Twister
Iterative MapReduce

Tutorial

http://www.iterativemapreduce.org/

Big Data for Science **Workshop**

Jaliya Ekanayake

Community Grids Laboratory,

Digital Science Center

Pervasive Technology Institute

Indiana University

PERVASIVE TECHNOLOGY INSTITUTE
INDIANA UNIVERSITY

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

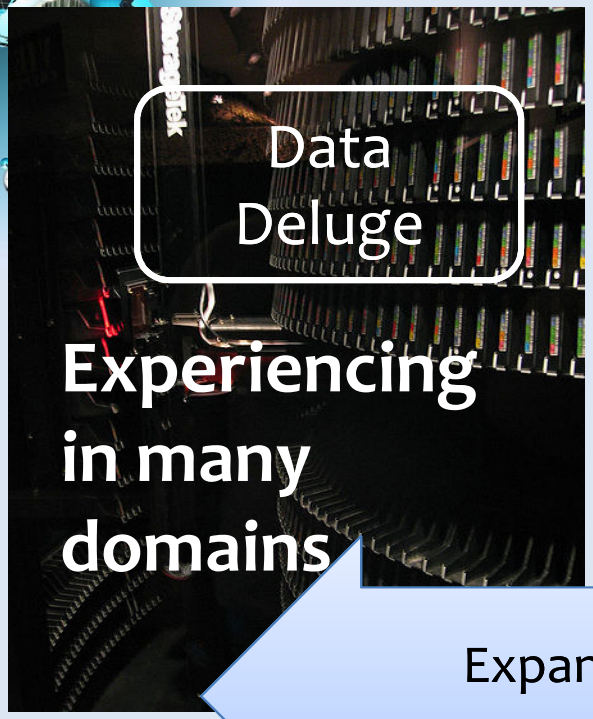
Resources

- 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 -I
- 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



MapReduce

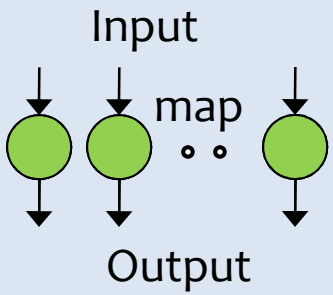
Classic Parallel Runtimes (MPI)

Data Centered, QoS

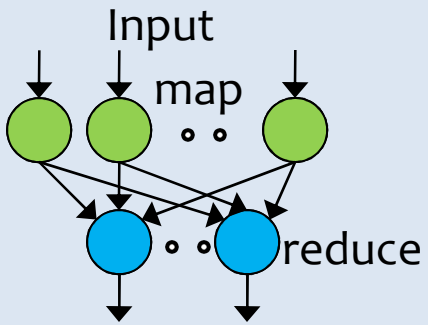
Efficient and Proven techniques

Expand the Applicability of MapReduce to more **classes** of Applications

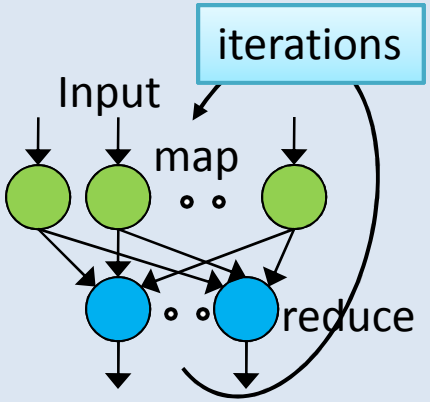
Map-Only



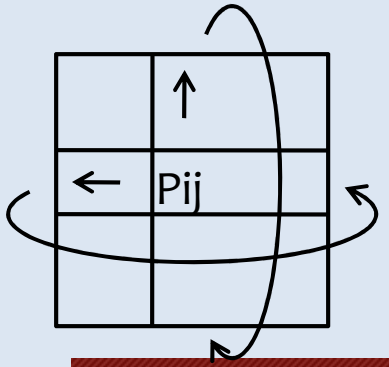
MapReduce



Iterative MapReduce

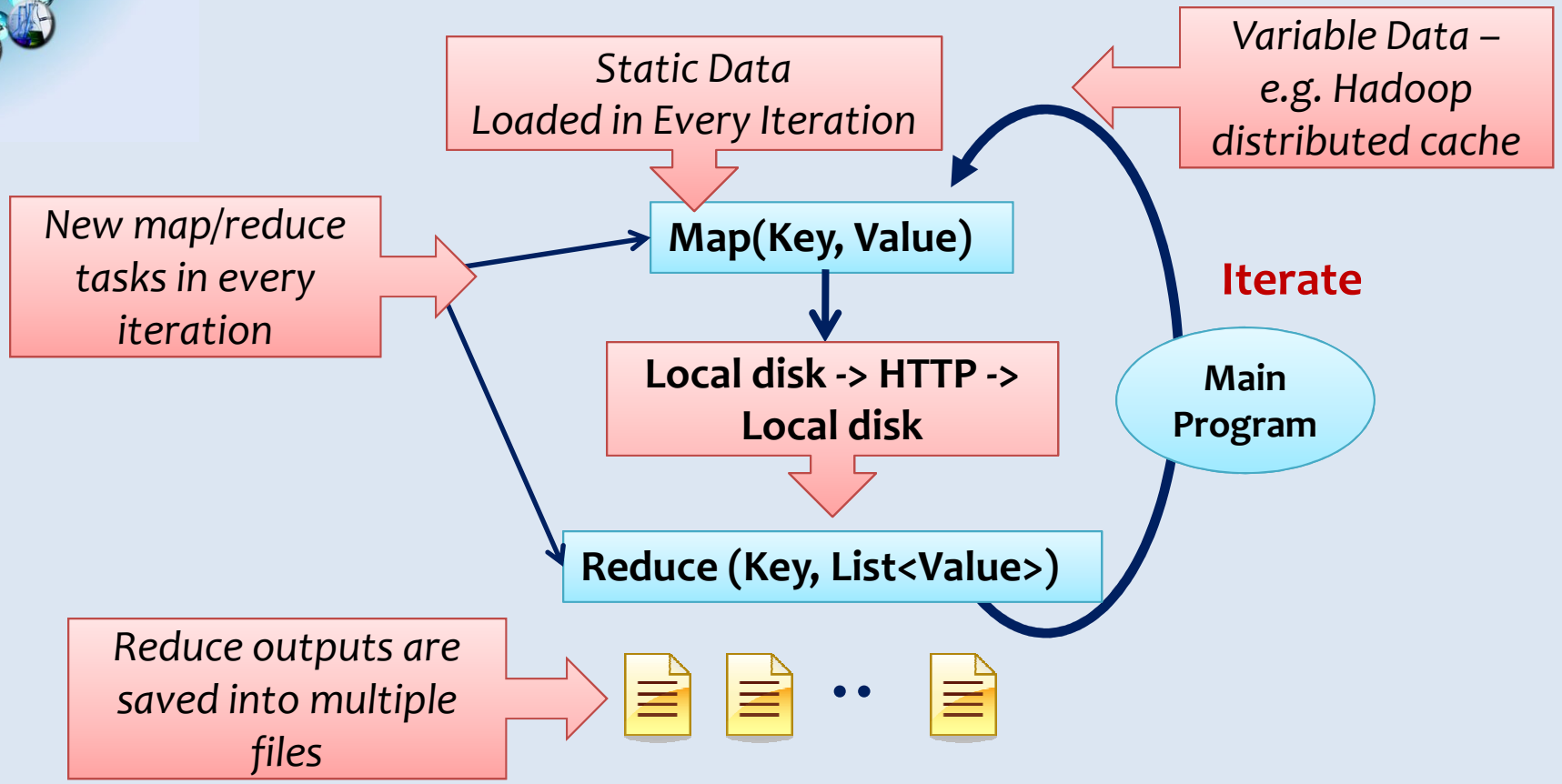


More Extensions



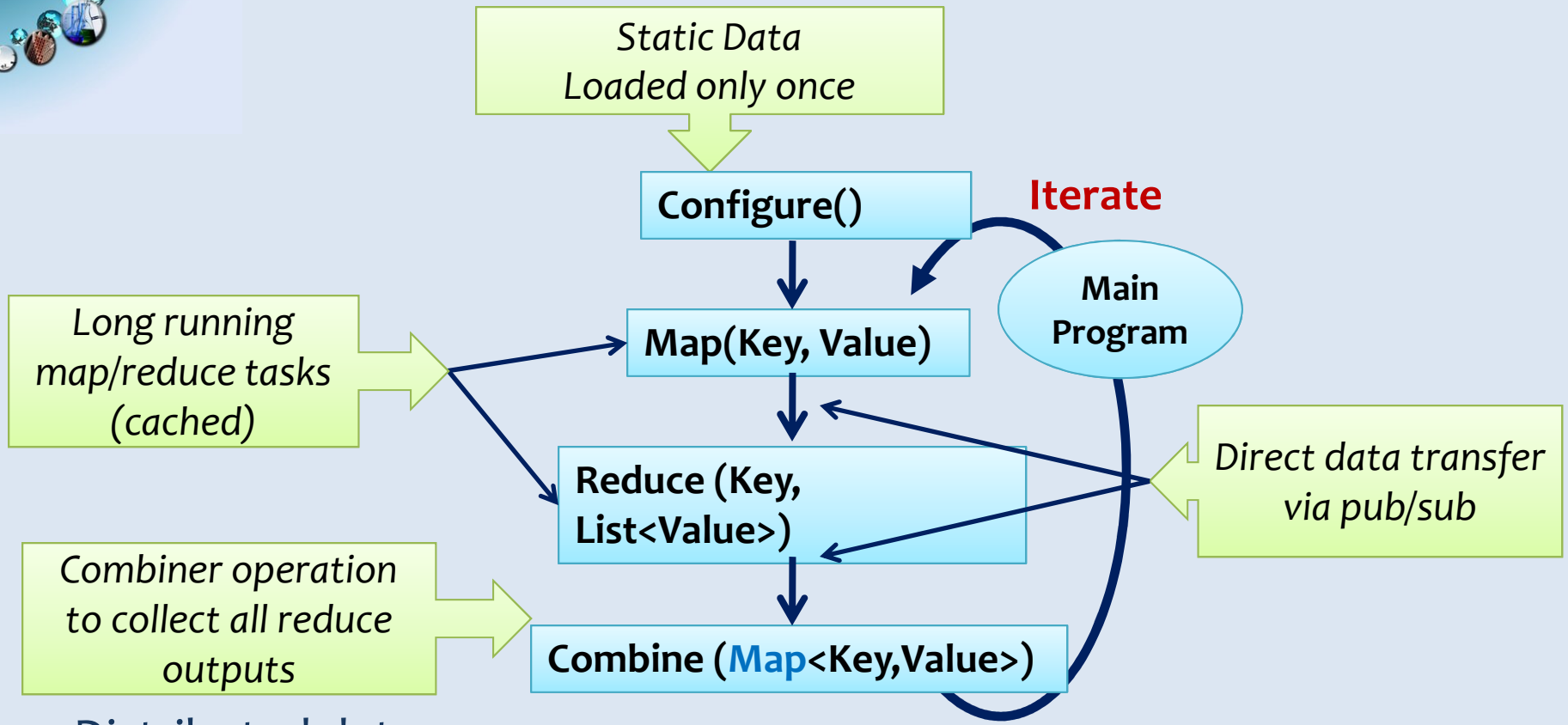
INDIANA UNIVERSITY

Iterative MapReduce using Existing Runtimes



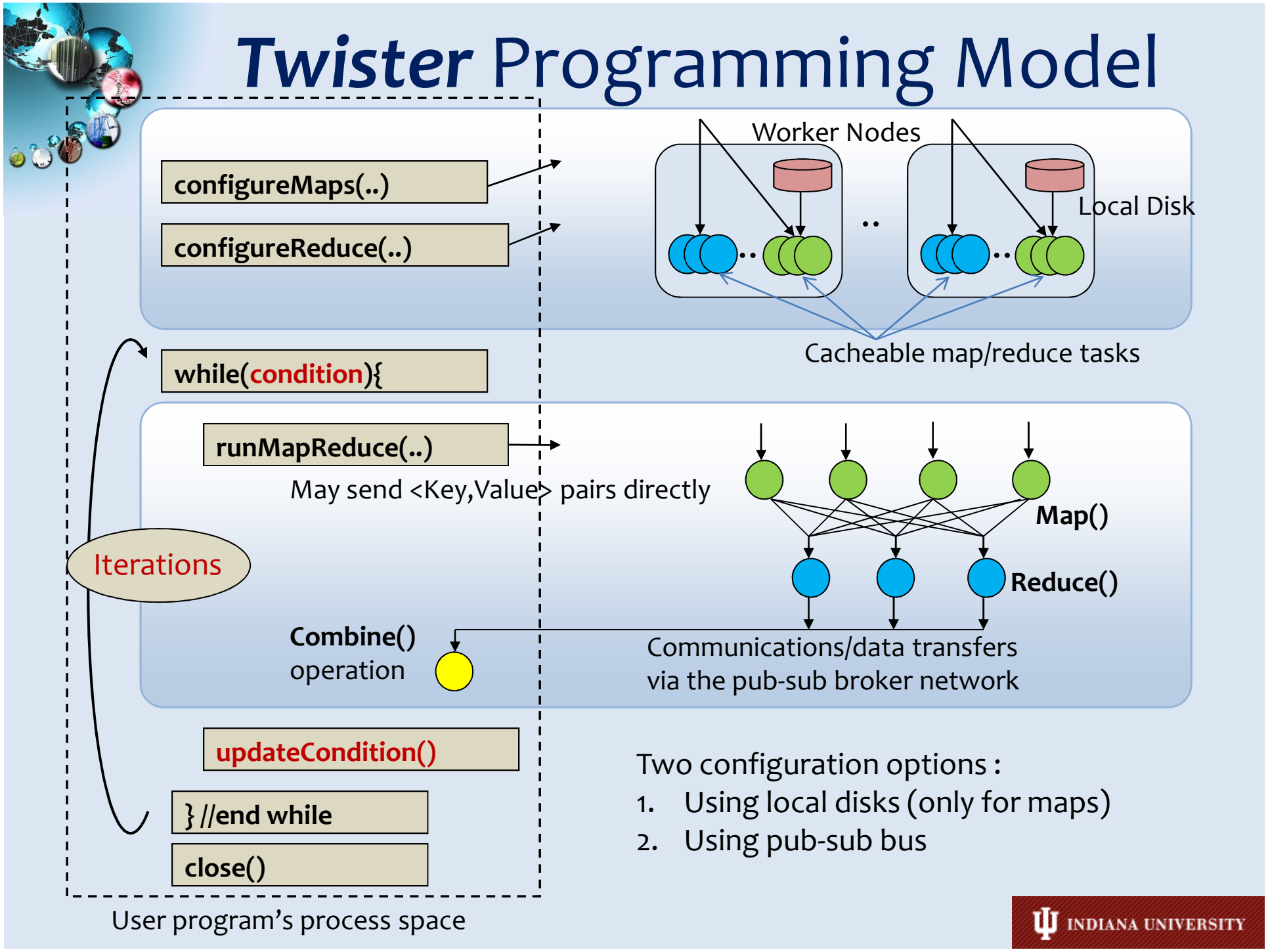
- Focuses mainly on single step map->reduce computations
- Considerable overheads from:
  - Reinitializing tasks
  - Reloading static data
  - Communication & data transfers

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

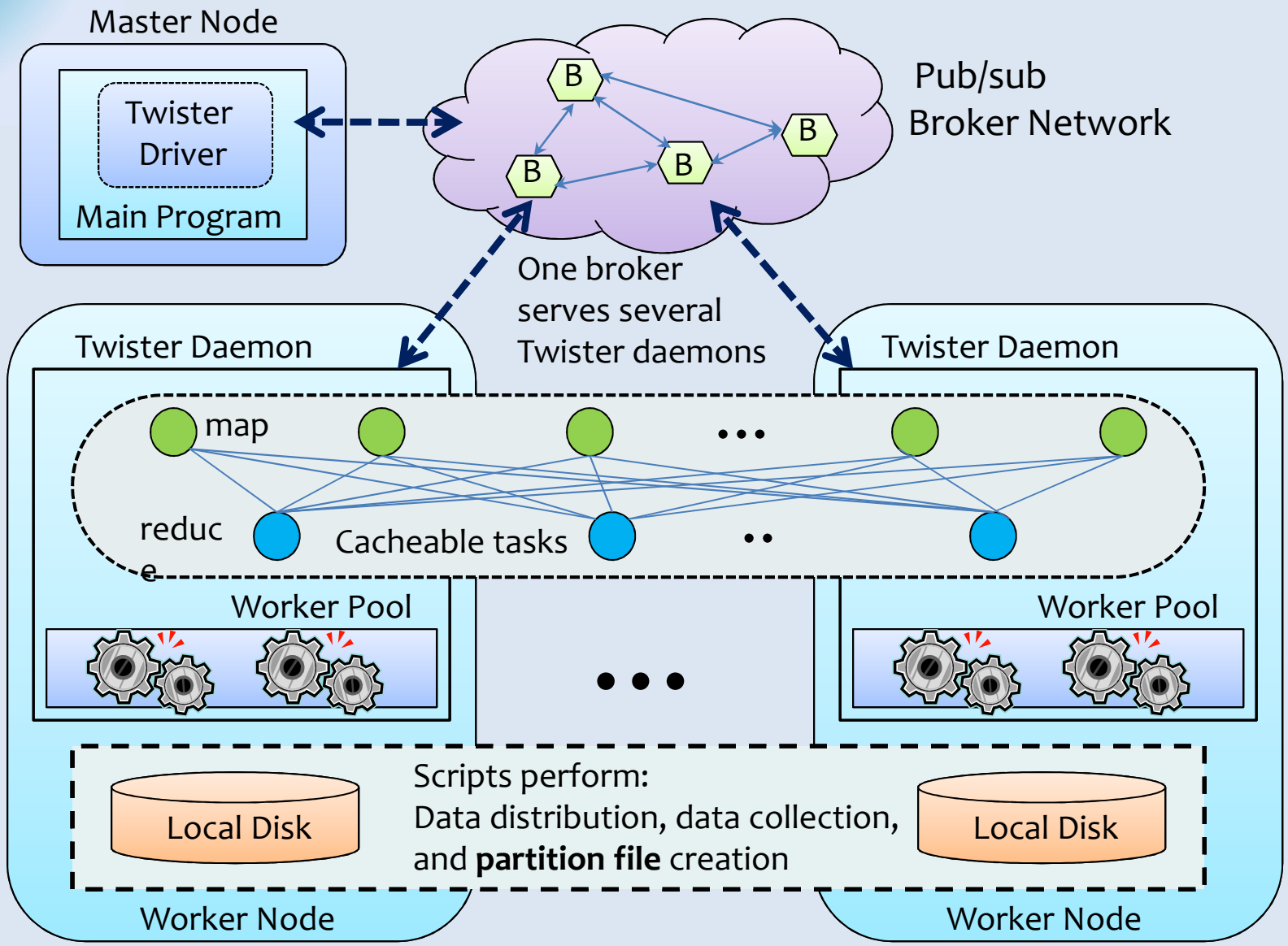
Twister Programming Model



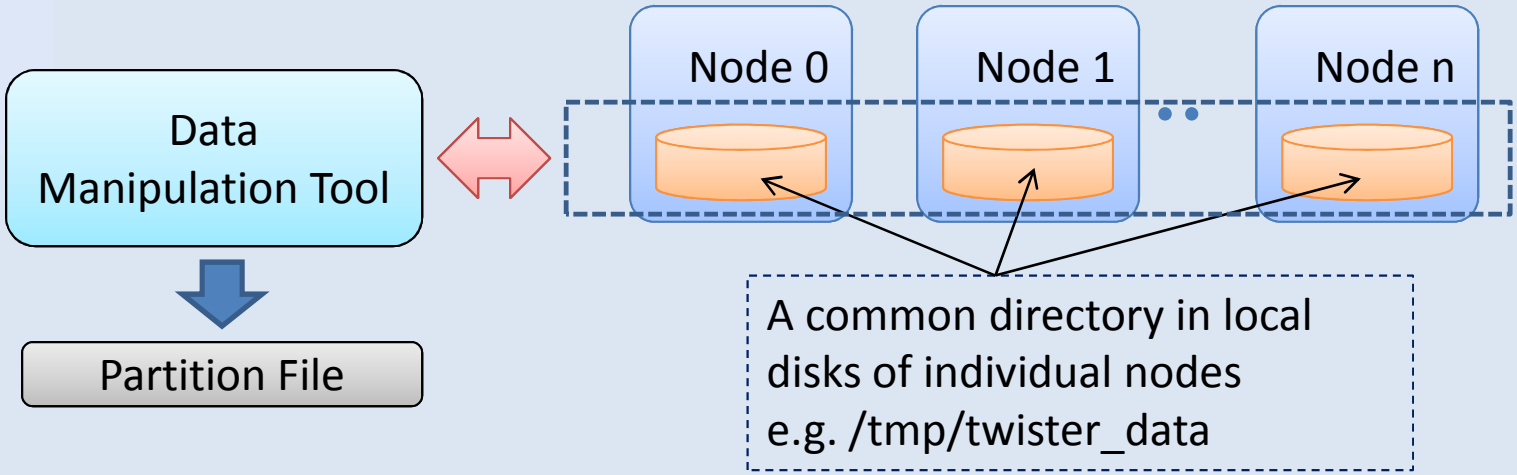
Two configuration options :

1. Using local disks (only for maps)
2. Using pub-sub bus

Twister Architecture



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`**

Partition File

File No	Node IP	Daemon No	File partition path
4	156.56.104.96	2	/home/jaliya/data/mds/GD-4D-23.bin
5	156.56.104.96	2	/home/jaliya/data/mds/GD-4D-0.bin
6	156.56.104.96	2	/home/jaliya/data/mds/GD-4D-27.bin
7	156.56.104.96	2	/home/jaliya/data/mds/GD-4D-20.bin
8	156.56.104.97	4	/home/jaliya/data/mds/GD-4D-23.bin
9	156.56.104.97	4	/home/jaliya/data/mds/GD-4D-25.bin
10	156.56.104.97	4	/home/jaliya/data/mds/GD-4D-18.bin
11	156.56.104.97	4	/home/jaliya/data/mds/GD-4D-15.bin

- Partition file allows duplicates
- One data partition may reside in multiple nodes
- In an event of failure, the duplicates are used to re-schedule the tasks

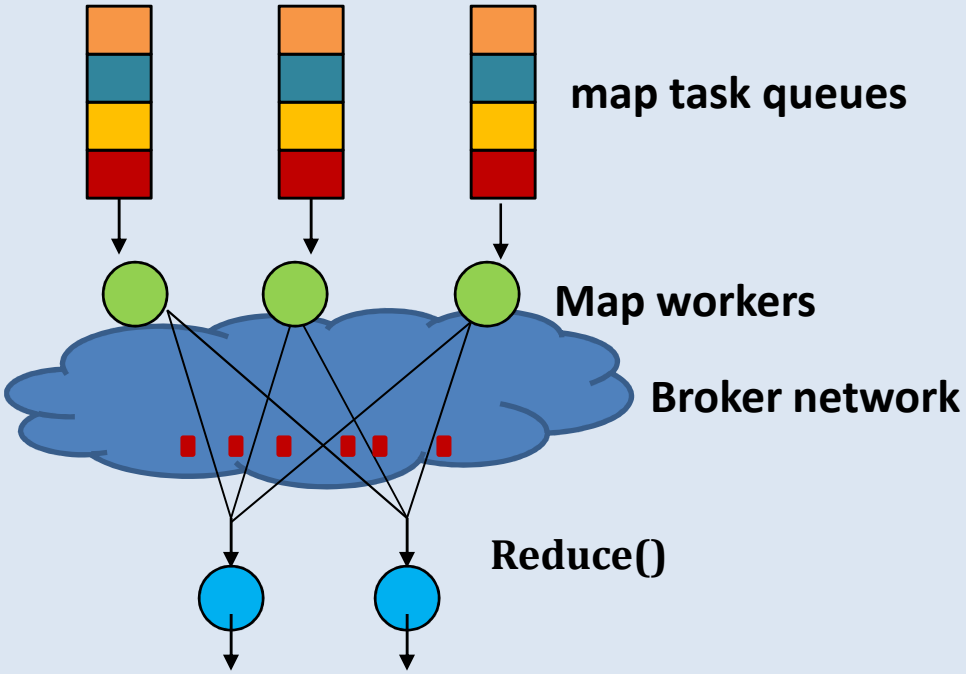
The use of pub/sub messaging

- 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

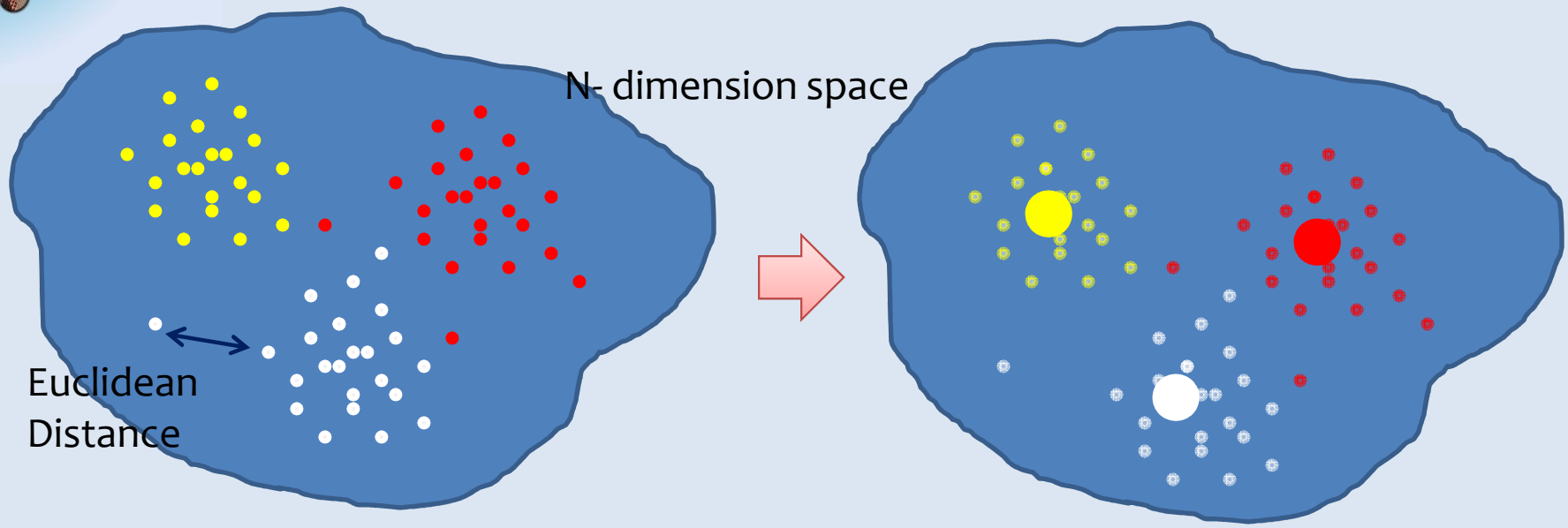
```
1.configureMaps(PartitionFile partitionFile)
2.configureMaps(Value[] values)
3.configureReduce(Value[] values)
4.runMapReduce()
5.runMapReduce(KeyValue[] keyValues)
6.runMapReduceBCast(Value value)
7.map(MapOutputCollector collector, Key key, Value val)
8.reduce(ReduceOutputCollector collector, Key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

Twister Tutorial

- Complete Tutorial
  - http://salsahpc.indiana.edu/tutorial/twister-intro.html

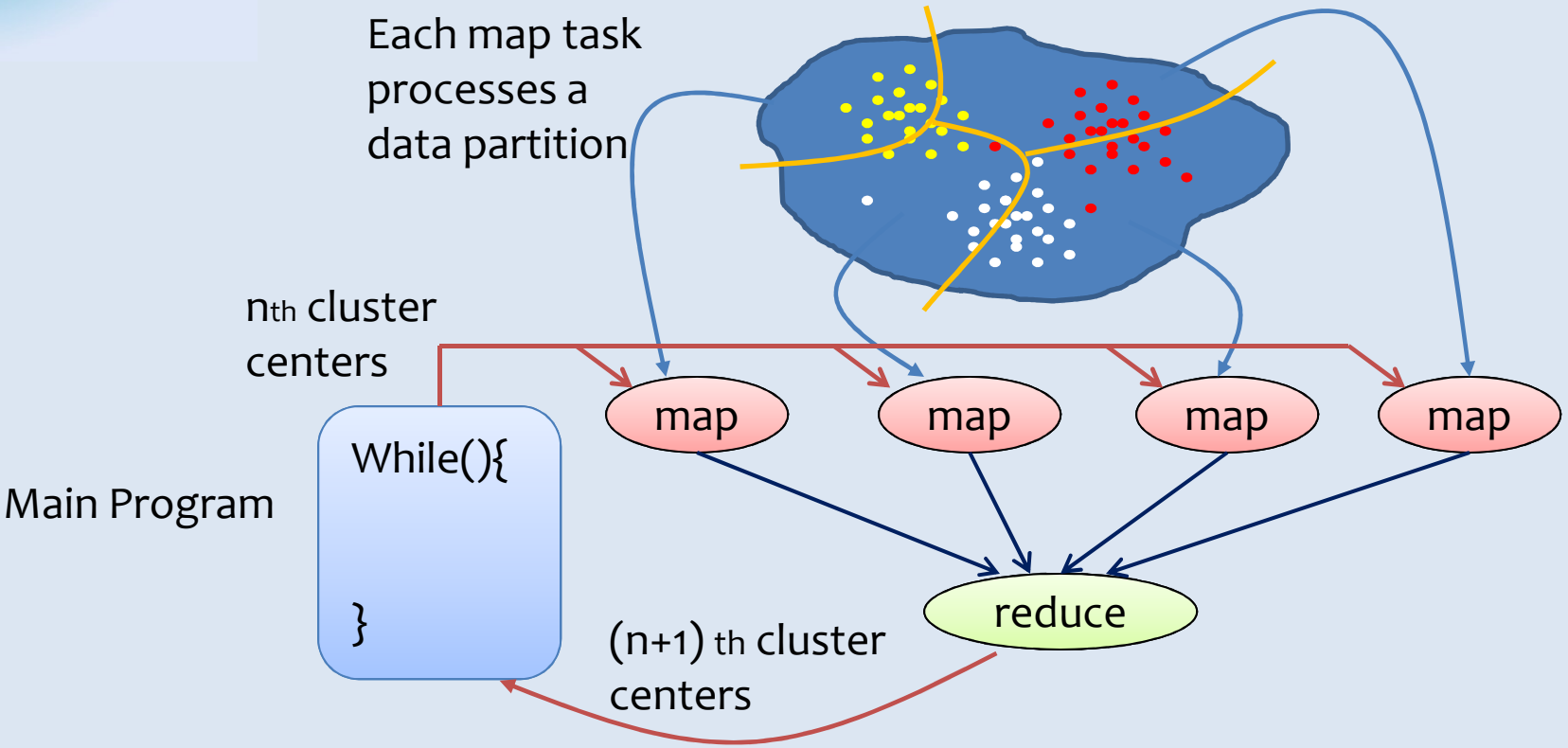
Questions?

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

K-Means Clustering - MapReduce



- Map tasks calculates 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

Code Walkthrough – Main Program

```
144         // Job Configurations
145         JobConf jobConf = new JobConf("kmeans-map-reduce"+ uuidGen.generateTimeBasedUUID());
146         jobConf.setMapperClass(KMeansMapTask.class);
147         jobConf.setReducerClass(KMeansReduceTask.class);
148         jobConf.setCombinerClass(KMeansCombiner.class);
149         jobConf.setNumMapTasks(numMapTasks);
150         jobConf.setNumReduceTasks(numReducers);
151
152         TwisterDriver driver = new TwisterDriver(jobConf);
153         driver.configureMaps(partitionFile);
```

```
170         //Main iteration for K-Means clustering
171         boolean complete = false;
172         while (!complete) {
173             monitor = driver.runMapReduceBCast(cData);
174             monitor.monitorTillCompletion();
175             DoubleVectorData newCData = ((KMeansCombiner) driver.getCurrentCombiner()).getResults();
176             totalError = getError(cData, newCData);
177             cData = newCData;
178             if (totalError < THRESHOLD) {
179                 complete = true;
180                 break;
181             }
182             loopCount++;
183         }
```

Code Walkthrough – map/reduce

```
83    /**
84     * Loads the vector data from a file. Since the map tasks are cached
85     * across iterations, we only need to load this data  once for all
86     * the iterations.
87     */
88    public void configure(JobConf jobConf, MapperConf mapConf)  throws TwisterException {
89        this.vectorData = new DoubleVectorData();
90        fileData = (FileData) mapConf.getDataPartition();
91        try {
92            vectorData.loadDataFromTextFile(fileData.getFileName());
93        } catch (Exception e) {
94            throw new TwisterException(e);
95        }
96    }
```

```
106   /**
107    * Map function for the K-means clustering. Calculates the Euclidean
108    * distance between  data points and the given cluster centers. Next it
109    * calculates the partial cluster centers as well.
110    */
111   public void map(MapOutputCollector collector, Key key, Value val)throws TwisterException {
112
113       double[][] data = vectorData.getData();
114       DoubleVectorData cData = new DoubleVectorData();
115
```

```
147           /**
148            * additional location carries the number of partial points to a
149            * particular centroid.
150            */
151           DoubleVectorData newCData = new DoubleVectorData(newCentroids,numCentroids, vecLen + 1);
152           collector.collect(new StringKey(outKey),new BytesValue(newCData.getBytes()));
```

Code Walkthrough – map/reduce

```
84    public void reduce(ReduceOutputCollector collector, Key key,
85            List<Value> values) throws TwisterException {
86
```

```
139             DoubleVectorData newCentroidData =
140                 new DoubleVectorData(newCentroids, numData, lenData);
141             collector.collect(key, new BytesValue(newCentroidData.getBytes()));
142
```

```
85    /***
86     * Combines the reduce outputs to a single value.
87     */
88    public void combine(Map<Key, Value> keyValues) throws TwisterException {
89        assert (keyValues.size() == 1);// There should be a single value here.
90        Iterator<Key> ite = keyValues.keySet().iterator();
91        Key key = ite.next();
92        BytesValue val = (BytesValue) keyValues.get(key);
93        try {
94            this.results.fromBytes(val.getBytes());
95        } catch (SerializationException e) {
96            throw new TwisterException(e);
97        }
98    }
99
```

Login into Futuregrid Accounts

1. ssh trainXXX@bigdata.[india, sierra].futuregrid.org
2. [train200@s1 ~]$ **qsub –I**

```
[train199@s1 ~]$ qsub -I
qsub: waiting for job 291814.s82 to start
qsub: job 291814.s82 ready

[train199@s10 ~]$
```

3. Create **3** command line windows (shells)
   - ssh trainXXX@bigdata.[india, sierra].futuregrid.org
   - ssh sxx

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

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

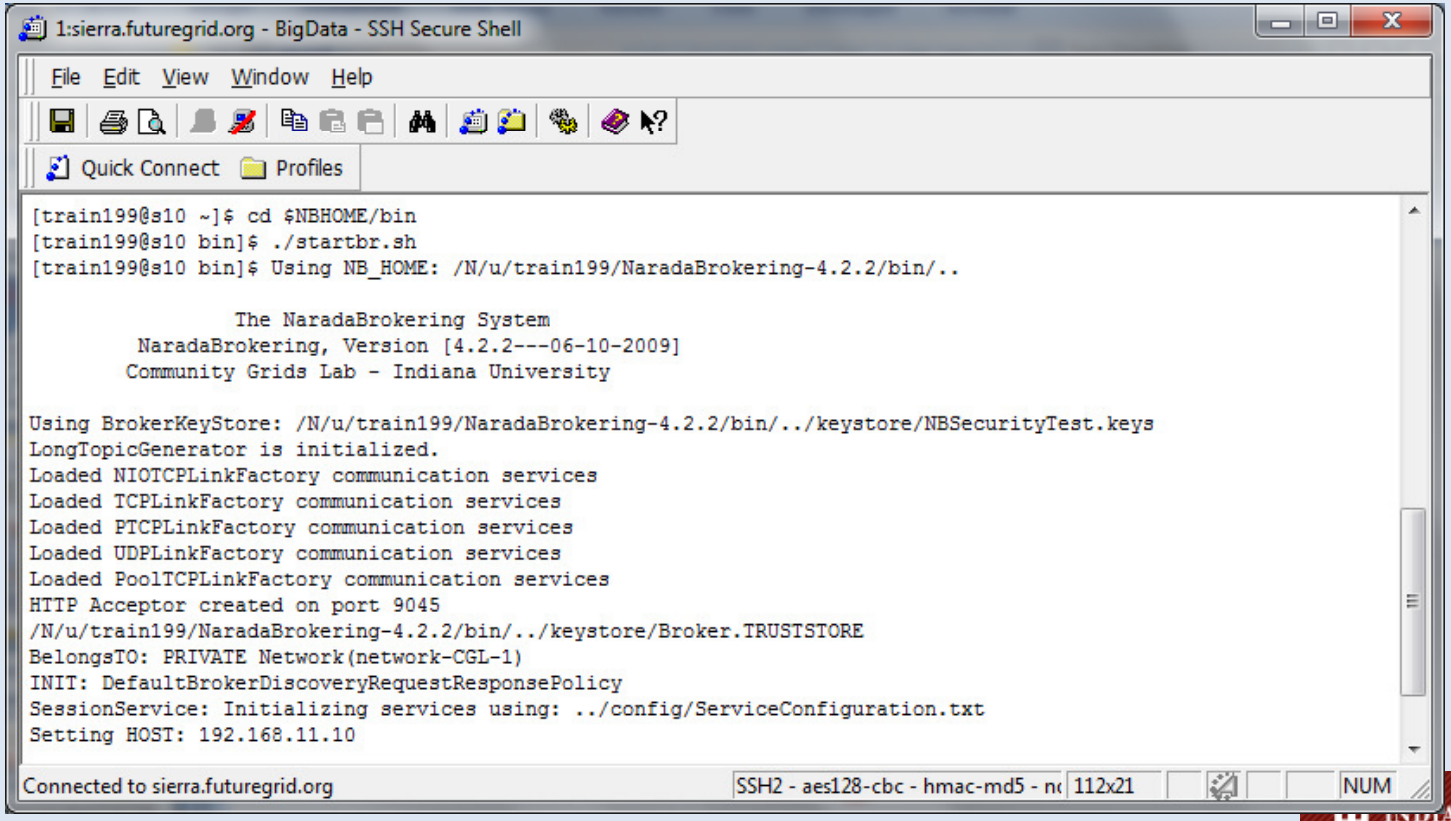
```
[train199@s10 ~]$
```

Start NaradaBrokering

In the first command windows (shell)

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

```
[train199@s10 ~]$ cd $NBHOME/bin
[train199@s10 bin]$ ./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 -I

```
[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

```
[train199@s10 bin]$ cd $TWISTER_HOME/bin
[train199@s10 bin]$ ./twister.sh mkdir kmeans
127.0.0.1:/N/u/train199/Twister/data/kmeans created.
[train199@s10 bin]$
```

Run K-Means Clustering (2)

In the third command window (shell)

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

```
[train001@s10 bin]$ ./twister.sh put $TWISTER_HOME/samples/kmeans/input kmeans
INPUT :Copying /N/u/train001/Twister/samples/kmeans/input/kmeans_1.txt to  127.0.0.1:/N/u/train001/Twister/data/kmeans
INPUT :Copying /N/u/train001/Twister/samples/kmeans/input/kmeans_0.txt to  127.0.0.1:/N/u/train001/Twister/data/kmeans
INPUT :Copying /N/u/train001/Twister/samples/kmeans/input/kmeans_4.txt to  127.0.0.1:/N/u/train001/Twister/data/kmeans
INPUT :Copying /N/u/train001/Twister/samples/kmeans/input/kmeans_2.txt to  127.0.0.1:/N/u/train001/Twister/data/kmeans
```

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

```
[train001@s10 bin]$ ./create_partition_file.sh kmeans kmeans_ $TWISTER_HOME/samples/kmeans/bin/kmeans.pf
TransmissionManager: All transfers have been completed
CommunicationsService: Closing Link => niotcp://127.0.0.1:3045
NIOTCPReceiverThread: Proceeding to close channel
NIOTCPReceiverThread: Proceeding to close channel
Partition file created.
```

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

The Output

```
461790 [pool-1-thread-4] DEBUG cgl.imr.worker.Reducer  - Reduce Task :0 terminating.
Loop 15 Completes.
112.45252105688243 , 379.0583823699088 , 111.17722395292489 ,
255.42047339062265 , 326.65264586160106 , 248.25282677521483 ,
122.278814846972 , 120.21586715867159 , 116.79227262860864 ,
372.9488701447464 , 121.16713527022806 , 117.14016453191711 ,
422.60639518210496 , 126.96544307427588 , 365.22497849154 ,
250.8247467438495 , 124.61722141823444 , 385.02561505065125 ,
114.42114738358137 , 383.84841574651944 , 386.2538406144983 ,
78.78692940175787 , 128.5942727530479 , 365.55670541536716 ,
386.9485784919654 , 385.76254635352285 , 390.559456118665 ,
390.4511322304353 , 377.7820016426141 , 111.73483515194181 ,
Total Time for kemeans : 3.686
Total loop count : 16
TransmissionManager: All transfers have been completed
CommunicationsService: Closing Link => niotcp://127.0.0.1:3045
NIOTCPReceiverThread: Proceeding to close channel
NIOTCPReceiverThread: Proceeding to close channel
0    [main] INFO  cgl.imr.client.TwisterDriver  - MapReduce computation termintated gracefully.
-------------------------------------------------------
Kmeans clustering took 3.838 seconds.
-------------------------------------------------------
2    [Thread-1] DEBUG cgl.imr.client.ShutdownHook  - Shutting down completed.
```

Once you are done please close Twister and then Naradabrokering

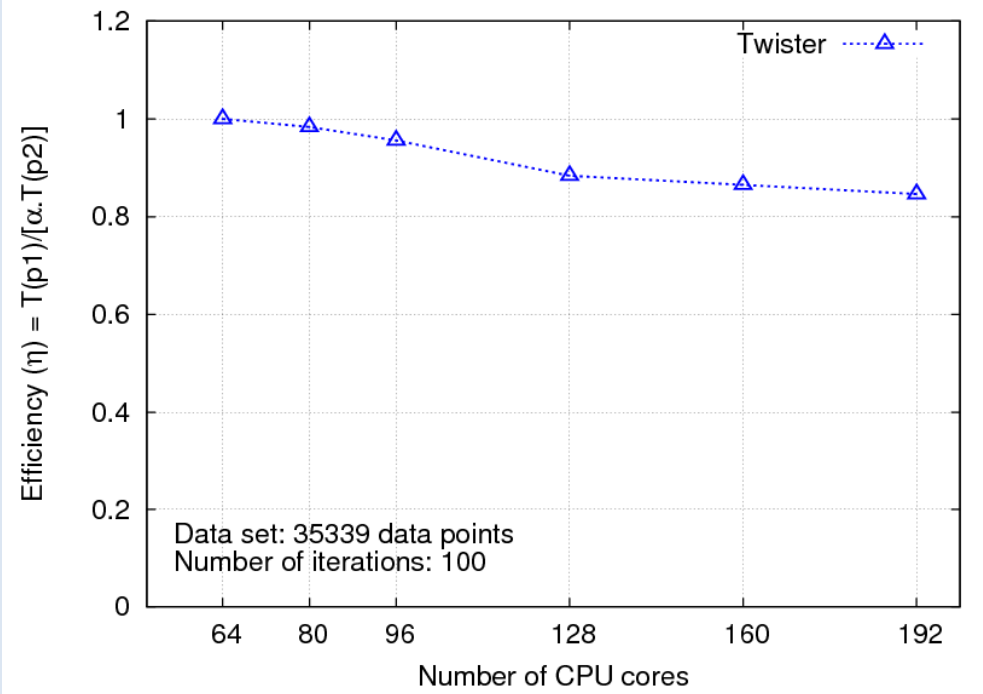
cd $TWISTER_HOME/bin
./stop_twister.sh

cd $NBHOME/bin
./stopbr.sh

Demo: Multi-dimensional Scaling

```
While(condition)
{
 <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]

[1] J. de Leeuw, "Applications of convex analysis to multidimensional scaling," *Recent Developments in Statistics, pp. 133-145, 1977.*

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…

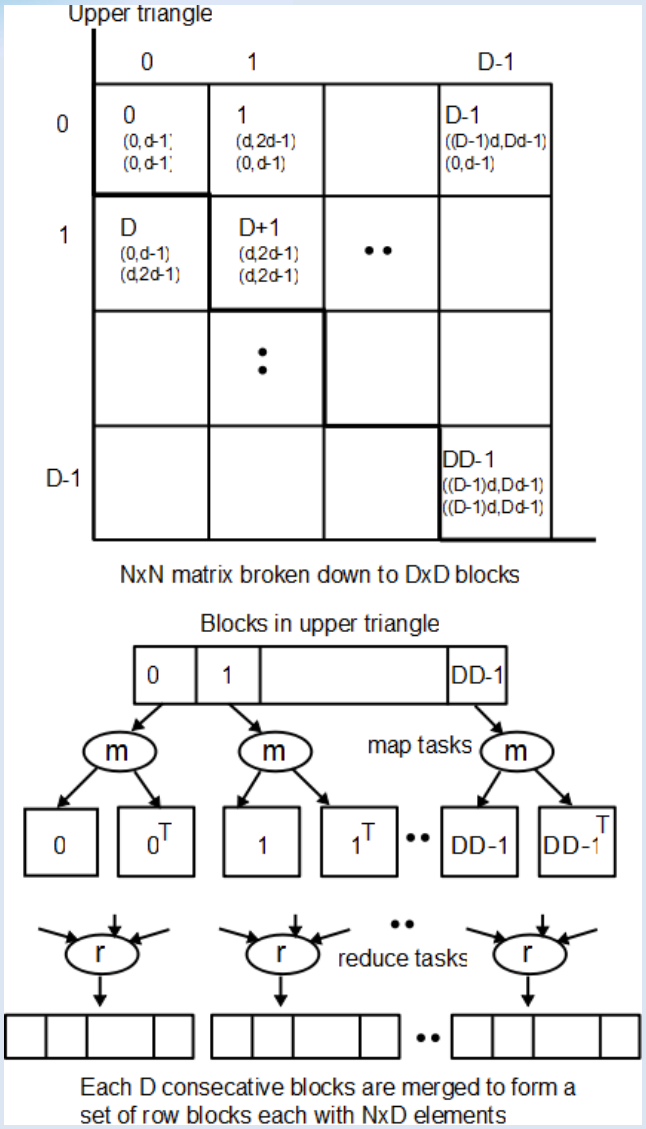
Performance Evaluation

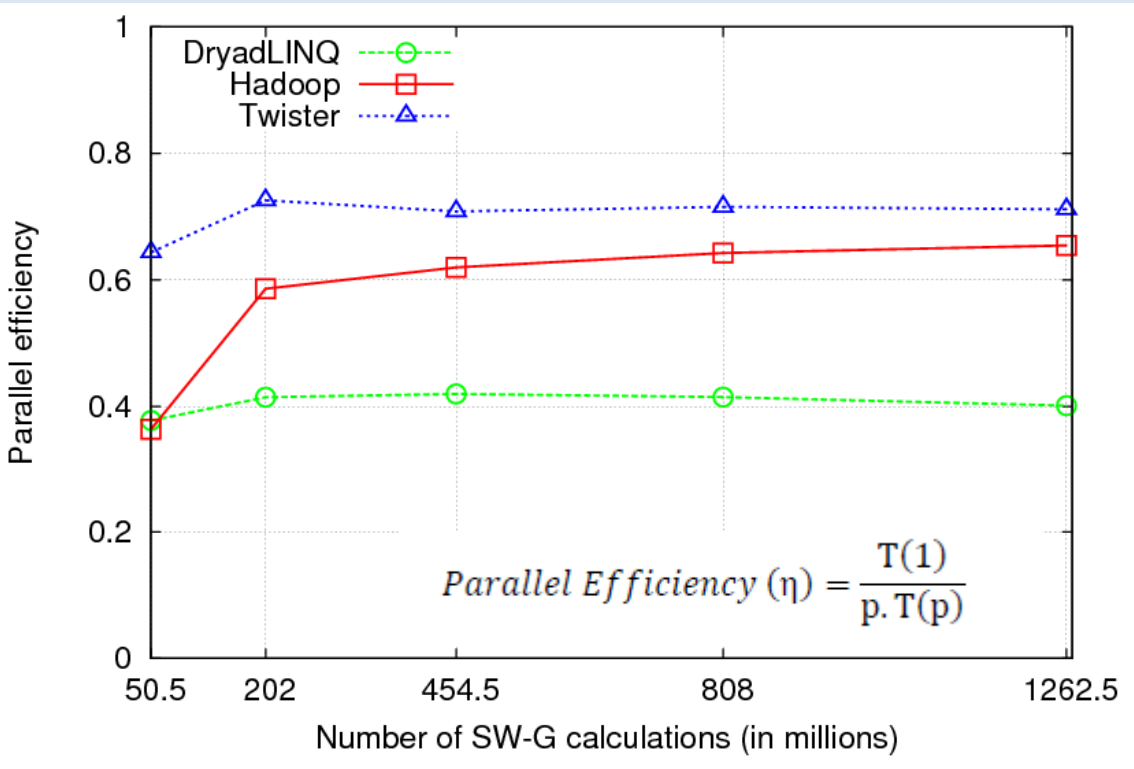
- **Hardware Configurations**

Cluster ID	Cluster-I	Cluster-II
# nodes	**32**	**230**
# CPUs in each node	**6**	**2**
# Cores in each CPU	**8**	**4**
Total CPU cores	**768**	**1840**
Supported OSs	Linux (Red Hat Enterprise Linux Server release 5.4 -64 bit) Windows (Windows Server 2008 - 64 bit)	Red Hat Enterprise Linux Server release 5.4 -64 bit

- 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

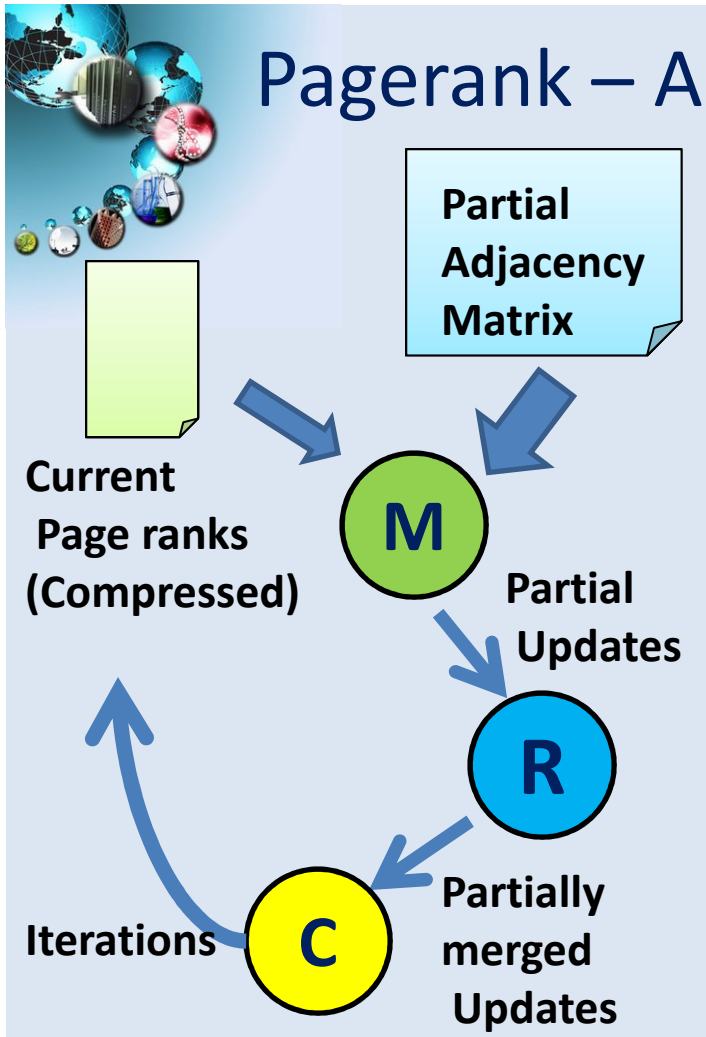


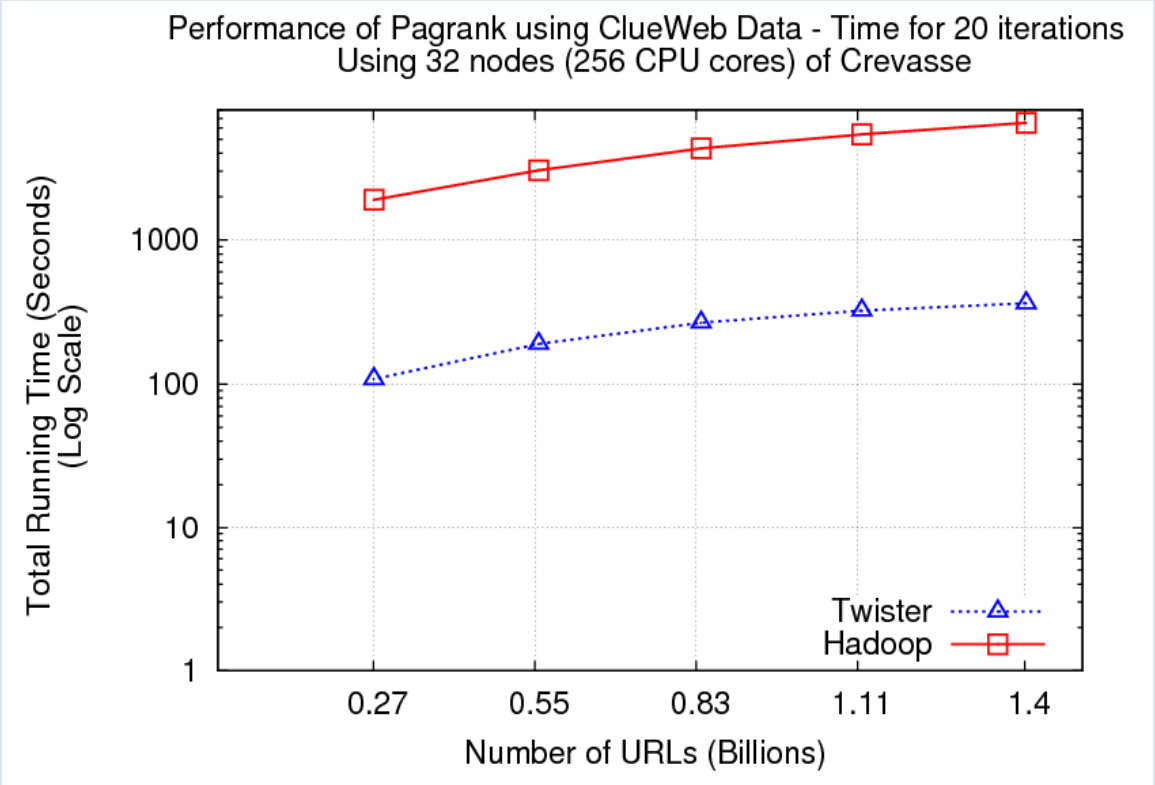


- Typical MapReduce computation
- Comparable efficiencies
- Twister performs the best

INDIANA UNIVERSITY

Pagerank – An Iterative MapReduce Algorithm





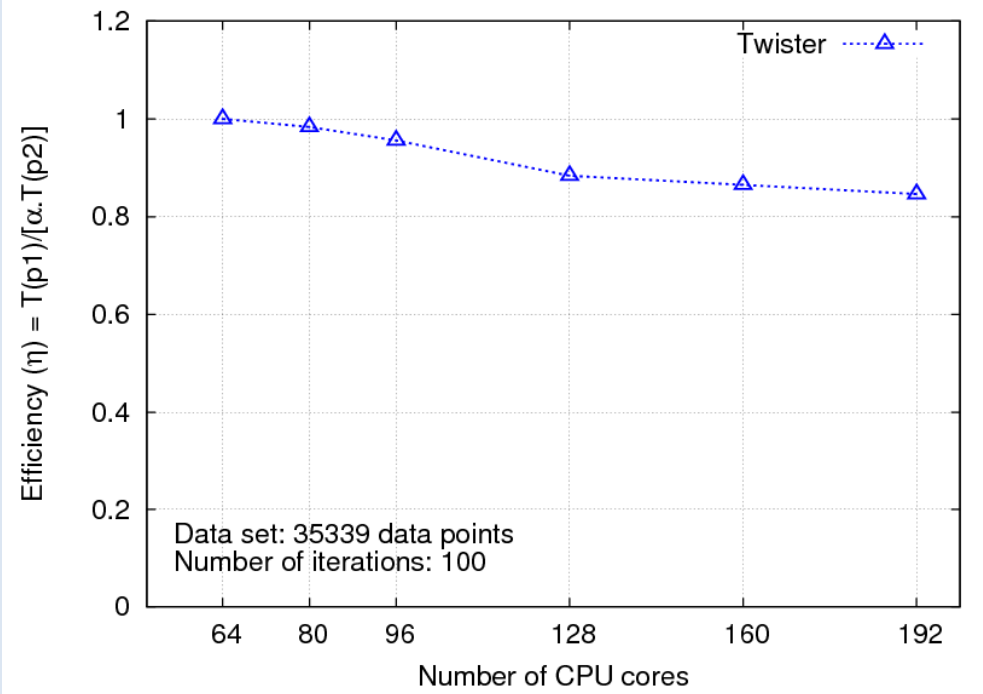
- Well-known pagerank algorithm [1]
- Used ClueWeb09 [2] (1TB in size) from CMU
- Reuse of map tasks and faster communication pays off

[1] Pagerank Algorithm, http://en.wikipedia.org/wiki/PageRank
[2] ClueWeb09 Data Set, http://boston.lti.cs.cmu.edu/Data/clueweb09/

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]

[1] J. de Leeuw, "Applications of convex analysis to multidimensional scaling," *Recent Developments in Statistics, pp. 133-145, 1977.*

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