Collective Computing for Scientific Big Data Analysis

Jialin Liu, **Yong Chen**, Suren Byna

September 1st, 2015

Outline

- Introduction
- Motivation
  - Two-Phase Collective I/O
  - Map-Reduce Computing Framework
- Collective Computing Framework and Preliminary Evaluation
  - Object I/O and Runtime Support
  - Map on Logic Subsets
  - Result Reduce and Construction
- Conclusion, Ongoing, and Future Work

Science Data Challenge

- Scientific simulations/applications have become highly data intensive
- Data-driven scientific discovery has become the fourth paradigm after experiment, theory, and simulation

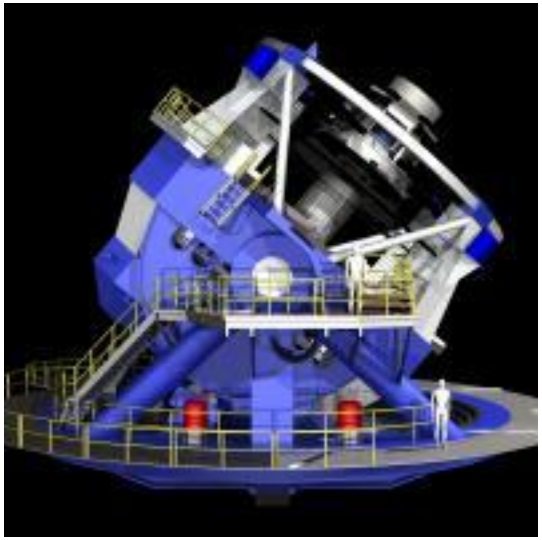
Data Requirements for Applications (2009)

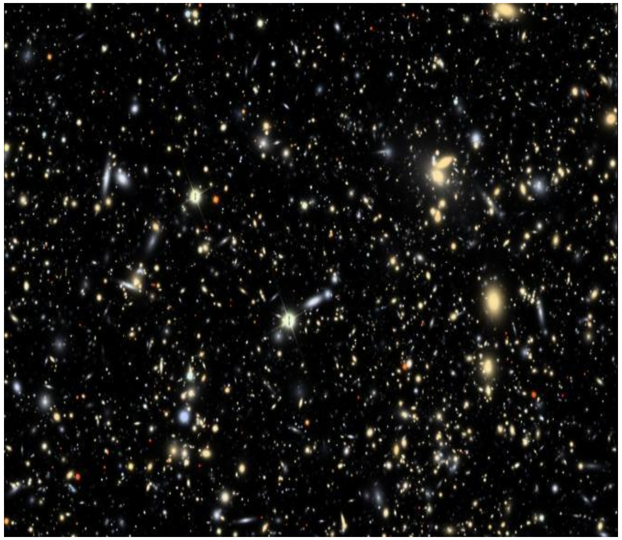
Project	On-Line	Off-Line
FLASH: Turbulent Nuclear Burning	75TB	300TB
Reactor Core Hydrodynamics	2TB	5TB
Computational Nuclear Structure	4TB	40TB
Computational Protein Structure	1TB	2TB
Performance Evaluation and Analysis	1TB	1TB
Kinetics and Thermodynamics of Metal	5TB	100TB
Climate Science	10TB	345TB
Parkinson's Disease	2.5TB	50TB
Plasma Microturbulence	2TB	10TB
Lattice QCD	1TB	44TB
Thermal Striping in Sodium Cooled Reactors	4TB	8TB
Gating Mechanisms of Membrane Proteins	10TB	10TB

Source: R. Ross et. al., Argonne National Laboratory

Science Data Challenge (cont.)

- Collected data from instruments increases rapidly too
  - Large Synoptic Survey Telescope capturing ultra-high-resolution images of the sky every 15 seconds, every night, for at least 10 years. More than 100 petabytes (about 20 million DVD, 4.7GB each) of data, 2022

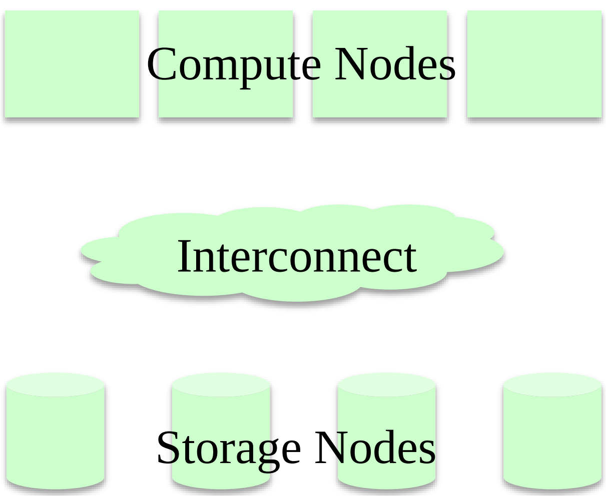




Source: LSST

Processing Data in HPC

- HPC architecture, hierarchical I/O stack
  - Traditional HPC: powerful compute nodes, high speed interconnect (e.g IB), petabytes storage, etc.
  - HPC I/O stack: scientific I/O libraries (e.g HDF5/PnetCDF/ADIOS), I/O middleware (MPI-IO), file systems (Lustre, GPFS, PVFS, etc.)

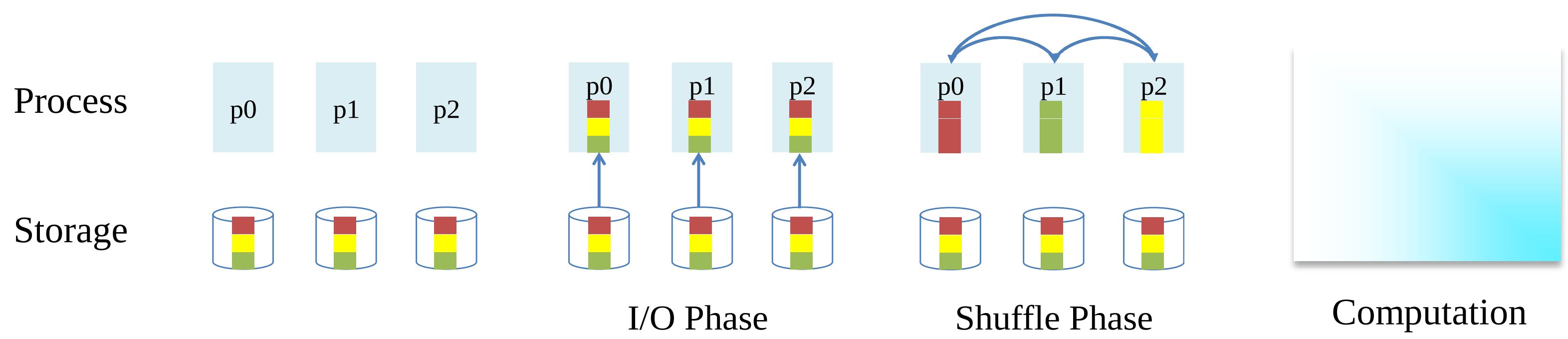


HPC Architecture

HPC I/O Software

Processing Data with Collective I/O

- Traditional Two-Phase Collective I/O
  - Non-contiguous access
  - Multiple iterations



- Problems
  - Traditional HPC: Move data from storage to compute nodes, then compute
  - Collective-IO: Computation start only when data are **completely** ready in memory

Processing Data with MapReduce

- MapReduce Computing Paradigm
  - Map step: Each worker node applies the "map()" function to the local data
  - *Shuffle step*: Worker nodes redistribute data based on the output
  - Reduce step: Worker nodes now process each group of output data, per key, in parallel.

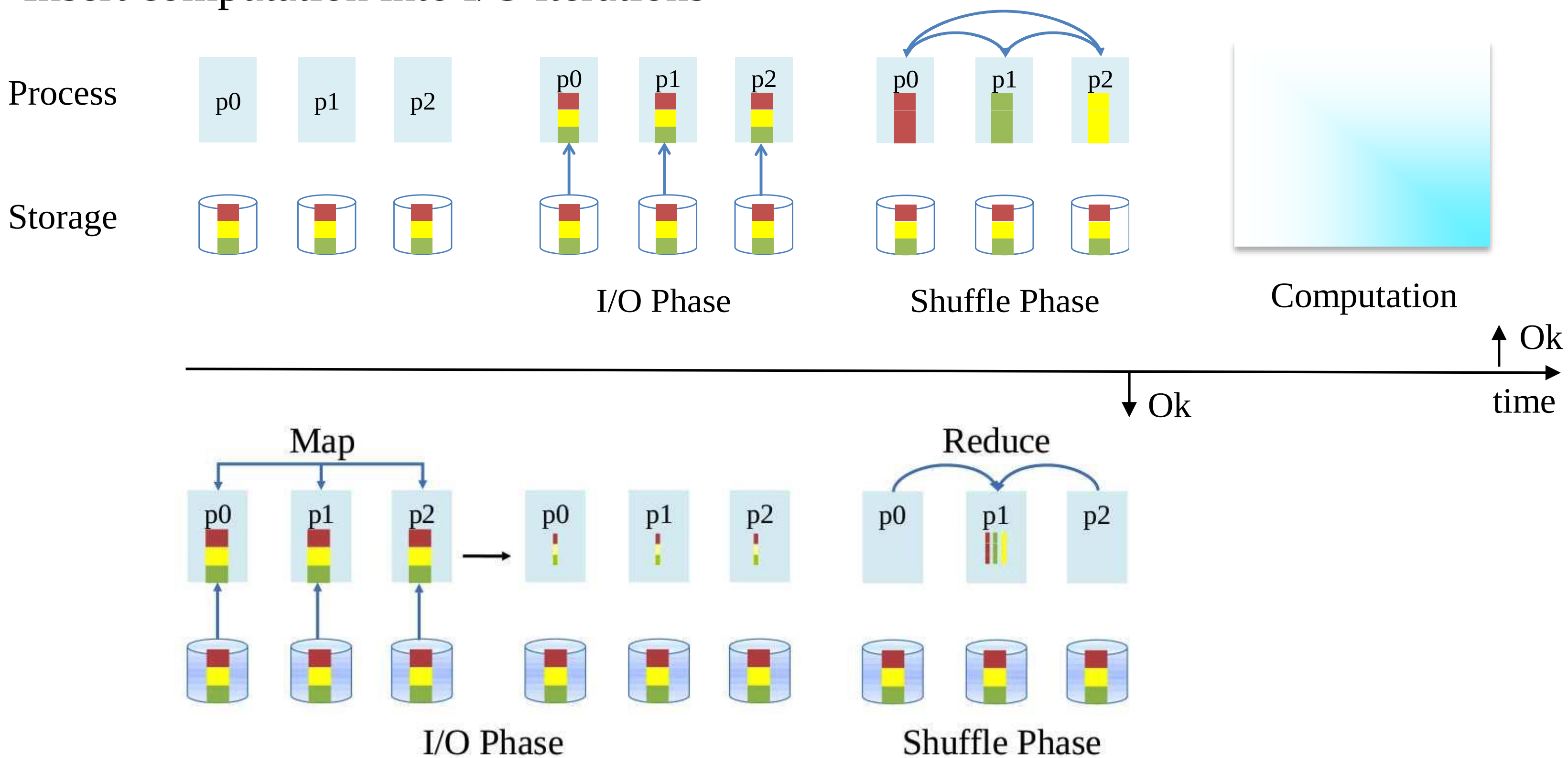
- Similarity vs Difference

Similarity	
Phase	They both have a two-phase workflow.
Order	The two phases is restricted such that the second phase can not start before the first one.
Locality	Locality is desired in both frameworks. Mapreduce's task allocation prefers a locality-aware strategy [20], while two-phase collective I/O is originally a locality-aware non-contiguous I/O optimization.
Shuffle	Mapreduce shuffle the map output to the reducer nodes. Two-phase collective I/O shuffle the data among compute nodes after I/O phase in collective read.

Difference	
Motivation	Mapreduce is a simplified parallel computing framework while two-phase collective I/O focuses on parallel I/O.
Uses	Mapreduce is commonly used in cloud computing. Two-phase collective I/O is used often in MPI-dominated, high performance computing (HPC) paradigm.
Scalability	Mapreduce can be easily scaled into a large cluster. Two-phase collective I/O delivers unsatisfying performance due to high communication overhead when running in large scale.

Collective Computing: Concept

- Collective Computing
  - Collective I/O + “MapReduce”
  - Insert computation into I/O iterations



Collective Computing: Design

- Challenges
  - Represent the computation in the collective I/O
  - Collective I/O is performed at byte level, reveal logical view
  - Runtime support
  - Others: *computation balance*, *fault tolerance*

- Proposed Solution and Contribution
  - Break the two-phase I/O constraint and form a flexible collective computing paradigm.
  - Propose object I/O to integrate the analysis task within the collective I/O.
  - Design logical map to recognize the byte sequence.

- Object I/O

I/O

```
start[0] = (dim/nprocs)*rank;
count[0] = (dim/nprocs);
temp=(float *)malloc(SIZE*sizeof(float));
ncmpi_get_vara_float_all(
ncid, varid, start, count, temp);
```

Computation

```
for(i = 0;i < count[0];i++){
sum += temp[i];
}
MPI_Reduce(sum, SUM, size, MPI_FLOAT,
        MPI_SUM,0,MPI_COMM_WORLD);
```

Traditional Collective I/O

I/O

```
io.start[0] = (dim/nprocs)*rank;
io.count[0] = (dim/nprocs);
io.temp  = (float *)malloc(SIZE*sizeof(float));
io.mode  = collective;
io.block = false;
```

Computation

```
void compute(out, in, len, dtype)
for(i = 0;i < len;i++){
out += in[i];
}
MPI_Op_create((MPI_User_function*)compute,1,&op);
```

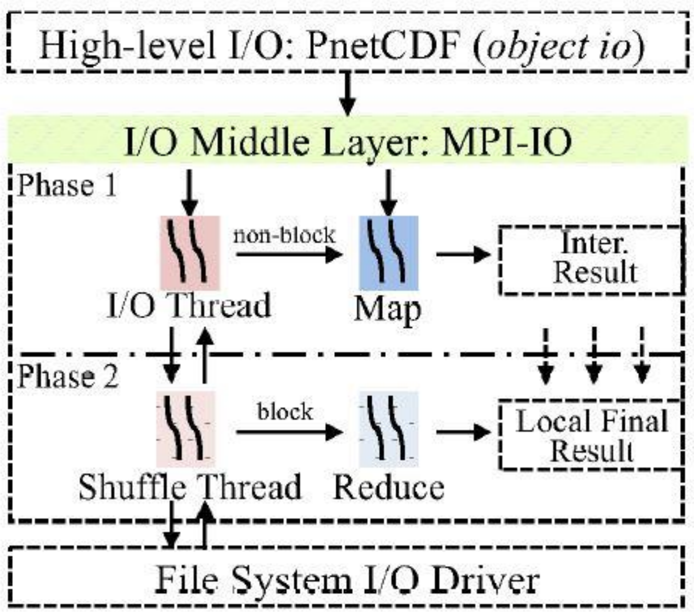
Object

```
ncmpi_object_get_vara_float(io,op);
```

Object I/O

Collective Computing: Design

- Runtime Support

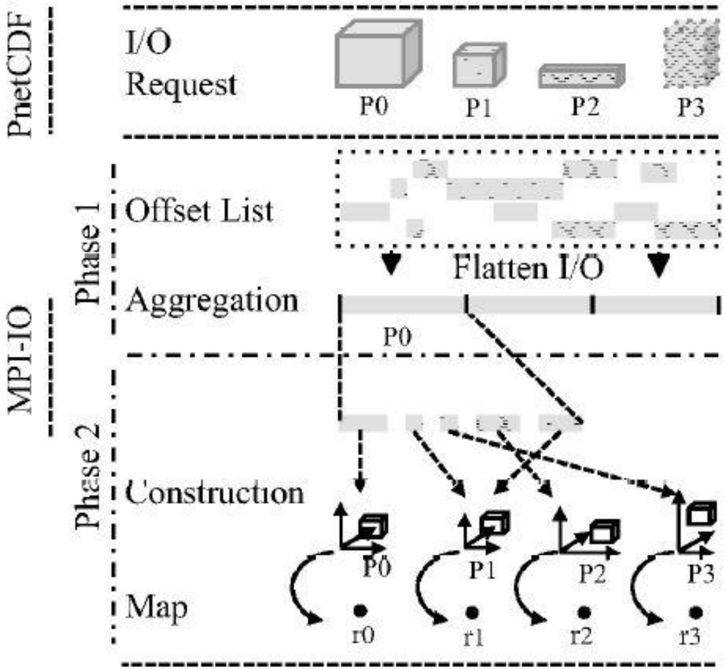


Collective Computing Runtime

The object I/O is declared in high-level I/O libraries, and passed into MPI-IO layer

Collective Computing: Design

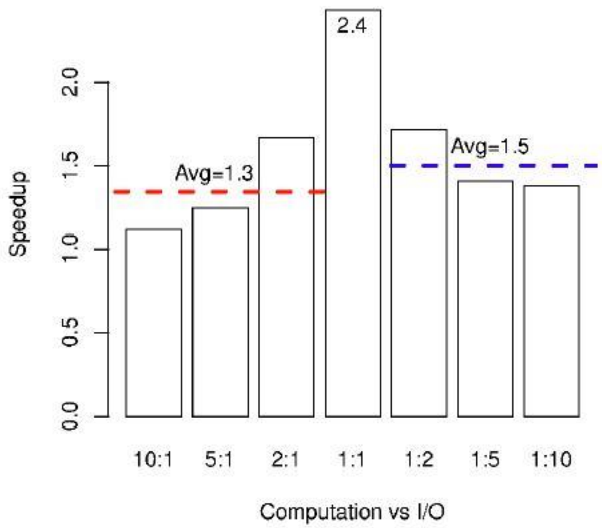
- Map on Logical Subsets



- Results Reduce and Construction
  - All-to-One
  - All-to-All

Evaluation

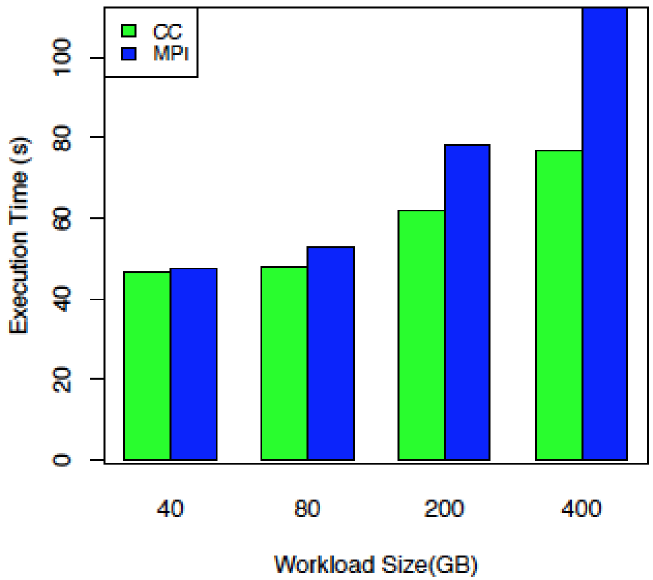
- Experimental Evaluation
  - Cray xe6, Hopper, 153216 cores, 212 terabytes memory, 2 petabytes disk
  - MPICH 3.1.2
  - Benchmark and applications, WRF, synthetic datasets, 800 GB
  - Computation: statistics, e.g., sum, average, etc



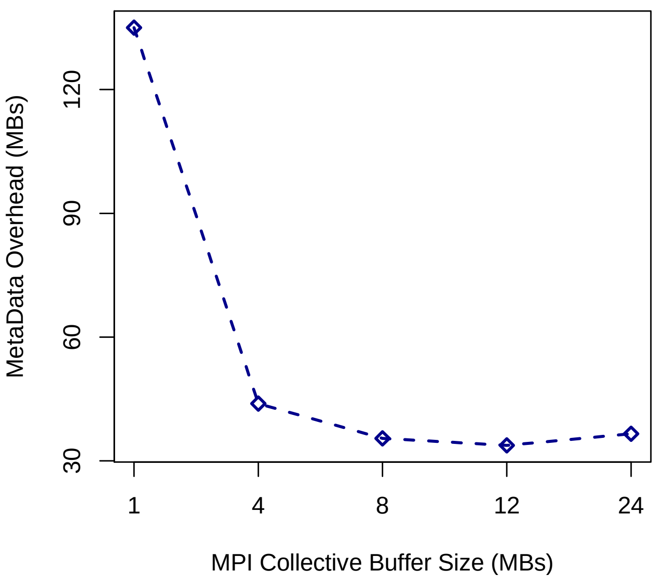
Speedup with Different Computation IO Ratio

Evaluation

- Experimental Evaluation
  - WRF model test
  - Storage overhead



WRF Model Test



Storage Overhead

Conclusion, Ongoing, and Future Work

- Related Work
  - Nonblocking Collective Operations
  - Combination of MPI and Mapreduce
- Conclusion
  - Traditional collective IO can not conduct analysis until the I/O is finished.
  - Collective computing intends to provide nonblocking computing paradigm
  - Breaks the two-phase I/O constraint: object I/O, logical map, runtime
  - 2.5X speedup
- Ongoing and future work
  - Balance computation on aggregator
  - Fault tolerance, handling loss of data and intermediate results

Q&A

Thank You!

For more info please visit: http://discl.cs.ttu.edu/