Mrs: MapReduce for Scientific Computing in Python

Andrew McNabb, **Jeff Lund**, and Kevin Seppi

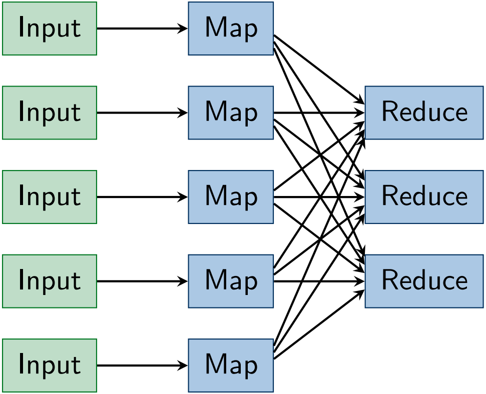
Brigham Young University

November 16, 2012

MapReduce

- Large scale problems require parallel processing
- Communication in parallel processing is hard
- MapReduce abstracts away interprocess communication
- User only has to identify which parts of the problem are embarrassingly parallel

MapReduce

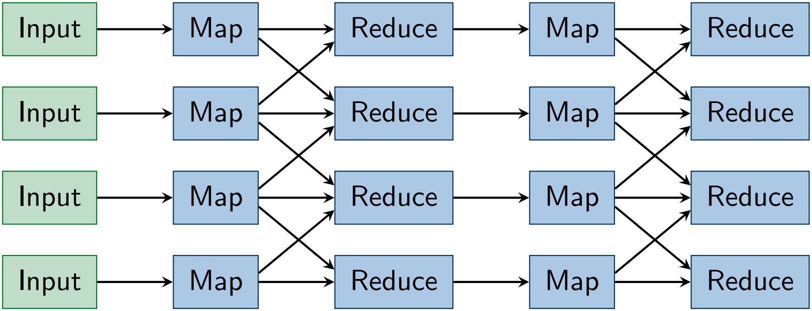

Input
Input
Input
Input
Input
Map
Map
Map
Map
Map
Reduce
Reduce
Reduce

WordCount

wordcount.py

```
import mrs

class WordCount(mrs.MapReduce):

    def map(self, line_num, line_text):
        for word in line_text.split():
            yield (word, 1)

    def reduce(self, word, counts):
        yield sum(counts)

if __name__ == '__main__':
    mrs.main(WordCount)
```

Iterative MapReduce



Hadoop

- Hadoop is the most widely used open source MapReduce implementation
- Hadoop was designed for big data, not scientific computing
- Requires the use of HDFS and a dedicated cluster

MapReduce in Scientific Computing

What does an ideal MapReduce implementation look like in the context of scientific computing?

Ease of Development

- Rapid prototyping
- Testability
- Debuggability

Ease of Development

WordCount.java

```
public class WordCount {
 public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
     ) throws IOException, InterruptedException {
   StringTokenizer itr =
      new StringTokenizer(value.toString());
   while (itr.hasMoreTokens()) {
    word.set(itr.nextToken());
    context.write(word, one);
   }
  }
 }
 public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key,
     Iterable<IntWritable> values, Context context
     ) throws IOException, InterruptedException {
   int sum = 0;
   for (IntWritable val : values) {
    sum += val.get();
   }
   result.set(sum);
   context.write(key, result);
  }
 }
 public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  String[] otherArgs = new
     GenericOptionsParser(conf, args).getRemainingArgs();
  if (otherArgs.length != 2) {
   System.err.println("Usage: wordcount <in> <out>");
   System.exit(2);
  }
  Job job = new Job(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job,
     new Path(otherArgs[0]));
  FileOutputFormat.setOutputPath(job,
     new Path(otherArgs[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

Ease of Deployment

- Dedicated cluster vs. supercomputers and private cluster
- Work with any filesystem
- Work with any scheduler

Ease of Deployment

pbs-hadoop.sh

```
# Step 1: Find the network address.
ADDR=$(/sbin/ip -o -4 addr list "$INTERFACE"
    |sed -e 's;^.*inet \(.*\)/.*$;\1;')

# Step 2: Set up the Hadoop configuration.
export HADOOP_LOG_DIR=$JOBDIR/log
mkdir $HADOOP_LOG_DIR
export HADOOP_CONF_DIR=$JOBDIR/conf
cp -R $HADOOP_HOME/conf $HADOOP_CONF_DIR
sed -e "s/MASTER_IP_ADDRESS/$ADDR/g"
    -e "s@HADOOP_TMP_DIR@$JOBDIR/tmp@g" \
    -e "s/MAP_TASKS/$MAP_TASKS/g" \
    -e "s/REDUCE_TASKS/$REDUCE_TASKS/g" \
    -e "s/TASKS_PER_NODE/$TASKS_PER_NODE/g" \
    <$HADOOP_HOME/conf/hadoop-site.xml \
    >$HADOOP_CONF_DIR/hadoop-site.xml

# Step 3: Start daemons on the master.
HADOOP="$HADOOP_HOME/bin/hadoop"
$HADOOP namenode -format # format the hdfs
$HADOOP_HOME/bin/hadoop-daemon.sh start namenode
$HADOOP_HOME/bin/hadoop-daemon.sh start jobtracker

# Step 4: Start daemons on the slaves.
ENV=". $HOME/.bashrc;
    export HADOOP_CONF_DIR=$HADOOP_CONF_DIR;
    export HADOOP_LOG_DIR=$HADOOP_LOG_DIR"
pbsdsh -u bash -c "$ENV; $HADOOP datanode" &
pbsdsh -u bash -c "$ENV; $HADOOP tasktracker" &
sleep 15

# Step 5: Run the User Program
$HADOOP dfs -put $INPUT $HDFS_INPUT
$HADOOP jar $PROGRAM ${ARGS[@]}
$HADOOP dfs -get $HDFS_OUTPUT $OUTPUT

# Step 6: Stop daemons on the slaves and master.
kill %2 # kill tasktracker
kill %1 # kill datanode
$HADOOP_HOME/bin/hadoop-daemon.sh stop jobtracker
$HADOOP_HOME/bin/hadoop-daemon.sh stop namenode
```

Other Issues

- Iterative performance
- Fault tolerance
- Interoperability

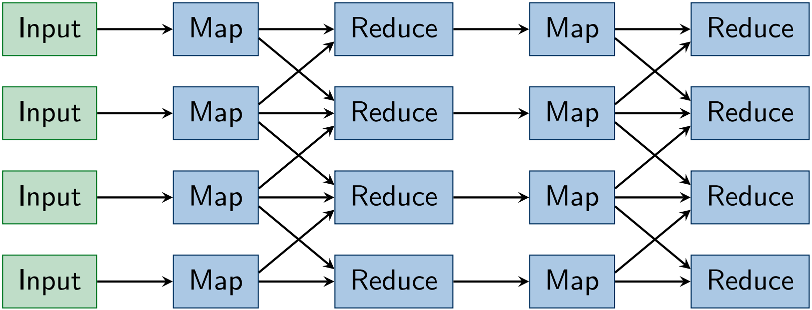
What is Mrs?

- Aims to be a simple to use MapReduce framework
- Implemented in pure Python
- Designed with scientific computing in mind

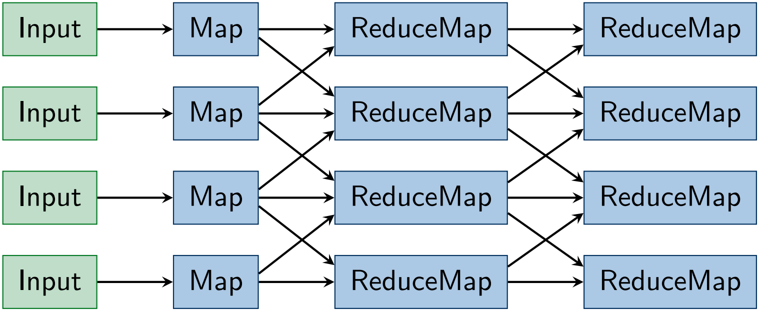
Why Python?

- Python is nearly ubiquitous
- Mrs needs no dependencies outside of standard library
- Familiarity and readability
- Easy interoperability
- Debugging and testing
- One downside: GIL

Iterative MapReduce



Iterative MapReduce: ReduceMap



Automatic Serialization

- Serialization happens every time a tasks communicates with another machine
- Mrs automatically handles this with pickle
- Hadoop requires Writable classes everywhere

Debugging: Run Modes

- Serial
- Mock Parallel
- Parallel

Debugging: Random Number Generators

- Seeding random number generators makes results reproducible
- Need different seed for each task
- Mrs has random function which lets you create a random number generator with an arbitrary number of offset parameters
- ex. rand = self.random(id, iter)

Performance and Case Studies

Interpreter overhead does not preclude good performance for Mrs.

We demonstrate on three different problems:

- Halton Sequence:
  CPU bound benchmark
- Particle Swarm Optimization:
  CPU bound application
- Walk Analysis:
  IO bound application

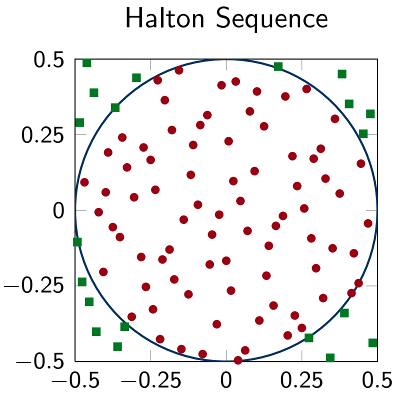
Performance and Case Studies

Optimization Story:

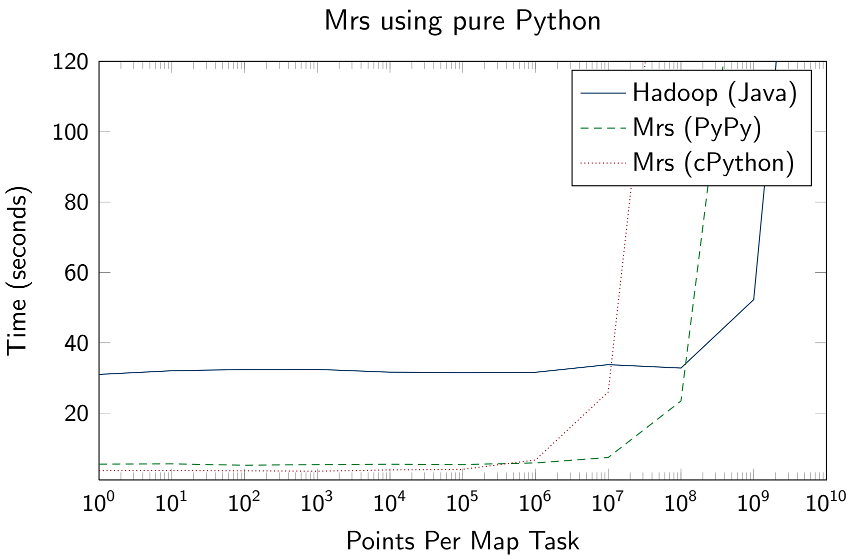
- Make sure you have the right algorithm
- Careful profiling
- Run with PyPy
- Rewrite critical path in C

Monte Carlo Pi Estimation

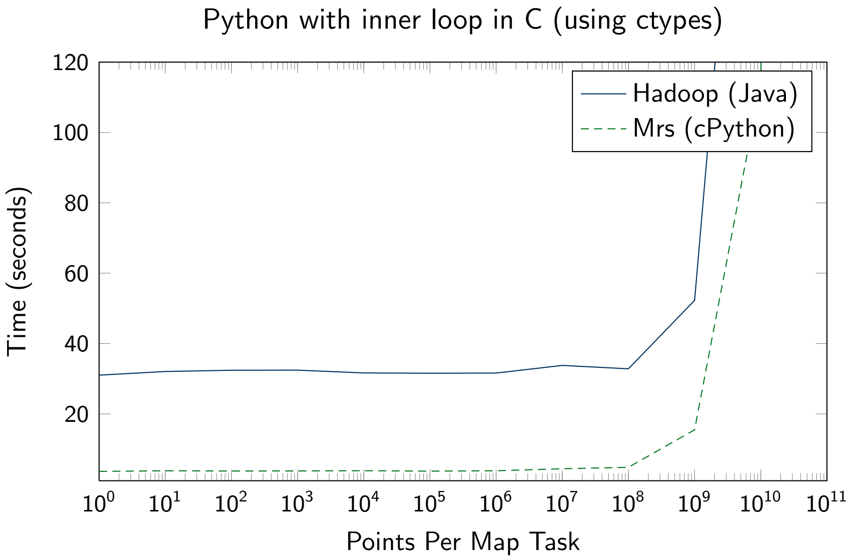
- Monte Carlo algorithm for computing the value of π by generating random points in a square
- Very little data, but computationally intense
- We can control how much computation each map task performs



Monte Carlo Pi Estimation

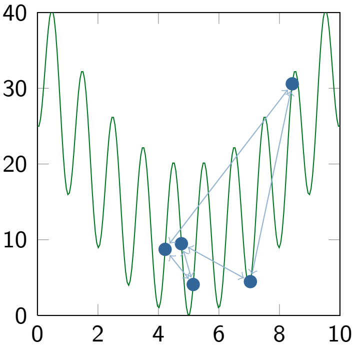


Monte Carlo Pi Estimation

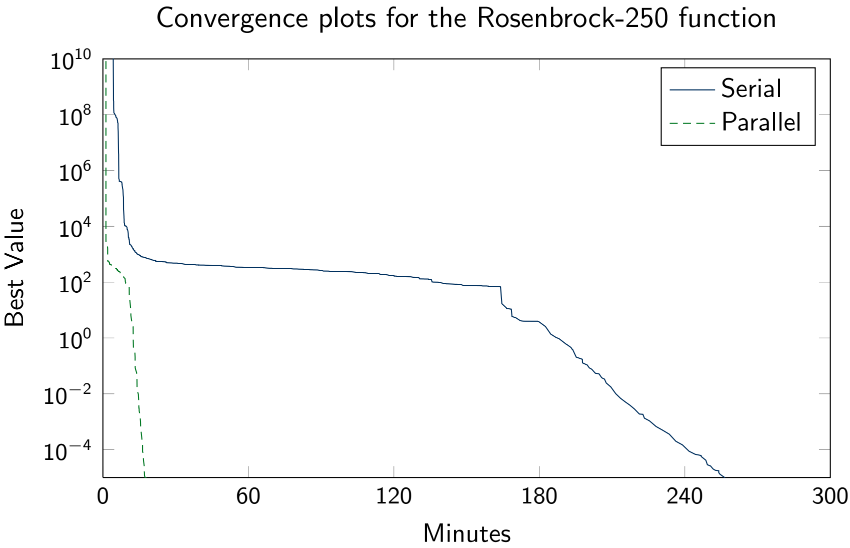


Particle Swarm Optimization

- Inspired by simulations of flocking birds
- Particles interact while exploring
- Map: motion and function evaluation
- Reduce: communication
- CPU bound problem



Particle Swarm Optimization



Walk Analyzer

- Involves analyzing random walks in a graph
- Heavy IO bound
- Average Hadoop Time: 1:06:53
- Average Mrs Time: 52:55

Where to find Mrs

Mrs Homepage with links to source, documentation, mailing list, etc:

`http://code.google.com/p/mrs-mapreduce`

In case you forget the url, just google "mrs mapreduce" :)

Other cool features I neglected to mention...

- Reduce merge sort
- Asyncronous MapReduce
- Concurrent Convergence Checks
- Memory Logging
- Merge Sort Reduce Dataset
- Custom Serializer