Big Data Center - Data Management Initiative

Quincey Koziol
BDC Summit
July 18, 2018
Data Management Technologies

- **Scientific Data Analytics Software Stack**
  - Deep Learning on HPC IO Platforms

![Diagram showing the stack of data management technologies]

- Python Runtime
- TensorFlow
- TensorFlow Native Modules
- NetCDF
- HDF5
- MPI-IO
- POSIX
- Lustre
- Python Libraries (e.g. PyTables)
- Python Modules
Data Management Challenges

- Achieving IO Performance is hard
  - Tuning I/O is complex
    - Even along “the happy path”
    - Multi-layered stack
    - Scores of tunable parameters
  - What is not measured cannot be improved!
    - Determine baseline for every layer
    - Measure overhead for each component
    - Iterate: diff against baseline  tune  measure

Diagram:
- Application
  - HDF5
  - MPI-IO
  - POSIX
  - Lustre
  - HPC I/O
Data Management Challenges

• “Big Data” adds new and different complexity
  • More moving parts and layers
    • More permutations to analyze
    • Middleware/Runtime/Library implementation issues
  • Big Data >>> O (GB)
    • KNL Nodes on Cori:
      • 96 GB DDR4
      • 16 GB MCDRAM
  • Scientific Datasets Typically:
    • Very Large: O (TB, PB)
    • Very Complex: Hundreds of channels, Extreme Resolution
Factors Affecting I/O Performance

• **Workload Type (I/O, Network, Memory, Compute)**
  - Is I/O even the bottleneck?
    - What proportion of application runtime spent doing I/O?
    - Use case may be compute/memory bound
  - Extract and run I/O kernel (i.e. “null” computation)
  - Compare against known baselines:
    - Realistic Peak Throughput on Cori
      - 3 GB/s per Lustre OST (Theoretical)
      - ~1 GB/s per Haswell Core
Factors Affecting I/O Performance

• File System (Physical Layout)
  • Parallelism: Serial vs Parallel
  • Concurrency: Exclusive vs Shared
    • How many Lustre clients per OST?
    • How many threads (or cores) per client? (KNL requires more)
  • Striping Layout: width, count, alignment

• Data Format (Logical Layout)
  • Text vs Binary, Raw vs Compressed
  • Row-major vs Column-Major order, Chunked
Factors Affecting I/O Performance

- Access Pattern: Sequential vs Strided vs Random
  - Does access pattern match file layout?
    - Reading columns from data stored row-wise
    - Misalignment of data transfers with stripe boundaries
  - Use I/O profiling tools to examine pattern
    - e.g. Darshan; but only works with MPI
HDF5 in TensorFlow

• Requirements
  • Feed data from HDF5 files to TensorFlow
    • And possibly other data analytics frameworks in the future
  • Needs to be fast and multi-threaded.
  • Buffer data for efficient shuffling
  • Allow overlapping I/O with computation
  • Flexible / Tunable with respect to
    • Number of files
    • HDF5 datatypes and array shapes
HDF5 in TensorFlow

• Implementations
  • Python-based File Iterator (Naïve)
    • Easy to implement and change; Customized for specific dataset
    • Not multi-threaded and cannot be used for shared files.
  • File Queue Runner (Deprecated)
    • Threaded I/O when working with multiple files; data prefetching
    • Tedious to implement, deadlock prone
  • Dataset API (Recommended)
    • Threaded I/O when working with multiple files; data prefetching
    • Must be implemented for every File System
HDF5 in TensorFlow

• Preliminary Results

• Read 80 GiB from 4 HDF5 files stored on Lustre

![Bar chart showing read performance in MB/s for different configurations: Naive (1 thread; buffered), Queue (1 thread; unbuffered), Dataset (1 thread; unbuffered). Haswell and KNL results are compared.](chart.png)
Possible Next Steps

• **Benchmarking & Analysis**
  • Will need more real science applications to tune
  • Develop synthetic benchmarks from IO Kernels

• **Extending existing libraries for IO scaling**
  • Parallel I/O support for TensorFlow
    • FileSystem Interface?
    • MPI-IO over Lustre?
  • Multi-threading support for HDF5

• **Non-MPI Scaling & Parallel I/O**
  • Scale TensorFlow using novel exascale technologies?
Possible Next Steps

- Explore I/O strategies for scaling Deep Learning
  - Hyperparameter Optimization
    - Single data stream, no partitioning, multiple models
    - I/O Strategy: Embarrassingly Parallel, Shuffle (e.g. MPI_Allgatherv)
  - Data Parallelism
    - Multiple data streams, vertical partitioning, replicated model
    - I/O Strategy: Embarrassingly parallel, Stripe aligned
  - Model Parallelism
    - Multiple data streams, horizontal partitioning, distributed model
    - I/O Strategy: Read Coalescing, Chunking
• Evaluate Applications
  • Instrument Applications and Frameworks
  • Find I/O Bottlenecks
  • Determine Solutions and Implement Them
Outline

- HEP Benchmark
  - Time Breakdown and Bandwidth
- Climate Data Benchmark
  - Data Format
  - Time Breakdown and Bandwidth
- Future Exploration Scopes
Outline

- HEP Benchmark
  - Time Breakdown and Bandwidth
- Climate Data Benchmark
  - Data Format
  - Time Breakdown and Bandwidth
- Future Exploration Scopes
64 Nodes

![Bar chart showing average time taken per node (seconds) across different epochs.](chart.png)

### Average Time Taken Per Node (seconds)

**Epoch 1**: 51.88 (97.35 %)  
**Epoch 2**: 9.32 (18.84 %)  
**Epoch 3**: 9.76 (19.94 %)  
**Epoch 4**: 9.58 (19.87 %)  
**Epoch 5**: 10.67 (22.34 %)

<table>
<thead>
<tr>
<th>Task</th>
<th>Epoch 1</th>
<th>Epoch 2</th>
<th>Epoch 3</th>
<th>Epoch 4</th>
<th>Epoch 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>51.88 (97.35 %)</td>
<td>9.32 (18.84 %)</td>
<td>9.76 (19.94 %)</td>
<td>9.58 (19.87 %)</td>
<td>10.67 (22.34 %)</td>
</tr>
<tr>
<td>Load file</td>
<td>21.19 (3.82 %)</td>
<td>14.46 (2.85 %)</td>
<td>14.00 (2.78 %)</td>
<td>14.01 (2.73 %)</td>
<td>14.02 (3.07 %)</td>
</tr>
<tr>
<td>Training Iteration</td>
<td>481.80 (86.83 %)</td>
<td>481.54 (95.31 %)</td>
<td>479.41 (95.28 %)</td>
<td>488.67 (95.40 %)</td>
<td>431.76 (94.59 %)</td>
</tr>
</tbody>
</table>
128 Nodes

- Average Time Taken Per Node (seconds)

<table>
<thead>
<tr>
<th></th>
<th>Epoch 1</th>
<th>Epoch 2</th>
<th>Epoch 3</th>
<th>Epoch 4</th>
<th>Epoch 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>23.81 (8.24%)</td>
<td>4.81 (3.46%)</td>
<td>4.24 (1.66%)</td>
<td>4.25 (1.63%)</td>
<td>4.76 (2.12%)</td>
</tr>
<tr>
<td>Load File</td>
<td>10.45 (3.62%)</td>
<td>7.47 (2.27%)</td>
<td>7.01 (2.74%)</td>
<td>7.01 (2.69%)</td>
<td>7.01 (3.12%)</td>
</tr>
<tr>
<td>Training Iteration</td>
<td>254.60 (88.14%)</td>
<td>316.98 (96.27%)</td>
<td>244.05 (95.59%)</td>
<td>249.16 (95.68%)</td>
<td>212.98 (94.76%)</td>
</tr>
</tbody>
</table>
256 Nodes

![Bar chart showing average time taken per node (seconds) for different epochs and tasks.]

<table>
<thead>
<tr>
<th>Task</th>
<th>Epoch 1</th>
<th>Epoch 2</th>
<th>Epoch 3</th>
<th>Epoch 4</th>
<th>Epoch 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>11.57 (7.45 %)</td>
<td>2.55 (1.95 %)</td>
<td>2.34 (1.82 %)</td>
<td>2.11 (1.64 %)</td>
<td>2.13 (1.92 %)</td>
</tr>
<tr>
<td>Load File</td>
<td>5.03 (3.24 %)</td>
<td>3.96 (3.03 %)</td>
<td>3.50 (2.72 %)</td>
<td>3.50 (2.71 %)</td>
<td>3.50 (3.16 %)</td>
</tr>
<tr>
<td>Training Iteration</td>
<td>138.78 (89.31 %)</td>
<td>124.17 (95.02 %)</td>
<td>122.51 (95.45 %)</td>
<td>123.37 (95.65 %)</td>
<td>105.15 (94.92 %)</td>
</tr>
</tbody>
</table>
512 Nodes

![Bar chart showing average time taken per node for different epochs and operations.](512_nodes.png)
1024 Nodes

![Average Time Taken Per Node (seconds)]

<table>
<thead>
<tr>
<th>Epoch</th>
<th>Average Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.54 (4.95 %)</td>
</tr>
<tr>
<td>2</td>
<td>0.00 (0.00 %)</td>
</tr>
<tr>
<td>3</td>
<td>0.00 (0.00 %)</td>
</tr>
<tr>
<td>4</td>
<td>0.00 (0.00 %)</td>
</tr>
<tr>
<td>5</td>
<td>0.00 (0.00 %)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>Average Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>2.54 (4.95 %)</td>
</tr>
<tr>
<td>Load File</td>
<td>0.99 (1.93 %)</td>
</tr>
<tr>
<td>Training Iteration</td>
<td>47.77 (93.12 %)</td>
</tr>
</tbody>
</table>
Total Time Scale Out

- 31 -
Read Time Scale Out

![Graph showing Scale Out 5 Epochs Read Time]

- **Average Time Taken Per Node (seconds):**
  - 64: 91.21 (1.00X)
  - 128: 41.87 (1.09X)
  - 256: 20.71 (1.30X)
  - 512: 10.16 (1.12X)
  - 1024: 2.54 (2.25X)

- **Ideal Read Time:**
  - 64: 91.2054123171
  - 128: 45.6027061585
  - 256: 22.8013530793
  - 512: 11.4006765396
  - 1024: 5.70033826982
## Read Bandwidth Scale Out

### Scale Out 3 Epochs Read Bandwidth

<table>
<thead>
<tr>
<th>Count</th>
<th>Read Count</th>
<th>Read Time</th>
<th>Read Bandwidth</th>
<th>Ideal Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>91.2054123171</td>
<td>41.8738560997</td>
<td>20.7105450658</td>
<td>9.91656529215</td>
</tr>
<tr>
<td>128</td>
<td>91.2054123171</td>
<td>41.8738560997</td>
<td>20.7105450658</td>
<td>9.91656529215</td>
</tr>
<tr>
<td>256</td>
<td>91.2054123171</td>
<td>41.8738560997</td>
<td>20.7105450658</td>
<td>9.91656529215</td>
</tr>
<tr>
<td>512</td>
<td>91.2054123171</td>
<td>41.8738560997</td>
<td>20.7105450658</td>
<td>9.91656529215</td>
</tr>
<tr>
<td>1024</td>
<td>91.2054123171</td>
<td>41.8738560997</td>
<td>20.7105450658</td>
<td>9.91656529215</td>
</tr>
</tbody>
</table>

Bandwidth (GB/s)
Outline

- HEP Benchmark
  - Time Breakdown and Bandwidth
- Climate Data Benchmark
  - Data Format
    - Time Breakdown and Bandwidth
- Future Exploration Scopes
Outline

- HEP Benchmark
  - Time Breakdown and Bandwidth
- Climate Data Benchmark
  - Data Format
    - Time Breakdown and Bandwidth
- Future Exploration Scopes
64 Nodes
128 Nodes

![Graph showing Average Time Taken Per Node (seconds) for different epochs and read operations.]

<table>
<thead>
<tr>
<th></th>
<th>Asynchronous Read</th>
<th>Epoch 1</th>
<th>Epoch 2</th>
<th>Epoch 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>313.76 (100.00 %)</td>
<td>0.00 (0.00 %)</td>
<td>0.00 (0.00 %)</td>
<td>0.00 (0.00 %)</td>
</tr>
<tr>
<td>Training Iteration</td>
<td>0.00 (0.00 %)</td>
<td>2169.07 (100.00 %)</td>
<td>2153.28 (100.00 %)</td>
<td>2141.81 (100.00 %)</td>
</tr>
</tbody>
</table>
256 Nodes
### 512 Nodes

![Bar Chart]

<table>
<thead>
<tr>
<th></th>
<th>Asynchronous Read</th>
<th>Epoch 1</th>
<th>Epoch 2</th>
<th>Epoch 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>188.77 (100%)</td>
<td>0.00 (0%)</td>
<td>0.00 (0%)</td>
<td>0.00 (0%)</td>
</tr>
<tr>
<td><strong>Training Iteration</strong></td>
<td>0.00 (0%)</td>
<td>626.99 (100%)</td>
<td>599.85 (100%)</td>
<td>598.40 (100%)</td>
</tr>
</tbody>
</table>
Total Time Scale Out

Scale Out 3 Epochs

Average Time Taken Per Node (seconds)

<table>
<thead>
<tr>
<th></th>
<th>Read</th>
<th>Training Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>3640.89 (22.01%)</td>
<td>12898.48 (77.99%)</td>
</tr>
<tr>
<td>128</td>
<td>313.76 (4.63%)</td>
<td>6464.17 (95.37%)</td>
</tr>
<tr>
<td>256</td>
<td>160.26 (4.59%)</td>
<td>3329.12 (95.41%)</td>
</tr>
<tr>
<td>512</td>
<td>188.77 (9.37%)</td>
<td>1825.23 (90.63%)</td>
</tr>
</tbody>
</table>
### Read Time Scale Out

The graph illustrates the change in read time as the scale out increases. The read time decreases significantly as the number of nodes increases from 64 to 512.

<table>
<thead>
<tr>
<th>Scale Out (Nodes)</th>
<th>Read Time (seconds)</th>
<th>(X Factor)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>3640.89</td>
<td>1.00X</td>
</tr>
<tr>
<td>128</td>
<td>313.76 (5.80X)</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>160.26 (5.68X)</td>
<td></td>
</tr>
<tr>
<td>512</td>
<td>188.77 (2.41X)</td>
<td></td>
</tr>
</tbody>
</table>

The ideal read times are also shown:

<table>
<thead>
<tr>
<th>Scale Out (Nodes)</th>
<th>Ideal Read Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>3640.8909441</td>
</tr>
<tr>
<td>128</td>
<td>1820.4455472</td>
</tr>
<tr>
<td>256</td>
<td>910.222773602</td>
</tr>
<tr>
<td>512</td>
<td>455.111386801</td>
</tr>
</tbody>
</table>
Read Bandwidth Scale Out

![Graph showing read bandwidth scale out](image)
Outline

- HEP Benchmark
  - Time Breakdown and Bandwidth
- Climate Data Benchmark
  - Data Format
  - Time Breakdown and Bandwidth
- Future Exploration Scopes
Next Exploration Scopes

- Added Flag to Ignore Training
  - Can Try to Run the Tests with Training Disabled
- Added Provision to Test Checkpointing by *TimeLogger*
  - Can Try to Run the Tests with this feature
- Have to Perform Deeper Analysis on the Bandwidth for Parallel IO in Climate Data Benchmark
- To Have a Look into the *PyTorch Benchmark*
- To Try Running the Climate Data Tests Using Burst Buffer