Tutorial

Iterative MapReduce

http://www.iterativemapreduce.org/

Workshop

Jaliya Ekanayake
Community Grids Laboratory,
Digital Science Center
Pervasive Technology Institute
Indiana University
Acknowledgements to:

Team at IU

- Seung-Hee Bae,
- Jong Choi
- Saliya Ekanayake
- Geoffrey Fox
- Thilina Gunarathne
- Ryan Hartman
- Adam Lee
- Hui Li
- Judy Qiu
- Binging Shang
- Stephen Wu
- Ruan Yang
Resources

Twister Website  http://www.iterativemapreduce.org/
- Twister Tutorial Package
  - http://salsahpc.indiana.edu/tutorial/apps/Twister.zip
- Naradabrokering
  - http://www.naradabrokering.org/software.htm
- Account Info
  - trainXXX@bigdata.india.futuregrid.org OR trainXXX@bigdata.sierra.futuregrid.org
- Request a New Node
  - qsub -l
- Tutorial Pages
  - http://salsahpc.indiana.edu/tutorial/twister-intro.html
  - http://salsahpc.indiana.edu/tutorial/twister_install.htm
  - http://salsahpc.indiana.edu/tutorial/twister_wordcount_user_guide.htm
  - http://salsahpc.indiana.edu/tutorial/twister_blast_user_guide.htm
  - http://salsahpc.indiana.edu/tutorial/twister_kmeans_user_guide.htm
Contents

• Twister: Runtime for Iterative MapReduce
• Sample Application & MapReduce algorithm
• Code Walkthrough
• Hands-on exercise
Motivation

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

MapReduce

Iterative MapReduce

More Extensions

Input
map
Output

Input
map
reduce

iterations

Pij
Iterative MapReduce using Existing Runtimes

- Focuses mainly on single step map->reduce computations
- Considerable overheads from:
  - Reinitializing tasks
  - Reloading static data
  - Communication & data transfers
Iterative MapReduce using Twister

- Distributed data access
- 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

Static Data
Loaded only once

Configure()

Main Program

Iterate

Map(Key, Value)

Reduce (Key, List<Value>)

Combine (Map<Key,Value>)

Long running map/reduce tasks (cached)

Combiner operation to collect all reduce outputs

Direct data transfer via pub/sub
Twister Programming Model

configureMaps(..)
configureReduce(..)
runMapReduce(..)
while(condition){
while(condition){
runMapReduce(..)

May send <Key,Value> pairs directly

Combine() operation

Communications/data transfers via the pub-sub broker network

User program’s process space

Local Disk

Two configuration options:
1. Using local disks (only for maps)
2. Using pub-sub bus
Twister Architecture

Master Node
- Twister Driver
- Main Program

Pub/sub Broker Network
- One broker serves several Twister daemons

Twister Daemon
- Map
- Reduce
- Cacheable tasks

Worker Node
- Local Disk
- Worker Pool

Scripts perform:
- Data distribution, data collection,
- and partition file creation
Input/Output Handling

• **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>4</td>
<td>156.56.104.96</td>
<td>2</td>
<td>/home/jaliya/data/mds/GD-4D-23.bin</td>
</tr>
<tr>
<td>5</td>
<td>156.56.104.96</td>
<td>2</td>
<td>/home/jaliya/data/mds/GD-4D-0.bin</td>
</tr>
<tr>
<td>6</td>
<td>156.56.104.96</td>
<td>2</td>
<td>/home/jaliya/data/mds/GD-4D-27.bin</td>
</tr>
<tr>
<td>7</td>
<td>156.56.104.96</td>
<td>2</td>
<td>/home/jaliya/data/mds/GD-4D-20.bin</td>
</tr>
<tr>
<td>8</td>
<td>156.56.104.97</td>
<td>4</td>
<td>/home/jaliya/data/mds/GD-4D-23.bin</td>
</tr>
<tr>
<td>9</td>
<td>156.56.104.97</td>
<td>4</td>
<td>/home/jaliya/data/mds/GD-4D-25.bin</td>
</tr>
<tr>
<td>10</td>
<td>156.56.104.97</td>
<td>4</td>
<td>/home/jaliya/data/mds/GD-4D-18.bin</td>
</tr>
<tr>
<td>11</td>
<td>156.56.104.97</td>
<td>4</td>
<td>/home/jaliya/data/mds/GD-4D-15.bin</td>
</tr>
</tbody>
</table>

- Partition file allows duplicates
- One data partition may reside in multiple nodes
- In an event of failure, the duplicates are used to reschedule the tasks
The use of pub/sub messaging

- Intermediate data transferred via the broker network
- Network of brokers used for load balancing
  - Different broker topologies
- Interspersed computation and data transfer minimizes large message load at the brokers
- Currently supports
  - NaradaBrokering
  - ActiveMQ

E.g.
100 map tasks, 10 workers in 10 nodes

~ 10 tasks are producing outputs at once
Scheduling

• Twister supports long running tasks
• Avoids unnecessary initializations in each iteration
• Tasks are scheduled statically
  – Supports task reuse
  – May lead to inefficient resources utilization
• Expect user to randomize data distributions to minimize the processing skews due to any skewness in data
Fault Tolerance

Recover at iteration boundaries

• Does not handle individual task failures

• Assumptions:
  – Broker network is reliable
  – Main program & Twister Driver has no failures

• Any failures (hardware/daemons) result the following fault 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
Twister API

1. configureMaps(PartitionFile partitionFile)
2. configureMaps(Value[] values)
3. configureReduce(Value[] values)
4. runMapReduce()
5. runMapReduce(KeyValue[] keyValues)
6. runMapReduceBCast(Value value)
7. map(MapOutputCollector collector, Key key, Value val)
8. reduce(ReduceOutputCollector collector, Key key, List<Value> values)
9. combine(Map<Key, Value> keyValues)
Twister Tutorial

• Complete Tutorial
  – http://salsahpc.indiana.edu/tutorial/twister-intro.html
Questions?
K-Means Clustering

- Points distributions in n dimensional space
- Identify a given number of cluster centers
- Use Euclidean distance to associate points to cluster centers
- Refine the cluster centers iteratively
Each map task processes a data partition.

Main Program

While()

\[ n_{th} \text{ cluster centers} \]

While()

\[ (n+1)_{th} \text{ cluster centers} \]

- Map tasks calculate Euclidean distance from each point in its partition to each cluster center.
- Map tasks assign points to cluster centers and sum the partial cluster center values.
- Emit cluster center sums + number of points assigned.
- Reduce task sums all the corresponding partial sums and calculate new cluster centers.
// Job Configurations
JobConf jobConf = new JobConf("kmeans-map-reduce" + uuidGen.generateTimeBasedUUID());
jobConf.setMapperClass(KMeansMapTask.class);
jobConf.setReducerClass(KMeansReduceTask.class);
jobConf.setCombinerClass(KMeansCombiner.class);
jobConf.setNumMapTasks(numMapTasks);
jobConf.setNumReduceTasks(numReducers);

TwisterDriver driver = new TwisterDriver(jobConf);
driver.configureMaps(partitionFile);

// Main iteration for K-Means clustering
boolean complete = false;
while (!complete) {
    monitor = driver.runMapReduceBCast(cData);
    monitor.monitorTillCompletion();
    DoubleVectorData newCData = ((KMeansCombiner) driver.getCurrentCombiner()).getResults();
    totalError = getError(cData, newCData);
    cData = newCData;
    if (totalError < THRESHOLD) {
        complete = true;
        break;
    }
    loopCount++;
}
public void configure(JobConf jobConf, MapperConf mapConf) throws TwisterException {
    this.vectorData = new DoubleVectorData();
    fileData = (FileData) mapConf.getDataPartition();
    try {
        vectorData.loadDataFromFile(fileData.getFileName());
    } catch (Exception e) {
        throw new TwisterException(e);
    }
}

/**
 * Map function for the K-means clustering. Calculates the Euclidean
 * distance between data points and the given cluster centers. Next it
 * calculates the partial cluster centers as well.
 */

public void map(MapOutputCollector collector, Key key, Value val) throws TwisterException {
    double[][] data = vectorData.getData();
    DoubleVectorData cData = new DoubleVectorData();

    DoubleVectorData newCData = new DoubleVectorData(newCentroids, numCentroids, vecLen + 1);
    collector.collect(new StringKey(outKey), new BytesValue(newCData.getBytes()));
}
public void reduce(ReduceOutputCollector collector, Key key, 
List<Value> values) throws TwisterException {

DoubleVectorData newCentroidData = 
    new DoubleVectorData(newCentroids, numData, lenData);
collector.collect(key, new BytesValue(newCentroidData.getBytes()));

/**
 * Combines the reduce outputs to a single value.
 */
public void combine(Map<Key, Value> keyValues) throws TwisterException {
    assert (keyValues.size() == 1);// There should be a single value here.
    Iterator<Key> ite = keyValues.keySet().iterator();
    Key key = ite.next();
    BytesValue val = (BytesValue) keyValues.get(key);
    try {
        this.results.fromBytes(val.getBytes());
    } catch (SerializationException e) {
        throw new TwisterException(e);
    }
Login into Futuregrid Accounts

1. ssh trainXXX@bigdata.[india, sierra].futuregrid.org
2. [train200@s1 ~]$
   \textbf{qsub} \ -I

   \begin{verbatim}
   [train199@s1 ~]$
   \textbf{qsub} \ -I
   \textbf{qsub:} \ waiting \ for \ job \ 291814.s82 \ to \ start
   \textbf{qsub:} \ job \ 291814.s82 \ ready
   \end{verbatim}

   [train199@s10 ~]$ [train199@s10 ~]$ [train199@s10 ~]$

3. Create 3 command line windows (shells)
   \begin{itemize}
   \item ssh trainXXX@bigdata.[india, sierra].futuregrid.org
   \item ssh sxx
   \end{itemize}
Start NaradaBrokering

In the first command windows (shell)

1. cd $NBHOME/bin
2. ./startbr.sh
Start Twister

In the second command window (shell)

• cd $TWISTER_HOME/bin
• ./star_twister.sh

```
[train199@s10 bin]$ ./start_twister.sh
127.0.0.1
127.0.0.1
[train199@s10 bin]$ 0   [main] INFO  cgl.imr.worker.DaemonWorker - Daemon no: 0 started.
```

• If you see something like below
  – Make sure you are logged into the reserved node using qsub -l

```
[train199@s10 bin]$ 0   [main] INFO  cgl.imr.worker.DaemonWorker - Daemon no: 0 started.
2   [main] ERROR  cgl.imr.worker.TwisterDaemon - TwisterDaemon no0quiting due to error.
java.net.BindException: Address already in use
  at java.net.PlainSocketImpl.socketBind(Native Method)
```

• Edit `twister.properties` and change the following
  – daemon_port = 12500  //change this to something else

```
[train199@s10 bin]$: vi twister.properties
```

```
daemons_per_node = 1
workers_per_daemon = 8
pubsub_broker = NaradaBrokering
daemon_port = 18500
nodes_file = /N/u/train199/Twister/bin/nodes
app_dir = /N/u/train199/Twister/apps
data_dir = /N/u/train199/Twister/data
```
Run K-Means Clustering (1)

In the third command window (shell)

1. Go to the samples directory
   - cd $TWISTER_HOME/samples/kmeans/bin

2. Split data
   - The data is already partitioned and is in $TWISTER_HOME/samples/kmeans/input

3. Create a directory to hold these data
   - cd $TWISTER_HOME/bin
   - ./twister.sh mkdir kmeans
Run K-Means Clustering (2)

In the third command window (shell)

4. Distribute data
   - 
     ```
     ./twister.sh put 
     $TWISTER_HOME/samples/kmeans/input kmeans
     ```

5. Create a partition file
   - 
     ```
     ./create_partition_file.sh kmeans kmeans_  
     $TWISTER_HOME/samples/kmeans/bin/kmeans.pf
     ```

6. Run Twister Kmeans application
   - 
     ```
     cd $TWISTER_HOME/samples/kmeans/bin
     ./run_kmeans.sh init_cluster.txt 8 kmeans.pf
     ```
The Output

Once you are done please close Twister and then Naradabrokering

cd $TWISTER_HOME/bin
./stop_twister.sh

cd $NBHOME/bin
./stopbr.sh
Demo: Multi-dimensional Scaling

While(condition) {
  \( \langle X \rangle = [A] [B] \langle C \rangle \)
  \( C = \text{CalcStress}(\langle X \rangle) \)
}

While(condition) {
  \( \langle T \rangle = \text{MapReduce1}([B], \langle C \rangle) \)
  \( \langle X \rangle = \text{MapReduce2}([A], \langle T \rangle) \)
  \( C = \text{MapReduce3}(\langle X \rangle) \)
}

- Maps high dimensional data to lower dimensions (typically 2D or 3D)
- SMACOF (Scaling by Majorizing of COmplicated Function)[1]

Conclusions & Future Work

• Twister extends the MapReduce to iterative algorithms

• Several iterative algorithms we have implemented
  – K-Means Clustering
  – Pagerank
  – Matrix Multiplication
  – Multi dimensional scaling (MDS)
  – Breadth First Search

• Integrating a distributed file system
• Programming with side effects yet support fault tolerance
Questions?

Thank you!
More Applications

• Saliya will present from here...
Performance Evaluation

- **Hardware Configurations**

<table>
<thead>
<tr>
<th>Cluster ID</th>
<th>Cluster-I</th>
<th>Cluster-II</th>
</tr>
</thead>
<tbody>
<tr>
<td># nodes</td>
<td>32</td>
<td>230</td>
</tr>
<tr>
<td># CPUs in each node</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td># Cores in each CPU</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Total CPU cores</td>
<td>768</td>
<td>1840</td>
</tr>
<tr>
<td>Supported OSs</td>
<td>Linux (Red Hat Enterprise Linux Server release 5.4 - 64 bit) Windows (Windows Server 2008 - 64 bit)</td>
<td>Red Hat Enterprise Linux Server release 5.4 - 64 bit</td>
</tr>
</tbody>
</table>

- We use the academic release of DryadLINQ, Apache Hadoop version 0.20.2, and Twister for our performance comparisons.
- Both Twister and Hadoop use JDK (64 bit) version 1.6.0_18, while DryadLINQ and MPI uses Microsoft .NET version 3.5.
Pair wise Sequence Comparison using Smith Waterman Gotoh

- Typical MapReduce computation
- Comparable efficiencies
- Twister performs the best
Pagerank – An Iterative MapReduce Algorithm

- Well-known pagerank algorithm [1]
- Used ClueWeb09 [2] (1TB in size) from CMU
- Reuse of map tasks and faster communication pays off

Multi-dimensional Scaling

While(condition) {
    <X> = [A] [B] <C>
    C = CalcStress(<X>)
}

While(condition) {
    <T> = MapReduce1([B],<C>)
    <X> = MapReduce2([A],<T>)
    C = MapReduce3(<X>)
}

- Maps high dimensional data to lower dimensions (typically 2D or 3D)
- SMACOF (Scaling by Majorizing of COmplicated Function)[1]

Related Work

- General MapReduce References:
  - Google MapReduce
  - Apache Hadoop
  - Microsoft DryadLINQ
  - Pregel: Large-scale graph computing at Google
  - Sector/Sphere
  - All-Pairs
  - SAGA: MapReduce
  - Disco