Bringing the HPC Reconstruction Algorithms to Big Data Platforms

Nikolay Malitsky

New York Scientific Data Summit: Data-Driven Discovery

August 17, 2016
Outline

- Spark as an integrated platform for the Big Data and Big Computing applications
- Spark In-Situ Data Access Approach
- Ptychographic Application
- Spark-Based Distributed Deep Learning Solvers
- SHARP-SPARK Project
- Summary
Closing a Gap between Big Data and Big Computing

Ecosystems*

Big Data

Big Computing

Motivation: New Frontiers

Leaders: Spark MPI

*G. Fox at al. HPC-ABDC High Performance Computing Enhanced Apache Big Data Stack, CCGrid, 2015
Three Directions

- Spark Model + MPI-oriented extension
- MPI Model + Spark-oriented extension
- New Model

topic of this talk
Spark ecosystem and proposed extensions for experimental facilities

1. Big Volume + Variety

2. HPC Computing

3. Big Velocity
DATA LAYER
Database vs HDF5 models

No SQL

Array-Oriented Model:
- SciDB, MonetDB, etc.

Column Family Model:
- Bigtable, Cassandra, etc.

Document-Oriented Model:
- MongoDB, CouchDB, etc.

Graph Model:
- Neo4j, etc.

HDF5

The key data model concepts:

- **Group** - a collection of objects (including groups)
- **Dataset** - a multidimensional array of data elements with attributes and other metadata
- **Datatype** - a description of a specific class of data element
- **Attribute** - a named data value associated with a group, dataset, or named datatype

Present version: single file oriented
Large-Scale HDF5-Oriented Development

Exascale Fast-Forward I/O and Storage Stack*
Time: 2012-2014
Team: Intel, The HDF Group, EMC, Cray

Spark-based Proposal

Intel, The HDF Group, EMC, and Cray. FF-Storage Final Report, June 2014
Spark In-Situ Data Access Approach

Supported file formats:
- Text files (plain, JSON, CSV)
- Hadoop InputFormat with arbitrary key and value
- Hadoop SequenceFile with arbitrary key and value
- Object files with the RDD values previously saved using the Java/Python serialization
- HDF5 (research projects*)

SQL and NoSQL Databases:
- Java Database Connectivity (JDBC)
- HBase
- Cassandra
- MongoDB
- Neo4j

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).

\[ \psi_j = P(r-r_j)O(r) \]

Ptychography Algorithm (in math)

Iteration loop based on the difference map\(^1\):

\[
\psi^{n+1} = \psi^n + \beta \Delta(\psi^n)
\]

\[
\Delta = \pi_1 \phi_2 - \pi_2 \phi_1
\]

\[
f_i(\psi) = (1 + \gamma_i) \pi_i(\psi) - \gamma_i \psi
\]

Projection operators associated with the modulus and overlap constraints:

\[
\pi_a(\psi): \psi \rightarrow F^* \frac{F_{\psi}}{|F_{\psi}|} \sqrt{I}
\]

\[
\pi_o(\psi): \psi \rightarrow P(\mathbf{r} - \mathbf{r}_j)O(\mathbf{r})
\]

Object and probe updates from the minimization of the cost function\(^2\):

\[
\varepsilon = ||\Psi - \Psi^0||^2 = \sum_j \sum_r |\psi_j(\mathbf{r}) - P^0(\mathbf{r} - \mathbf{r}_j)O^0(\mathbf{r})|^2
\]

\[
\frac{\partial \varepsilon}{\partial O^0} = 0: O^0(\mathbf{r}) = \frac{\sum_j \psi_j(\mathbf{r})P^*(\mathbf{r} - \mathbf{r}_j)}{\sum_j |P(\mathbf{r} - \mathbf{r}_j)|^2}
\]

\[
\frac{\partial \varepsilon}{\partial P^0} = 0: P^0(\mathbf{r}) = \frac{\sum_j \psi_j(\mathbf{r} + \mathbf{r}_j)O^*(\mathbf{r} + \mathbf{r}_j)}{\sum_j |O(\mathbf{r} + \mathbf{r}_j)|^2}
\]

---


\(^2\) P. Thibault et al. Probe retrieval in ptychographic coherent diffractive imaging, Ultramicroscopy, 109, 2009
**Ptychography Approach (in pictures*)**

**Sample space**

\[ \pi_0(\psi): \psi \rightarrow P(r-r_j)O(r) \]

**Detector space**

- **FFT**
- **propagate**
- **replace magnitudes**
- **FFT\(^{-1}\)**
- **propagate back**

\[ O^0(r') = \frac{\sum_j \psi_j(r)P^*(r-r_j)}{\sum_j |P(r-r_j)|^2} \]

\[ P^0(r) = \frac{\sum_j \psi_j(r+r_j)O^*(r+r_j)}{\sum_j |O(r+r_j)|^2} \]

\[ \pi_a(\psi): \psi \rightarrow F^* \frac{F\psi}{|F\psi|} \sqrt{I} \]

---

S. Marchesini, Fast Scalable methods for ptychographic imaging, SHARP workshop, LBNL, Oct 8, 2014
**SHARP GPU-Based Solver and NSLS-II Application**

### SHARP-NSLS2

![Class Diagram for SHARP-NSLS2](image)

### Functions and Times

<table>
<thead>
<tr>
<th>Functions</th>
<th>Time, s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>256 frames</td>
</tr>
<tr>
<td>Modulus &amp; overlap projections</td>
<td>0.06</td>
</tr>
<tr>
<td>Probe update</td>
<td>0.025</td>
</tr>
<tr>
<td>Object update</td>
<td>0.03</td>
</tr>
</tbody>
</table>
Multi-GPU Approach

ALS Streaming Pipeline

(1) S. Marchesini et al. SHARP: a distributed, GPU-based ptychographic solver, LBNL-1003977, 2016

SPARK-BASED DISTRIBUTED DEEP LEARNING SOLVERS
Deep Learning Approach – 1 of 2

Deep learning is an active area of machine learning, achieving a state-of-the-art performance in multiple application domains, ranging from visual object recognition to reinforcement learning. The major category of methods is based on multi-layer (deep) architectures using the convolution neural network model.

A brief (and incomplete) history of the convolution neural network model:


Deep Learning Approach – 2 of 2

- **DistBelief, Google**: J. Dean et al. Large Scale Distributed Deep Networks, NIPS 2012

- **Caffe, UC Berkeley**: http://caffe.berkeleyvision.org/
  Y. Jia et al., Caffe: Convolution Architecture for Fast Feature Embedding, ACM International Conference on Multimedia, 2014

- **TensorFlow, Google**: https://www.tensorflow.org/
  Open source: Nov 2015, Distributed version: April 2016
SparkNet*

Philipp Moritz, Robert Nishihara, Ion Stoica, and Michael Jordan. AMPLab, UC Berkeley

URL: https://github.com/amplab/SparkNet

API: Scala
Engine Wrapper: Scala/Java/C++

```
var nets = trainData.foreachPartition( data => {
  var net = Net (…)
  net.setTrainingData(data)
  net }

var weights = initialWeights (…)

for ( i <- 1 to 1000){
  var broadcastWeights = broadcast(weights)
  nets.map(net => net.setWeights(broadcastWeights.value))
  weights = nets.map(net => {
    net.train (50)
    net.getWeights() }).mean()
}
```

CaffeOnSpark*

Andy Feng, Jun Shi, and Mridul Jain. Yahoo Big ML Team

URL: https://github.com/yahoo/CaffeOnSpark

API: Python/Scala

Engine Wrapper: Scala/Java/C++

Inter-Worker Interface (C++):
- Ethernet/TCP
- InfiniBand/RDMA
- GPU or CPU

TensorSpark*

Christopher Nguyen, Chris Smith, Ushnish De, Vu Pham, and Nanda Kishore. Arimo

URL: https://github.com/adatao/tensorspark

API: Python
Engine Wrapper: Python/C++


Fragment of the Driver script

# define a worker function that calls the TensorFlow wrapper
def train_partition(partition):
    return TensorSparkWorker(...).train_partition(partition)

# access the Spark context
sc = pyspark.SparkContext(...)

# load data on distributed workers and cache them in memory
training_rdd = sc.textFile(...).cache()

# start the Tornado-based parameter server
param_server = ParameterServer(...)  
param_server.start()

# start a training loop
for i in range(num_epochs):
    # run the train_partition function on distributed workers
    training_rdd.mapPartitions(train_partition).collect()
**PySpark**

```
In [24]:
def send_arrays(x):
    t1 = datetime.now();
    results = [];
    for i in range(0, 10):
        results.append(np.arange(1000000, dtype = np.float))
    t2 = datetime.now();
    print "create_arrays. length: ", len(results), " time: ", (t2 - t1)
    return results

create_arrays. length: 10 time: 0:00:00.008749

In [25]:
t1 = datetime.now()
send_arrays_rdd = sc.parallelize(xrange(100, partitions + 1), partitions).map(send_arrays);
ms = send_arrays_rdd.collect();
t2 = datetime.now();
print "collect. length: ", len(ms[0]), " time: ", (t2 - t1);

collect. length: 10 time: 0:00:01.126674
```

https://cwiki.apache.org/confluence/display/SPARK/PySpark+Internals
def wf(args):
    comm = Communicator.createCommunicator(args['rank'], args['size'])

    # 1. allocate buffers used in the peer-to-peer communication
    imageSize = 2*1000000
    comm.allocate(imageSize*4)

    # 2. connect to the address server and exchange the RDMA addresses
    comm.connect(args['addr'])

    # define a local array (e.g. image)
    a = np.zeros(imageSize, dtype=np.float32)
    a[imageSize-1] = 1.0

    # 3. sum peers’ arrays for several iterations
    t1 = datetime.now()
    for i in range(0, 10):
        comm.allSum(a)
    t2 = datetime.now()

    # prepare and return the benchmark results
    out = {
        'a' : a[imageSize-1],
        'time' : (t2-t1),
    }
    comm.release()
    return out

<table>
<thead>
<tr>
<th>Approach</th>
<th>Time, s</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI Allreduce based on MVAPICH2</td>
<td>0.013</td>
</tr>
<tr>
<td>SHARP-SPARK based on the CaffeOnSpark library</td>
<td>0.016</td>
</tr>
</tbody>
</table>

Worker’s function of the benchmark application
Summary

- Outlined Spark as an integrated platform for the Big Data and Big Computing applications at experimental facilities.

- Presented the SHARP-SPARK application highlighting the MPI-oriented development of the Spark computational model.
Acknowledgement

**SHARP Team, CAMERA, LBNL:** H. Krishnan, S. Marchesini, T. Perciano, J. Sethian, D. Shapiro

**CaffeOnSpark Team, Yahoo:** A. Feng, J. Shi, M. Jain

**HXN Group, NSLS-II, BNL:** X. Huang

**Control Group, NSLS-II, BNL:** M. Cowan, L. Flaks, A. Heroux, K. Lauer, R. Petkus

**Computational Science Lab, CSI, BNL:** N. D’ Imperio, D. Zhihua

**Information Technology Division, BNL:** R. Perez