H5Spark: Bridging the I/O Gap between Spark and Scientific Data Formats on HPC Systems

Jialin Liu, Evan Racah, Quincey Koziol, Richard Shane Canon, Alex Gittens, Lisa Gerhardt, Suren Byna, Mike F. Ringenburg, Prabhat

Jialnliu@lbl.gov
National Energy Research Scientific Computing Center(NERSC)
H5Spark: Outline

- Introduction, Spark
- Motivation
- H5Spark Design
- H5Spark Evaluation
- H5Spark Future
Apache Spark is an open source cluster computing framework

- Developed at UCB AMPLab, 2014 v1.0, 2016 v2.0
  - Actively developed, 1000+ contributors in 2015
- Productive programming interface
  - 6 vs 28 lines of code compare to hadoop mapreduce
- Implicit data parallelism
- Fault-tolerance

Spark for Data-intensive Computing

- Streaming processing
- SQL
- Machine learning, MLlib
- Graph processing
H5Spark: Porting Spark onto HPC

• Advantages of Porting Spark onto HPC
  – A more productive API for data-intensive computing
  – Relieve the users from concurrency control, communication and memory management with traditional MPI model.
  – Embarrassingly parallel computing, \texttt{data.map(f)}
  – Fault tolerance, \texttt{recompute()}

• But Scientific Data Formats in HPC not Supported
  – HDF5/ netCDF are among the top 5 libraries at NERSC, 2015
    • 750+ unique users @NERSC, million of users worldwide
  – 1987, NCSA&UIUC. NASA send HDF-EOS to 2.4 millions end users
  – Hierarchical data organization
  – Parallel I/O
H5Spark: Data in Spark

• RDD: Resilient Distributed Datasets
  – Read-only, partitioned collection of records in Spark
  – RDD can contain any type of Python/Java/Scala objects
  – Fault Tolerant

• Transformations on RDD
  – Filter, map, join, etc

• Actions on RDD
  – Reduce, collect, etc

• Spark operations are lazy

• RDD allows in-memory processing
  – rdd.cache() or rdd.persist()
  – Good for iterative or interactive processing
H5Spark: Data in Spark

```
RDD[String]
Hello
Hello
Hello
welcome
to
Spark
...

.map(word => (word, 1))

RDD[(String, Int)]
Hello 1
Hello 1
Hello 1
welcome 1
to 1
Spark 1
...
```
H5Spark: Data in HDF5

- Hierarchical Data Format v5
H5Spark: Support HDF5 in Spark

• What does Spark have in reading various data formats?
  – Textfile, sc.textFile()
  – Parquet, sc.read.parquet()
  – Json, sc.read.json()
  – HDF5, sc.read.hdf5()

• Challenges: Functionality and Performance
  – How to transform an HDF5 dataset into an RDD?
  – How to utilize the HDF5 I/O libraries in Spark?
  – How to enable parallel I/O on HPC?
  – What is the impact of Lustre striping?
  – What is the effect of caching on IO in Spark?
H5Spark: Software Overview

- Scala/Python implementation
  - Spark favors Scala and Python
  - H5Spark uses HDF5 java library
  - Underneath is HDF5 C posix library
  - No MPIIO support
- H5Spark as a standalone package
  - Users can load it in their Spark applications
  - H5Spark module on Cori
  - sbt package------> h5spark_2.10-1.0.jar
- Open source
  - Github: https://github.com/valiantljk/h5spark
H5Spark: Design

- RDD Seeder
- Metadata Analyzer
- Hyperslab Partitioner
- RDD Constructor

Lustre File System

- HDF5 Group
- Dataset

RDD

CUG, May 2016
**H5Spark: From HDF5 to RDD**

- **Input:**

<table>
<thead>
<tr>
<th>HDF5 File Path:</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset Name:</td>
<td>( v )</td>
</tr>
<tr>
<td>SparkContext:</td>
<td>( sc )</td>
</tr>
<tr>
<td>*Spark Partition:</td>
<td>( p )</td>
</tr>
</tbody>
</table>

*Spark Partition determines the degree of parallelism = MPI processes + OpenMP

\( p > \text{num of cores} \)

- **Output:** RDD: \( r \)

- **Under the Hood:** reading HDF5 into RDD
  
  - Adjust partitions \( p = p > \text{dim}[sid] \? \text{dim}[sid]:p \)
  
  - Determine hyperslab \( \text{offset}[i]=\text{dim}[sid]/p * i \)
  
  - Seed RDD \( r\_\text{seed} = \text{sc}.\text{parallelize}(\text{offset, } p) \)
  
  - Perform parallel I/O \( r\_\text{seed}.\text{flatmap}(h5\text{read}(f,v)) \)
## H5Spark: How to Use

- **H5Spark APIs**

<table>
<thead>
<tr>
<th>Function</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>h5read</td>
<td>A RDD of double array</td>
</tr>
<tr>
<td>h5read_point</td>
<td>A RDD of (key, value) pair</td>
</tr>
<tr>
<td>h5read_vec</td>
<td>A RDD of vector</td>
</tr>
<tr>
<td>h5read_irow</td>
<td>A RDD of indexed row</td>
</tr>
<tr>
<td>H5read_imat</td>
<td>A RDD of indexed row matrix</td>
</tr>
</tbody>
</table>

- **Correspond to Spark MLlib interface**

```java
import org.apache.spark.mllib.linalg
```

Data Type: Vector, labeled point, matrix, indexedrowmatrix, etc
H5Spark: How to Use

• Sample codes, H5Spark vs MPI

1. `val sc = new SparkContext()
2. val rdd = h5read (sc, f, v, p)
3. sc.stop()`

H5Spark Parallel Read

---

1. `MPI_Init(&argc, &argv);`
2. `MPI_Comm_size(comm, &mpi_size);`
3. `MPI_Comm_rank(comm, &mpi_rank);`
4. `hid_t fapl = H5Pcreate(H5P_FILE_ACCESS);`
5. `H5Pset_fapl_mpio(fapl, comm, info);`
6. `file= H5Fopen(f, H5F_ACC_RDONLY, fapl);`
7. `dataset= H5Dopen(file, v, H5P_DEFAULT);`
8. `hid_t dataspace = H5Dget_space(dataset);`
9. `hsize_t offset[rank];`
10. `hsize_t count[rank];`
11. `hsize_t rest = dims_out[0] % mpi_size;`
12. `if(mpi_rank != (mpi_size - 1)){`
13. `count[0] = dims_out[0]/mpi_size;`
14. }else{`
15. `count[0] = dims_out[0]/mpi_size + rest;`
16. }
17. `offset[0] = dims_out[0]/mpi_size * mpi_rank;`
18. `for(i=1; i<rank; i++){`
19. `offset[i] = 0;`
20. `count[i] = dims_out[i];`
21. }
22. `hid_t hyperid=H5Sselect_hyperslab(dataspace,`
23. `H5S_SELECT_SET, offset, NULL, count, NULL);`
24. `hsize_t rankmemsize=1;`
25. `for(i=0; i<rank; i++) rankmemsize*=count[i];`
26. `hid_t memspace = H5Screate_simple(rank,count,NULL);`
27. `double * data_t=(double *)malloc(sizeof(double)*rankmemsize);`
28. `H5Dread(dataset, H5T_NATIVE_DOUBLE, memspace,`
29. `dataspace, H5P_DEFAULT, data_t);`
30. `MPI_Finalize()`

MPI Parallel Read
H5Spark: Evaluation

• About the System
  – Cori, Phase 1, Cray XC40 supercomputer, 1600 compute nodes, 248 Lustre OSTs
  – Each compute node has 32 cores with 128 GB RAM in total. The peak I/O bandwidth is 700GB/s.

• Experimental Setup
  – PCA on 2.2 TB global ocean temperature data, 16 TB CAM5 atmosphere data.
  – 2.2TB, 16 TB, HDF5 format, Double precision
  – Number of nodes: 45, 90, 135, 1600
  – Stripe counts: 1, 8, 24, 72, 144, 248

CAM5, 16TB, Finding the principal causes of variability in large scale 3D fields.
H5Spark: Evaluation

- Scaling/Profiling H5Spark with Lustre Striping
  - 45 nodes, 1440 cores, 3000 partitions, 2.2TB data, 1MB stripe size

H5Spark Bandwidth, 2.2TB Climate Data

Spark Tasks Launch Delay and GC Cost

I/O Bandwidth with Lustre Striping

OST should be another factor in Spark’s scheduling besides CPU/Memory
H5Spark: Evaluation

- Scaling H5Spark with Partitions
  - 45 nodes, 2.2TB

The number of partitions can be tuned, based on the workloads and resources.
Scaling H5Spark with Executors and/or Partitions

- 2.2TB, 45,95,135 nodes

**Lesson:** Increase the number of Executors and Partitions at the same time.
H5Spark: Evaluation

- H5Spark has been tested at full scale on Cori phase 1

<table>
<thead>
<tr>
<th>Tests</th>
<th>Size(TB)</th>
<th>I/O(s)</th>
<th>B/W(GB/s)</th>
<th>OSTs</th>
<th>Executors</th>
<th>Partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>135 nodes</td>
<td>2.2</td>
<td>37</td>
<td>59.7</td>
<td>144</td>
<td>135</td>
<td>9000</td>
</tr>
<tr>
<td>Full scale</td>
<td>16</td>
<td>120</td>
<td>136.5</td>
<td>144</td>
<td>1522</td>
<td>52100</td>
</tr>
</tbody>
</table>
H5Spark: Evaluation

- H5Spark Python vs Scala

<table>
<thead>
<tr>
<th>Version</th>
<th>I/O(s)</th>
<th>B/W(GB/s)</th>
<th>Speedup</th>
<th>Mem(GB)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>162</td>
<td>13.65</td>
<td>1</td>
<td>479</td>
<td>1</td>
</tr>
<tr>
<td>Scala</td>
<td>90</td>
<td>24.56</td>
<td>1.8</td>
<td>2210</td>
<td>4.61</td>
</tr>
</tbody>
</table>

Scala is faster than Python
H5Spark: Evaluation

- H5Spark vs MPI-IO

MPI scales well with OSTs
H5Spark scales well with Nodes (while MPI saturates the I/O)
Again: Storage on HPC is an important scheduling factor

Partitions are also increased
H5Spark: Evaluation

- H5Spark@LBNL vs SciSpark@NASA
  - https://github.com/valiantljk/h5spark

Architecture of SciSpark

H5Spark vs SciSpark
H5Spark: Conclusion & Future Work

• H5Spark:
  – An efficient HDF5 file loader for Spark
  – Users can now use Spark perform big data analysis on HDF5 data
  – H5Spark gets closer to MPIIO

• H5Spark Future
  – Spark I/O finer profiling/ lazy evaluation
  – Parallel write/filter
  – Storage-aware scheduling
Thanks SciSpark Team
Thank Douglas Jacobsen@NERSC,
Jey Kottaalam@Amplab