Streaming, Storing, and Sharing Big Data for Light Source Science

Justin M Wozniak <wozniak@mcs.anl.gov>
Kyle Chard, Ben Blaiszik, Michael Wilde, Ian Foster

Argonne National Laboratory

At STREAM 2015
Oct. 27, 2015
Advanced Photon Source (APS)
Advanced Photon Source (APS)

- Moves electrons at >99.999999% of the speed of light.
- Magnets bend electron trajectories, producing x-rays, highly focused onto a small area.
- X-rays strike targets in 35 different laboratories – each a lead-lined, radiation-proof experiment station.
- Scattering detectors produce images containing experimental results.
## Distance from Top Light Sources to Top Supercomputer Centers

<table>
<thead>
<tr>
<th>Light Source</th>
<th>Distance to Top10 Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIRIUS, Brazil</td>
<td>&gt; 5000Km, TACC, USA</td>
</tr>
<tr>
<td>BAP, China</td>
<td>2000Km, Tihane-2, China</td>
</tr>
<tr>
<td>MAX, Sweden</td>
<td>800Km, Jülich Germany</td>
</tr>
<tr>
<td>PETRA III, Germany</td>
<td>500Km, Jülich Germany</td>
</tr>
<tr>
<td>ESRF, France</td>
<td>400Km, Lugano, Switzerland</td>
</tr>
<tr>
<td>Spring 8, Japan</td>
<td>100Km, K-Machine, Kobe, Japan</td>
</tr>
<tr>
<td>APS, IL, USA</td>
<td>~1Km, ALCF &amp; MCS*, ANL, USA</td>
</tr>
</tbody>
</table>

*ANL Computing Divisions  
ALCF: Argonne Leadership Computing Facility  
MCS: Mathematics & Computer Science
Proximity means we can closely couple computing in novel ways. Terabits/s in the near future are possible.
Goals and tools

TALK OVERVIEW
Goals

- **Automated data capture and analysis pipelines**
  *To boost productivity during beamtime*

- **Integration with high-performance computers**
  *To integrate experiment and simulation*

- **Effective use of large data sets**
  *Maximize utility of high-resolution, high-frame-rate detectors and automation*

- **High interactivity and programmability**
  *Improve the overall scientific process*
Tools

- **Swift**
  *Workflow language with very high scalability*

- **Globus Catalog**
  *Annotation system for distributed data*

- **Globus Transfer**
  *Parallel data movement system*

- **NeXpy/NXFS**
  *GUI with connectivity to Catalog and Python remote object services*
High performance workflows

SWIFT
Goals of the Swift language

Swift was designed to handle many aspects of the computing campaign

- Ability to integrate **many application components** into a new workflow application
- Data structures for complex data organization
- Portability - separate site-specific configuration from application logic
- Logging, provenance, and plotting features

- **Today,** we will focus on running scripted applications on large streaming data sets
Swift programming model: All progress driven by concurrent dataflow

(int r) myproc (int i, int j)
{
    int x = A(i);
    int y = B(j);
    r = x + y;
}

- A() and B() implemented in native code
- A() and B() run in concurrently in different processes
- r is computed when they are both done

- This parallelism is *automatic*
- Works recursively throughout the program’s call graph
Swift programming model

- **Data types**
  ```swift
  int i = 4;
  int A[];
  string s = "hello world";
  ```

- **Mapped data types**
  ```swift
  file image<"snapshot.jpg">;
  ```

- **Structured data**
  ```swift
  image A[]<array_mapper...>;
  type protein {
    file pdb;
    file docking_pocket;
  }
  protein p<ext; exec=protein.map>;
  ```

- **Conventional expressions**
  ```swift
  if (x == 3) {
    y = x+2;
    s = strcat("y: ", y);
  }
  ```

- **Parallel loops**
  ```swift
  foreach f,i in A {
    B[i] = convert(A[i]);
  }
  ```

- **Data flow**
  ```swift
  merge(analyze(B[0], B[1]),
       analyze(B[2], B[3]));
  ```

- **Swift: A language for distributed parallel scripting.** J. Parallel Computing, 2011
Swift/T: Distributed dataflow processing

Had this: (Swift/K)

For extreme scale, we need this: (Swift/T)

Swift/T: Enabling high-performance workflows

- Write site-independent scripts
- Automatic parallelization and data movement
- Run native code, script fragments as applications

Features for Big Data analysis

• **Location-aware scheduling**
  User and runtime coordinate data/task locations

- Application:
  Dataflow, annotations

- Runtime:
  Hard/soft locations

- Distributed data


• **Collective I/O**
  User and runtime coordinate data/task locations

- Application:
  I/O hook

- Runtime:
  MPI-IO transfers

- Distributed data

- Parallel FS

Next steps for streaming analysis

- **Integrated streaming solution**
  Combine parallel transfers and stages with distributed in-memory caches

- **Parallel, hierarchical data ingest**
  Implement fast bulk transfers from experiment to variably-sized ad hoc caches

- **Retain high programmability**
  Provide familiar programming interfaces
Abstract, extensible MapReduce in Swift

```swift
main {
    file d[];
    int N = string2int(argv("N"));
    // Map phase
    foreach i in [0:N-1] {
        file a = find_file(i);
        d[i] = map_function(a);
    }
    // Reduce phase
    file final <"final.data"> = merge(d, 0, tasks-1);
}

(file o) merge(file d[], int start, int stop) {
    if (stop-start == 1) {
        // Base case: merge pair
        o = merge_pair(d[start], d[stop]);
    } else {
        // Merge pair of recursive calls
        n = stop-start;
        s = n % 2;
        o = merge_pair(merge(d, start, start+s),
                        merge(d, start+s+1, stop));
    }
}
```

- User needs to implement `map_function()` and `merge()`
- These may be implemented in native code, Python, etc.
- Could add annotations
- Could add additional custom application logic
Hercules/Swift

- Want to run arbitrary workflows over distributed filesystems that expose data locations: **Hercules** is based on **Memcached**
  - Data analytics, post-processing
  - Exceed the generality of MapReduce without losing data optimizations

- Can **optionally** send a Swift task to a particular location with simple syntax:
  ```
  foreach i in [0:N-1] {
    location L = locationFromRank(i);
    @location=L f(i);
  }
  ```

- Can obtain ranks from hostnames:
  ```
  int rank = hostmapOneWorkerRank("my.host.edu");
  ```

- Can now specify location constraints:
  ```
  location L = location(rank, HARD|SOFT, RANK|NODE);
  ```

- Much more to be done here!
Annotation system for distributed scientific data

GLOBUS CATALOG
Catalog Goals

- **Group** data based on use, not location
  - Logical grouping to organize, reorganize, search, and describe usage
- **Annotate** with characteristics that reflect content ...
  - Capture as much existing information as possible
  - Share datasets for collaboration- user access control
- **Operate** on datasets as units
- Research data lifecycle is **continuous and iterative**:
  - Metadata is created (automatically and manually) throughout
  - Data provenance and linkage between raw and derived data
- **Most often**:
  - Data is grouped and acted on collectively
    - Views (slices) may change depending on activity
  - Data and metadata changes over time
  - Access permissions are important (and also change)
Catalog Data Model

- **Catalog**: a hosted resource that enables the grouping of related datasets
- **Dataset**: a virtual collection of (schema-less) metadata and distributed data elements
- **Annotation**: a piece of metadata that exists within the context of a dataset or data member
  - Specified as key-value pairs
- **Member**: a specific data item (file, directory) associated with a dataset
Web interface for annotations
High-speed wide area data transfers

GLOBUS TRANSFER
Globus Transfer

Supercomputers and Campus Clusters

Object Storage

Block/Drive Storage

Instance Storage

Personal Resources

Globus Connect

Globus Connect

Globus Connect

Globus Endpoints

Transfer

Synchronize

Share

InCommon/CILogon

OpenID

MyProxy

OAuth

Globus Nexus

Globus Connect

Globus Connect

Globus Connect

Globus Connect
Globus Transfer

- Reliable, secure, high-performance file transfer and synchronization
- “Fire-and-forget” transfers
- Automatic fault recovery
- Seamless security integration
- 10x faster than SCP
The Petrel research data service

- High-speed, high-capacity data store
- Seamless integration with data fabric
- Project-focused, self-managed
Rapid and remote structured data visualization

NEXPY / NXFS
NeXpy: A Python Toolbox for Big Data

- A toolbox for manipulating and visualizing arbitrary NeXus data of any size
- A scripting engine for GUI applications
- A portal to Globus Catalog
- A demonstration of the value of combining:
  - a flexible data model
  - a powerful scripting language

http://nexpy.github.io/nexpy

$ pip install nexpy
Mullite
NeXpy in the Pipeline

- Use of NeXpy throughout the analysis pipeline
The NeXus File Service (NXFS)

NXFS Performance

- Faster than application-agnostic remote filesystem technologies
  - Compared Pyro to Chirp and SSHFS from inside ANL (L) and AWS EC2 (W)
- Plus ability to invoke remote methods!

- File open \((10^{-1}s)\)
- Metadata read \((10^{-2}s)\)
- Pixel read \((1s)\)

Operation and Time Scale
Near Field – High Energy Diffraction Microscopy
Collaboration with APS Sector 1: Jon Almer, Hemant Sharma, et al.

CASE STUDY: NF-HEDM
Determining the crystal structure of metals non-destructively

Gold calibrant wire

Confidence Index Orientation Map Tomo reconstruction Confidence Index
NF-HEDM

0: Generate Parameters
FOP.c
50 tasks
25s/task
¼ CPU hours
Manual
Swift/K

Detector

Dataset
360 files
4 GB total

Workflow Control Script
Bash

Globus Catalog
Scientific Metadata
Workflow Progress

Up to
2.2 M CPU hours
per week!

1: Imaging
Median calc
75s (90% I/O)
MedianImage.c
Uses Swift/K
Peak Search
15s per file
Swift/K

Reduced Dataset
360 files
5 MB total

2: Convert bins to Network
Endian format.
2 min for all files.
Swift/T

3: Analysis Pass
FitOrientation.c
60s/task (PC)
1667 CPU hours
600s/task (BG/Q)
16,670 CPU hours
Swift/T

Orthros
(All data in NFS)

Blue Gene/Q

feedback to experiment
High-Energy Diffraction Microscopy

- Near-field high-energy diffraction microscopy discovers metal grain shapes and structures
- The experimental results are greatly improved with the application of Swift-based cluster computing (RED indicates higher confidence in results)
Big picture: Task-based HPC on Big Data

- Existing C code assembled into scalable HPC program with Swift/T
- Problem: Each task must consumes ~500 MB of experimental data
- Runs on the Blue Gene/Q
- Relevant to Big Data – HPC convergence
- Could use Swift/T data locality annotations for high-level, data location-aware programming
Intended use of broadcast operation

- Grain orientation optimization workflow runs on BG/Q once data is there
- Each task needs to read all input from a given dataset
- Desire to use MPI-IO before running tasks
Big Data Staging with MPI-IO

- Solution: Broadcast experimental data on HPC system with MPI-IO
- Tasks consume data normally from node-local storage
Scalability result: End-to-end

- 101 GB/s
- 21 GB/s
- 8K cores
Scalability result: Stage+Write

- This plot breaks I/O hook into 1) stage+write and 2) read phases
- Read phase is node-local: consistently 10.8 ± 0.1 s
NF-HEDM: Conclusions

- Blue Gene/Q can be used for big data problems and a many-task programming model
  - Just broadcast the data to compute nodes first with MPI-IO

- The Swift I/O hook enables efficient I/O in a many-task model
  - Reduces I/O time by factor of 4.7!

- Connecting HPC to a real-time experiment saved an experiment by detecting a loose cable

- Code is now being reused by about 5 different groups
  - Now must accommodate extra users on HPC resources!
Summary

- Described Big Data + HPC application: X-ray crystallography
- Described four relevant tools:
  - Swift
  - Globus Catalog
  - Globus Transfer
  - NeXpy/NXFS
- Described path forward, integrating tools for streaming workflows
- Thanks to the organizers
- Thanks to our application collaborators
- Questions?

  - http://swift-lang.org