TÜBİTAK BİLİŞİM VE BİLGİ GÜVENLİĞİ
İLERİ TEKNOLOJİLER ARAŞTIRMA MERKEZİ

www.bilgem.tubitak.gov.tr

b3lab
BULUT BİLİŞİM VE BÜYÜK VERİ ARAŞTIRMA LABORATUVARI
CLOUD COMPUTING & BIG DATA RESEARCH LABORATORY

SPARK

Sunum Planı

Giriş

Spark Shell

SparkContext

Resilient Distributed Dataset (RDD)

Aksiyon ve Dönüşümler

Eşli RDD

Spark Uygulamasının Bileşenleri

Spark Uygulama İzleme Arayüzü

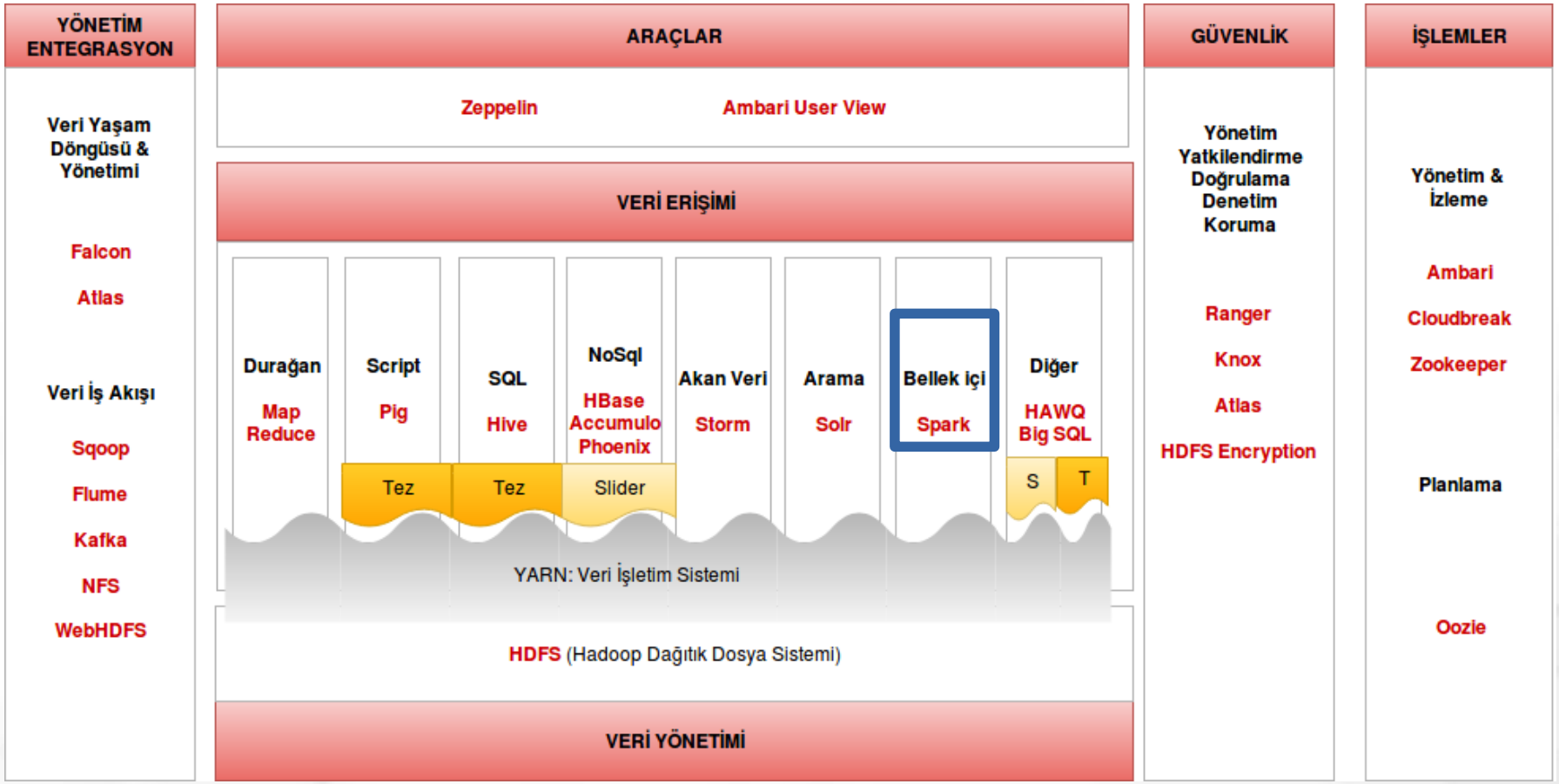
Spark Uygulaması Oluşturma ve Çalıştırma

Spark SQL / DataFrame İşlemleri

Spark Streaming

Spark MLLib

Giriş


YÖNETİM ENTEGRASYON
Veri Yaşam Döngüsü & Yönetimi
Falcon
Atlas
Veri İş Akışı
Sqoop
Flume
Kafka
NFS
WebHDFS
ARAÇLAR
Zeppelin
Ambari User View
VERİ ERİŞİMİ
Durağan
Map Reduce
Script
Pig
SQL
Hive
NoSql
HBase
Accumulo
Phoenix
Akan Veri
Storm
Arama
Solr
Bellek içi
Spark
Diğer
HAWQ
Big SQL
Tez
Tez
Slider
S
T
YARN: Veri İşletim Sistemi
HDFS (Hadoop Dağıtık Dosya Sistemi)
VERİ YÖNETİMİ
GÜVENLİK
Yönetim
Yatkilendirme
Doğrulama
Denetim
Koruma
Ranger
Knox
Atlas
HDFS Encryption
İŞLEMLER
Yönetim & İzleme
Ambari
Cloudbreak
Zookeeper
Planlama
Oozie

Giriş

Spark

bellek içi veri kümeleri üzerinde hızlı işlemler yapabilen
açık kaynak
Apache lisanslı
bir büyük veri çözümüdür.

Scala, Python, R, Java dillerini destekler.

Hadoop ile entegre

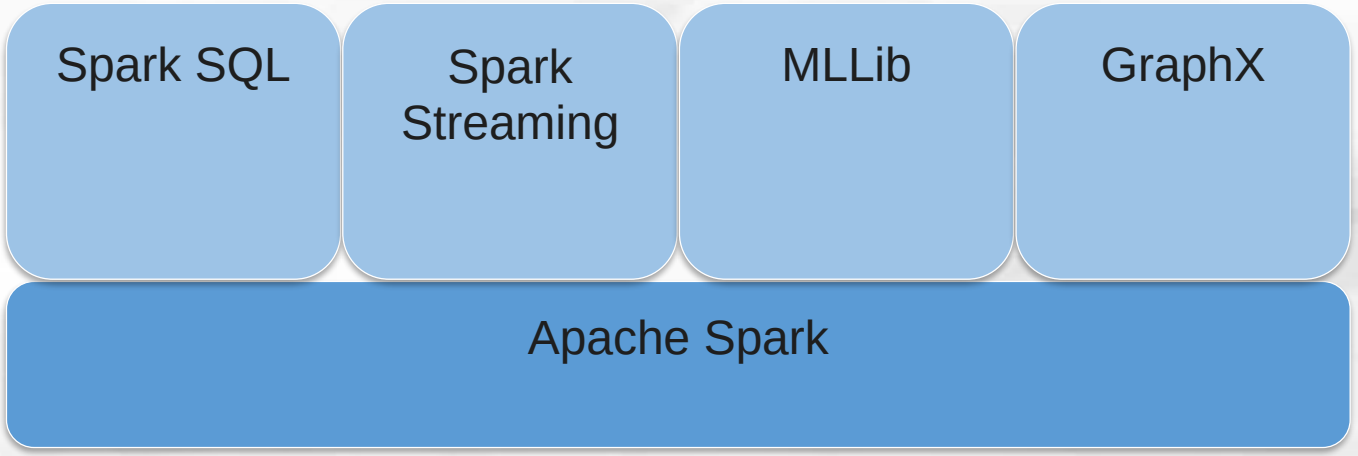
RDD (Resilient Distributed Dataset)

Esnek, dağıtık veri kümesi

Spark'ın Avantajları

Uygulama programlama arayüzleri (API)

- Data Frame/SQL
- Akan veri işleme
- Makine öğrenmesi
- Çizge algoritmaları için



Spark Kullanan Firmalara Örnekler

Spark ve MapReduce Karşılaştırması

MapReduce çok sayıda Disk I/O içerir.
Spark daha çok veriyi bellekte tutar.

Spark, daha yüksek seviyeli API sunar.

Aynı map/reduce işlemi, MapReduce'ta ~40 satırda, Spark ile ~3 satırda yapılabilir.

Spark Shell

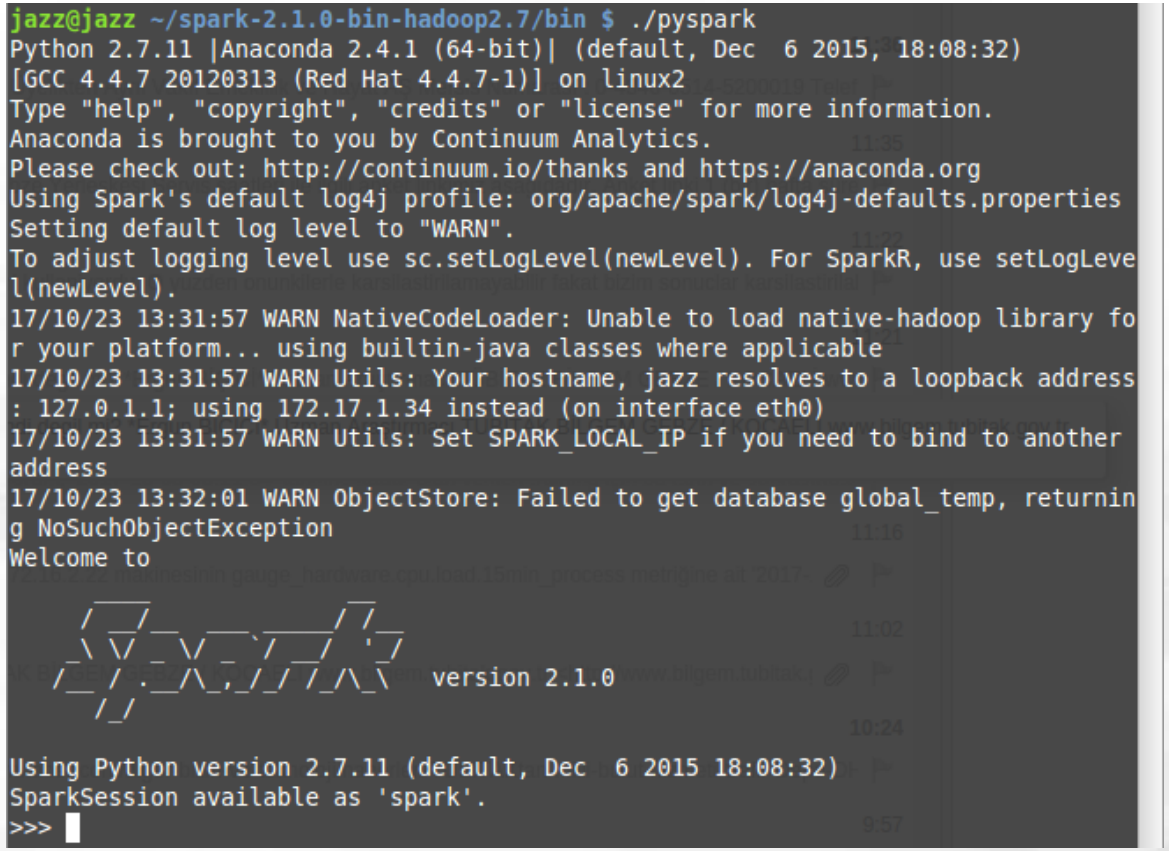
Etkileşimli Spark arayüzü

Pyhton:

```
$ pyspark
```

Scala:

```
$ spark-shell
```



SparkContext

Spark kümesiyle olan bağlantıyı temsil eder.

Spark Shell tarafından otomatik oluşturulur.

Uygulamada kullanıcı tarafından oluşturulur.

SparkContext

Özellikler

- **`sc.appName`**: Spark uygulama ismi
- **`sc.master`**: Spark Master (`local`, `yarn-client`, gibi)
- **`sc.version`**: Kullanılan Spark versiyonu

Fonksiyonlar

- **`sc.parallelize()`**: Veriden RDD oluşturmak için
- **`sc.textFile()`**: Bir metin dosyasından RDD oluşturmak için
- **`sc.stop()`**: SparkContext'in durdurulması

Resilient Distributed Dataset (RDD)

Her RDD bir veya daha fazla bölümden (partition) oluşur.
Bölüm sayısını kullanıcı kontrol edebilir.
Bölümleme sayesinde dağıtık çalışma mümkün olmaktadır.

RDD Oluşturma

Veri, dosya(lar)dan yüklenebilir (lokal, HDFS, vb).

```
rdd1 = sc.textFile("file://dosya/yolu/dosyam.txt")

rdd2 = sc.textFile("hdfs://namenode:8020/dizin1/veri.txt)

rdd3 = sc.textFile("dizin2/*.txt")

rdd4 = sc.textFile("veri1.txt,veri2.txt")
```

RDD Oluşturma

Veri, `parallelize` fonksiyonu ile yüklenebilir.

```
rdd5 = sc.parallelize([2, 4, 7, 9, 18])

rdd6 = sc.parallelize(["kuzu", "deve", "inek"])

veri1 = ("ali okula gitti")
rdd7 = sc.parallelize([veri1])
```

Aksiyonlar (Actions) ve Dönüşümler (Transformations)

Bir RDD üzerinde iki farklı şey uygulanabilir:

Aksiyonlar (Actions)

Bir sonuç döndürür. Örn.:

```
rdd1.count()
```

Dönüşümler (Transformations)

Bir RDD yeni bir RDD'ye dönüştürülür. Örn.:

```
rdd2 = rdd1.filter(lambda line: “Spark” in line)
```

Fonksiyonel Programlama

Fonksiyonlar, başka fonksiyonlara girdi olarak gönderilir.

Anonim fonksiyonlar

Tanımlı olmayan fonksiyonların satır içinde (inline) gönderilmesi

```
flatMap(lambda line: line.split(" ") )
```

Fonksiyon argumanı Fonksiyon gövdesi

filter() Dönüşümü

Bir koşula göre bazı elemanların tutulmasını sağlar.

```
rdd = sc.parallelize([1, 2, 3, 4, 5])
rdd2 = rdd.filter(lambda x:  x%2 == 1)
[1, 3, 5]
```

map() Dönüşümü

Bir RDD'nin her bir elemanına bir fonksiyon uygulanmasını sağlar.

```
rdd=sc.parallelize([1, 2, 3, 4, 5])
rdd.map(lambda x: x*3+1).collect()

[4, 7, 10, 13, 16]
```

flatMap() Dönüşümü

Bir RDD'nin her bir elemanına bir fonksiyon uygulanmasını sağlar.

```
rdd = sc.textFile("egitimler.txt")
```

Büyük Veri Eğitimi
Spark Eğitimi
Hive Eğitimi
Hbase Eğitimi

```
rdd1 = rdd.map(lambda line: line.split(" "))
```

["Büyük", "Veri", "Eğitimi"]
["Spark", "Eğitimi"]
["Hive", "Eğitimi"]
["Hbase", "Eğitimi"]

```
rdd2 = rdd.flatMap(lambda line: line.split(" "))
```

"Büyük"
"Veri"
"Eğitimi"
"Spark"
"Eğitimi"
...
"Hbase"
"Eğitimi"

collect() Aksiyonu

RDD'nin tamamını döndürür.

```
rdd = sc.parallelize([8, -21, 9, 103, 20, 7])
rdd.collect()
```

```
[8, -21, 9, 103, 20, 7]
```

count() Aksiyonu

RDD'deki eleman sayısını döndürür.

```
veri = [8, -21, 9 , 103, 20, 7]
rdd = sc.parallelize(veri)
rdd.count()
```

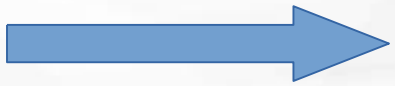
```
6
```

Eşli RDD (Pair RDD)

Elemanları anahtar-değer tipinde olan RDD'dir.

Normal RDD'lerden map() veya flatmap() ile oluşturulabilir.

```
rdd = sc.textFile("egitimler.txt")
rdd1 = rdd.flatMap(lambda satir: satir.split(' '))
rdd2 = rdd1.map(lambda kelime: (kelime,1))
rdd2.collect()
```



“Büyük”	(“Büyük”, 1)
“Veri”	(“Veri”, 1)
“Eğitimi”	(“Eğitimi”, 1)
“Spark”	(“Spark”, 1)
“Eğitimi”	(“Eğitimi”, 1)
...	...
“Hbase”	(“Hbase”, 1)
“Eğitimi”	(“Eğitimi”, 1)

Eşli RDD Dönüşümü

reduceByKey ile anahtar bazında reduce işlemi yapılır.

```
rdd2.reduceByKey(lambda a,b: a+b).collect()
```

RDD'lerle Çalışma

Dönüşümler “tembel”dir.

Dönüşümler için Directed Acyclic Graph (DAG) oluşturulur.

RDD/sonraki RDD üzerinde bir aksiyon çağırılana kadar işlem yapılmaz.

```
rdd1 = sc.textFile("hdfs://egitimler.txt”)
rdd2 = rdd1.flatMap(lambda satir: line.satir(" "))
rdd3 = rdd2.map(lambda kelime: (kelime, 1))
rdd4 = rdd3.reduceByKey(lambda a, b: a + b)

rdd4.saveAsTextFile("hdfs://kelime-sayisi.txt”)
```

DAG oluşturulur.

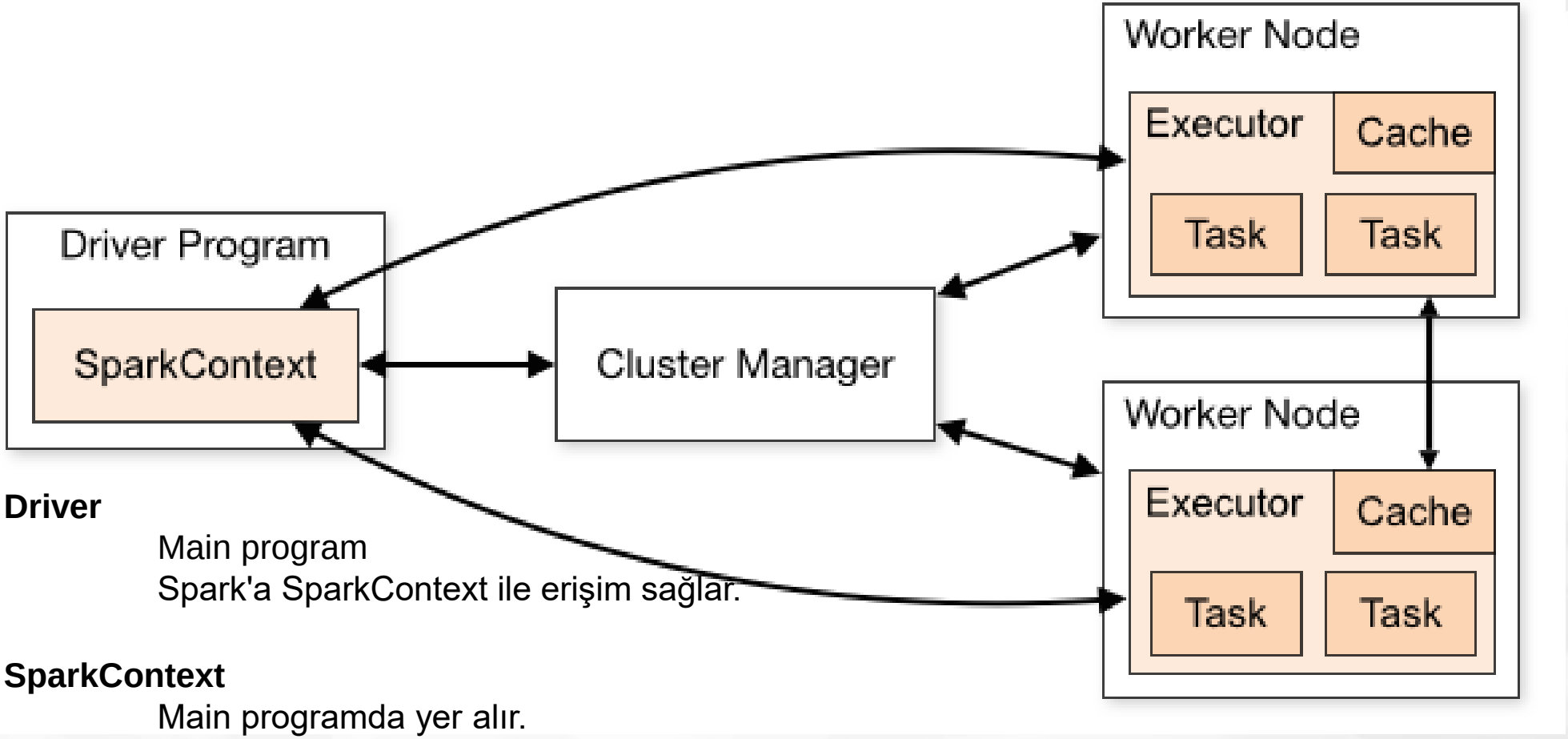
DAG çalıştırılır.

RDD'lerle Çalışma

RDD'ler herhangi tipte eleman içerebilir:

- int, float, bool, …
- string, list, array, …
- karışık veri tipi

Spark Uygulamasının Bileşenleri (Mimari)



Driver

Main program
Spark'a SparkContext ile erişim sağlar.

SparkContext

Main programda yer alır.
Farklı tipte cluster manager'a bağlanabilir (Standalone, Hadoop YARN gibi)

Executor

Alt işler (tasks), Executor'lara gönderilir.

Spark Task, Stage, Job

Job (iş), aşamalardan oluşur.

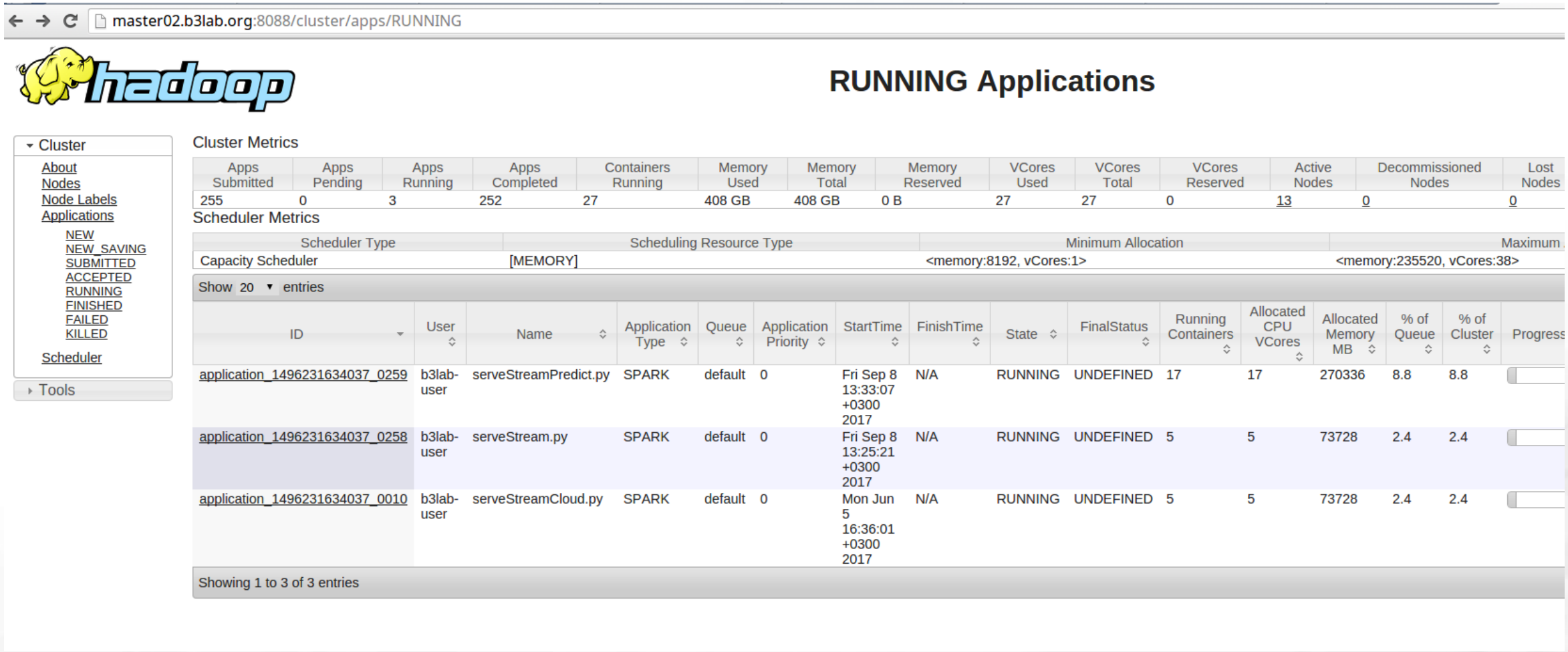
Shuffle (karıştırma) gerektiren işlemlere göre aşamalara bölünür.

Stage (aşama), görevlerden meydana gelir.

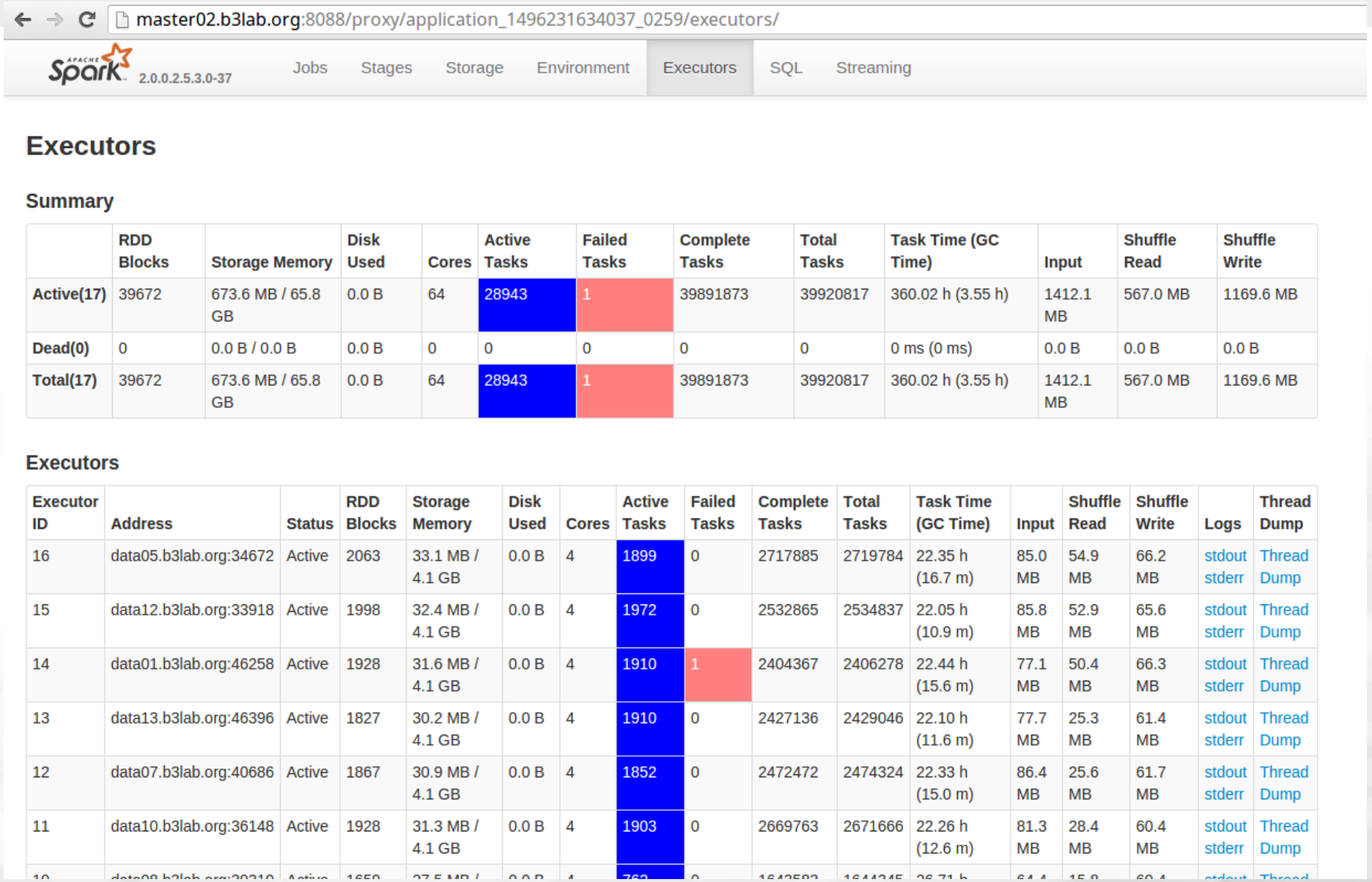
Task (görev), Spark'taki temel çalışma birimidir.

Task → Stage → Job

Spark Uygulama İzleme Arayüzü



Spark Uygulama İzleme Arayüzü – Executors



Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
Active(17)	39672	673.6 MB / 65.8 GB	0.0 B	64	28943	1	39891873	39920817	360.02 h (3.55 h)	1412.1 MB	567.0 MB	1169.6 MB
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
Total(17)	39672	673.6 MB / 65.8 GB	0.0 B	64	28943	1	39891873	39920817	360.02 h (3.55 h)	1412.1 MB	567.0 MB	1169.6 MB

Executors

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
16	data05.b3lab.org:34672	Active	2063	33.1 MB / 4.1 GB	0.0 B	4	1899	0	2717885	2719784	22.35 h (16.7 m)	85.0 MB	54.9 MB	66.2 MB	stdout stderr	Thread Dump
15	data12.b3lab.org:33918	Active	1998	32.4 MB / 4.1 GB	0.0 B	4	1972	0	2532865	2534837	22.05 h (10.9 m)	85.8 MB	52.9 MB	65.6 MB	stdout stderr	Thread Dump
14	data01.b3lab.org:46258	Active	1928	31.6 MB / 4.1 GB	0.0 B	4	1910	1	2404367	2406278	22.44 h (15.6 m)	77.1 MB	50.4 MB	66.3 MB	stdout stderr	Thread Dump
13	data13.b3lab.org:46396	Active	1827	30.2 MB / 4.1 GB	0.0 B	4	1910	0	2427136	2429046	22.10 h (11.6 m)	77.7 MB	25.3 MB	61.4 MB	stdout stderr	Thread Dump
12	data07.b3lab.org:40686	Active	1867	30.9 MB / 4.1 GB	0.0 B	4	1852	0	2472472	2474324	22.33 h (15.0 m)	86.4 MB	25.6 MB	61.7 MB	stdout stderr	Thread Dump
11	data10.b3lab.org:36148	Active	1928	31.3 MB / 4.1 GB	0.0 B	4	1903	0	2669763	2671666	22.26 h (12.6 m)	81.3 MB	28.4 MB	60.4 MB	stdout stderr	Thread Dump

Spark Uygulaması Oluşturma - Örnek


```
import os
import sys
from pyspark import SparkContext, SparkConf

if __name__ == "__main__":
        conf = SparkConf() \
                .setAppName("App Name") \
                .setMaster("yarn-client")
        sc = SparkContext(conf=conf)
        sc.textFile("myfile.txt")
        ## Do Stuff
        sc.stop()
```

Spark Uygulamasının Çalıştırılması - Örnek


```
spark-submit --master yarn-client --num-executors 4 \
--executor-memory 8g /dizin1/uygulama1.py
```

Önemli Konfigürasyon Ayarları

conf/spark-defaults.conf

./bin/spark-submit --conf …

```
conf = SparkConf()
conf.set('spark.serializer',
            'org.apache.spark.serializer.KryoSerializer’)
```

Spark SQL

Yapısal veri işlemek için Spark modülü

RDD'den farklı olarak, verinin yapısı ve işlemlerle ilgili daha fazla bilgi sağlanır.

Spark SQL ile veri iki şekilde sorgulanabilir:

SQL sorguları ile

DataFrame / Dataset API ile

Java, Scala, Python and R desteklenmektedir.

Çeşitli kaynaklara erişim sağlanabilir:

Hive, Avro, Parquet, ORC, JSON, and JDBC.

Spark SQL – Başlangıç Noktası: SparkSession


```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

Spark SQL – DataFrame Oluşturma

DataFrame, kavramsal olarak ilişkisel veritabanındaki tabloya karşılık gelir.

SparkSession ile, uygulamalar mevcut bir RDD, Hive tablosu veya Spark veri kaynaklarından DataFrame oluşturabilir.

```
df = spark.read.json("examples/src/main/resources/people.json")

df.show()

# +----+-------+
# | age|   name|
# +----+-------+
# |null|Michael|
# |  30|   Andy|
# |  19| Justin|
# +----+-------+
```

Spark SQL – DataFrame İşlemleri

Şemanın ağaç yapısında görüntülenmesi

```
df.printSchema()


# root
# |-- age: long (nullable = true)
# |-- name: string (nullable = true)
```

Spark SQL – DataFrame İşlemleri

Sadece “name” kolonunun görüntülenmesi

```
df.select("name").show()
```

```
# +-------+
# |   name|
# +-------+
# |Michael|
# |   Andy|
# | Justin|
# +-------+
```

Spark SQL – DataFrame İşlemleri

21 yaşından büyük olanların görüntülenmesi

```
df.filter(df['age'] > 21).show()


# +---+----+
# |age|name|
# +---+----+
# |  30|Andy|
# +---+----+
```

Spark SQL – DataFrame İşlemleri

Yaşa göre kişilerin sayılması

```
df.groupBy("age").count().show()
```

```
# +----+-----+
# | age|count|
# +----+-----+
# |  19|    1|
# |null|    1|
# |  30|    1|
# +----+-----+
```

Spark SQL – SQL Sorgusu Çalıştırma

sql fonksiyonu ile programsal olarak SQL sorguları çalıştırılabilir.

Sonuç, DataFrame olarak döndürülür.

```
# Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")

sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()

# +----+-------+
# | age|   name|
# +----+-------+
# |null|Michael|
# |  30|   Andy|
# |  19| Justin|
# +----+-------+
```

Spark Streaming

Ölçeklenebilir, hataya dayanıklı, akan veri işleme uygulamaları

HDFS, Flume, Kafka, Twitter ve ZeroMQ'dan veri okunabilir.

Özel veri kaynakları tanımlanabilir.

Spark Streaming

Spark Streaming veriyi yığınlara (batches) bölerek işler.

DStream (discretized stream)

Bir RDD dizisine karşılık gelir.

Kafka, Flume, Kinesis vb kaynaklardan veya,

bir Dstream'e işlemler uygulanmasıyla oluşturulabilir.

Scala, Java, Python

Spark Structured Streaming

- Yapılandırılmış akan veri için

- DStreams'ten daha yeni API olan

- Datasets ve DataFrames kullanılıyor.

Spark Structured Streaming – Örnek

SparkSession oluşturulur.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode
from pyspark.sql.functions import split

spark = SparkSession \
    .builder \
    .appName("StructuredNetworkWordCount") \
    .getOrCreate()
```

Spark Structured Streaming – Örnek (devam)


```
# Create DataFrame representing the stream of input lines from connection to
localhost:9999
lines = spark \
    .readStream \
    .format("socket") \
    .option("host", "localhost") \
    .option("port", 9999) \
    .load()

# Split the lines into words
words = lines.select(
   explode(
       split(lines.value, " ")
   ).alias("word")
)

# Generate running word count
wordCounts = words.groupBy("word").count()
```

Spark Structured Streaming – Örnek (devam)


```
 # Start running the query that prints the running counts to the console
query = wordCounts \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .start()

query.awaitTermination()
```

Spark Structured Streaming – Örnek (devam)

Veri sunucusu olarak:

```
$ nc -lk 9999
```

Başka bir terminalde:

```
$ ./bin/spark-submit
examples/src/main/python/sql/streaming/structured_network_wordcount.py localhost
9999
```

Spark Structured Streaming – Örnek (devam)


```
# TERMINAL 1:
# Running Netcat

$ nc -lk 9999
apache spark
apache hadoop
...
```

Python | Scala | Java | R

```
# TERMINAL 2: RUNNING structured_network_wordcount.py

$ ./bin/spark-submit examples/src/main/python/sql/streaming/structured_network_wordcount.py localhost 9999

-------------------------------------------
Batch: 0
-------------------------------------------
+------+-----+
| value|count|
+------+-----+
|apache|    1|
| spark|    1|
+------+-----+

-------------------------------------------
Batch: 1
-------------------------------------------
+------+-----+
| value|count|
+------+-----+
|apache|    2|
| spark|    1|
|hadoop|    1|
+------+-----+
...
```

Spark Streaming – DataFrame ve SQL İşlemleri / Makine Öğrenmesi

Akan veri üzerinde DataFrame ve SQL işlemleri kolaylıkla kullanılabilir.

Akan veri için MLlib tarafından sağlanan algoritmalar

Streaming Linear Regression, Streaming Kmeans vb

Veya:

Çevrimdışı olarak model öğrenilerek, akan veri üzerine uygulanması

Spark MLlib

Java, Scala, Python, R

Makine öğrenmesi algoritmaları
sınıflandırma, regresyon, kümeleme ve ortak filtreleme

Özellik çıkarma (feature extraction), dönüşüm, boyut azaltma ve seçim

Makine öğrenmesi boru hattı oluşturma (Pipelines)

Algoritma, model ve boru hatlarını kaydetme / yükleme

Yardımcı programlar: doğrusal cebir, istatistik, veri yönetimi vb.

Spark MLlib

RDD tabanlı MLlib API bakım modunda (Spark 2.0 itibariyle)

spark.mllib paketi

DataFrame tabanlı API birincil durumda

spark.ml paketi

Spark MLlib – Logistic Classification Örneği


```
from pyspark.ml.classification import LogisticRegression

# Load training data
training = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")

lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# Fit the model
lrModel = lr.fit(training)

# Print the coefficients and intercept for logistic regression
print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))
```

Teşekkür Ederiz..

www.b3lab.org

TÜBİTAK BİLGEM Gebze Yerleşkesi, Gebze, Kocaeli, TÜRKİYE, 41470
T: **+90 262 675 23 92** E: **b3lab.iletisim@tubitak.gov.tr**