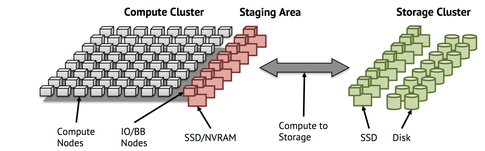
A Framework for Exascale Analysis Shipping

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

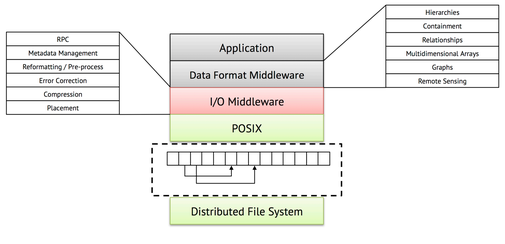
The Road to Exascale

Exascale systems that are slated for the end of this decade will include up to a million of compute nodes running about a billion threads of execution. In this scenario, traditional methods that ameliorate I/O bottlenecks don't work anymore. *I/O Staging*[1][2] proposes the designation of a portion of the high-end nodes to manage I/O.



The POSIX barrier

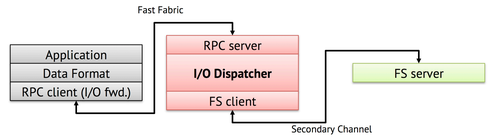
Since most of the applications assume a POSIX interface to storage, existing middleware either transparently handle the I/O operations, appearing as regular POSIX calls to applications; or modify the I/O API as little as possible in order to minimize the impact on production codes.



As we move towards the exascale goal, this way of interfacing with storage becomes more of an obstacle. Many features provided by distributed file systems are hidden by the POSIX layer and in many cases have to be replicated by existing middleware.

Fast Forward I/O and Storage Initiative

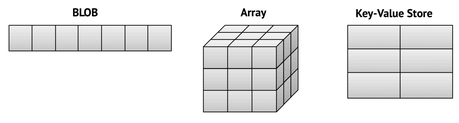
DOE's Fast Forward Storage and I/O[3] project is aimed at merging the features of existing middleware into a next-generation storage and I/O stack. Applications or data format libraries interface against the I/O Dispatcher (IOD) interface (shown below), which manages the staging area and interfaces directly with the distributed file system.



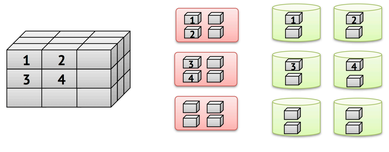
I/O Dispatcher Interface

The interface exposes many features: transactions, asynchronous I/O, object-based storage, sharding, placement and formatting.

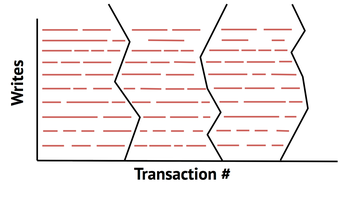
Object-based



Sharding and Placement

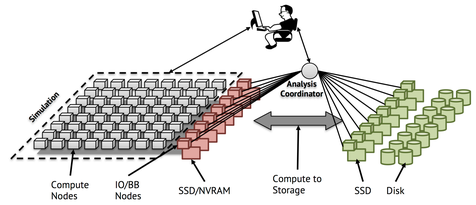


Transactions



Exascale Analysis

We device a master/worker architecture running on the I/O nodes (and storage cluster nodes) that is in charge of receiving, planning and executing user-defined analytical tasks.



The usage flow is the following:

1. Launch simulation on compute cluster.
2. Timestamp computed and dumped to I/O nodes.
3. Interactively explore (query) data on I/O nodes.
4. Ship analysis to I/O nodes.

Analysis Shipping

The user launches analytical tasks by shipping a python script to the coordinator. The following is an example of an analytical (ipython-based) session:

```
In [1]: connect ('$IOD_ADDRESS')
In [2]: t = newtask ('/path/to/analysis.py')
In [3]: ship(t)
In [4]: t.status
task 't' has been queued
In [5]: t.status
task 't' has started
```

Analysis Planning and Execution

To illustrate, consider data-movement minimization. We achieve this through the following procedure:

1. Coordinator requests the sharding of referenced objects.
2. Coordinator communicates sharding metadata to workers.
3. Expose sharding information to analytical environment.
4. Each worker operates on local shards.

Scriptable Analysis

```
def execute():

  ds = h5file['/group1/dataset1']

  for s in local_shards[ds]:
    # local_shards are loaded into the analysis context.
    # metadata associated to them can be obtained
    res = user_defined_shard_management(s)

    # script can make use of the inter-ION communicator.
    # the neighborhood is application-specific; can be
    # implemented as a library (the "neighbors" function
    # in this case)
    iod_comm.send(res, neighbors(s))

    # possibly write to a new dataset
    new_ds = container['/group2/newdataset']

    # specify the optimal layout for new objects
    new_ds.set_layout(get_layout())

    # obtain value, by coordinating with neighbors
    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
  #   * create shard-to-process topology (neighborhood)
```

Conclusion and Outlook

We have introduced our vision of an analysis execution engine that runs on top of the FastForward IOD interface. With this foundation, we can apply many proven data management techniques from other domains, such as those from *Analytics Databases* and *Big Data* systems.

Acknowledgements

We would like to thank all the Fast Forward team for insightful discussions. Part of this work was done while the main author worked as a summer Intern at **Intel**.

[1] Liu et al., *On the Role of Burst Buffers in Leadership-class Storage Systems.* http://dx.doi.org/10.1109/MSST.2012.6232369
[2] Lofstead et al., *Adaptable, metadata rich IO methods for portable high performance IO.* http://dx.doi.org/10.1109/IPDPS.2009.5161052
[3] *DOE Extreme-Scale Technology Acceleration. FastForward.* https://asc.llnl.gov/fastforward/