# The HDInsight Pi Estimator Sample

This sample topic shows how to run a simple MapReduce program that estimates the value of the mathematical constant Pi on the Windows Azure HDinsight service with Azure Powershell.

The program uses a statistical (quasi-Monte Carlo) method to estimate the value of Pi. Points placed at random inside of a unit square also fall within a circle inscribed within that square with a probability equal to the area of the circle, Pi/4. The value of Pi can be estimated from the value of 4R where R is the ratio of the number of points that are inside the circle to the total number of points that are within the square. The larger the sample of points used, the better the estimate is.

The PiEstimator Java code that contains the mapper and reducer functions is available for inspection below. The mapper program generates a specified number of points placed at random inside of a unit square and then counts the number of those points that are inside the circle. The reducer program accumulates points counted by the mappers and then estimates the value of Pi from the formula 4R, where R is the ratio of the number of points counted inside the circle to the total number of points that are within the square.

The script provided for this sample submits a Hadoop JAR job and is set up to run with a value 16 maps, each of which is required to compute 10 million sample points by the parameter values. These parameter values can be changed to improve the estimated value of Pi. For reference, the first 10 decimal places of Pi are 3.1415926535.

The .jar file that contains the files needed by Hadoop on Azure to deploy the application is a .zip file and is available for download. You can unzip it with various compression utilities then explore the files at your convenience.

The other samples that are available to help you get up to speed using the Windows Azure HDInsight Service to run MapReduce jobs are listed on [Running the HDInsight Samples](file:///C:\en-us\manage\services\hdinsight\get-started-hdinsight\run-samples) along with links to instructions on how to run them.

**You will learn:**

* How to use Windows Azure PowerShell to run the Pi Estimator MapReduce program on Windows Azure HDInsight.
* What a MapReduce program written in Java looks like.

**Prerequisites**:

* You must have a Windows Azure Account. For options on signing up for an account see [Try Windows Azure out for free](http://www.windowsazure.com/en-us/pricing/free-trial/) page.
* You must have provisioned an HDInsight cluster. For instructions on the various ways in which such clusters can be created, see [Provision HDInsight Clusters](file:///C:\en-us\manage\services\hdinsight\provision-hdinsight-clusters\)
* You must have installed Windows Azure PowerShell and the HDInsight Powershell Tools, and have configured them for use with your account. For instructions on how to do this, see [Install and configure PowerShell for HDInsight](file:///C:\en-us\manage\services\hdinsight\configure-powershell-for-hdinsight\)

**Outline**  
This topic shows you how to run the sample, presents the Java code for the Pi Estimator MapReduce program, summarizes what you have learned, and outlines some next steps. It has the following sections.

1. [Run the Sample with Windows Azure PowerShell](#run-sample)
2. [The Java Code for the Pi Estimator MapReduce Program](#java-code)
3. [Summary](#summary)
4. [Next Steps](#next-steps)

## Run the Sample with Windows Azure PowerShell

1. Open Windows Azure PowerShell and Notepad.
2. Copy and paste the following code into Notepad.
3. ### Provide the Windows Azure subscription name and the HDInsight cluster name.
4. $subscriptionName = "myAzureSubscriptionName"
5. $clusterName = "myClusterName"
6. Set the values for the two variable: $subscriptionname, $clustername to the subscripion and cluster you are using. Copy and paste this modified code into PowerShell and then press **Enter** to run.
7. The Pi Estimator MapReduce job is defined using the New-AzureHDInsightMapReduceJobDefinition cmdlet. The two arguments specify, respectively, the number of maps to create and the number of samples generated per map. Copy and paste the following code into PowerShell and press **Enter**.
8. ### Create a MapReduce job definition for the Pi Estimator job
9. ### There two arguments. The first number indicates how many maps to create (default is 16).
10. ### The second number indicates how many samples are generated per map (10 million by default).
11. ### So this program uses 10\*10 million = 160 million random points to make its estimate of Pi.
12. $piEstimatorJobDefinition = New-AzureHDInsightMapReduceJobDefinition -JarFile "wasb:///example/jars/hadoop-examples.jar" -ClassName "pi" –Arguments “16”, “10000000”
13. The Pi Estimator MapReduce job is run by piping the job defintion with the Start-AzureHDInsightJob, which requires the cluster and subscription names. Use the Wait-AzureHDInsightJob to get status on the run. Copy and paste the following code into PowerShell and press **Enter**.
14. ### Run the Pi Estimator MapReduce job.
15. $piJob = $piEstimatorJobDefinition | Start-AzureHDInsightJob -Cluster $clusterName –Subscription $subscriptionName
16. ### Wait for the job to complete.
17. $piJob | Wait-AzureHDInsightJob –Subscription $subscriptionName -WaitTimeoutInSeconds 3600
18. Use the Get-AzureHDInsightJobOutput to display the results in PowerShell. Copy and paste the following code into PowerShell and press **Enter** to see the result.
19. ### Print the standard error file of the MapReduce job
20. Get-AzureHDInsightJobOutput -Cluster $clusterName -Subscription $subscriptionName -JobId $piJob.JobId -StandardError

## The Java Code for the Pi Estimator MapReduce Program

/\*\*  
\* Licensed to the Apache Software Foundation (ASF) under one  
\* or more contributor license agreements. See the NOTICE file  
\* distributed with this work for additional information  
\* regarding copyright ownership. The ASF licenses this file  
\* to you under the Apache License, Version 2.0 (the  
\* "License"); you may not use this file except in compliance  
\* with the License. You may obtain a copy of the License at  
\*  
\* http://www.apache.org/licenses/LICENSE-2.0  
\*  
\* Unless required by applicable law or agreed to in writing, software  
\* distributed under the License is distributed on an "AS IS" BASIS,  
\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. \* See the License for the specific language governing permissions and  
\* limitations under the License.  
\*/

package org.apache.hadoop.examples;

import java.io.IOException; import java.math.BigDecimal; import java.util.Iterator;

import org.apache.hadoop.conf.Configured; import org.apache.hadoop.fs.FileSystem; import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.BooleanWritable; import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.SequenceFile; import org.apache.hadoop.io.Writable; import org.apache.hadoop.io.WritableComparable; import org.apache.hadoop.io.SequenceFile.CompressionType; import org.apache.hadoop.mapred.FileInputFormat; import org.apache.hadoop.mapred.FileOutputFormat; import org.apache.hadoop.mapred.JobClient; import org.apache.hadoop.mapred.JobConf; import org.apache.hadoop.mapred.MapReduceBase; import org.apache.hadoop.mapred.Mapper; import org.apache.hadoop.mapred.OutputCollector; import org.apache.hadoop.mapred.Reducer; import org.apache.hadoop.mapred.Reporter; import org.apache.hadoop.mapred.SequenceFileInputFormat; import org.apache.hadoop.mapred.SequenceFileOutputFormat; import org.apache.hadoop.util.Tool; import org.apache.hadoop.util.ToolRunner;

//A Map-reduce program to estimate the value of Pi  
//using quasi-Monte Carlo method.  
//  
//Mapper:  
//Generate points in a unit square  
//and then count points inside/outside of the inscribed circle of the square.  
//  
//Reducer:  
//Accumulate points inside/outside results from the mappers.  
//Let numTotal = numInside + numOutside.  
//The fraction numInside/numTotal is a rational approximation of  
//the value (Area of the circle)/(Area of the square),  
//where the area of the inscribed circle is Pi/4  
//and the area of unit square is 1. //Then, Pi is estimated value to be 4(numInside/numTotal).  
//

public class PiEstimator extends Configured implements Tool {  
//tmp directory for input/output  
static private final Path TMPDIR = new Path(PiEstimator.class.getSimpleName() + "TMP3141592654");

//2-dimensional Halton sequence {H(i)},  
//where H(i) is a 2-dimensional point and i >= 1 is the index.  
//Halton sequence is used to generate sample points for Pi estimation.  
private static class HaltonSequence {  
// Bases  
static final int[] P = {2, 3};  
//Maximum number of digits allowed  
static final int[] K = {63, 40};

private long index;  
private double[] x;  
private double[][] q;  
private int[][] d;

//Initialize to H(startindex),  
//so the sequence begins with H(startindex+1).  
HaltonSequence(long startindex) {  
index = startindex;  
x = new double[K.length];  
q = new double[K.length][];  
d = new int[K.length][];  
for(int i = 0; i < K.length; i++) {  
q[i] = new double[K[i]];  
d[i] = new int[K[i]];  
}

for(int i = 0; i < K.length; i++) { long k = index; x[i] = 0;

for(int j = 0; j < K[i]; j++) {  
q[i][j] = (j == 0? 1.0: q[i][j-1])/P[i];  
d[i][j] = (int)(k % P[i]); k = (k - d[i][j])/P[i];  
x[i] += d[i][j] \* q[i][j]; }  
} }

//Compute next point.  
//Assume the current point is H(index). //Compute H(index+1).  
//@return a 2-dimensional point with coordinates in [0,1)^2  
double[] nextPoint() {  
index++;  
for(int i = 0; i < K.length; i++) {  
for(int j = 0; j < K[i]; j++) {  
d[i][j]++; x[i] += q[i][j];  
if (d[i][j] < P[i]) {  
break;  
}  
d[i][j] = 0;  
x[i] -= (j == 0? 1.0: q[i][j-1]);  
}  
}  
return x;  
}  
}

//Mapper class for Pi estimation.  
//Generate points in a unit square and then  
//count points inside/outside of the inscribed circle of the square.  
public static class PiMapper extends MapReduceBase implements Mapper {

//Map method.  
//@param offset samples starting from the (offset+1)th sample.  
//@param size the number of samples for this map  
//@param out output {ture->numInside, false->numOutside}  
//@param reporter  
public void map(LongWritable offset, LongWritable size, OutputCollector out, Reporter reporter) throws IOException {

final HaltonSequence haltonsequence = new HaltonSequence(offset.get());  
long numInside = 0L;  
long numOutside = 0L;

for(long i = 0; i < size.get(); ) { //generate points in a unit square final double[] point = haltonsequence.nextPoint();

//count points inside/outside of the inscribed circle of the square final double x = point[0] - 0.5; final double y = point[1] - 0.5; if (xx + yy > 0.25) { numOutside++; } else { numInside++; }

//report status i++; if (i % 1000 == 0) { reporter.setStatus("Generated " + i + " samples."); } }

//output map results out.collect(new BooleanWritable(true), new LongWritable(numInside)); out.collect(new BooleanWritable(false), new LongWritable(numOutside)); } }

//Reducer class for Pi estimation.  
//Accumulate points inside/outside results from the mappers.  
public static class PiReducer extends MapReduceBase implements Reducer, Writable> {

private long numInside = 0; private long numOutside = 0; private JobConf conf; //configuration for accessing the file system

//Store job configuration.  
@Override  
public void configure(JobConf job) { conf = job; }

// Accumulate number of points inside/outside results from the mappers.  
// @param isInside Is the points inside?  
// @param values An iterator to a list of point counts  
// @param output dummy, not used here.  
// @param reporter

public void reduce(BooleanWritable isInside, Iterator values, OutputCollector<WritableComparable, Writable> output, Reporter reporter) throws IOException { if (isInside.get()) { for(; values.hasNext(); numInside += values.next().get()); } else { for(; values.hasNext(); numOutside += values.next().get()); } }

//Reduce task done, write output to a file.  
@Override  
public void close() throws IOException { //write output to a file Path outDir = new Path(TMP\_DIR, "out"); Path outFile = new Path(outDir, "reduce-out"); FileSystem fileSys = FileSystem.get(conf); SequenceFile.Writer writer = SequenceFile.createWriter(fileSys, conf, outFile, LongWritable.class, LongWritable.class, CompressionType.NONE); writer.append(new LongWritable(numInside), new LongWritable(numOutside)); writer.close(); } }

//Run a map/reduce job for estimating Pi.  
//@return the estimated value of Pi.  
public static BigDecimal estimate(int numMaps, long numPoints, JobConf jobConf )  
throws IOException { //setup job conf jobConf.setJobName(PiEstimator.class.getSimpleName());

jobConf.setInputFormat(SequenceFileInputFormat.class);

jobConf.setOutputKeyClass(BooleanWritable.class); jobConf.setOutputValueClass(LongWritable.class); jobConf.setOutputFormat(SequenceFileOutputFormat.class);

jobConf.setMapperClass(PiMapper.class); jobConf.setNumMapTasks(numMaps);

jobConf.setReducerClass(PiReducer.class); jobConf.setNumReduceTasks(1);

// turn off speculative execution, because DFS doesn't handle  
// multiple writers to the same file.  
jobConf.setSpeculativeExecution(false);

//setup input/output directories  
final Path inDir = new Path(TMPDIR, "in");final Path outDir = new Path(TMPDIR, "out");  
FileInputFormat.setInputPaths(jobConf, inDir); FileOutputFormat.setOutputPath(jobConf, outDir);

final FileSystem fs = FileSystem.get(jobConf);  
if (fs.exists(TMPDIR)) {throw new IOException("Tmp directory " + fs.makeQualified(TMPDIR) + " already exists. Please remove it first."); }  
if (!fs.mkdirs(inDir)) {  
throw new IOException("Cannot create input directory " + inDir);  
}

//generate an input file for each map task  
try { for(int i=0; i < numMaps; ++i) { final Path file = new Path(inDir, "part"+i);  
final LongWritable offset = new LongWritable(i \* numPoints);  
final LongWritable size = new LongWritable(numPoints); final SequenceFile.Writer writer = SequenceFile.createWriter( fs, jobConf, file, LongWritable.class, LongWritable.class, CompressionType.NONE); try { writer.append(offset, size); } finally { writer.close();  
} System.out.println("Wrote input for Map #"+i); }

//start a map/reduce job  
System.out.println("Starting Job"); final long startTime = System.currentTimeMillis(); JobClient.runJob(jobConf); final double duration = (System.currentTimeMillis() - startTime)/1000.0; System.out.println("Job Finished in " + duration + " seconds");

//read outputs  
Path inFile = new Path(outDir, "reduce-out"); LongWritable numInside = new LongWritable(); LongWritable numOutside = new LongWritable(); SequenceFile.Reader reader = new SequenceFile.Reader(fs, inFile, jobConf); try { reader.next(numInside, numOutside); } finally { reader.close(); }

//compute estimated value return BigDecimal.valueOf(4).setScale(20) .multiply(BigDecimal.valueOf(numInside.get())) .divide(BigDecimal.valueOf(numMaps)) .divide(BigDecimal.valueOf(numPoints)); } finally { fs.delete(TMP\_DIR, true); } }

//Parse arguments and then runs a map/reduce job.  
//Print output in standard out.  
//@return a non-zero if there is an error. Otherwise, return 0.  
public int run(String[] args) throws Exception { if (args.length != 2) { System.err.println("Usage: "+getClass().getName()+" "); ToolRunner.printGenericCommandUsage(System.err); return -1; }

final int nMaps = Integer.parseInt(args[0]); final long nSamples = Long.parseLong(args[1]);

System.out.println("Number of Maps = " + nMaps); System.out.println("Samples per Map = " + nSamples);

final JobConf jobConf = new JobConf(getConf(), getClass()); System.out.println("Estimated value of Pi is " + estimate(nMaps, nSamples, jobConf)); return 0; }

//main method for running it as a stand alone command.  
public static void main(String[] argv) throws Exception { System.exit(ToolRunner.run(null, new PiEstimator(), argv)); } }

## Summary

In this tutorial, you saw how to deploy a MapReduce job on an Hadoop cluster hosted on the Windows Azure HDinsight Service and how to use Monte Carlo methods that require and generare large datasets that can be managed by this service.

## Next Steps

For tutorials running other samples and providing instructions on using Pig, Hive, and MapReduce jobs on Windows Azure HDInsight with Windows Azure PowerShell, see the following topics:

* [Sample: 10GB GraySort](file:///C:\en-us\manage\services\hdinsight\sample-10gb-graysort\)
* [Sample: Wordcount](file:///C:\en-us\manage\services\hdinsight\sample-wordcount\)
* [Sample: C# Steaming](file:///C:\en-us\manage\services\hdinsight\sample-csharp-streaming\)
* [Sample: Scoop Import/Export](file:///C:\en-us\manage\services\hdinsight\sample-sqoop-import-export\)
* [Tutorial: Using Pig](file:///C:\en-us\manage\services\hdinsight\using-pig-with-hdinsight\)
* [Tutorial: Using Hive](file:///C:\en-us\manage\services\hdinsight\using-hive-with-hdinsight\)
* [Tutorial: Using MapReduce](file:///C:\en-us\manage\services\hdinsight\using-mapreduce-with-hdinsight\)