A Framework for Exascale Analysis Shipping

Ivo Jimenez, Carlos Maltzahn (UCSC)
Jerome Soumagne, Ruth Aydt, Quincey Koziol (HDF Group)
Jay Lofstead (Sandia National Labs)

October 25, 2013
Evolution of Storage and I/O
Evolution of Storage and I/O

Compute Cluster

Compute Nodes

Compute to Storage

Storage Cluster

Disk

Application

HDF5, NetCDF, etc.

POSIX client

POSIX server

FS Services
Evolution of Storage and I/O
Evolution of Storage and I/O

Compute Cluster

Compute Nodes

Compute to Storage

Storage Cluster

Disk

Application
- Data Format Middleware
- MPI-IO, Parallel HDF5, etc.
- POSIX client

POSIX server
- FS Services
Evolution of Storage and I/O

Compute Cluster

Storage Cluster

Compute to Storage

Disk

Application
Data Format Middleware
I/O Middleware
POSIX client

POSIX server
FS Services
Evolution of Storage and I/O
Evolution of Storage and I/O
Exascale

- End of decade
- Millions of processors
- Billions of processes
- Storage is a major bottleneck (again)
The POSIX barrier
DOE FastForward Storage and I/O

[Diagram showing the flow of data and interactions between different components such as Application, Data Format Middleware, RPC client (I/O fwd.), RPC server, I/O Middleware, POSIX client, POSIX server, FS Services, I/O Dispatcher, Data Format Middleware, I/O Dispatcher client, FS client, and FS Services.]

11
DOE FastForward Storage and I/O

• Application controls the I/O stack
• Support complex usage patterns
• Merge/develop technology into a unified next-generation I/O Dispatcher (IOD) API:
  • Manage the staging area
  • Handle atomicity and durability of checkpoints
  • Expose non-POSIX FS services
  • Interact with long-term storage
Staging Area Management
Staging Area Management
Staging Area Management
Staging Area Management
Staging Area Management
Staging Area Management

Compute Cluster ↔ Staging Area ↔ Storage Cluster
Staging Area Management
Staging Area Management
Atomic, Durable Transactions
Atomic, Durable Transactions

Compute Cluster → Staging Area → Storage Cluster

Writes

Transaction #
Atomic, Durable Transactions

- Compute Cluster
- Staging Area
- Storage Cluster

Writes

Transaction #
Atomic, Durable Transactions
Atomic, Durable Transactions
FS Services: Object-based
FS Services: Object Layout
What about Analysis?

• Currently:
  • Treated as a second-class citizen
  • Usually runs off-line (but slow)
  • Ideal is to run in-situ, but is hard at exascale

• Middleware:
  • Modify data as it comes into the staging area
  • Injects analysis and visualization *awareness* into the POSIX-formated data

• Problems:
  • Hard to co-schedule with simulation:
  • Resource contention
  • Impossible to generate results with in-situ alone
  • Temporal analysis can be difficult
New Possibilities

- Rich metadata = Smarter execution
- Consider execution alternatives
- Optimize for distinct objectives
- Dynamic, user-defined code execution
Analysis Shipping
IOD Console

In [1]: iod = connect('\$IOD_ADDRESS')
In [1]: iod = connect('$IOD_ADDRESS')
In [2]: j = iod.new_job('GTC')
In [1]: iod = connect('$IOD_ADDRESS')
In [2]: j = iod.new_job('GTC')
In [3]: iod.execute(j)
    enqueuing job 'j'
In [1]: iod = connect('$IOD_ADDRESS')
In [2]: j = iod.new_job('GTC')
In [3]: iod.execute(j)
   enqueuing job 'j'
In [4]: j.status(j)
   job 'j' is running
In [1]: iod = connect('$IOD_ADDRESS')
In [2]: j = iod.new_job('GTC')
In [3]: iod.execute(j)
enqueuing job 'j'
In [4]: j.status(j)
   job 'j' is running
In [5]: j.highest_committed_transaction(j)
   highest HCT for 'j': 6
In [1]: iod = connect('$IOD_ADDRESS')
In [2]: j = iod.new_job('GTC')
In [3]: iod.execute(j)
    enqueuing job 'j'
In [4]: j.status(j)
    job 'j' is running
In [5]: j.highest_committed_transaction(j)
    highest HCT for 'j': 6
In [6]: t = new iod.analysis('/path/to/analysis.py')
In [1]: iod = connect('IOD_ADDRESS')
In [2]: j = iod.new_job('GTC')
In [3]: iod.execute(j)
   enqueuing job 'j'
In [4]: j.status(j)
   job 'j' is running
In [5]: j.highest_committed_transaction(j)
   highest HCT for 'j': 6
In [6]: t = new iod.analysis('/path/to/analysis.py', 6)
In [7]: iod.ship(t)
   task 't' has been queued
In [1]: iod = connect('$$IOD_ADDRESS$$')
In [2]: j = iod.new_job('GTC')
In [3]: iod.execute(j)
   enqueuing job 'j'
In [4]: j.status(j)
   job 'j' is running
In [5]: j.highest_committed_transaction(j)
   highest HCT for 'j': 6
In [6]: t = new iod.analysis('/path/to/analysis.py', 6)
In [7]: iod.ship(t)
   task 't' has been queued
In [8]: t.status
   task 't' has started
User Context (Analysis API)

- Communicator among I/O nodes to allow:
  - User-defined process topology
  - Handle unalignment
  - Handle “ghost” regions between neighbors
- Metadata about referenced objects:
  - Sharding
  - Format
  - Multiple copies
  - Placement
Example:

```python
def execute_analysis():

ds = h5file['/group1/dataset1']

for s in local_shards[ds]:
    res = user_defined_shard_management(s)

    # communicate with neighborhood
    iod_comm.send(res, neighbors(s))

    # write output, in sharding-optimal way
    new_ds = container['/group2/newdataset']
    new_ds.set_layout(get_layout())

    # coordinating with neighbors to write
    iod_comm.receive(value, coord, neighbors(s))
    new_ds[coord] = value

def user_defined_shard_management(shard):
    # possible user-defined handling of shards:
    # * deal with unaligned shards
    # * handle "ghost" regions
    # * create shard-to-process assignment
```
Status and Future Work

• Current:
  • Extensions on FastForward’s prototype
  • Implemented data-movement optimization
  • Finishing experimental setup

• Future:
  • Modularize optimization phase
  • Incorporate other features of IOD
  • Add declarative interface
  • Plug to existing workflow managers
Thanks!