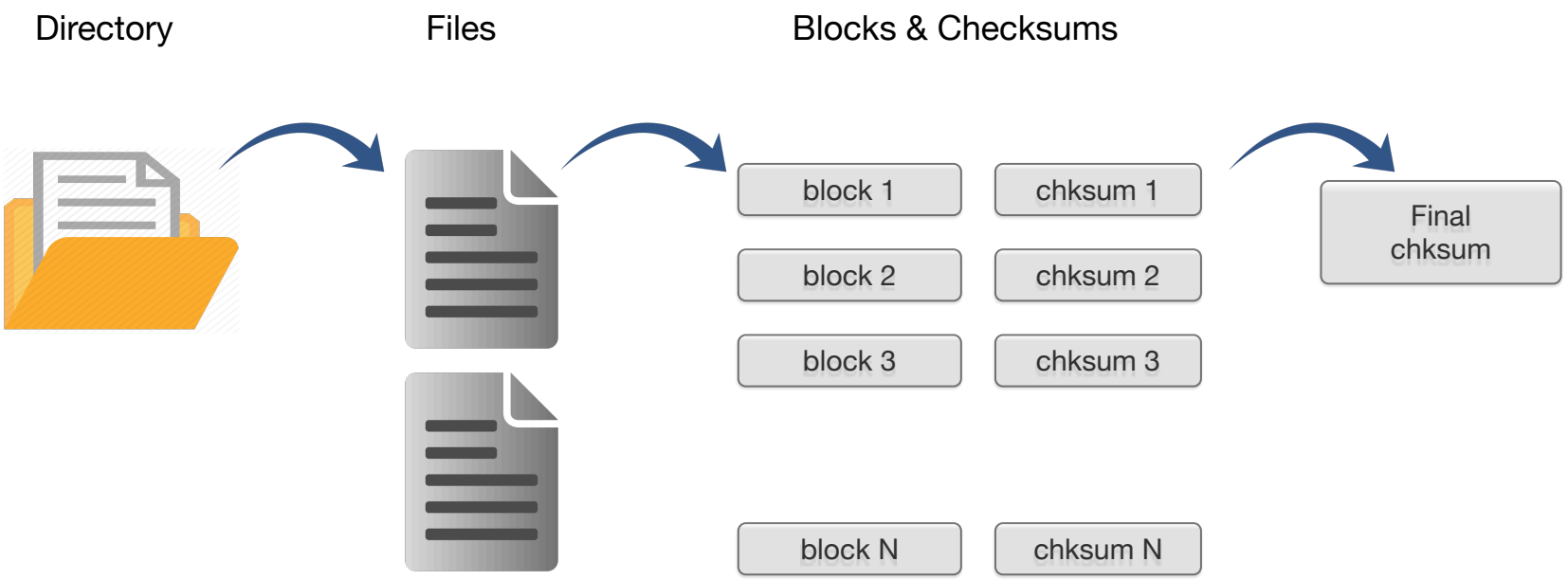


Adaptive Chunking



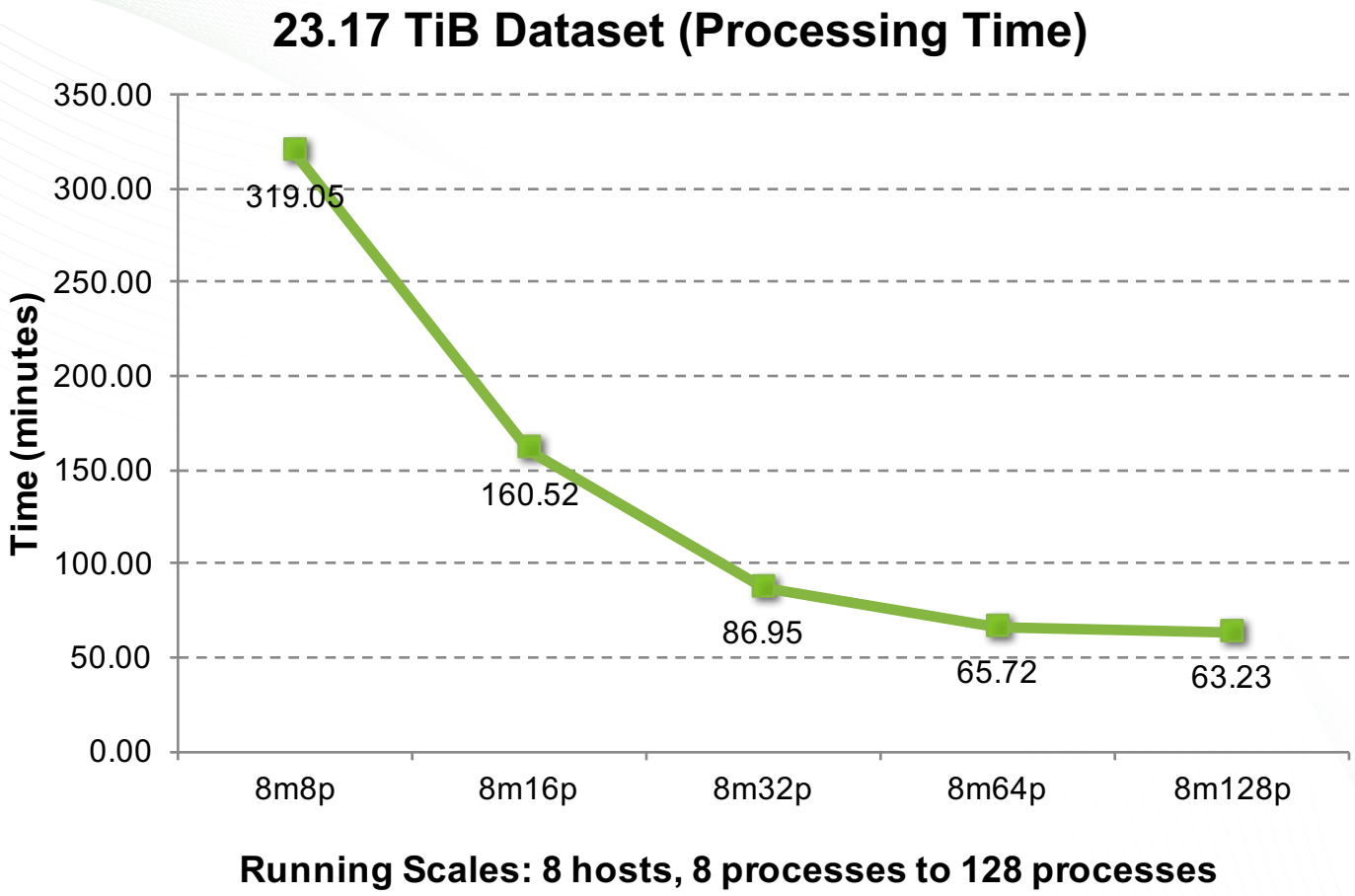
Large chunks – you may miss the opportunity for parallelization
Small chunks – you may have too much overhead

Parallel Checksumming

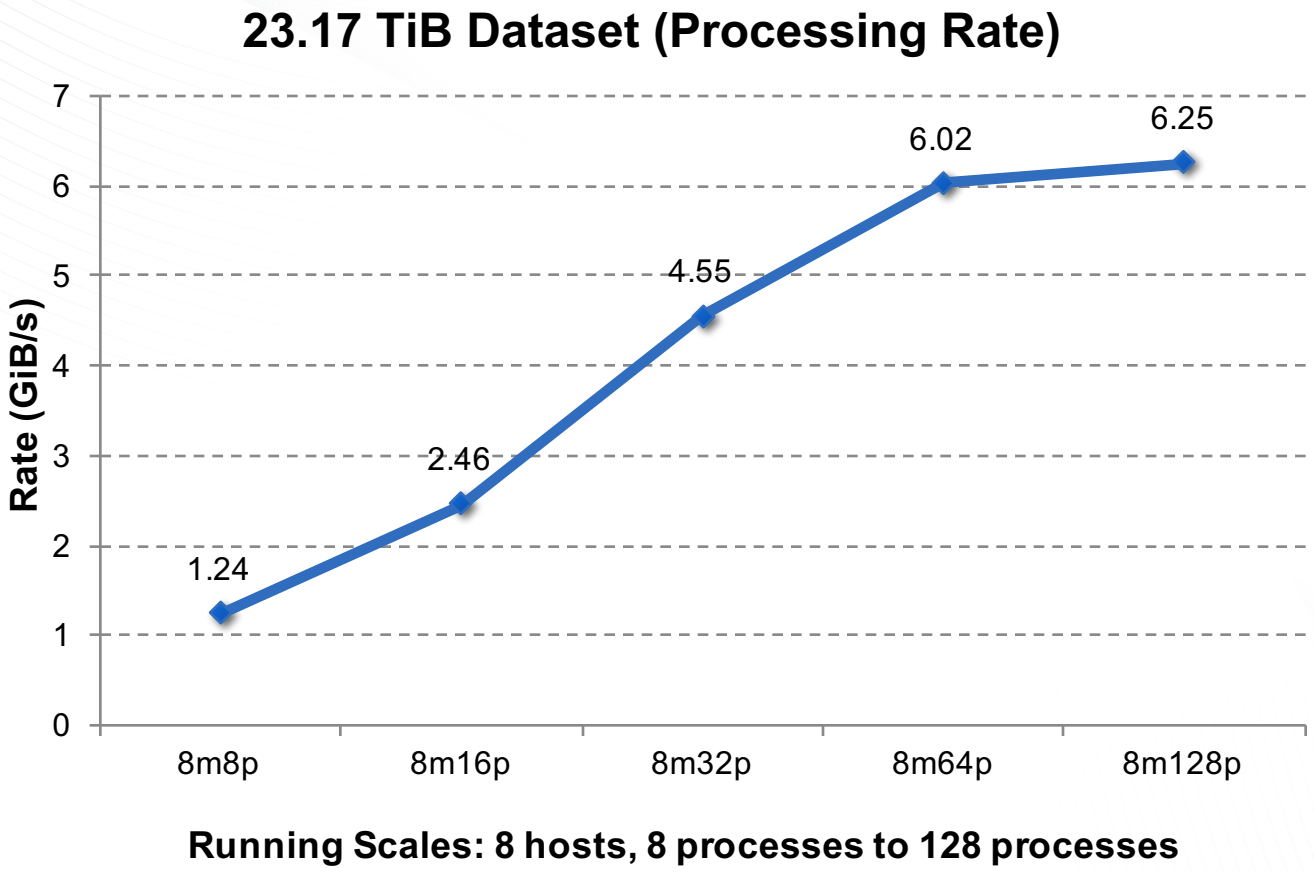


(1) Post copy verification
(2) On-the-fly dataset signature
(3) Post copy dataset signature
(4) Compare two datasets

Parallel Checksumming: Scaling



Parallel Checksumming: Scaling



Checkpoint and Restart

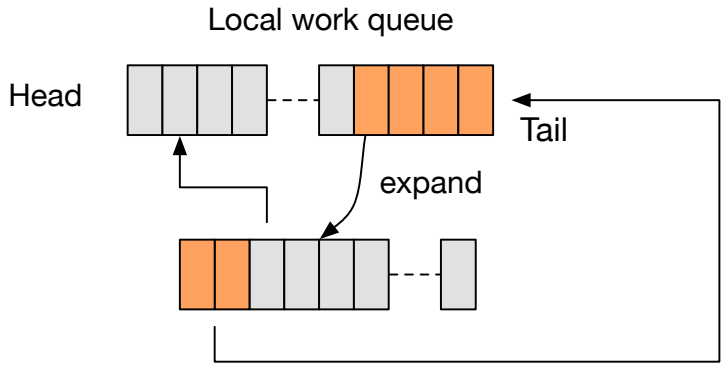
- For serial application, resume would have been trivial.
- For FCP, it is conceptually simple:
  - Write checkpoint file periodically
  - Read it back and resume the work
- Devil is in the details:
  - Fail when doing parallel walk
  - Fail before first checkpoint
  - Fail during the writing of the checkpoint
  - Fail when workload (stealing) message is in flight
  - Should each rank have its own checkpoint?
  - What if next time users launch differently (e.g. # of mpi)?
  - After restart, what bookkeeping needed to show correct progress?

Extreme Scalability: Out-Of-Core

Each Metadata object = 112 bytes (File system)
Each Chunk object = 64 bytes (Work queue)

500 million objects = 56 GB

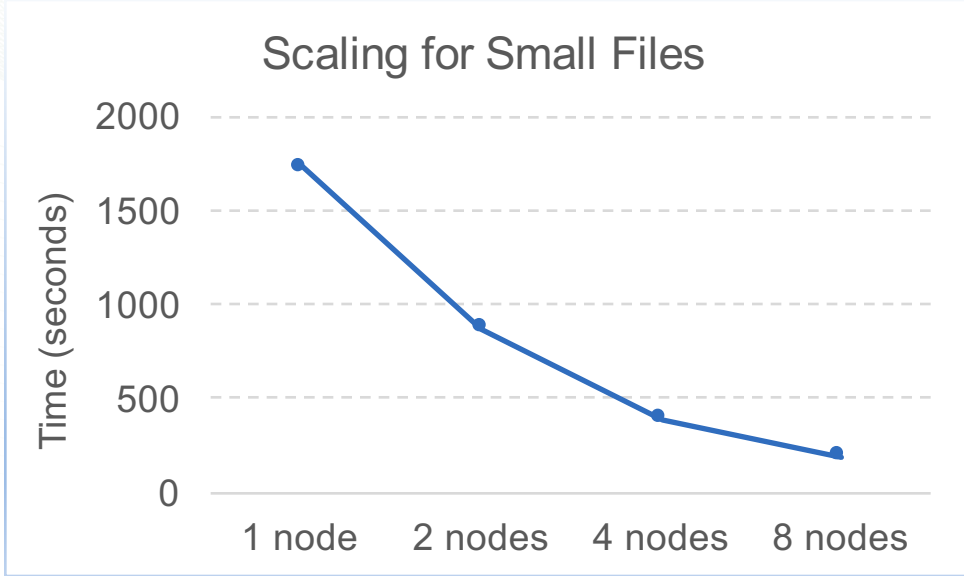
Optimization: queue file objects at the front directory object at the back for extreme unbalanced directories



We also considered:

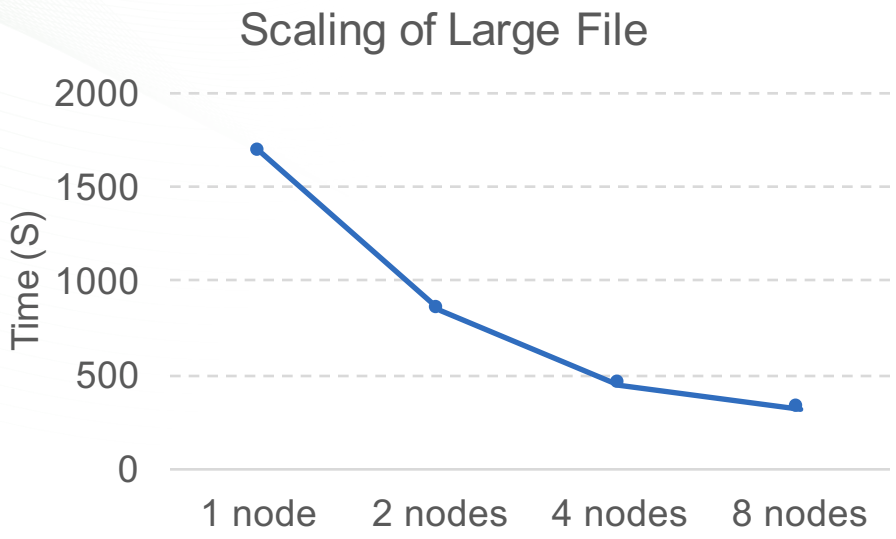
K-V store
Database
Distributed task queue or cache

Evaluation: Small Files



- Real application dataset
- 100,000 small as in less than 4k files, default striping

Evaluation: Large Files



- 1TB file
- 32 stripes, np=8
- Stripe matters

Conclusions

- We presented a parallel data copy tool, under the umbrella of a suite of tools developed at OLCF.
- It is built on the core assumptions and principles of:
  - Ubiquitous MPI in cluster environment
  - Work-stealing pattern
  - Distributed termination for scalability and self-stabilization
- It has shown promising results at our production site:
  - Large scale profiling
  - Large scale parallel checksumming
  - Large scale file transfer
- Source availability:
  - http://www.github.com/olcf/pcircle