Iterative MapReduce for Scientific Computing and Data Analytics

Roy Campbell
Reza Farivar

Judy Qiu, Bingjing Zhang, Mattei Zaharia et. al.
Iterative Computations

Youtube Video Suggestion

PageRank

BFS

Clustering

Pattern Recognition
Intel’s Application Stack
Motivation

• Iterative algorithms are commonly used in many domains

• Traditional MapReduce and classical parallel runtimes cannot solve iterative algorithms efficiently
  – Hadoop: Repeated data access to HDFS, no optimization to data caching and data transfers
  – MPI: no nature support of fault tolerance and programming interface is complicated
What is Iterative MapReduce

- **Iterative MapReduce**
  - Mapreduce is a Programming Model instantiating the paradigm of bringing computation to data
  - Iterative Mapreduce extends Mapreduce programming model and support iterative algorithms for Data Mining and Data Analysis

- **Interoperability**
  - Is it possible to use the same computational tools on HPC and Cloud?
  - Enabling scientists to focus on science not programming distributed systems

- **Reproducibility**
  - Is it possible to use Cloud Computing for Scalable, Reproducible Experimentation?
  - Sharing results, data, and software
Iterative MapReduce Frameworks

- **Twister**\(^1\)
  - Map->Reduce->Combine->Broadcast
  - Long running map tasks (data in memory)
  - Centralized driver based, statically scheduled.
- **Daytona**
  - Microsoft Iterative MapReduce on Azure using cloud services
  - Architecture similar to Twister
- **Haloop**
  - On disk caching, Map/reduce input caching, reduce output caching
- **Spark**
  - Iterative MapReduce Using Resilient Distributed Dataset to ensure the fault tolerance
- **PIC: Partitioned Iterative Convergence**
  - Partition, solve local problems, merge the results. Iterate.
- **Pregel**
  - Graph processing from Google
- **Mahout**
  - Apache project for supporting data mining algorithms
Others

- Network Levitated Merge\textsuperscript{[8]}
  - RDMA/Infiniband based shuffle & merge
- Mate-EC2\textsuperscript{[9]}
  - Local reduction object
- Asynchronous Algorithms in MapReduce\textsuperscript{[10]}
  - Local & global reduce
- MapReduce online\textsuperscript{[11]}
  - online aggregation, and continuous queries
  - Push data from Map to Reduce
- Orchestra\textsuperscript{[12]}
  - Data transfer improvements for MR
- iMapReduce\textsuperscript{[13]}
  - Async iterations, One to one map & reduce mapping, automatically joins loop-variant and invariant data
- CloudMapReduce\textsuperscript{[14]} & Google AppEngine MapReduce\textsuperscript{[15]}
  - MapReduce frameworks utilizing cloud infrastructure services
# Applications & Different Interconnection Patterns

<table>
<thead>
<tr>
<th>Map Only</th>
<th>Classic MapReduce</th>
<th>Iterative MapReduce</th>
<th>Loosely Synchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Input" /> → <img src="image2" alt="map" /> → <img src="image3" alt="Output" /></td>
<td><img src="image4" alt="Input" /> → <img src="image5" alt="map" /> → <img src="image6" alt="reduce" /></td>
<td><img src="image7" alt="Input" /> → <img src="image8" alt="map" /> → <img src="image9" alt="reduce" /></td>
<td><img src="image10" alt="iterations" /> → <img src="image11" alt="Pij" /></td>
</tr>
</tbody>
</table>

## Map Only
- **CAP3 Analysis**
  - Document conversion (PDF -> HTML)
  - Brute force searches in cryptography
  - Parametric sweeps
- **Document conversion**
- **Brute force searches in cryptography**
- **Parametric sweeps**

## Classic MapReduce
- **High Energy Physics (HEP) Histograms**
- **SWG gene alignment**
- **Distributed search**
- **Distributed sorting**
- **Information retrieval**

## Iterative MapReduce
- **Expectation maximization algorithms**
- **Clustering**
- **Linear Algebra**
- **Kmeans**
- **Deterministic Annealing Clustering**
- **Multidimensional Scaling (MDS)**
- **Solving Differential Equations and**
- **particle dynamics with short range forces**

## Loosely Synchronous
- Many MPI scientific applications utilizing wide variety of communication constructs including local interactions

---

## Domain of MapReduce and Iterative Extensions

![Flowchart](image12)
Data Intensive Kmeans Clustering

— Image Classification: 1.5 TB; 500 features per image; 10k clusters
1000 Map tasks; 1GB data transfer per Map task node
Iterative MapReduce

Data Deluge

Experiencing in many domains

MapReduce

Data Centered, QoS

Classic Parallel Runtimes (MPI)

Efficient and Proven techniques

Expand the Applicability of MapReduce to more classes of Applications

Map-Only

Input → map → Output

MapReduce

Input → map → reduce

Iterative MapReduce

Input → map → reduce → iterations

More Extensions

Pij

More Extensions

Input → map → reduce → iterations

More Extensions
A Programming Model for Iterative MapReduce

- Distributed data access
- In-memory MapReduce
- Distinction on static data and variable data (data flow vs. δ flow)
- Cacheable map/reduce tasks (long running tasks)
- Combine operation
- Support fast intermediate data transfers
Twister Iterative MapReduce

Infrastructure for Iterative MapReduce Programming

- Distinction on static and variable data
- Configurable long running (cacheable) map/reduce tasks
- Pub/sub messaging based communication/data transfers
- Broker Network for facilitating communication
Twister Programming Model

Main program’s process space

- `configureMaps(..)`
- `configureReduce(..)`

while (`condition`){
- `runMapReduce(..)`
- `monitorTillCompletion(..)`

• Main program may contain many MapReduce invocations or iterative MapReduce invocations

Communications/data transfers via the pub-sub broker network & direct TCP

Worker Nodes

Local Disk

Communications/data transfers

Combine() operation

UpdateCondition()

}//end while

close()
Twister Design Features

Concepts and Features in Twister

- Static Data Loaded only once
- Long running map/reduce task threads (cached)
- Combine operation to collect all reduce outputs
- Direct TCP Broadcast/Scatter transfer of dynamic KeyValue pairs
- Direct data transfer via pub/sub
- Fault detection and recovery support between iterations
Twister APIs

1. `configureMaps()`
2. `configureMaps(String partitionFile)`
3. `configureMaps(List<Value> values)`
4. `configureReduce(List<Value> values)`

5. `addToMemCache(Value value)`
6. `addToMemCache(List<Value> values)`
7. `cleanMemCache()`

8. `runMapReduce()`
9. `runMapReduce(List<KeyValuePair> pairs)`
10. `runMapReduceBCast(Value value)`
11. `map(MapOutputCollector collector, Key key, Value val)`
12. `reduce(ReduceOutputCollector collector, Key key, List<Value> values)`
13. `combine(Map<Key, Value> keyValues)`
Twister Architecture

Master Node
Twister Driver
Main Program

Twister Daemon

Worker Pool
Local Disk
Worker Node

Scripts perform:
Data distribution, data collection, and partition file creation

Pub/Sub Broker Network and Collective Communication Service

map
reduce
Cacheable tasks

...
Data Storage and Caching

- Use local disk to store static data files
- Use Data Manipulation Tool to manage data
- Use partition file to locate data
- Data on disk can be cached into local memory
- Support using NFS (version 0.9)
Data Manipulation Tool

- Provides basic functionality to manipulate data across the local disks of the compute nodes
- Data partitions are assumed to be files (Contrast to fixed sized blocks in Hadoop)
- Supported commands:
  - mkdir, rmdir, put, putall, get, ls
  - Copy resources
  - Create Partition File

A common directory in local disks of individual nodes e.g. /tmp/twister_data
### Partition File

<table>
<thead>
<tr>
<th>File No</th>
<th>Node IP</th>
<th>Daemon No</th>
<th>File Partition Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>149.165.229.10</td>
<td>0</td>
<td>/tmp/zhangbj/data/kmeans/km_142.bin</td>
</tr>
<tr>
<td>8</td>
<td>149.165.229.34</td>
<td>24</td>
<td>/tmp/zhangbj/data/kmeans/km_382.bin</td>
</tr>
<tr>
<td>16</td>
<td>149.165.229.147</td>
<td>78</td>
<td>/tmp/zhangbj/data/kmeans/km_370.bin</td>
</tr>
<tr>
<td>24</td>
<td>149.165.229.12</td>
<td>2</td>
<td>/tmp/zhangbj/data/kmeans/km_57.bin</td>
</tr>
</tbody>
</table>

- Partition file allows duplicates to show replica availability
- One data file and its replicas may reside in multiple nodes
- Provide information required for caching
Pub/Sub Messaging

• For small control messages only
• Currently support
  – NaradaBrokering: single broker and manual configuration
  – ActiveMQ: multiple brokers and auto configuration
Broadcast

- Use **addToMemCache** operation to broadcast dynamic data required by all tasks in each iteration

  ```python
  MemCacheAddress memCacheKey
  = driver.addToMemCache(centroids);
  TwisterMonitor monitor
  = driver.runMapReduceBCast(memCacheKey);
  ```

- Replace original broker-based methods to direct TCP data transfers
- Algorithm auto selection
- Topology-aware broadcasting
Twister Communications

- **Broadcasting**
  - Data could be large
  - Chain & MST

- **Map Collectives**
  - Local merge

- **Reduce Collectives**
  - Collect but no merge

- **Combine**
  - Direct download or Gather
Twister Broadcast Comparison
Sequential vs. Parallel implementations

Per Iteration Cost (Before)

- Combine
- Shuffle & Reduce
- Map
- Broadcast

Per Iteration Cost (After)

Time (Unit: Seconds)
Twister Broadcast Comparison: Ethernet vs. InfiniBand

InfiniBand Speed Up Chart – 1GB bcast

- Ethernet (blue)
- InfiniBand (red)

Chart shows the comparison of broadcast speed between Ethernet and InfiniBand, indicating a significant speed advantage for InfiniBand.
Failure Recovery

• Recover at iteration boundaries
• Does not handle individual task failures

• Any Failure (hardware/daemons) result the following failure handling sequence
  – Terminate currently running tasks (remove from memory)
  – Poll for currently available worker nodes (& daemons)
  – Configure map/reduce using static data (re-assign data partitions to tasks depending on the data locality)
  – Re-execute the failed iteration
Berkeley Spark

• Extend the MapReduce model to better support two common classes of analytics apps:
  – **Iterative** algorithms (machine learning, graphs)
  – **Interactive** data mining

• Enhance programmability:
  – Integrate into Scala programming language
  – Allow interactive use from Scala interpreter
Motivation

Most current cluster programming models are based on *acyclic data flow* from stable storage to stable storage.
Motivation

• Most current cluster programming models are based on *acyclic data flow* from stable storage to stable storage

**Benefits of data flow**: runtime can decide where to run tasks and can automatically recover from failures
Motivation

• Acyclic data flow is inefficient for applications that repeatedly reuse a *working set* of data:
  – **Iterative** algorithms (machine learning, graphs)
  – **Interactive** data mining tools (R, Excel, Python)

• With current frameworks, apps reload data from stable storage on each query
Solution: Resilient Distributed Datasets (RDDs)

• Allow apps to keep working sets in memory for efficient reuse
• Retain the attractive properties of MapReduce
  – Fault tolerance, data locality, scalability
• Support a wide range of applications
Programming Model

Resilient distributed datasets (RDDs)

– Immutable, partitioned collections of objects
– Created through parallel transformations (map, filter, groupBy, join, ...) on data in stable storage
– Can be cached for efficient reuse

Actions on RDDs

– Count, reduce, collect, save, ...
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split(\"\t\")(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
...
```

**Result:** scaled to 1 TB data in 5-7 sec (vs 170 sec for on-disk data)
RDD Fault Tolerance

RDDs maintain *lineage* information that can be used to reconstruct lost partitions

Ex: messages = textFile(...).filter(_.startsWith("ERROR")) .map(_.split('\t')(2))

```
HDFS File
  filter (func = _.contains(...))
Filtered RDD
  map (func = _.split(...))
Mapped RDD
```
Example: Logistic Regression

Goal: find best line separating two sets of points
Example: Logistic Regression

```scala
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: "+ w)
```
Logistic Regression Performance

![Graph showing running time vs number of iterations for Hadoop and Spark, with 127 s/iteration and first iteration 174 s, further iterations 6 s.](image)
Spark Applications

- In-memory data mining on Hive data (Conviva)
- Predictive analytics (Quantifind)
- City traffic prediction (Mobile Millennium)
- Twitter spam classification (Monarch)

Collaborative filtering via matrix factorization

...
• Aggregations on many keys w/ same WHERE clause

• 40× gain comes from:
  – Not re-reading unused columns or filtered records
  – Avoiding repeated decompression
  – In-memory storage of deserialized objects
Frameworks Built on Spark

• Pregel on Spark (Bagel)
  – Google message passing model for graph computation
  – 200 lines of code

• Hive on Spark (Shark)
  – 3000 lines of code
  – Compatible with Apache Hive
  – ML operators in Scala
Implementation

Runs on Apache Mesos to share resources with Hadoop & other apps
Can read from any Hadoop input source (e.g. HDFS)

• No changes to Scala compiler
Spark Scheduler

Dryad-like DAGs
Pipelines functions within a stage
Cache-aware work reuse & locality
Partitioning-aware to avoid shuffles

A: = cached data partition
Interactive Spark

Modified Scala interpreter to allow Spark to be used interactively from the command line

Required two changes:

– Modified wrapper code generation so that each line typed has references to objects for its dependencies

– Distribute generated classes over the network
Understanding MapReduce in Context

Support Scientific Simulations (Data Mining and Data Analysis)

Applications


Programming Model

High Level Language

Cross Platform Iterative MapReduce (Collectives, Fault Tolerance, Scheduling)

Runtime

Security, Provenance, Portal

Services and Workflow

Runtime

Distributed File Systems

Object Store

Data Parallel File System

Storage

Infrastructure

Linux HPC Bare-system

Virtualization

Amazon Cloud

Windows Server HPC Bare-system

Virtualization

Azure Cloud

Grid Appliance

Hardware

CPU Nodes

GPU Nodes