

Cloud Based Data Engineering

Lecture 7: Spark

Spark

Apache Spark is a fast and general-purpose cluster computing system.

- Provides high-level APIs in Java, Scala, Python and R.
- Optimized engine that supports general execution graphs.
- Can manage "big data" collections with a small set of high-level primitives like map, filter, group-by, and join.
- Supports a rich set of higher-level tools including Spark SQL.



RDD - Resilient Distributed Datasets

- RDDs behave like Python collections (e.g. lists).
- Spark iteratively **apply functions** to every item of these collections in parallel to produce new RDDs.
- The data is distributed across nodes in a cluster of computers.
- Functions implemented in Spark can work in parallel across elements of the collection.
- RDDs automatically rebuilt on machine failure.

Operations in Distributed DBs

Two types of operations: **transformations** and **actions**

- Transformations are **lazy** (not computed immediately)
- Transformations are executed when an action is run

Transformations

map()
filter()
union()
distinct()
groupBy()
...

Actions

reduce()
collect()
count()
first()
saveAsTextFile()
..

Sample Spark transformations

map(func): passing each element of the source through func

filter(func): selecting those elements of the source on which returns true

union/intersection(otherDataset): contains the union/intersection of elements

join(otherDataset, [numTasks]): When called on datasets of type (K, V) and (K, W),

returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key.

Outer joins are supported through leftOuterJoin, rightOuterJoin, and fullOuterJoin.

Operations in Distributed DBs - Transformation

map() flatMap()

join()

filter()

cogroup()

mapPartitions() mapPartitionsWithIndex()

cartesian()

sample()

pipe()

union() intersection() distinct()

coalesce()

groupBy() groupByKey()

repartition()

reduceBy() reduceByKey()

partitionBy()

sortBy() sortByKey()

Sample Spark Actions

reduce(func): Aggregate the elements of the dataset using a function func (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.

collect(): Return all the elements of the dataset as an array at the driver program.

This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.

count(): Return the number of elements in the dataset.

Actions cause calculations to be performed
transformations just set things up (lazy evaluation)

Operations in Distributed DBs - Actions

reduce()

saveAsTextFile()

collect()

saveAsSequenceFile()

count()

saveAsObjectFile()

first()

countByKey()

take()

foreach()

takeSample()

saveToCassandra()

takeOrdered()

Spark – RDD Persistence

- You can persist (cache) an RDD
- When you persist an RDD, each node stores any partitions of it that it computes in memory and reuses them in other actions on that dataset (or datasets derived from it)
- Allows future actions to be much faster (often >10x).
- Can choose storage level (MEMORY_ONLY, DISK_ONLY, MEMORY_AND_DISK, etc.)
- Can manually call unpersist()

RDD – Key Observations

- RDDs can be created from Hadoop InputFormats (such as HDFS files), “parallelize()” datasets, or by transforming other RDDs (you can stack RDDs)
- Actions can be applied to RDDs; actions force calculations and return values
- Lazy evaluation: Nothing computed until an action requires it
- RDDs are best suited for applications that apply the same operation **to all elements**

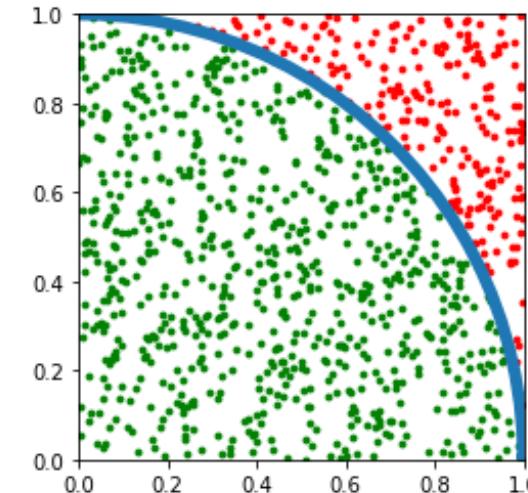
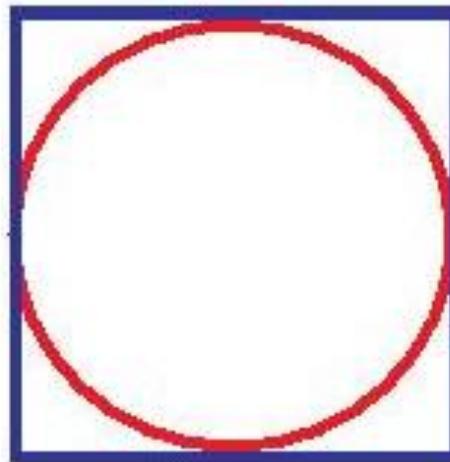
Life Cycle of Spark Program

1. Create input RDDs from external data or parallelize a collection in your driver program.
2. Lazily transform them to define new RDDs using transformations like `filter()` or `map()`
3. Ask Spark to cache() any intermediate RDDs that will need to be reused.
4. Launch actions such as count() and collect() to kick off a parallel computation, which is then optimized and executed by Spark.

Spark – First Application

Goal: estimate π using a stochastic method

- Obviously, there are more efficient ways to figure out digits of pi
- This example will help build a conceptual understanding

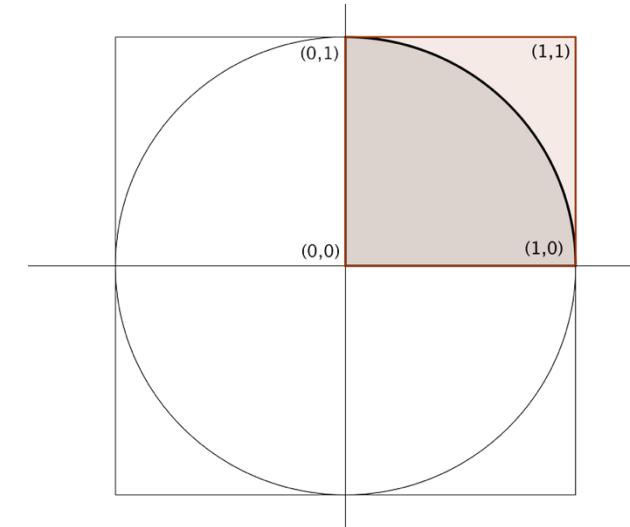
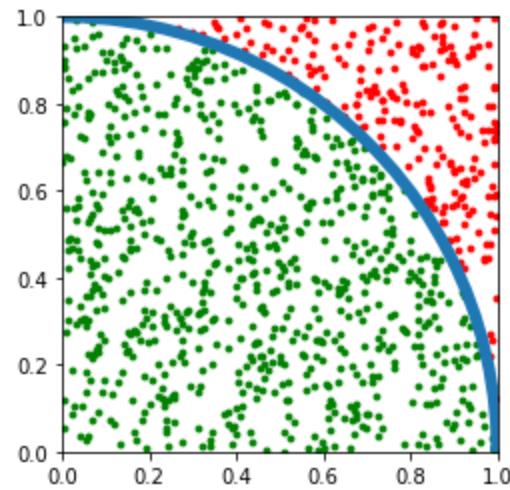


Spark – First Application

Use a **circle** of radius 1 and center in (0,0)

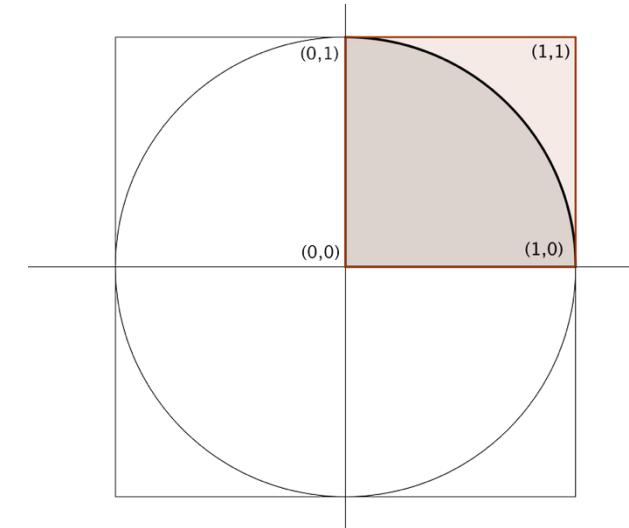
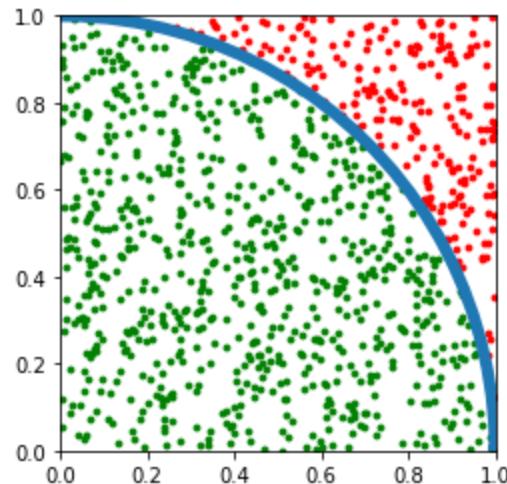
- The ratio of the area of the circle to the area of the square is:

$$\frac{\pi R^2}{2 \cdot 2} = \frac{\pi \cdot 1^2}{4} = \frac{\pi}{4}$$



Spark – First Application

We will use this key fact in order to estimate π by randomly selecting points with the square, and checking whether they are within the circle.



Estimate π - Algorithm

- Generate n points uniformly distributed within the square.
- Let X be a random variable equal to 1 if a randomly generated point falls within the circle, and 0 otherwise.
- Output $\frac{4}{n} \cdot \sum X_i$

Estimate π - Algorithm

By the algorithm, X is a binomial random variable, which makes:

$$X_i \sim \text{Bernoulli} \left(\frac{\pi}{4} \right)$$

$$E[X_i] = \frac{\pi}{4}$$

$$E[4 \cdot X_i] = 4 \cdot \frac{\pi}{4} = \pi$$

But how accurate is this method?

How accurate is this method?

Consider the variance of our estimate:

$$Var\left[\frac{4}{n} \cdot \sum X_i\right] = \frac{16}{n^2} \cdot Var\left[\sum X_i\right] = \frac{16}{n} \cdot Var[X]$$

Since X_i is a Bernoulli R.V. with $p = \frac{\pi}{4}$ the variance:

$$Var(X) = \frac{\pi}{4} \cdot \left(1 - \frac{\pi}{4}\right)$$

However, we will assume we don't know π . We need to compute the sample variance of X .

Sample Variance of X

We can calculate this retrospectively, since X is a Bernoulli R.V:

Ex: When n = 1000, we estimate pi to be 3.188. That is,

$$\frac{4}{1000} \cdot \sum X_i = 3.188 \leftrightarrow \sum X_i = 797$$

797 of our points fell within the circle,

making x = 1 for 797 of our trials, and zero for the other 203

Hypothetically...

With this method for estimating π , how many trials are necessary for us to be at least 95% sure that our error is less than 1×10^{-10} ?

About 1.036×10^{21}



Estimate π - Spark

```
def sample(p):  
    x, y = np.random(), np.random()  
    return 1 if x*x + y*y < 1 else 0  
  
count = spark.parallelize(range(0, NUM_SAMPLES)).map(sample).reduce(lambda a, b: a + b)  
print("Pi is", 4.0 * count / NUM_SAMPLES)
```

Spark DataFrames

Enable wider audiences beyond “Big Data” engineers to leverage the power of distributed processing.

- Inspired by data frames in R and Python (Pandas)
- Designed from the ground-up to support modern big data and data science applications
- Extension to the existing RDD API

DataFrames Are..

- The preferred abstraction in Spark
- Strongly typed collection of distributed elements
- Built on Resilient Distributed Datasets (RDD)
- Immutable once constructed

With DataFrames You Can..

With DataFrames you can:

- Track lineage information to efficiently recompute lost data
- Enable operations on collection of elements in parallel

construct DataFrames:

- by parallelizing existing collections (e.g., Pandas DataFrames)
- by transforming an existing DataFrames
- from files in HDFS or any other storage system (e.g., Parquet)

Apache Spark: Libraries “on top” of core that come with it

- Spark SQL
- Spark Streaming – stream processing of live datastreams
- MLlib - machine learning
- GraphX – graph manipulation
 - extends Spark RDD with Graph abstraction: a directed multigraph with properties attached to each vertex and edge.

Gray sort competition: Winner

	Hadoop MR Record	Spark Record (2014)
Data Size	102.5 TB	100 TB
Elapsed Time	72 mins	23 mins
# Nodes	2100	206
# Cores	50400 physical	6592 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network
Sort rate	1.42 TB/min	4.27 TB/min
Sort rate/node	0.67 GB/min	20.7 GB/min

Sort benchmark, Daytona Gray: sort of 100 TB of data (1 trillion records)