bigdata

WeizhouS

3/19/2019

Fundamentals of BIG DATA

3 V's of big data

- Volumne
- Variety
- Velocity

Concepts and Terminology

- Clustered computing: collection of resources of mltiple machines
- Parallel computing: Simultaneous computation
- Distributed computing: Collection of nodes(networked computers) that run parallelly
- Batch processing: Breaking job into small pieces
- Real-time processing: immediate processing of data

Big Data processing systems

- Hadoop/MapReduce: Scalable and fault tolerant framework written in Java
- Apache Spark: General purpose and lightning fast cluster computing system

Features of Apache Spark frame work

- Distributed cluster computing framework
- Efficient in-memory computations for large data sets
- Lighting fast data processing framework
- Provides support for Java, Scala, Python, R and SQL

Apache Spark Components

- Core: RDD API
- Spark SQL
- MLlib
- GraphX
- Spark Streaming

Spark modes of deployment

- **Local mode:** Single machine, convenient for testing
- **Cluster mode:** Set of pre-defined machines, good for production
- Workflow: Local → clusters
- *No code change necessary*

PySpark: Spark with Python

Overview

- Apache Spark is written in Scala
- PySpark is to support Python with Spark
- Similar speed as Scala
- APIs similar to *Pandas* and *Scikit-learn*

Spark shell

- Interactive environment for Spark jobs
- All interaction with data on disk or in memory
- Spark-shell for Scala
  PySpark-shell for Python
  SparkR for R

PySpark shell

- Python-based command line tool
- Allows interface with Spark data strucures
- Support connecting to a cluster

SparkContext

- Entry point into the world of Spark
- PySpark has a default SparkContext called sc

```
sc.version
sc.pythonVer
sc.master
rdd = sc.parallelize([1,2,3])
rdd2 = sc.textFile('text.txt')
```

Anonymous functions

- Lambda functions
- Efficient with map() and filter()
- Inline function definition or to defer excution of a code
- No return statement

```
lambda arguments: expression
map(fuction, list)
filter(function, list)
```

RDD

RDD = Resilient Distributed Datasets

- Resilient: Ability to withstand failures
- Distributed: Spanning across multiple machines
- Datasets: Collection of partitioned data

How to create RDDs?

- Parallelizing an existing collection of objects
- External datasets:
  - HDFS
  - Amazon S3 bucket
  - text file
- Existing RDDs

Partitioning in PySpark

- A partition is a logical division of a large distriuted data set

```
numRDD = sc.parallelize(range(10), minPartitions = 6)
fileRDD = sc.textFile("bigdata.md", minPartitions = 6)
numRDD.getNumPartions()
```

RDD operations in PySpark

- Transformations crate new RDDs
- Actions perform computation on the RDDs

Transformations

- Lazy evaluation
- Basic transformations: map() flatmap() filter() uninion()

Actions

- Return a value
- Basic actions: collect() take(N) first() count()

Pair RDDs

Introduction

- Close to real life datasets: key/value pair

Creating pair RDDs

- From a list of key value tuple
- From a existing RDD

Transformations

- Regular transformation should pass functions that operate on key value pairs
- reduceByKey(func) groupByKey() sortByKey() join

More ACTIONS:

- reduce()
- saveAsTextFile()

```
RDD.coalesce(1).saveAsTextFile("tempFile")
```

- countByKey()
- collectAsMap(): return dictionary

PySpark DataFrames

- Immutable distributed collection of data with named columns
- Designed for structured (relational database) and semi-structured data (JSON)
- Dataframe API is available in Python, R, Scala, and Java
- Dataframe in PySpark support both SQL query(`SELECT * FROM table`) or expression methods (`df.select()`)

SparkSession

- Provides a single point of entry to interact with Spark DataFrames
- SparkSession is used to create DataFrame, register DataFrames, execute SQL queries
- Available in PySpark shell as spark

Creating DataFrames in PySpark

- From existing RDDs using SparkSession's createDataFrame()
- From data sources(csv, json, txt) using SparkSession's read method
- Schema controls the data and helps DataFrames to optimize queries
- Schema provides information about column name, type of data, empty values etc

```
df = spark.createDataFrame(RDD, schema=list(names))
df = spark.read.csv("text.csv", header=True, inferSchema=True)
```

Interaction with PySpark DataFrames

Operators

- Transformation: `select()` `filter()` `grouby()` `orderby()` `dropDuplicates()` `withColumnRenamed()`
- Actions: `printSchema()` `head()` `show()` `count()` `columns()` `describe()`

Interacting with DataFrames using PySpark SQL

- DataFrame API provides a programmatic domain-specific language (DSL) for data
- SQL queries can be concise and easier to understand and portable

```
df.createOrReplaceTempView("table1")
df2 = spark.sql("SELECT field1 from table1")
query = '''SELECT field1 from table1'''
df2 = spark.sql(query)
```

Data Visualization in PySpark using DataFrames

- In Python: Matplotlib, Seaborn, Bokeh
- For PySpark DataFrames:
  - pyspark_dist_explore: `hist()` `distplot()` `pandas_histogram()`
  - toPandas()
  - HandySpark library

Pandas DataFrame vs PySpark DataFrame

- Pandas DataFrames are in-memory, single-server based structures
- Operations on PySpark run in parallel
- Not lazy operations vs lazy transformations
- Mutable vs immutable
- Pandas API support more operations

PySpark MLlib

- MLlib is component of Apache Spark for machine learning
- Tools include:
  - ML Algorithms: collaborative filtering, classification and clustering
  - Featurization: feature extraction, transformation, dimensionality reduction, and selection
  - Pipelines: tools for constructing, evaluating and tuning ML Pipelines

Why PySpark MLlib?

- sklearn only work for small datasets on a single machine
- Spark's MLlib algorithms are designed for parallel processing on a cluster
- Support Scala, Java and R
- Provides a high-level API to build machine learning pipelines

PySpark MLlib Algorithms: 3C

- Classification and Regression:
  - Linear SVMs
  - Logistic regression
  - Decision trees
  - Random forests
  - Gradient-boosted trees
  - Naive Bayes
  - Linear least squares
  - Lasso
  - Ridge regression
  - Isotonic regression
- Collaborative filtering:
  - Alternating least squares
- Clustering:
  - K-means
  - Gaussian mixture
  - Power iteration clustering (PIC)
  - Bisecting K-means

- Streaming K-Means

```
from pyspark.mllib.recommendation import ALS
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
from pyspark.mllib.clustering import KMeans
```

Collaborative filtering

- Finding users that share common interests
- Used for recommender systems
- Two approaches:
  - User-User Collaborative filtering
  - Item-Item Collaborative filtering

Rating class

- a tuple (user, product, rating)

randomSplit()

- Randomly splits RDD with provided weights and returns multiple RDDs

```
train, test = rdd.randomSplit([0.7, 0.3])
```

Alternating Least Squares (ALS)

- `model = ALS.train(ratings, rank, iterations)`
- `preds = model.predictAll(unratedRDD)`

Classification

Vectors and LabelledPoint()

```
denseVec = Vectors.dense([1.1, 0, 3.3])
sparseVec = Vectors.sparse(3, [0, 2], [1.1, 3.3])

pt = LabeledPoint(1, [1.1, 0, 3.3])

HashingTF()
```

- RDD of LabelledPoint to be trained