Building Near-Real-Time Processing Pipelines with the Spark-MPI Platform

Nikolay Malitsky, Aashish Chaudhary, Patrick O'Leary, Matt Cowan, Sebastien Joudain, Marcus Hanwell, and Kerstin Kleese Van Dam

Kitware

New York Scientific Data Summit: Data-Driven Discovery

August 7, 2017

BROOKHAVEN NATIONAL LABORATORY

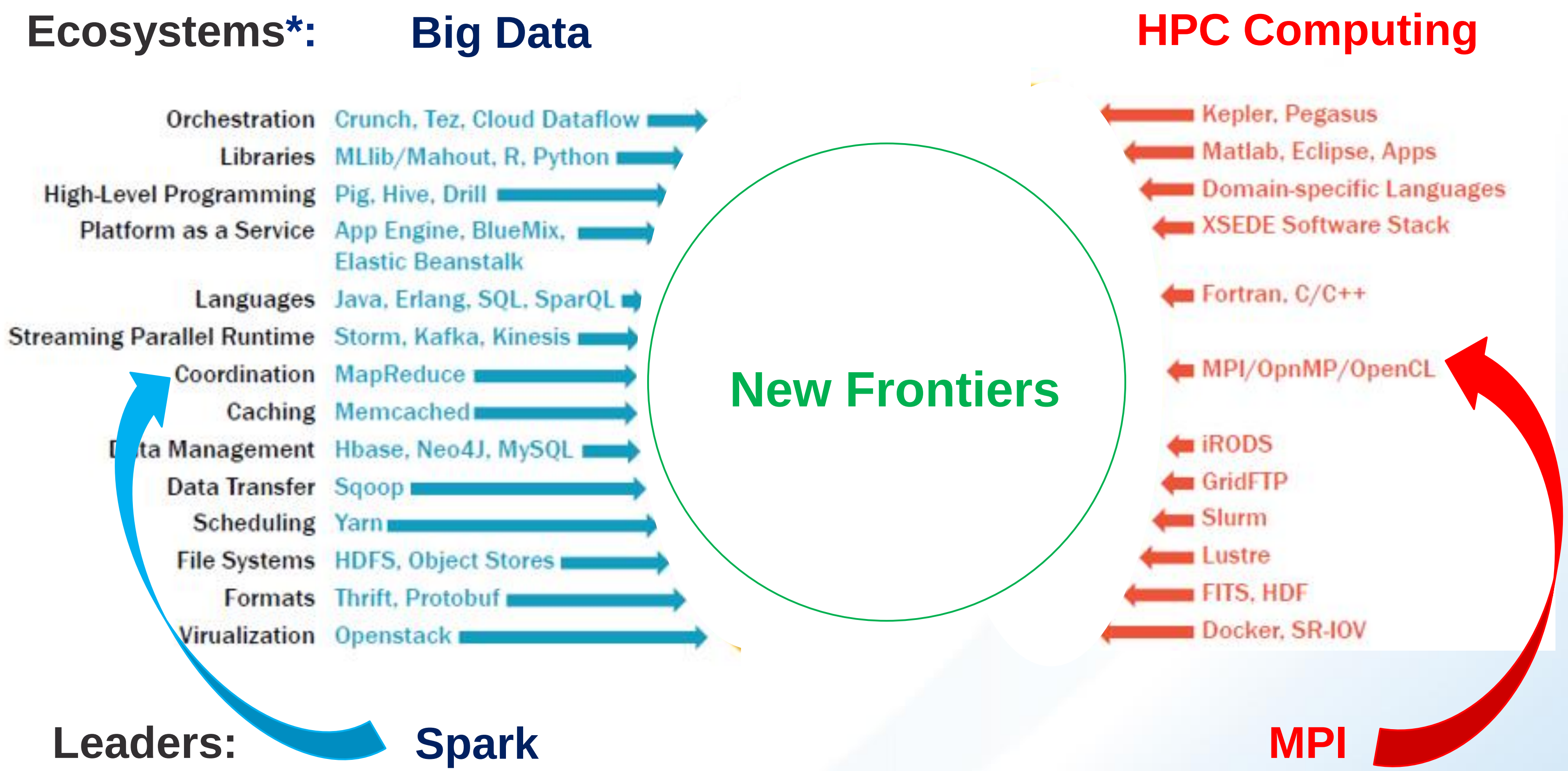
a passion for discovery

U.S. DEPARTMENT OF ENERGY | Office of Science

Outline

- Rationale
- Spark-MPI approach
- Ptychographic application
- Tomographic application
- Future directions

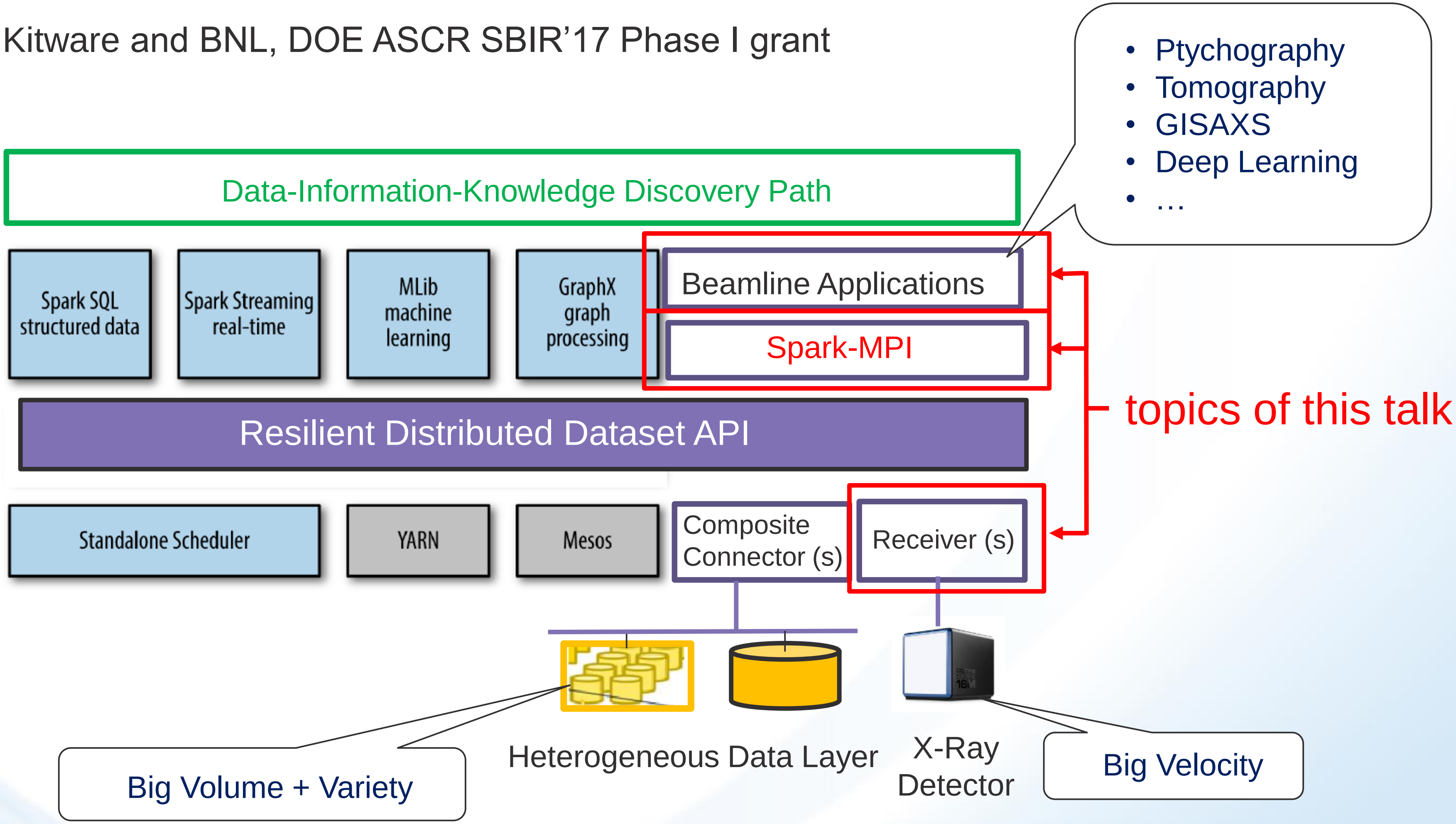
Closing a Gap between Big Data and HPC Computing



*Geoffrey Fox et al. HPC-ABDC High Performance Computing Enhanced Apache Big Data Stack, CCGrid, 2015

In Situ, Streaming, Data- and Compute-Intensive Platform for Experimental Data

Kitware and BNL, DOE ASCR SBIR'17 Phase I grant



BROOKHAVEN NATIONAL LABORATORY

Approaching the Fifth Paradigm of Cognitive Applications

by consolidating Big Data frameworks on HPC clusters

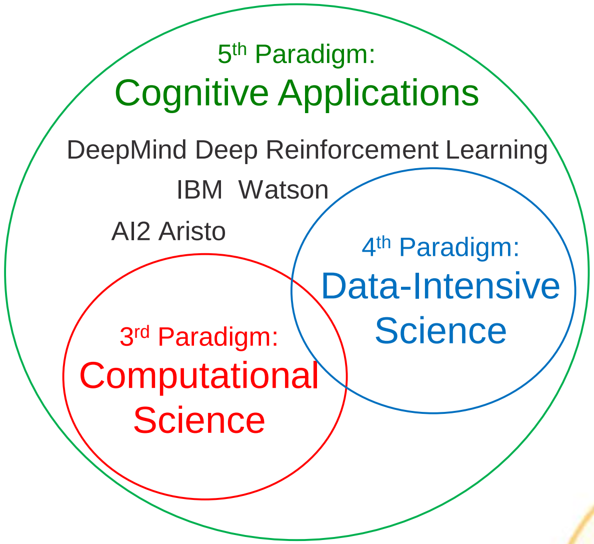


Figure 1: The Fifth Paradigm[1]

- The consolidation of the HPC and Big Data machine learning technologies represents the prerequisite for developing the next paradigm of cognitive applications

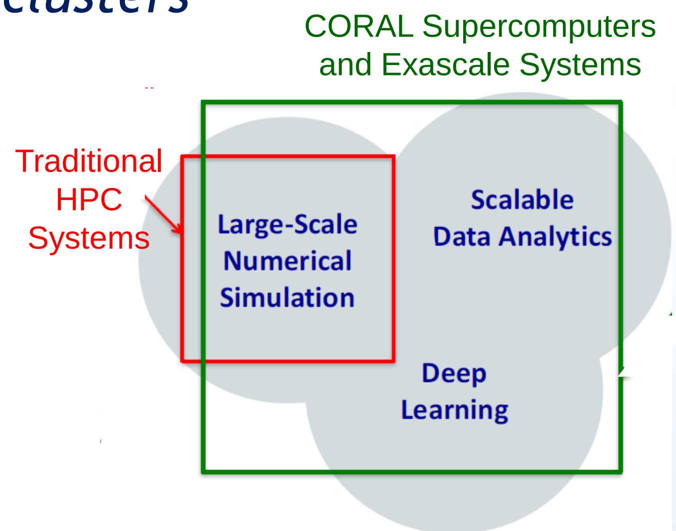


Figure 2: Integration of Simulation, Data Analytics, and Machine Learning[2]

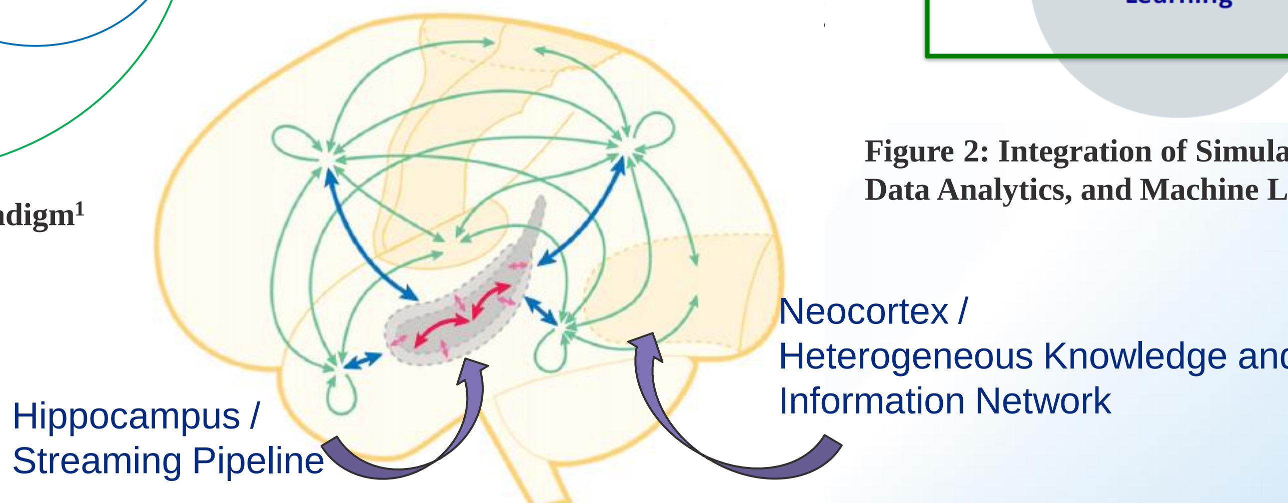


Figure 3: Complementary Learning Systems[3]

---

(1) Nikolay Malitsky, Approaching the Fifth Paradigm, Future Online Analysis Platform Workshop, 2017

(2) Rick Stevens, Deep Learning in Cancer: Example for BDEC, BDEC, 2017

(3) Dharshan Kumaran, Demis Hassabis, and James L. McClelland, What Learning Systems do Intelligent Agents Need ? Complementary Learning Systems, Trends in Cognitive Sciences, 2016

SPARK-MPI APPROACH

Spark-MPI AllReduce Demo

https://github.com/SciDriver/spark-mpi/tree/master/examples/spark/

Create the rdd collection associated with the MPI workers

```
rdd = sc.parallelize(env, partitions)
```

Define the MPI application

PMI Server variables

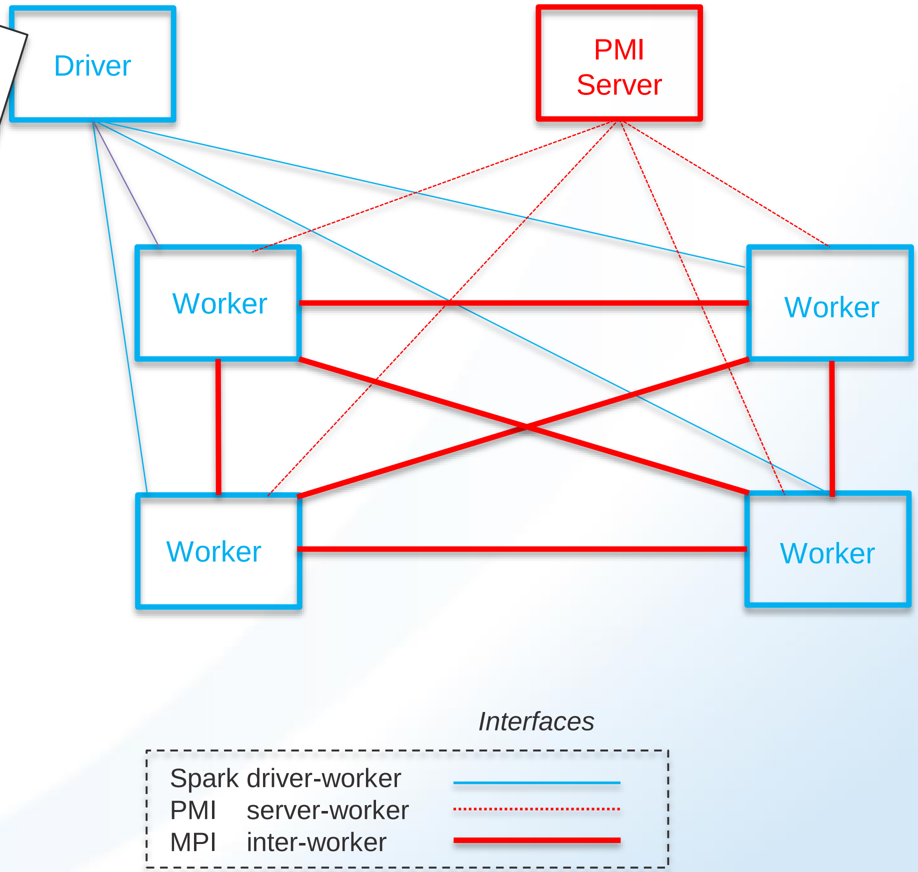
```
def allreduce(kvs):

    os.environ["PMI_PORT"] = kvs["PMI_PORT"]
    os.environ["PMI_ID"] = str(kvs["PMI_ID"])

    from mpi4py import MPI

    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()

    # image

    n = 2*1000000
    sendbuf = np.arange(n, dtype=np.float32)
    recvbuf = np.arange(n, dtype=np.float32)

    sendbuf[n-1] = 5.0;

    t1 = datetime.now()
    comm.Allreduce(sendbuf, recvbuf, op=MPI.SUM)
    t2 = datetime.now()

    out = {
       'rank' : rank,
       'time' : (t2-t1),
       'sum'  : recvbuf[n-1]
    }
    return out
```

MPI interface

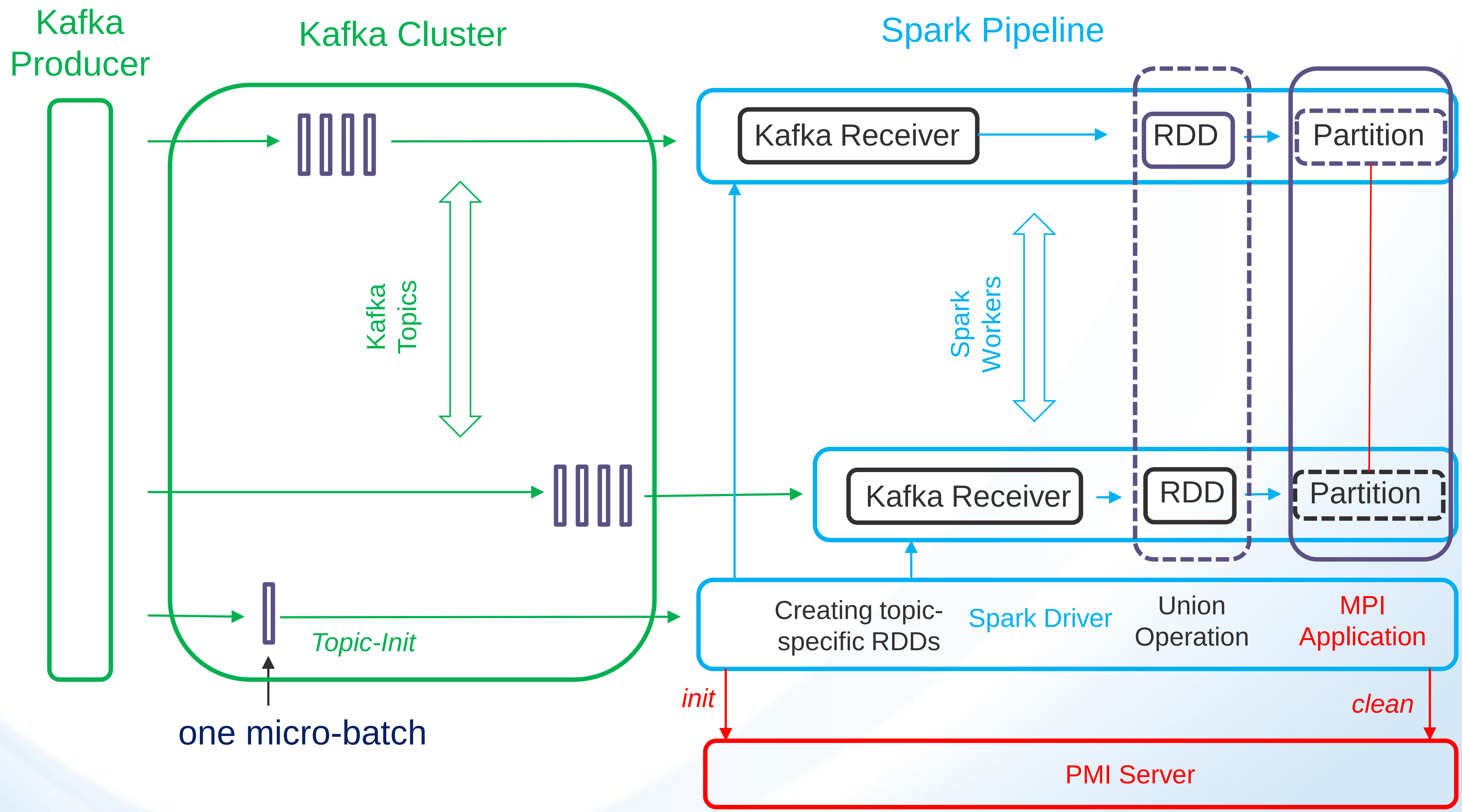
Run MPI application on Spark workers and collect the results

```
results = rdd.map(allreduce).collect()
for out in results:
    print ("rank: ", out['rank'], ", sum: ", out['sum'], ", 
```

```
rank:  0 , sum:  20.0 , processing time:  0:00:00.014500
rank:  1 , sum:  20.0 , processing time:  0:00:00.015380
rank:  2 , sum:  20.0 , processing time:  0:00:00.014479
rank:  3 , sum:  20.0 , processing time:  0:00:00.015245
```



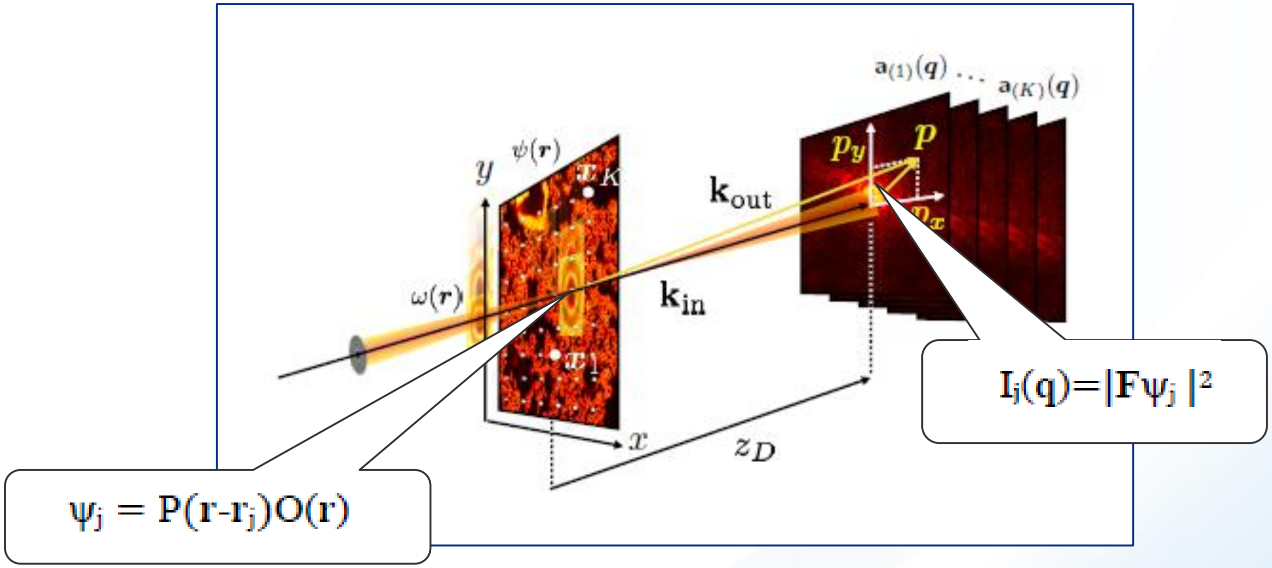
Streaming Demo with the Kafka Streaming Platform and Spark-MPI



PTYCHOGRAPHIC APPLICATION

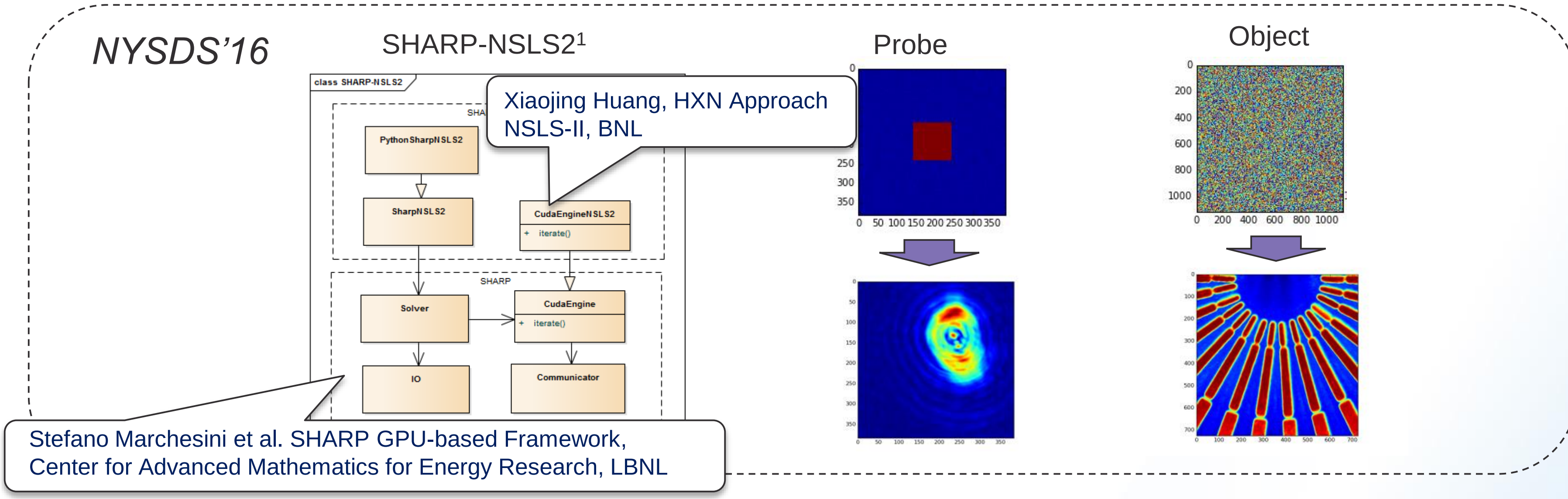
Ptychography

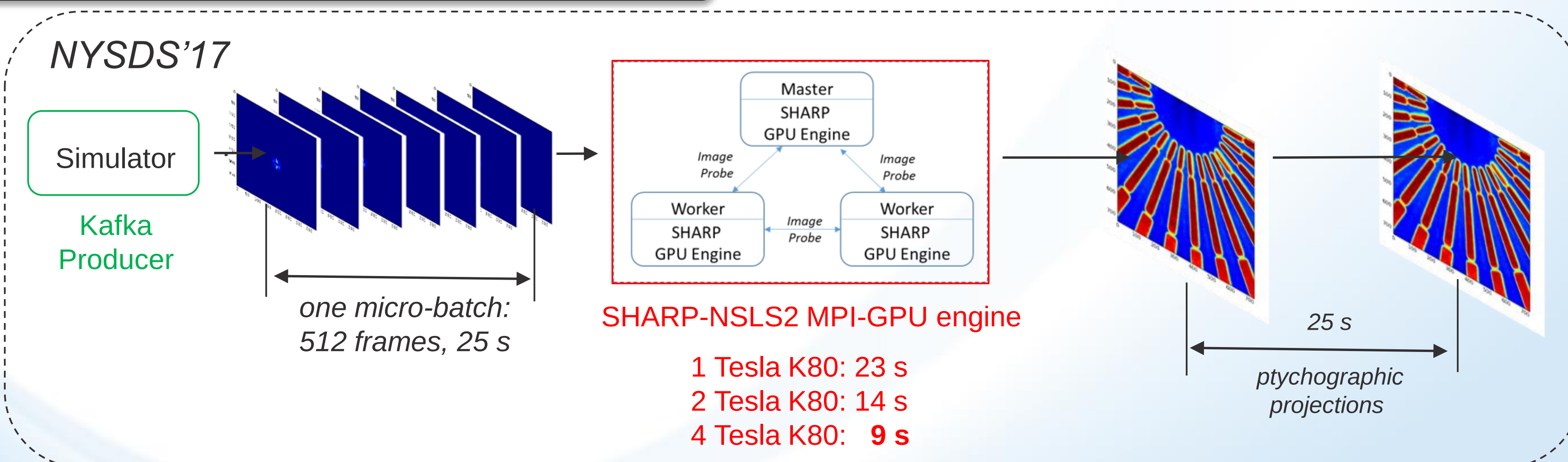
Ptychography is one of the essential image reconstruction techniques used in light source facilities. This method consists of measuring multiple diffraction patterns by scanning a finite illumination (also called the probe) on an extended specimen (the object).



Stefano Marchesini et al. SHARP: a distributed, GPU-based ptychographic solver, J. Appl. Cryst., 49, 2016

Near-Real-Time Ptychographic Pipeline





[1] Nikolay Malitsky. Bringing the HPC Reconstruction Algorithms to Big Data Platforms, NYSDS, 2016

Kitware

TOMOGRAPHIC APPLICATION

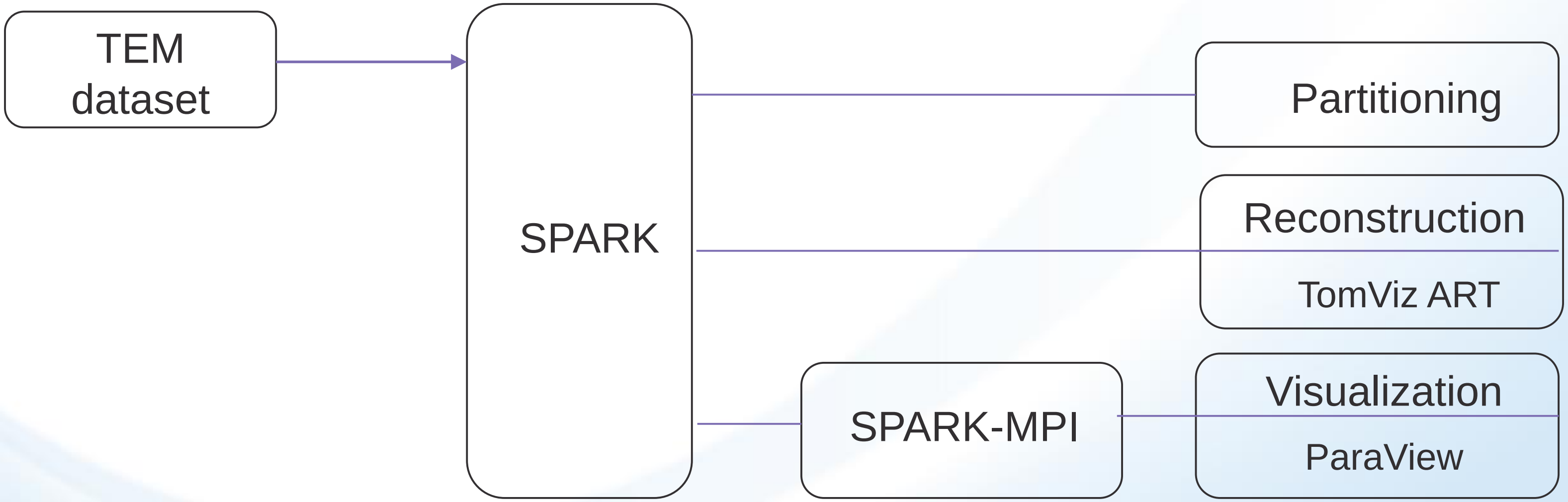
Transmission Electron Microscopy

Three-dimensional (3D) characterization of materials at the nano- and meso-scale has become possible with transmission and scanning transmission electron microscopes (S/TEM. We are using publicly accessible data from our collaborators referred to in a nature article (https://dx.doi.org/10.6084/m9.figshare.c.2185342). All of the files contained in the dataset are in 16 bit tiff image format.

Objective: Perform near real-time reconstruction leveraging existing serial and MPI-enabled parallel algorithms, image processing, and then visualization in parallel using parallel rendering

Tomographic Pipeline

- Load a transmission electron microscopy (TEM) dataset into a Spark RDD
- Repartition the data to ensure the neighboring pixel are in the same partition for reconstruction algorithm.
- Apply Algebraic Reconstruction Technique (ART) implemented in TomViz on each partition in parallel using the Spark map-collect operations
- Gather the resulting reconstruction 3D dataset and render it using ParaView



Scientific Data Ingestion – Column Storage

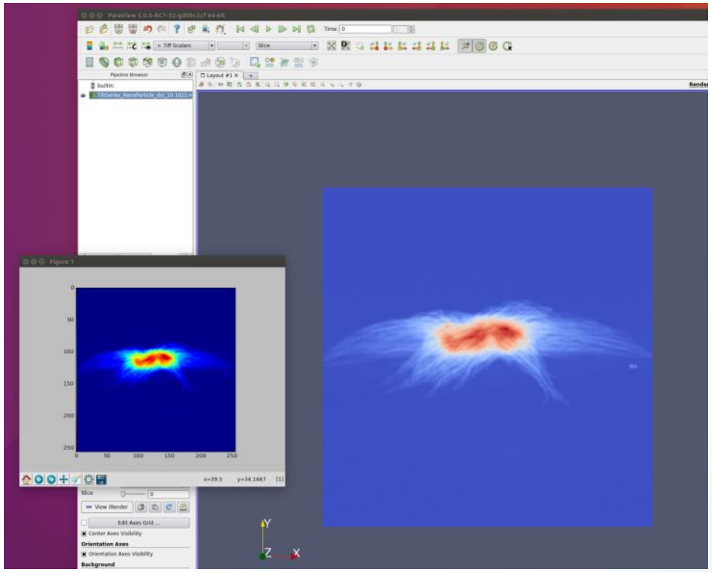
Goal: Ingest scientific data into Spark/Hadoop ecosystem

Input: TEM data

Approach: TEM data ingested into HDFS using Parquet (https://parquet.apache.org) format which provided both RDD and DataFrame.

Added following metadata to the parquet format:

- Tilt angles (data for each slice, hence this would be an array),
- Pixel size (dimensionality) (nanometers usually), and
- Shape of the array

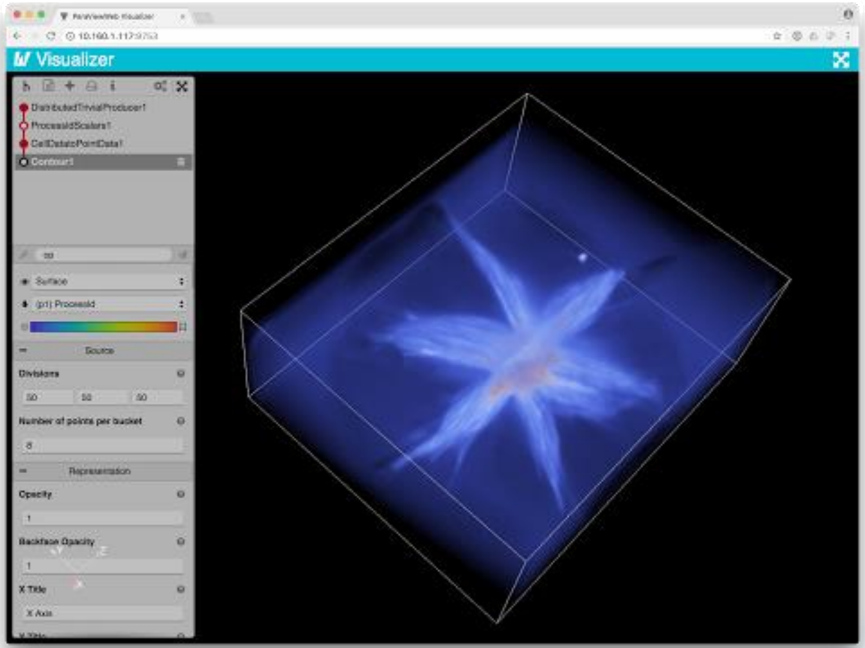


Another approach is reading the data as NUMPY and using Spark parallelize to distribute it across nodes

Perform a conversion from the NumPy array to a VTK data structure before sending the dataset to the ParaView

Parallel Visualization using ParaView and MPI

Perform the rendering and visualization utilizing ParaView that employs the MPI environment for parallelization

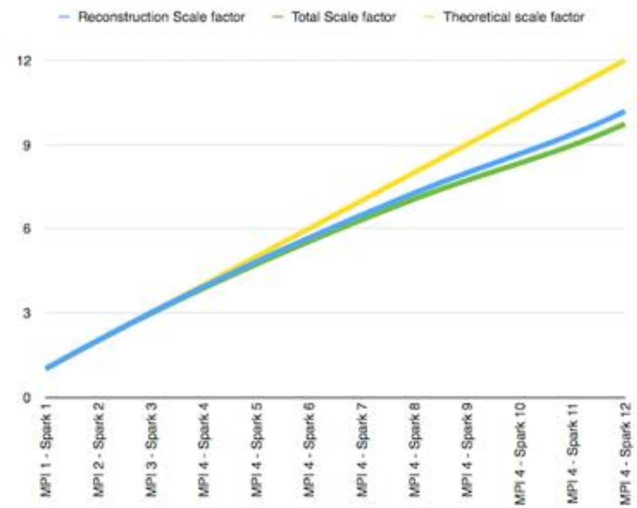


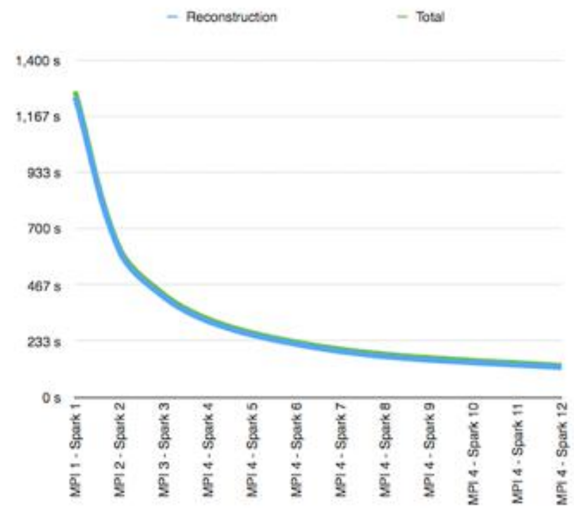

```
def parallelVisualize(recon):
        (iSize, jSize, kSize) = recon.shape

    os.environ["PMI_PORT"] = pmi_port
    os.environ["PMI_ID"] = str(idx)
    os.environ["PV_ALLOW_BATCH_INTERACTION"] = "1"
    os.environ["DISPLAY"] = ":0"

    // Convert reconstruction array into VTK format
    arr = recon.ravel(order='A')
    vtkarray = numpy_support.numpy_to_vtk(arr)
    vtkarray.SetName('Scalars')

    dataset = vtkImageData()
    minX = 0
    maxX = 0
    for i in range(idx + 1):
        minX = maxX
        maxX += xSizes[i]
    dataset.SetExtent(minX, maxX - 1, 0, sizeY - 1, 0, sizeY - 1)
    dataset.GetPointData().SetScalars(vtkarray)
    vtkDistributedTrivialProducer.SetGlobalOutput('Spark', dataset)

        // Import VTK and ParaView modules here

    pm = vtkProcessModule.GetProcessModule()
    class _VisualizerServer(pv_wamp.PVServerProtocol):
        dataDir = '/data'
        groupRegex = "[0-9]+\\.[0-9]+\\.|[0-9]+\\."
        excludeRegex = "^\\.|~$|^\\$"
        allReaders = True
        viewportScale=1.0
        viewportMaxWidth=2560
        viewportMaxHeight=1440
        def initialize(self):
            .....
    args = Options()
    if pm.GetPartitionId() == 0:
        producer = simple.DistributedTrivialProducer()
        producer.UpdateDataset = ''
        producer.UpdateDataset = 'Spark'
        producer.WholeExtent = [0, sizeX - 1, 0, sizeY - 1, 0, sizeY - 1]
        server.start_webserver(options=args, protocol=_VisualizerServer)
        pm.GetGlobalController().TriggerBreakRMIs()
    yield (idx, targetPartition)

//
results = rdd.mapPartitionsWithIndex(processPartition).collect()
for out in results:
    print(out)
```

Performance

Parallel execution time using Spark infrastructure

	Tiff read	Gather	Reconstruction	MPI Gather	MPI share	Total	Reconstruction Scale factor	Total Scale factor	Theoretical scale factor
MPI 1 - Spark 1	2.184	6.474	1247.631	13.991	1.284	1271.563	1.0	1.0	1.0
MPI 2 - Spark 2	1.883	3.598	607.032	6.999	1.462	620.974	2.1	2.0	2.0
MPI 3 - Spark 3	1.925	2.382	415.907	5.443	2.048	427.704	3.0	3.0	3.0
MPI 4 - Spark 4	2.096	1.996	318.026	3.657	1.824	327.599	3.9	3.9	4.0
MPI 4 - Spark 5	1.738	1.370	264.787	3.750	1.656	273.302	4.7	4.7	5.0
MPI 4 - Spark 6	2.095	1.489	219.104	3.844	1.600	228.132	5.7	5.6	6.0
MPI 4 - Spark 7	1.717	1.018	194.830	4.245	2.435	204.246	6.4	6.2	7.0
MPI 4 - Spark 8	1.735	1.135	171.594	3.902	1.833	180.199	7.3	7.1	8.0
MPI 4 - Spark 9	1.903	0.844	157.032	3.915	1.950	165.644	7.9	7.7	9.0
MPI 4 - Spark 10	1.713	0.933	143.657	3.994	2.131	152.428	8.7	8.3	10.0
MPI 4 - Spark 11	1.993	0.738	134.416	4.112	1.866	143.126	9.3	8.9	11.0
MPI 4 - Spark 12	1.621	0.782	122.454	3.920	1.810	130.587	10.2	9.7	12.0





Metrics using 128 GB of memory, and 16 Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz processors for the 256x256x74 TEM demonstration that produced a 256x256x256 reconstructed dataset.

Varied number of Spark workers (from 1 to 12), which performs the reconstruction, and the number of MPI ranks (from 1 to 4), which implements the visualization.

The total of ART went down to ~300 secs, an improvement of 6x over the implementation in TomViz

FUTURE DIRECTIONS

Climate Application

Problem Description: Perform insitu analysis and visualization in high resolution climate models

Sample data - The NASA Earth Exchange Globally Downscaled Projections (NEX-GDDP) dataset (https://cds.nccs.nasa.gov/nex-gddp) is comprised of downscaled climate scenarios for the globe that are derived from General Circulation Model (GCM) runs conducted under the Coupled Model Intercomparison Project Phase 5 (CMIP5) and across two of the four greenhouse gas emissions scenarios known as Representative Concentration Pathways (RCPs). The total size of the NEX-GDDP dataset is 12 TB with individual file size of 750MB.

Advancing Image Reconstruction Pipelines with Deep Learning Approaches

In collaboration with Shantenu Jha

- Development of deep learning approaches for analyzing reconstructed images
- Development of streaming MPI-based deep learning applications
- Development of machine learning approaches for steering pre-processing and reconstruction algorithms.