Twister2: A High-Performance Big Data Programming Environment


Geoffrey Fox, May 21, 2018
Judy Qiu, Supun Kamburugamuve
Department of Intelligent Systems Engineering
gcf@indiana.edu, http://www.dsc.soic.indiana.edu/, http://spidal.org/

Work with Shantenu Jha, Kannan Govindarajan, Pulasthi Wickramasinghe, Gurhan Gunduz, Ahmet Uyar
Abstract

• We analyse the components that are needed in programming environments for Big Data Analysis Systems with scalable HPC performance and the functionality of ABDS – the Apache Big Data Software Stack.

• One highlight is Harp-DAAL which is a machine library exploiting the Intel node library DAAL and HPC communication collectives within the Hadoop ecosystem.

• Another highlight is Twister2 which consists of a set of middleware components to support batch or streaming data capabilities familiar from Apache Hadoop, Spark, Heron and Flink but with high performance

• Twister2 covers bulk synchronous and data flow communication; task management as in Mesos, Yarn and Kubernetes; dataflow graph execution models; launching of the Harp-DAAL library; streaming and repository data access interfaces, in-memory databases and fault tolerance at dataflow nodes.

• Similar capabilities are available in current Apache systems but as integrated packages which do not allow needed customization for different application scenarios.
Requirements

- On general principles **parallel and distributed computing** have different requirements even if sometimes similar functionalities
  - Apache stack ABDS typically uses distributed computing concepts
  - For example, Reduce operation is different in MPI (Harp) and Spark
- Large scale simulation requirements are well understood
- Big Data requirements are not agreed but there are a few key use types
  1) **Pleasingly parallel** processing (including **local machine learning LML**) as of different tweets from different users with perhaps MapReduce style of statistics and visualizations; possibly Streaming
  2) **Database model** with queries again supported by MapReduce for horizontal scaling
  3) **Global Machine Learning GML** with single job using multiple nodes as classic parallel computing
  4) **Deep Learning** certainly needs HPC – possibly only multiple small systems
- Current workloads stress 1) and 2) and are suited to current clouds and to Apache Big Data Software (with no HPC)
  - This explains why Spark with poor GML performance can be so successful
Difficulty in Parallelism

Size of Synchronization constraints

There is also distribution seen in grid/edge computing

Need a toolkit covering all applications with same API but different implementations

- Loosely Coupled
  - Commodity Clouds
    - MapReduce as in scalable databases
    - Pleasingly Parallel
      - Often independent events
      - Current major Big Data category
    - Parameter sweep simulations
  - HPC Clouds
    - High Performance Interconnect
    - Global Machine Learning
      - e.g. parallel clustering
    - Linear Algebra at core
      - (typically not sparse)
    - Structured Adaptive Sparsity
      - Huge Jobs
  - Deep Learning
- Tightly Coupled
  - HPC Clouds/Supercomputers
    - Memory access also critical
  - Unstructured Adaptive Sparsity
    - Medium size Jobs
  - LDA
  - Graph Analytics e.g.
    - subgraph mining
  - Large scale simulations
  - Exascale Supercomputers

Spectrum of Applications and Algorithms
Need a toolkit covering 5 main paradigms with same API but different implementations

Six Computation Paradigms for Data Analytics

Note Problem and System Architecture as efficient execution says they must match

Classic Cloud Workload

[Diagrams of the six computation paradigms]

- BLAST Analysis
- Local Machine Learning
- Pleasingly Parallel
- High Energy Physics (HEP) Histograms
- Web search
- Recommender Engines
- Expectation Maximization
- Clustering
- Linear Algebra
- PageRank
- Classic MPI
- PDE Solvers and Particle Dynamics
- Graph
- Streaming images from Synchrotron sources, Telescopes, Internet of Things
- Difficult to parallelize
- Asynchronous parallel Graph

Global Machine Learning

These 3 are focus of Twister2 but we need to preserve capability on first 2 paradigms
Comparing Spark, Flink and MPI

• On Global Machine Learning GML.
Machine Learning with MPI, Spark and Flink

• Three algorithms implemented in three runtimes
  • Multidimensional Scaling (MDS)
  • Terasort
  • K-Means (drop as no time and looked at later)

• Implementation in Java
  • MDS is the most complex algorithm - three nested parallel loops
  • K-Means - one parallel loop
  • Terasort - no iterations

• With care, Java performance ~ C performance
• Without care, Java performance << C performance (details omitted)
Multidimensional Scaling: 3 Nested Parallel Sections

Kmeans also bad – see later

MPI Factor of 20-200 Faster than Spark/Flink

MDS execution time on **16 nodes**
with 20 processes in each node with varying number of points

MDS execution time with 32000 points on **varying number of nodes**.
Each node runs 20 parallel tasks
Spark, Flink No Speedup
Terasort

Sorting 1TB of data records

Partition the data using a sample and regroup

Terasort execution time in 64 and 32 nodes. Only MPI shows the sorting time and communication time as other two frameworks doesn't provide a clear method to accurately measure them. Sorting time includes data save time.

MPI-IB - MPI with Infiniband
Software
HPC-ABDS
HPC-FaaS
NSF 1443054: CIF21
DIBBs: Middleware and High Performance Analytics Libraries for Scalable Data Science
Ogres Application Analysis
HPC-ABDS and HPC-FaaS Software
Harp and Twister2 Building Blocks
SPIDAL Data Analytics Library

Network Science
Biomolecular Simulations
Polar Science

Software:
MIDAS
HPC-ABDS
HPC-ABDS

Integrated wide range of HPC and Big Data technologies.

I gave up updating list in January 2016!

January 29 2016
Different choices in software systems in Clouds and HPC. HPC-ABDS takes cloud software augmented by HPC when needed to improve performance.

16 of 21 layers plus languages.
Harp Plugin for Hadoop: Important part of Twister2

Work of Judy Qiu

Parallelism Model

MapReduce Model

MapCollective Model

Collective Communication

Architecture

Application

MapReduce Applications

MapCollective Applications

Framework

Harp

Resource Manager

MapReduce V2

YARN

Harp is an open-source project developed at Indiana University [6], it has:

- MPI-like collective communication operations that are highly optimized for big data problems.
- Harp has efficient and innovative computation models for different machine learning problems.

Map Collective Run time merges MapReduce and HPC

Run time software for Harp

- broadcast
- reduce
- allreduce
- allgather
- regroup
- push & pull
- rotate
Dynamic Rotation Control for Latent Dirichlet Allocation and Matrix Factorization SGD (stochastic gradient descent)

Model Parameters
From Rotation

Model Related Data

Other Model Parameters
From Caching

Computes until the time arrives, then starts model rotation to address load imbalance

Multi-Thread Execution
Datasets: 5 million points, 10 thousand centroids, 10 feature dimensions
10 to 20 nodes of Intel KNL7250 processors
Harp-DAAL has 15x speedups over Spark MLlib

K means

Datasets: 500K or 1 million data points of feature dimension 300
Running on single KNL 7250 (Harp-DAAL) vs. single K80 GPU (PyTorch)
Harp-DAAL achieves 3x to 6x speedups

PCA

Datasets: Twitter with 44 million vertices, 2 billion edges, subgraph templates of 10 to 12 vertices
25 nodes of Intel Xeon E5 2670
Harp-DAAL has 2x to 5x speedups over state-of-the-art MPI-Fascia solution

Subgraph

Harp v. Spark

Harp v. Torch

Harp v. MPI
Mahout and SPIDAL

- Mahout was Hadoop machine learning library but largely abandoned as Spark outperformed Hadoop
- SPIDAL outperforms Spark MLlib and Flink due to better communication and better dataflow or BSP communication.
- Has Harp-(DAAL) optimized machine learning interface
- SPIDAL also has community algorithms
  - Biomolecular Simulation
  - Graphs for Network Science
  - Image processing for pathology and polar science
Qiu Core SPIDAL Parallel HPC Library with Collective Used

- **DA-MDS** Rotate, AllReduce, Broadcast
- **Directed Force Dimension Reduction** AllGather, Allreduce
- **Irregular DAVS Clustering** Partial Rotate, AllReduce, Broadcast
- **DA Semimetric Clustering (Deterministic Annealing)** Rotate, AllReduce, Broadcast
- **K-means** AllReduce, Broadcast, AllGather DAAL
- **SVM** AllReduce, AllGather
- **SubGraph Mining** AllGather, AllReduce
- **Latent Dirichlet Allocation** Rotate, AllReduce
- **Matrix Factorization (SGD)** Rotate DAAL
- **Recommender System (ALS)** Rotate DAAL
- **Singular Value Decomposition (SVD)** AllGather DAAL
- **QR Decomposition (QR)** Reduce, Broadcast DAAL
- **Neural Network** AllReduce DAAL
- **Covariance** AllReduce DAAL
- **Low Order Moments** Reduce DAAL
- **Naive Bayes** Reduce DAAL
- **Linear Regression** Reduce DAAL
- **Ridge Regression** Reduce DAAL
- **Multi-class Logistic Regression** Regroup, Rotate, AllGather
- **Random Forest** AllReduce
- **Principal Component Analysis (PCA)** AllReduce DAAL

**DAAL** implies integrated on node with Intel DAAL Optimized Data Analytics Library
Implementing Twister2 in detail I

This breaks rule from 2012-2017 of not “competing” with but rather “enhancing” Apache
Twister2: “Next Generation Grid - Edge – HPC Cloud” Programming Environment

• Analyze the runtime of existing systems
  • Hadoop, Spark, Flink, Pregel Big Data Processing
  • OpenWhisk and commercial FaaS
  • Storm, Heron, Apex Streaming Dataflow
  • Kepler, Pegasus, NiFi workflow systems
  • Harp Map-Collective, MPI and HPC AMT runtime like DARMA
  • And approaches such as GridFTP and CORBA/HLA (!) for wide area data links

• A lot of confusion coming from different communities (database, distributed, parallel computing, machine learning, computational/ data science) investigating similar ideas with little knowledge exchange and mixed up (unclear) requirements

http://www.iterativemapreduce.org/
Integrating HPC and Apache Programming Environments

- **Harp-DAAL** with a kernel Machine Learning library exploiting the Intel node library DAAL and HPC communication collectives within the Hadoop ecosystem. The broad applicability of Harp-DAAL is supporting all 5 classes of data-intensive computation, from pleasingly parallel to machine learning and simulations.

- **Twister2** is a toolkit of components that can be packaged in different ways
  - Integrated batch or streaming data capabilities familiar from Apache Hadoop, Spark, Heron and Flink but with high performance.
  - Separate bulk synchronous and data flow communication;
  - Task management as in Mesos, Yarn and Kubernetes
  - Dataflow graph execution models
  - Launching of the Harp-DAAL library
  - Streaming and repository data access interfaces,
  - In-memory databases and fault tolerance at dataflow nodes. (use RDD to do classic checkpoint-restart)
Approach

• Clearly define and develop functional layers (using existing technology when possible)
• Develop layers as independent components
• Use *interoperable* common abstractions but multiple *polymorphic* implementations.
• Allow users to pick and choose according to requirements such as
  • Communication + Data Management
  • Communication + Static graph
• Use HPC features when possible
<table>
<thead>
<tr>
<th>Area</th>
<th>Component</th>
<th>Implementation</th>
<th>Comments: User API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture Specification</td>
<td>Coordination Points</td>
<td>State and Configuration Management; Change execution mode; save and reset state</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Execution</td>
<td>Program, Data and Message Level</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Semantics</td>
<td>Mapping of Resources to Bolts/Maps in Containers, Processes, Threads</td>
<td>Different systems make different choices - why?</td>
</tr>
<tr>
<td></td>
<td>Parallel Computing (Dynamic/Static)</td>
<td>Spark Flink Hadoop Pregel MPI modes</td>
<td>Owner Computes Rule</td>
</tr>
<tr>
<td></td>
<td>Resource Allocation</td>
<td>Plugins for Slurm, Yarn, Mesos, Marathon, Aurora</td>
<td>Client API (e.g. Python) for Job Management</td>
</tr>
<tr>
<td></td>
<td>Task migration</td>
<td>Monitoring of tasks and migrating tasks for better resource utilization</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Elasticity</td>
<td>OpenWhisk</td>
<td></td>
</tr>
<tr>
<td>Task System</td>
<td>Streaming and FaaS Events</td>
<td>Heron, OpenWhisk, Kafka/RabbitMQ</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task Execution</td>
<td>Process, Threads, Queues</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task Scheduling</td>
<td>Dynamic Scheduling, Static Scheduling, Pluggable Scheduling Algorithms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task Graph</td>
<td>Static Graph, Dynamic Graph</td>
<td>Support accelerators (CUDA,KNL)</td>
</tr>
<tr>
<td>Area</td>
<td>Component</td>
<td>Implementation</td>
<td>Comments</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------</td>
<td>-----------------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Communication API</td>
<td>Messages</td>
<td>Heron</td>
<td>This is user level and could map to multiple communication systems</td>
</tr>
<tr>
<td></td>
<td>Dataflow Communication</td>
<td>Fine-Grain Twister2 Dataflow communications: MPI, TCP and RMA</td>
<td>Streaming, ETL data pipelines,</td>
</tr>
<tr>
<td></td>
<td>BSP Communication</td>
<td>Conventional MPI, Harp</td>
<td>Define new Dataflow communication API and library</td>
</tr>
<tr>
<td></td>
<td>Map-Collective</td>
<td></td>
<td>MPI Point to Point and Collective API</td>
</tr>
<tr>
<td>Data Access</td>
<td>Static (Batch) Data</td>
<td>File Systems, NoSQL, SQL</td>
<td>Data API</td>
</tr>
<tr>
<td></td>
<td>Streaming Data</td>
<td>Message Brokers, Spouts</td>
<td>Data Transformation API;</td>
</tr>
<tr>
<td>Data Management</td>
<td>Distributed Data Set</td>
<td>Memory (immutable data),</td>
<td>Spark RDD, Heron Streamlet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>Check Pointing</td>
<td>Lightweight; Coordination Points; Spark/Flink, MPI and Heron models</td>
<td>Streaming and batch cases distinct; Crosses all components</td>
</tr>
<tr>
<td>Security</td>
<td>Storage, Messaging, execution</td>
<td>Research needed</td>
<td>Crosses all Components</td>
</tr>
</tbody>
</table>
Different applications at different layers

<table>
<thead>
<tr>
<th>Type of applications</th>
<th>Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Task System</td>
</tr>
<tr>
<td>Streaming</td>
<td>Distributed Data Set</td>
</tr>
<tr>
<td>Data Pipelines</td>
<td>Distributed Data Set</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>Distributed Shared Memory</td>
</tr>
<tr>
<td>FaaS</td>
<td>Stateless</td>
</tr>
</tbody>
</table>

- **Spark**
- **Flink**
- **Hadoop**
- **Heron**
- **Storm**
- **None**

*Suitable for Simple applications* (Ex – Pleasingly Parallel Applications)

*Suitable for Complex applications* (Ex – MDS, Complex ML Algorithms)
Implementing Twister2 in detail II

Look at Communication in detail
Communication Models

- **MPI Characteristics**: Tightly synchronized applications
  - Efficient communications (µs latency) with use of advanced hardware
  - In place communications and computations (Process scope for state)

- **Basic dataflow**: Model a computation as a graph
  - Nodes do computations with Task as computations and edges are asynchronous communications
  - A computation is activated when its input data dependencies are satisfied

- **Streaming dataflow**: Pub-Sub with data partitioned into streams
  - Streams are unbounded, ordered data tuples
  - Order of events important and group data into time windows

- **Machine Learning dataflow**: Iterative computations and keep track of state
  - There is both Model and Data, but typically only communicate the model
  - Collective communication operations such as AllReduce AllGather (no differential operators in Big Data problems)
  - Can use in-place MPI style communication
Twister2 Dataflow Communications

- **Twister:Net** offers two communication models
  - **BSP** (Bulk Synchronous Processing) communication using TC or MPI separated from its task management plus extra Harp collectives

- plus a new **Dataflow library DFW** built using MPI software but at data movement not message level
  - Non-blocking
  - Dynamic data sizes
  - Streaming model
    - Batch case is modeled as a finite stream
  - The communications are between a set of tasks in an arbitrary task graph
  - Key based communications
  - Communications spilling to disks
  - Target tasks can be different from source tasks
Twister:Net

- Communication operators are stateful
  - Buffer data
  - handle imbalanced dynamically sized communications,
  - act as a combiner
- Thread safe
- Initialization
  - MPI
  - TCP / ZooKeeper
- Buffer management
  - The messages are serialized by the library
- Back-pressure
  - Uses flow control by the underlying channel

Architecture

<table>
<thead>
<tr>
<th>Reduce</th>
<th>Gather</th>
<th>Partition</th>
<th>Broadcast</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllReduce</td>
<td>AllGather</td>
<td>Keyed-Partition</td>
<td></td>
</tr>
<tr>
<td>Keyed-Reduce</td>
<td>KeyedGather</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Batch and Streaming versions of above currently available
Bandwidth & Latency Kernel

Latency and bandwidth between two tasks running in two nodes

Bandwidth utilization of Flink, Twister2 and OpenMPI over 1Gbps, 10Gbps and IB with Flink on IPoIB

Latency of MPI and Twister:Net with different message sizes on a two-node setup
Flink, BSP and DFW Performance

Total time for Flink and Twister:Net for Reduce and Partition operations in 32 nodes with 640-way parallelism. The time is for 1 million messages in each parallel unit, with the given message size.

Latency for Reduce and Gather operations in 32 nodes with 256-way parallelism. The time is for 1 million messages in each parallel unit, with the given message size. For BSP-Object case we do two MPI calls with MPIAllReduce / MPIAllGather first to get the lengths of the messages and the actual call. InfiniBand network is used.
K-Means algorithm performance

Left: K-means job execution time on 16 nodes with varying centers, 2 million points with 320-way parallelism. Right: K-Means with 4, 8, and 16 nodes where each node having 20 tasks. 2 million points with 16000 centers used.
Partition the data using a sample and regroup

BSP algorithm waits for others to send messages in a ring topology and can be inefficient compared to DFW case where processes do not wait.

For DFW case, a single node can get congested if many processes send message simultaneously.
Twister:Net and Apache Heron for Streaming

Latency of Apache Heron and Twister:Net DFW (Dataflow) for Reduce, Broadcast and Partition operations in 16 nodes with 256-way parallelism
Robot Algorithms

Simultaneous Localization and Mapping

Map Built from Robot data

Robot with a Laser Range Finder

N-Body Collision Avoidance

Robots need to avoid collisions when they move
SLAM Simultaneous Localization and Mapping

End to end delays without any processing is less than 10ms

Hosted in FutureSystems OpenStack cloud which is accessible through IU network

Rao blackwellized particle filter based SLAM

Gateway Message Brokers RabbitMQ, Kafka

Streaming SLAM Algorithm
Apache Storm

Multiple streaming workflows

A stream application with some tasks running in parallel

Sending to pub-sub

Sending to Persisting to storage

Streaming workflow
Performance of SLAM Storm v. Twister2

Twister2 Implementation speedup.

Storm Implementation Speedup
Implementing Twister2 in detail III

State

Iterative MapReduce
http://www.iterativemapreduce.org/
Resource Allocation

• Job Submission & Management
  • twister2 submit
• Resource Managers
  • Slurm
  • Nomad
  • Kubernetes
  • Mesos
It takes around 5 seconds to initialize a worker in Kubernetes.
It takes around 3 seconds to initialize a worker in Mesos.
When 3 workers are deployed in one executor or pod, initialization times are faster in both systems.
Task System

• Generate computation graph dynamically
  • Dynamic scheduling of tasks
  • Allow fine grained control of the graph

• Generate computation graph statically
  • Dynamic or static scheduling
  • Suitable for streaming and data query applications
  • Hard to express complex computations, especially with loops

• Hybrid approach
  • Combine both static and dynamic graphs
Task Graph Execution

- Task Scheduler is pluggable
- Executor is pluggable
- Scheduler running on all the workers

Scheduling Algorithms
- Streaming
  - Round robin
  - First fit
- Batch
  - Data locality aware
But internally to each job you can also elegantly express algorithm as dataflow but with more stringent performance constraints.

Coarse Grain Dataflows links jobs in such a pipeline:

- Data preparation
- Clustering
- Dimension Reduction
- Visualization

Internal Execution Dataflow Nodes:

- Reduce
- HPC Communication
- Maps

Corresponding to classic Spark K-means Dataflow:

- \( P = \text{loadPoints()} \)
- \( C = \text{loadInitCenters()} \)
- for (int \( i = 0; i < 10; i++ \)) {
  - \( T = P.\text{map().withBroadcast}(C) \)
  - \( C = T.\text{reduce()} \)
}

Dataflow at Different Grain sizes
Workflow vs Dataflow: Different grain sizes and different performance trade-offs

Workflow Controlled by Workflow Engine or a Script

Dataflow application running as a single job

The dataflow can expand from Edge to Cloud
NiFi Workflow
Systems State

- **State** is handled differently in systems
  - CORBA, AMT, MPI and Storm/Heron have long running tasks that preserve state
  - Spark and Flink preserve datasets across dataflow node using in-memory databases
  - All systems agree on coarse grain dataflow; only keep state by exchanging data

Spark Kmeans Dataflow

- \( P = \text{loadPoints}() \)
- \( C = \text{loadInitCenters}() \)

```
for (int i = 0; i < 10; i++) {
    T = P.map().withBroadcast(C)
    C = T.reduce()
}
```

Save State at Coordination Point
Store C in RDD
Fault Tolerance and State

• Similar form of **check-pointing** mechanism is used already in HPC and Big Data
  • although HPC informal as doesn’t typically specify as a dataflow graph
  • Flink and Spark do better than MPI due to use of **database** technologies; MPI is a bit harder due to richer state but there is an obvious integrated model using RDD type snapshots of MPI style jobs

• Checkpoint **after each stage of the dataflow graph** (**at location of intelligent dataflow nodes**)
  • Natural synchronization point
  • Let’s allows user to choose when to checkpoint (not every stage)
  • Save state as user specifies; Spark just saves Model state which is insufficient for complex algorithms
Implementing Twister2 Futures

Iterative MapReduce
http://www.iterativemapreduce.org/
Twister2 Timeline: End of August 2018

• Twister:Net Dataflow Communication API
  • Dataflow communications with MPI or TCP

• Harp for Machine Learning (Custom BSP Communications)
  • Rich collectives
  • Around 30 ML algorithms

• HDFS Integration

• Task Graph
  • Streaming - Storm model
  • Batch analytics - Hadoop

• Deployments on Docker, Kubernetes, Mesos (Aurora), Nomad, Slurm
Twister2 Timeline: End of December 2018

• Native MPI integration to Mesos, Yarn
• Naiad model based Task system for Machine Learning
• Link to Pilot Jobs
• Fault tolerance
  • Streaming
  • Batch
• Hierarchical dataflows with Streaming, Machine Learning and Batch integrated seamlessly
• Data abstractions for streaming and batch (Streamlets, RDD)
• Workflow graphs (Kepler, Spark) with linkage defined by Data Abstractions (RDD)
• End to end applications
Twister2 Timeline: After December 2018

• Dynamic task migrations
• RDMA and other communication enhancements
• Integrate parts of Twister2 components as big data systems enhancements (i.e. run current Big Data software invoking Twister2 components)
  • Heron (easiest), Spark, Flink, Hadoop (like Harp today)
• Support different APIs (i.e. run Twister2 looking like current Big Data Software)
  • Hadoop
  • Spark (Flink)
  • Storm
• Refinements like Marathon with Mesos etc.
• Function as a Service and Serverless
• Support higher level abstractions
  • Twister:SQL
Summary of Twister2: Next Generation HPC Cloud + Edge + Grid

• We have built a high performance data analysis library SPIDAL
• We have integrated HPC into many Apache systems with HPC-ABDS with rich set of collectives
• We have done a preliminary analysis of the different runtimes of Hadoop, Spark, Flink, Storm, Heron, Naiad, DARMA (HPC Asynchronous Many Task) and identified key components
• There are different technologies for different circumstances but can be unified by high level abstractions such as communication/data/task API’s
• Apache systems use dataflow communication which is natural for distributed systems but slower for classic parallel computing
  • No standard dataflow library (why?). Add Dataflow primitives in MPI-4?
• HPC could adopt some of tools of Big Data as in Coordination Points (dataflow nodes), State management (fault tolerance) with RDD (datasets)
• Could integrate dataflow and workflow in a cleaner fashion
• Not clear so many big data and resource management approaches needed