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

Jalnliu@lbl.gov
National Energy Research Scientific Computing Center(NERSC)

CUG, May 2016

H5Spark: Outline

- Introduction, Spark
- Motivation
- H5Spark Design
- H5Spark Evaluation
- H5Spark Future

H5Spark: Big Data Analytics, Spark

- 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

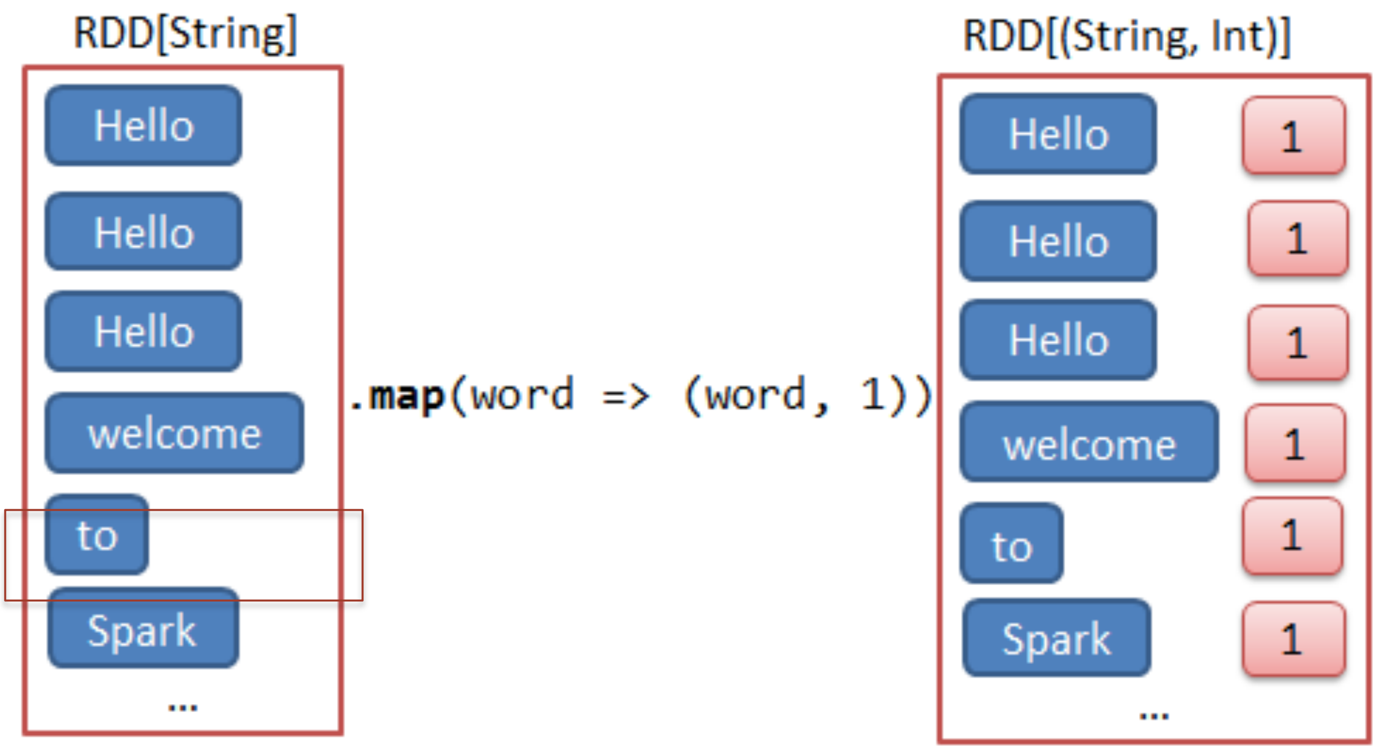
- Advantages of Porting Spark onto HPC
  - A more productive API for data-intensive computing
  - Relieve the users from concurrency control, communication and memory management with traditional MPI model.
  - Embarrassingly parallel computing, *data.map(f)*
  - Fault tolerance, *recompute()*

- 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

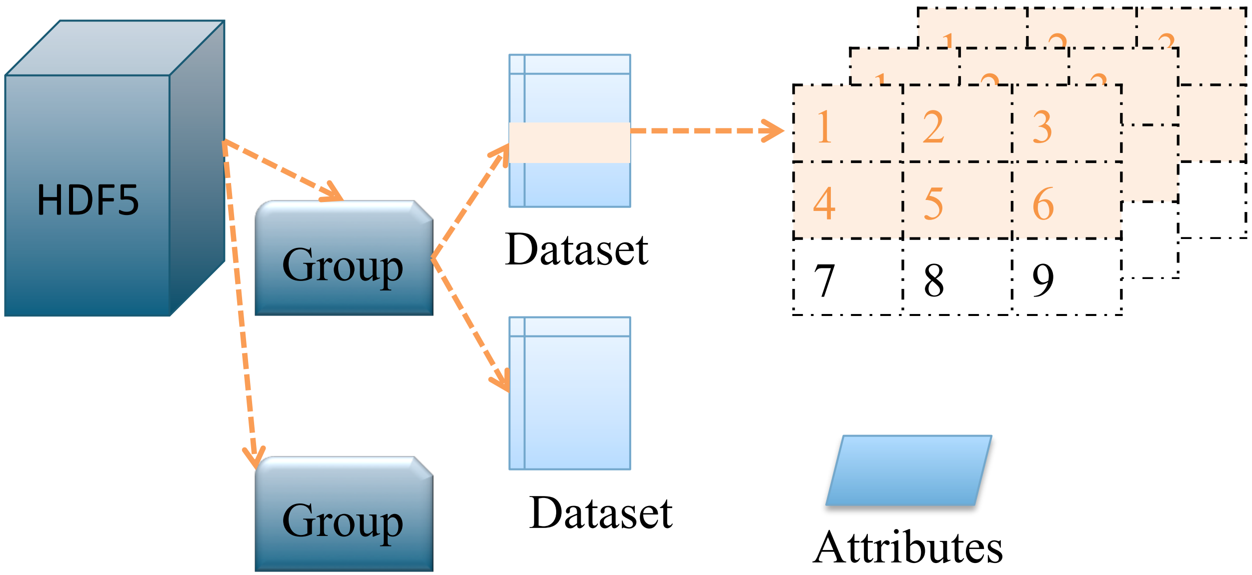
- RDD: Resilient Distributed Datasets
  - Read-only, partitioned collection of records in Spark
  - RDD can contain any type of Python/Java/Scala objects
  - Fault Tolerant
- Transformations on RDD
  - Filter, map, join, etc
- Actions on RDD
  - Reduce, collect, etc
- Spark operations are lazy
- RDD allows in-memory processing
  - rdd.cache() or rdd.persist()
  - Good for iterative or interactive processing

H5Spark: Data in Spark



H5Spark: Data in HDF5

- Hierarchical Data Format v5



H5Spark: Support HDF5 in Spark

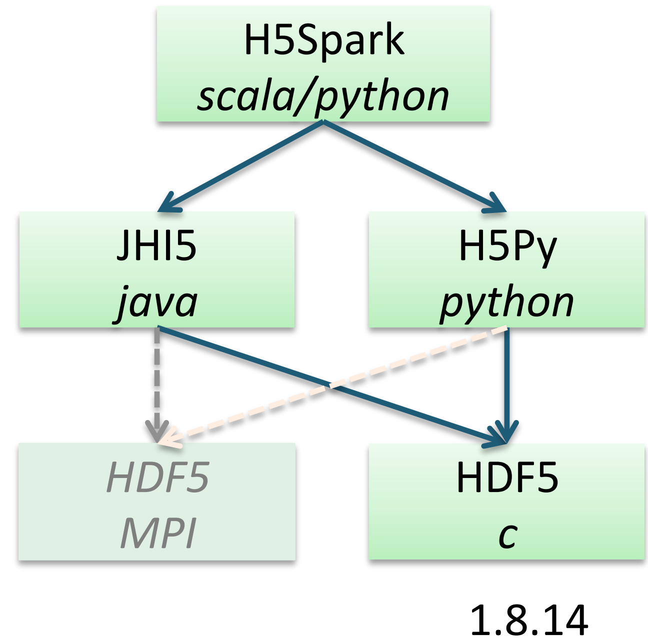
- What does Spark have in reading various data formats?
  - Textfile, sc.textFile()
  - Parquet, sc.read.parquet()
  - Json, sc.read.json()
  - **HDF5, sc.read.hdf5()**

HDF5 → Parquet?

- 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?

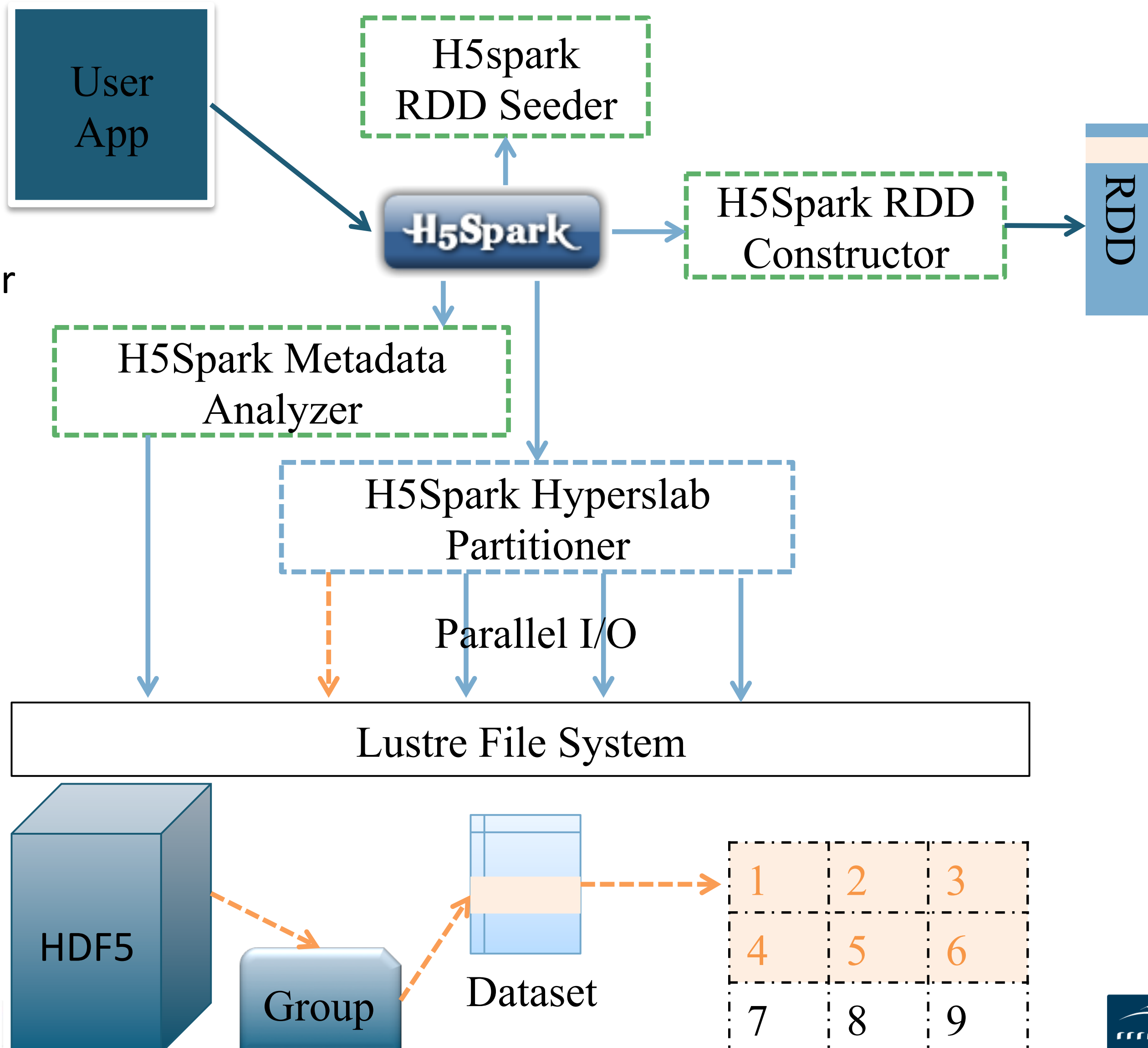
H5Spark: Software Overview

- 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



H5Spark: From HDF5 to RDD

- **Input:**

HDF5 File Path:	f
Dataset Name:	v
SparkContext:	sc
*Spark Partition:	p

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

+OpenMP

p > num of cores

- **Output**: RDD: ***r***
- **Under the Hood**: reading HDF5 into RDD
  - Adjust partitions *p= p > dim[sid] ? dim[sid]:p*
  - Determine hyperslab *offset[i]=dim[sid]/p * i*
  - Seed RDD *r_seed = sc.parallelize(offset, p)*
  - Perform parallel I/O *r_seed.flatmap(h5read(f,v))*

H5Spark: How to Use

- H5Spark APIs

Input: sc, f, v, p	
Functions	Output
h5read	A RDD of double array
h5read_point	A RDD of (key, value) pair
h5read_vec	A RDD of vector
h5read_irow	A RDD of indexed row
H5read_imat	A RDD of indexed row matrix

- Correspond to Spark MLlib interface

  import org.apache.spark.mllib.linalg

    DataType: Vector, labeled point, matrix, indexedrowmatrix, etc

H5Spark: How to Use

- Sample codes, H5Spark vs MPI

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

H5Spark Parallel Read

=

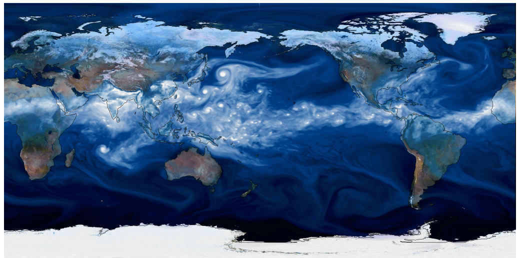
```
MPI_Init(&argc, &argv);
MPI_Comm_size(comm, &mpi_size);
MPI_Comm_rank(comm, &mpi_rank);
hid_t fapl = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(fapl, comm, info);
file= H5Fopen(f, H5F_ACC_RDONLY, fapl);
dataset= H5Dopen(file, v, H5P_DEFAULT);
hid_t dataspace = H5Dget_space(d
hsize_t offset[rank];
hsize_t count[rank];
hsize_t rest = dims_out[0] % m
if(mpi_rank != (mpi_size -1)){
  count[0] = dims_out[0]/mpi_size;
}else{
  count[0] = dims_out[0]/mpi_size + rest;
}
offset[0] = dims_out[0]/mpi_size * mpi_rank;
for(i=1; i<rank; i++){
 offset[i] = 0;
 count[i] = dims_out[i];
}
hid_t hyperid=H5Sselect_hyperslab(dataspace,
                   H5S_SELECT_SET, offset, NULL, count, NULL);
hsize_t rankmemsize=1;
for(i=0; i<rank; i++)   rankmemsize*=count[i];
hid_t memspace = H5Screate_simple(rank,count,NULL);
double * data_t=(double *)malloc(sizeof(double)*rankmemsize);
H5Dread(dataset, H5T_NATIVE_DOUBLE, memspace,
       dataspace, H5P_DEFAULT, data_t);
MPI_Finalize()
```

Parallelism

MPI Parallel Read

H5Spark: Evaluation

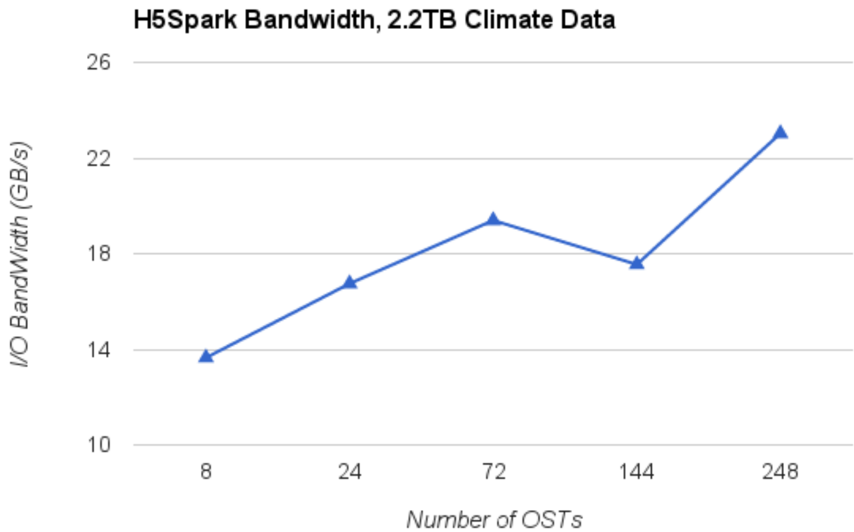
- About the System
  - Cori, Phase 1, Cray XC40 supercomputer, 1600 compute nodes, 248 Lustre OSTs
  - Each compute node has 32 cores with 128 GB RAM in total. The peak I/O bandwidth is 700GB/s.
- Experimental Setup
  - PCA on 2.2 TB global ocean temperature data, 16 TB CAM5 atmosphere data.
  - 2.2TB, 16 TB, HDF5 format, Double precision
  - Number of nodes: 45, 90, 135, 1600
  - Stripe counts: 1, 8, 24, 72, 144, 248



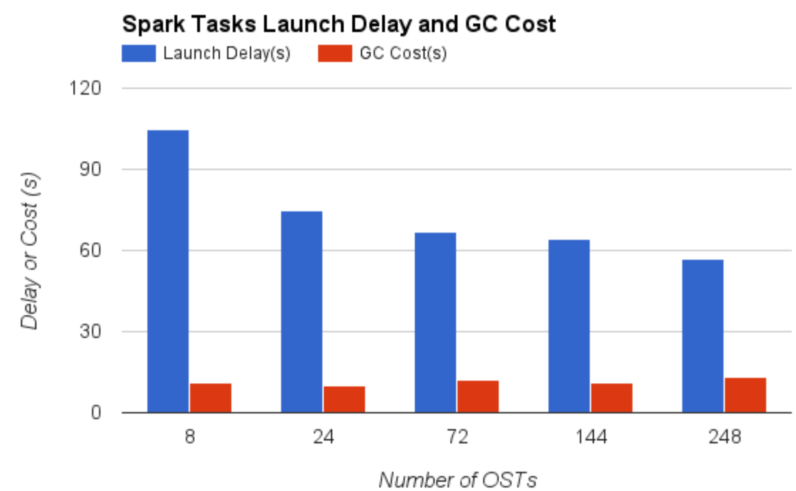
CAM5, 16TB, Finding the principal causes of variability in large scale 3D fields.

H5Spark: Evaluation

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



I/O Bandwidth with Lustre Striping

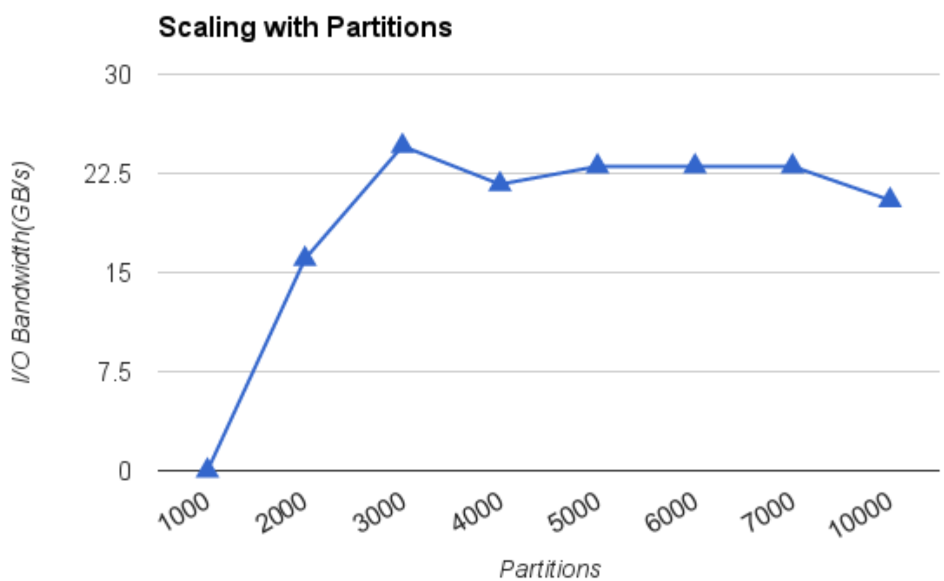


H5Spark Tasks Launching Delay

OST should be another factor in Spark's scheduling besides CPU/Memory

H5Spark: Evaluation

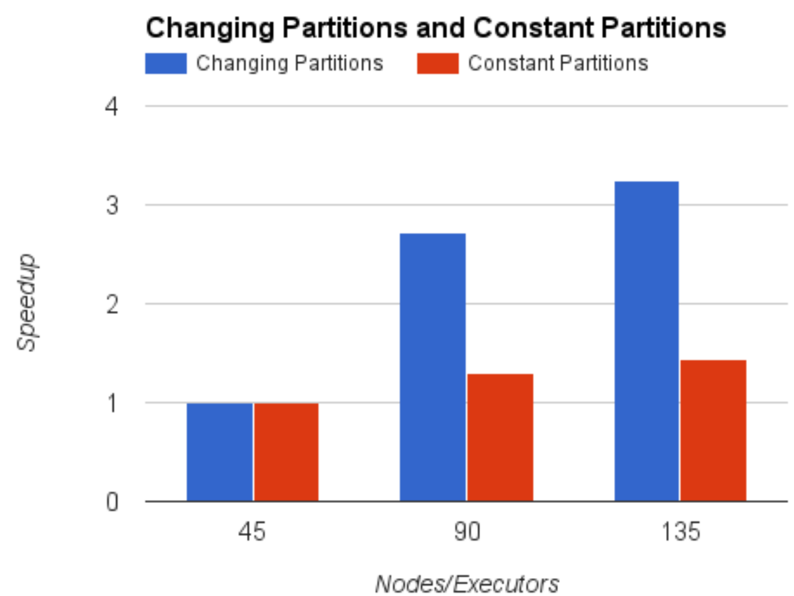
- Scaling H5Spark with Partitions
  - 45 nodes, 2.2TB



The number of partitions can be tuned, based on the workloads and resources

H5Spark: Evaluation

- 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

Tests	Size(TB)	I/O(s)	B/W(GB/s)	OSTs	Executors	Partitions
135 nodes	2.2	37	59.7	144	135	9000
Full scale	16	120	136.5	144	1522	52100

H5Spark: Evaluation

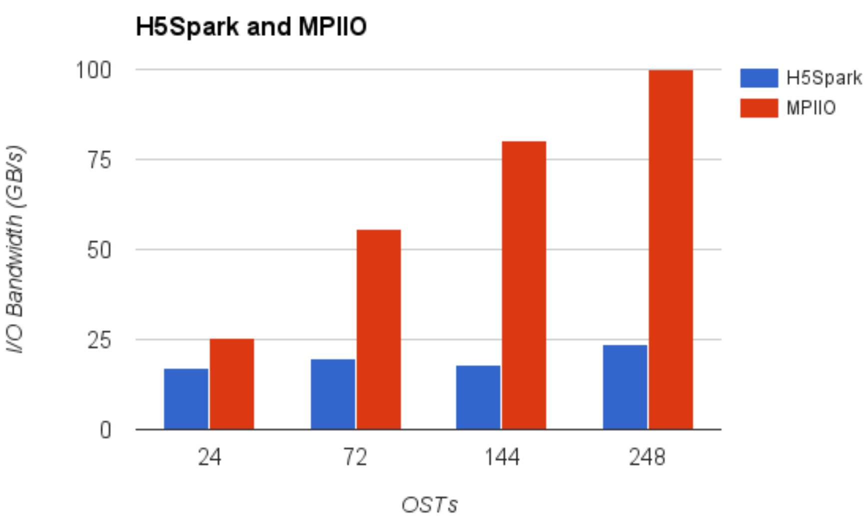
- H5Spark Python vs Scala

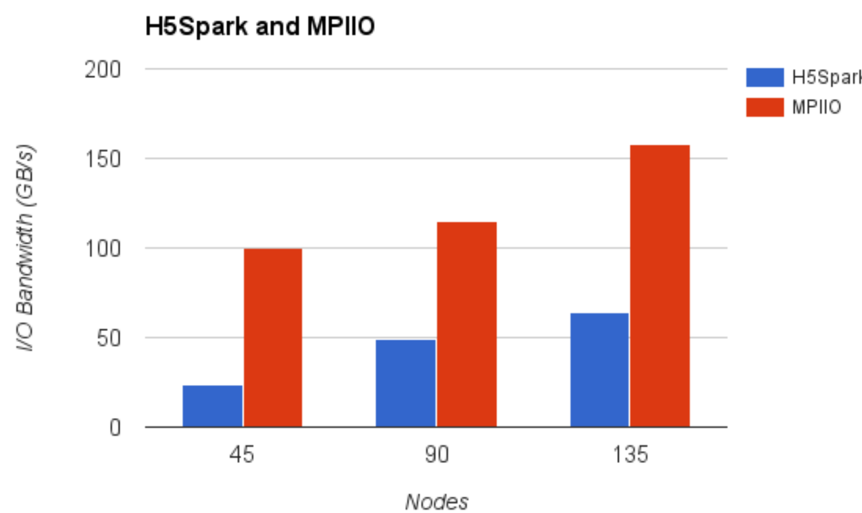
Version	I/O(s)	B/W(GB/s)	Speedup	Mem(GB)	Ratio
Python	162	13.65	1	479	1
Scala	90	24.56	1.8	2210	4.61

Scala is faster than Python

H5Spark: Evaluation

- H5Spark vs MPI-IO



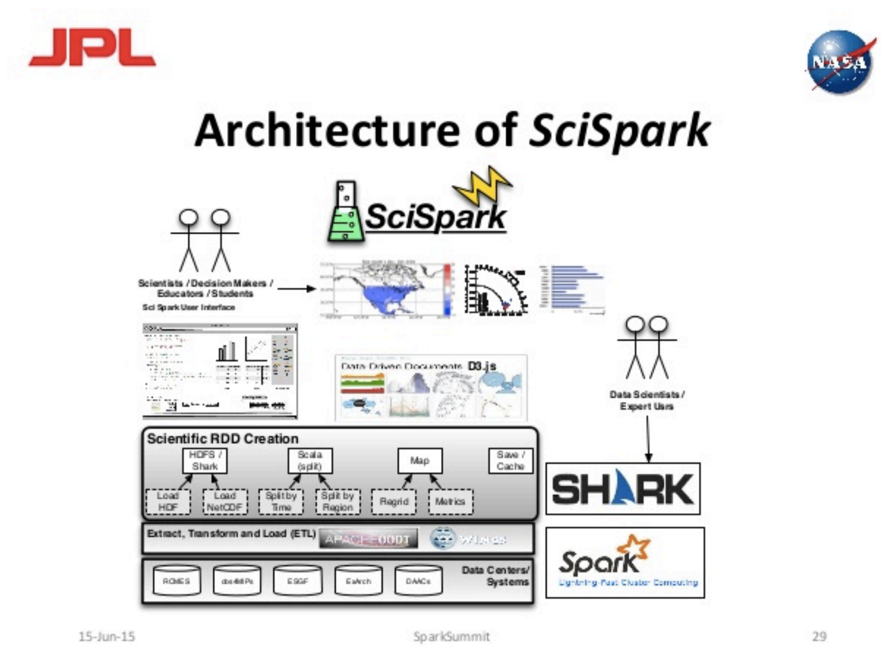


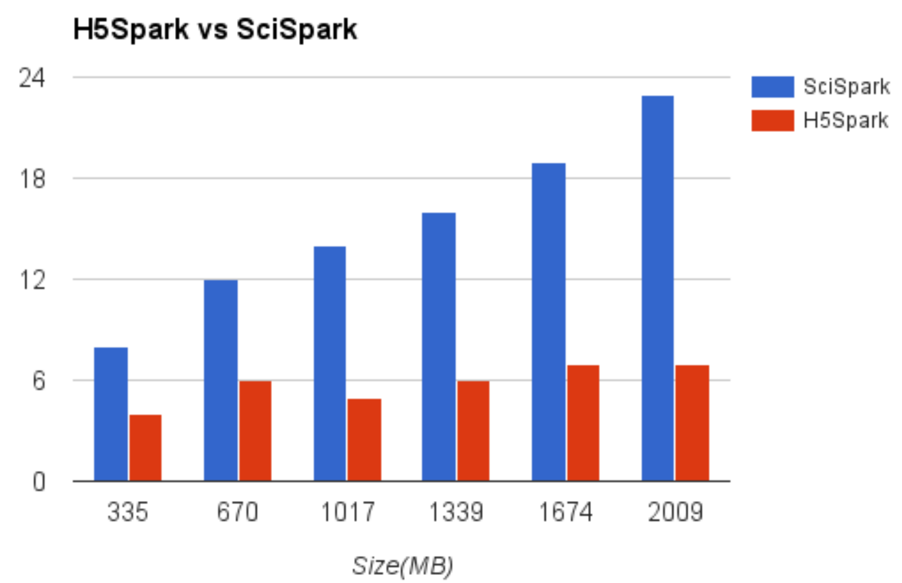
Partitions are also increased

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

H5Spark: Evaluation

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





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