BLM433
Dağıtık Sistemlere Giriş

Dr. Süleyman Eken

Bilgisayar Mühendisliği
Kocaeli Üniversitesi

Sunum Planı

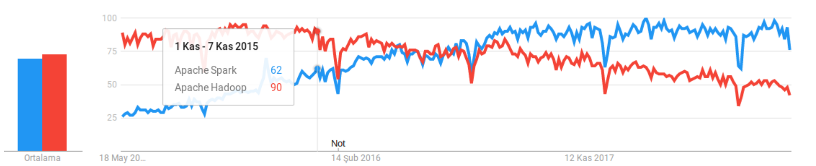
- Açık Kaynak Büyük Veri Araçları
  - Apache Spark, Niçin?
- Apache Spark Nedir?
  - RDD, dönüşüm ve aksiyonlar, RDD sürekliliği
- Spark Uygulama Geliştirme, Shell'den çalıştırma
  - Demo 1: Scala, Java (wordcount)
- Spark Kütüphaneleri
  - Spark SQL, MLlib, Streaming, GraphX
  - Demo 2: Streaming uygulama (NetworkWordCount)
- Big Learning
  - Spark ML paketi, algoritmaları, pipeline yapısı,
  - Demo 3: Sınıflandırma (Naive Bayes)
- Sonuç

Büyük Veri için Açık Kaynak Araçlar

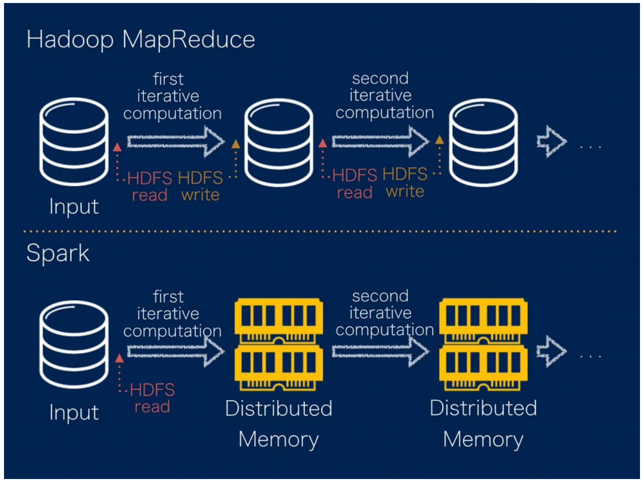
- Büyük veri analitiği platformları
  - Apache Hadoop, Apache Spark, GridGain, HPCC Systems, Storm vs
- Büyük veri depolama sistemleri
  - Cassandra, HBase, MongoDB, Neo4J, CouchDB, OrientDB, Hibari, Riak, Hive, vs
- Büyük veri iş zekası araçları
  - Talend, Jaspersoft, Jedox, Pentaho, SpagoBI, BIRT vs
- Büyük veri madenciliği araçları
  - RapidMiner, Apache Mahout, Weka, KEEL, Rattle vs
- Büyük veri dosyaları toplama ve transfer araçları
  - Apache Lucene, Sqoop, Flume, Chukwa vs
- Diğer araçlar
  - Terracotta, Avro, Oozie, Zookeeper vs

MapReduce Problemleri ve Çözüm

- MapReduce Problemleri:
  - Çoğu problemi map-reduce olarak tanımlamak zor (batch-processing birçok usecase'e uymuyor)
  - Diskte kalıcılık genellikle bellek içi (in-memory) çalışmadan daha yavaş
  - Map-reduce programlamak zor



- Alternatif: Apache Spark
  - MapReduce yerine kullanılabilir genel amaçlı bir işleme motoru
  - in-memory hesaplamalar yapar

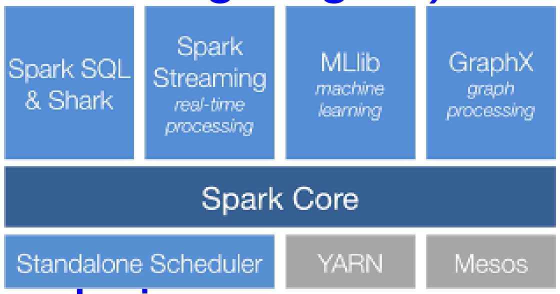


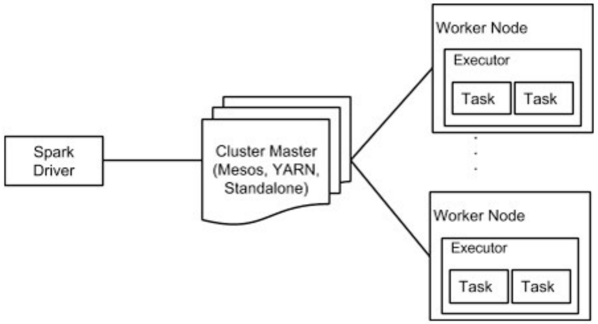
Apache Spark Tarihçesi

- 2009'da Berkeley AMPLab'da Matei Zaharia tarafından başlatıldı (Mesos'u test için).
  - Linux-Mesos, Ubuntu-DC/OS
  - AMP = Algorithms Machines People
  - AMPLab is integrating Algorithms, Machines, and People to make sense of Big Data
- 2010'da açık kaynak halini aldı.
- 2013'te Apache Software Foundation tarafından desteklendi.
- 2014'te top level proje haline geldi.
- Mayıs 2014'te Spark 1.0, Kasım 2016'ta 2.0 piyasaya sürüldü.
- 2015 yılı itibariyle 1000e yakın contributor'e sahip.

Apache Spark

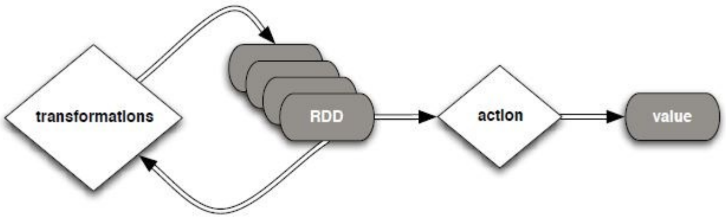
- Map-reduce yerine daha büyük operasyon seti (transformations & actions) tanımlayan işleme motoru (processing engine)
- Açık kaynak yazılım
- Java, Scala, Python destekler
- Scala ile yazılmıştır.
- SQL, MLlib, streaming, GraphX kütüphaneleri
- Standalone veya cluster'lar (YARN veya Mesos mode) üzerinde çalışabilir.
- Anahtar yapısı: Esnek Dağıtılmış Veri Kümesi (Resilient Distributed Dataset RDD)





Esnek Dağıtılmış Veri Kümesi-RDD

- Spark'ın birincil soyutlaması
- Elementlerin dağıtık topluluğu (dist. collection of elements)
- Cluster'da paralleştirilmiş
- RDD üzerinde iki tip operasyon vardır:
  - Transformasyonlar-geriye değer donusu yok, transforme edilen RDD icin pointer doner (map, filter, groupBy, FlatMap gibi) RDD->RDD
  - Aksiyonlar (actions)-geriye değer donusu var (count, collect, reduce, top(N), saveAsTextFile gibi) RDD->value
- Hataya toleranslı
- Caching
- Sabit, değiştirilmez (immutable)



Dönüşümler ve Aksiyonlar

Transformations	$map(f : \mathrm{T} \Rightarrow \mathrm{U})$	: $\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$filter(f : \mathrm{T} \Rightarrow \mathrm{Bool})$	: $\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$
	$flatMap(f : \mathrm{T} \Rightarrow \mathrm{Seq[U]})$	: $\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[U]}$
	$sample(fraction : \mathrm{Float})$	: $\mathrm{RDD[T]} \Rightarrow \mathrm{RDD[T]}$ (Deterministic sampling)
	$groupByKey()$	: $\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, Seq[V])]}$
	$reduceByKey(f : (\mathrm{V},\mathrm{V}) \Rightarrow \mathrm{V})$	: $\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$union()$	: $(\mathrm{RDD[T]}, \mathrm{RDD[T]}) \Rightarrow \mathrm{RDD[T]}$
	$join()$	: $(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (V, W))]}$
	$cogroup()$	: $(\mathrm{RDD[(K, V)]}, \mathrm{RDD[(K, W)]}) \Rightarrow \mathrm{RDD[(K, (Seq[V], Seq[W]))]}$
	$crossProduct()$	: $(\mathrm{RDD[T]}, \mathrm{RDD[U]}) \Rightarrow \mathrm{RDD[(T, U)]}$
	$mapValues(f : \mathrm{V} \Rightarrow \mathrm{W})$	: $\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, W)]}$ (Preserves partitioning)
	$sort(c : \mathrm{Comparator[K]})$	: $\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
	$partitionBy(p : \mathrm{Partitioner[K]})$	: $\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{RDD[(K, V)]}$
Actions	$count()$	: $\mathrm{RDD[T]} \Rightarrow \mathrm{Long}$
	$collect()$	: $\mathrm{RDD[T]} \Rightarrow \mathrm{Seq[T]}$
	$reduce(f : (\mathrm{T},\mathrm{T}) \Rightarrow \mathrm{T})$	: $\mathrm{RDD[T]} \Rightarrow \mathrm{T}$
	$lookup(k : \mathrm{K})$	: $\mathrm{RDD[(K, V)]} \Rightarrow \mathrm{Seq[V]}$ (On hash/range partitioned RDDs)
	$save(path : \mathrm{String})$	: Outputs RDD to a storage system, *e.g.*, HDFS

Table 2: Transformations and actions available on RDDs in Spark. Seq[T] denotes a sequence of elements of type T.

Esnek Dağıtılmış Veri Kümesi-RDD 2

- RDD olusturmanın uc yolu vardır:
  - Driver program icinde var olan koleksiyonu paralel hale getirme
    - val data = 1 to 10000
    - val distData = sc.parallize(data)
  - Harici depolama biriminde olan bir verisetini referanslama
    - val readmeFile = sc.TextFile(“Readme.md”)
  - Var olan RDD'den transformasyon yolu ile
    - distData.filter(...)
- Veriseti Hadoop tarafından desteklenen herhangi bir depolama olabilir:
  - Lokal dosya sistemi, HDFS, Cassandra, HBase, Amazon S3 vb.
- Desteklenen dosya tipleri
  - Metin (text) dosyaları, SequenceFiles, Hadoop InputFormat

RDD operasyoları Temeller (Scala)

- “sc” is a “Spark context” – textFile transforms the file into an RDD
  - val textFile = sc.textFile("README.md")
- Return number of items (lines) in this RDD; count() is an action
  - textFile.count()
- Demo filtering. Filter is a tranform.
  - val linesWithSpark = textFile.filter(line => line.contains("Spark"))
- Demo chaining – how many lines contain “Spark”? count() is an action.
  - textFile.filter(line => line.contains("Spark")).count()
- Length of line with most words. Reduce is an action.
  - textFile.map(line => line.split(" ").size).reduce((a, b) => if (a > b) a else b)
- Word count – traditional map-reduce. collect() is an action
  - val wordCounts = textFile.flatMap(line => line.split(" "))

    .map(word => (word, 1))

    .reduceByKey((a, b) => a + b)
  - wordCounts.collect()

RDD Sürekliliği(persistence) veya depolanması

- RDD persistence icin iki metot var:
  - persist()
  - cache() → sadece MEMORY_ONLY ile sağlar
    - Serialization (marshalling)- Turn data into a stream of bytes that can be stored
    - deserialization - Turn a stream of bytes back into a copy of the original object.

Storage Level	Meaning
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER	Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as the levels above, but replicate each partition on two cluster nodes.
OFF_HEAP (experimental)	Store RDD in serialized format in Tachyon. Compared to MEMORY_ONLY_SER, OFF_HEAP reduces garbage collection overhead and allows executors to be smaller and to share a pool of memory, making it attractive in environments with large heaps or multiple concurrent applications. Furthermore, as the RDDs reside in Tachyon, the crash of an executor does not lead to losing the in-memory cache. In this mode, the memory in Tachyon is discardable. Thus, Tachyon does not attempt to reconstruct a block that it evicts from memory.

Spark Kurulumu

- https://www.tutorialspoint.com/apache_spark/apache_spark_installation.htm
- http://spark.apache.org/docs/latest/spark-standalone.html
-
- Scala IDE for Eclipse
- http://scala-ide.org/download/sdk.html

Spark Uygulaması Geliştirme

- SparkContext, spark cluster'a bağlanma şeklini temsil eder.
  - val sc = new SparkContext("local", "Simple App")
  - val sc = new SparkContext("spark://yourhostname:7077", "Simple App")
  - local[*] uses as many threads as there are cores.

SPARK master parameter

Master URL	Meaning
`local`	Run Spark locally with one worker thread (i.e. no parallelism at all).
`local[K]`	Run Spark locally with K worker threads (ideally, set this to the number of cores on your machine).
`spark://HOST:PORT`	Connect to the given Spark standalone cluster master. The port must be whichever one your master is configured to use, which is 7077 by default.
`mesos://HOST:PORT`	Connect to the given Mesos cluster. The port must be whichever one your is configured to use, which is 5050 by default. Or, for a Mesos cluster using ZooKeeper, use mesos://zk://....
`yarn-cluster`	Connect to a YARN cluster in cluster mode. The cluster location will be found based on HADOOP_CONF_DIR.

Spark Uygulaması Geliştirme-2

- Spark applications requires certain dependencies.
- Must have a compatible Scala version to write applications.
  - e.g Spark 1.1.1 uses Scala 2.10.
- To write a Spark application, you need to add a Maven dependency on Spark.
  - Spark is available through Maven Central at:

```
groupId = org.apache.spark
artifactId = spark-core_2.10
version = 1.1.1
```

- To access a HDFS cluster, you need to add a dependency on *hadoop-client* for your version of HDFS

```
groupId = org.apache.hadoop
artifactId = hadoop-client
version = <your-hdfs-version>
```

Spark Örneklerinin Shell den Çalıştırılması

- Spark yüklenince beraberinde bazı örnekler gelir.
  - Scala, Java, Python ve R örnekleri examples/src/main dizininde bulunur.
- Java veye Scala örnekleri bin/run-example <class> [params]
  - run-example SparkPi 10
- Python örnekleri bin/spark-submit <py file> [params]
  - spark-submit /usr/local/spark/examples/src/main/python/pi.py
- R örnekleri bin/spark-submit <class> [params]
  - spark-submit examples/src/main/r/dataframe.R

  ile çalıştırılır.
- http://spark.apache.org/docs/latest/programming-guide.html

Demo 1

- Java ile worcount
- Scala ile wordcount

Spark Kütüphaneleri-Apache SQL

- SQL, HiveQL, Scala'da ifade edilen ilişkisel sorguların yapılmasını sağlar.
- Yapılı ve yarı yapılı veriler için SchemaRDD veri soyutlamasını kullanır.
- SchemaRDD; var olan RDD'lerden, JSON, Hive, Cassandra gibi veritabanlarından olusturulabilir.

1. At the command-line, copy the Hue sample_07 data to HDFS:

```
$ hdfs dfs -put HUE_HOME/apps/beeswax/data/sample_07.csv /user/hdfs
```

   where *HUE_HOME* defaults to `/opt/cloudera/parcels/CDH/lib/hue` (parcel installation) or `/usr/lib/hue` (package installation).

2. Start `spark-shell`:

```
$ spark-shell
```

3. Create a Hive table:

```
scala> sqlContext.sql("CREATE TABLE sample_07 (code string,description string,total_emp int,sala
```

4. Load data from HDFS into the table:

```
scala> sqlContext.sql("LOAD DATA INPATH '/user/hdfs/sample_07.csv' OVERWRITE INTO TABLE sample_0
```

5. Create a DataFrame containing the contents of the sample_07 table:

```
scala> val df = sqlContext.sql("SELECT * from sample_07")
```

6. Show all rows with salary greater than 150,000:

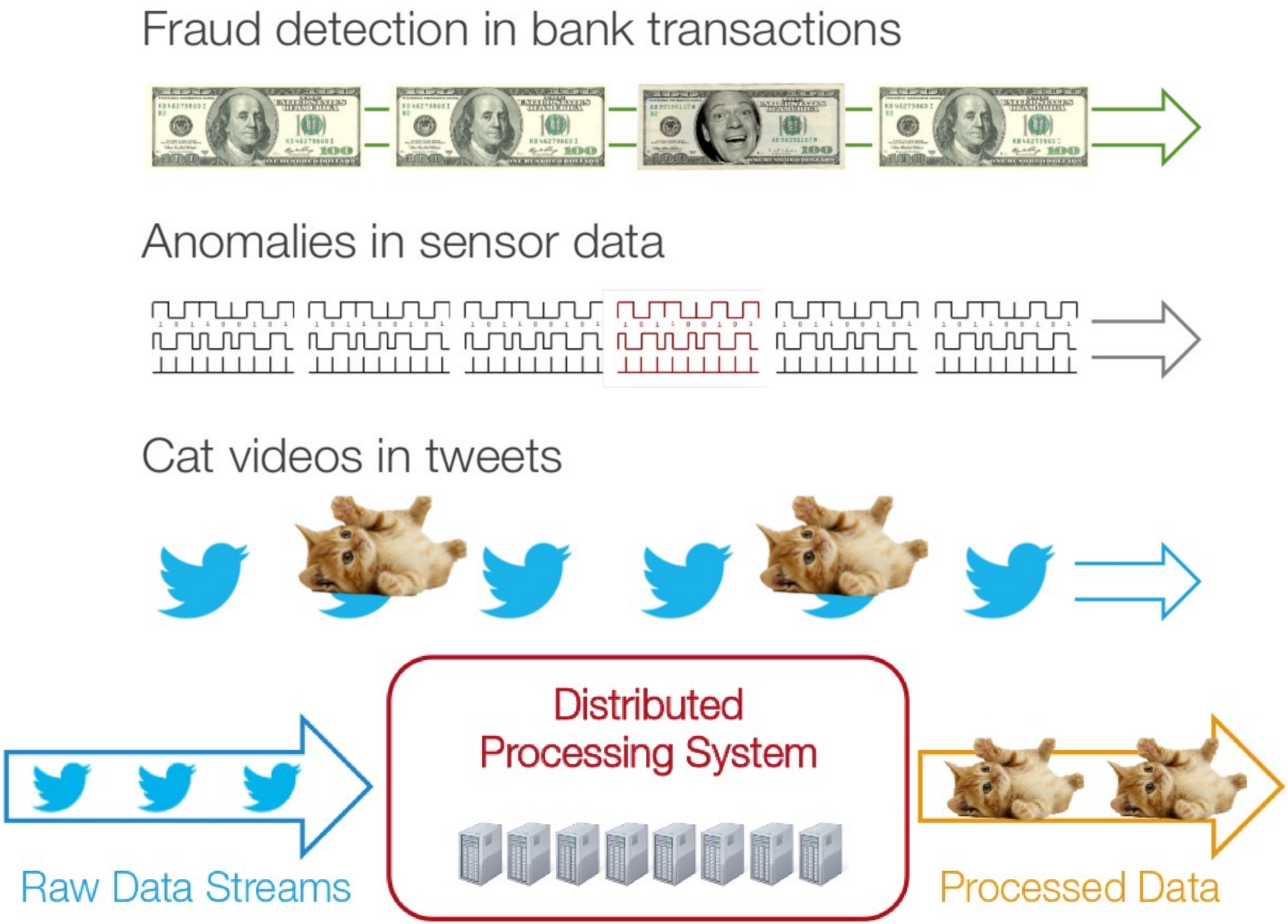
```
scala> df.filter(df("salary") > 150000).show()
```

   The output should be:

```
+-------+--------------------+---------+------+
|   code|         description|total_emp|salary|
+-------+--------------------+---------+------+
|11-1011|    Chief executives|   299160|151370|
|29-1022|Oral and maxillof...|     5040|178440|
|29-1023|       Orthodontists|     5350|185340|
|29-1024|     Prosthodontists|      380|169360|
|29-1061|   Anesthesiologists|    31030|192780|
|29-1062|Family and genera...|   113250|153640|
|29-1063| Internists, general|    46260|167270|
|29-1064|Obstetricians and...|    21340|183600|
|29-1067|            Surgeons|    50260|191410|
|29-1069|Physicians and su...|   237400|155150|
+-------+--------------------+---------+------+
```

Demo: run-example sql.JavaSparkSQL

Spark Kütüphaneleri-Streaming



Spark Kütüphaneleri-Streaming-2

- Batch processing ve stream processing
- Akan (streaming/continuous group of data records) veri: log files, transactions, sensors, server trafiği, online searches vs
- DStreams API kullanır.
- İnput'tan (Kafka, HBase, Flume, Twitter, HDFS vs) gelen veri microbatch'lere ayrılar (pre-defined interval (N seconds)), RDD dizisi gibi dusunulebilir.
- Her microbatch'e RDD operasyonları uygulanır.
- Sonuç HDFS, veritabanlarına vs yazılabilir.

Spark Kütüphaneleri-Streaming-3

- Some of the most interesting use cases of Spark Streaming include the following:
  - Uber, the company behind ride sharing service, uses Spark Streaming in their continuous Streaming ETL pipeline to collect terabytes of event data every day from their mobile users for real-time telemetry analytics.
  - Pinterest, the company behind the visual bookmarking tool, uses Spark Streaming, MemSQL and Apache Kafka technologies to provide insight into how their users are engaging with Pins across the globe in real-time.
  - Netflix uses Kafka and Spark Streaming to build a real-time online movie recommendation and data monitoring solution that processes billions of events received per day from different data sources.

- Other real world examples of Spark Streaming include:
  - Supply chain analytics
  - Real-time security intelligence operations to find threats
  - Ad auction platform
  - Real-time video analytics to help with personalized, interactive experiences to the viewers

Spark Kütüphaneleri-Streaming-4

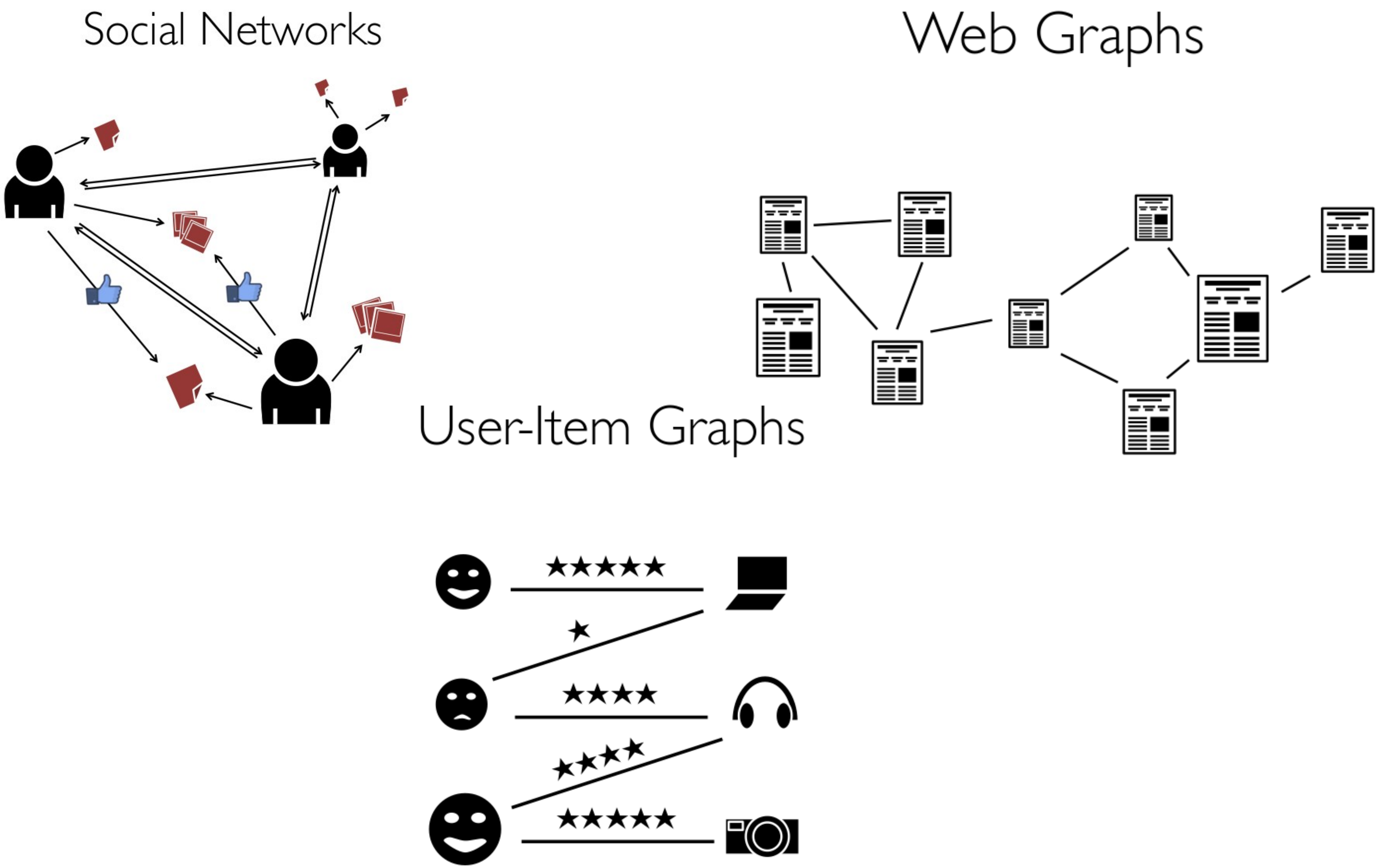
- Dstreams + RDDs
- MLlib, GraphX ile beraber kullanılabilir.
  - Offline learning, online prediction
  - Online learning and prediction

```
// Learn model offline
val model = KMeans.train(dataset, ...)
// Apply model online on stream
val kafkaStream = KafkaUtils.createDStream(...)
kafkaStream.map { event =>
model.predict(featurize(event)) }
```

Demo 2

- NetworkWordCount orneği
- Lokal sistemde haberleşme için bir port aç
  - nc -lk 9999
- Apache spark consol'dan (port 9999) text verisini bir stream data olarak alabilir.
  - run-example streaming.NetworkWordCount localhost 9999

Spark Kütüphaneleri-GraphX



Graf-Paralel Algoritmalar

Collaborative Filtering

- » Alternating Least Squares
- » Stochastic Gradient Descent
- » Tensor Factorization

Structured Prediction

- » Loopy Belief Propagation
- » Max-Product Linear Programs
- » Gibbs Sampling

Semi-supervised ML

- » Graph SSL
- » CoEM

Community Detection

- » Triangle-Counting
- » K-core Decomposition
- » K-Truss

Graph Analytics

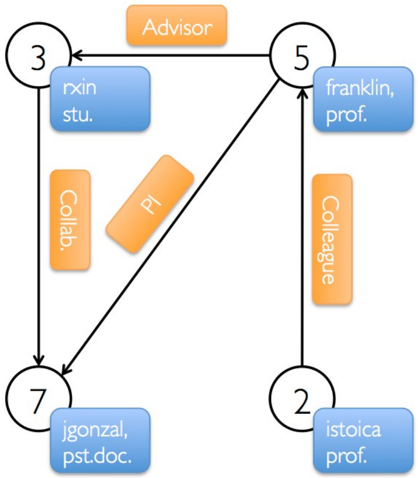
- » PageRank
- » Personalized PageRank
- » Shortest Path
- » Graph Coloring

Classification

- » Neural Networks

Graf Oluşturma ve Bazı Operatörler

Property Graph



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

```
/** Summary of the functionality in the property graph */
class Graph[VD, ED] {
  // Information about the Graph ====================
  val numEdges: Long
  val numVertices: Long
  val inDegrees: VertexRDD[Int]
  val outDegrees: VertexRDD[Int]
  val degrees: VertexRDD[Int]
  // Views of the graph as collections =============
  val vertices: VertexRDD[VD]
  val edges: EdgeRDD[ED]
  val triplets: RDD[EdgeTriplet[VD, ED]]
  ...
  ...
```

```
// Assume the SparkContext has already been constructed
val sc: SparkContext
// Create an RDD for the vertices
val users: RDD[(VertexId, (String, String))] =
  sc.parallelize(Array((3L, ("rxin", "student")), (7L, ("jgonzal", "postdoc")),
                       (5L, ("franklin", "prof")), (2L, ("istoica", "prof"))))
// Create an RDD for edges
val relationships: RDD[Edge[String]] =
  sc.parallelize(Array(Edge(3L, 7L, "collab"),    Edge(5L, 3L, "advisor"),
                       Edge(2L, 5L, "colleague"), Edge(5L, 7L, "pi")))
// Define a default user in case there are relationship with missing user
val defaultUser = ("John Doe", "Missing")
// Build the initial Graph
val graph = Graph(users, relationships, defaultUser)
// Count all users which are postdocs
graph.vertices.filter { case (id, (name, pos)) => pos == "postdoc" }.count
// Count all the edges where src > dst
graph.edges.filter(e => e.srcId > e.dstId).count
```

Big Learning Kütüphaneleri

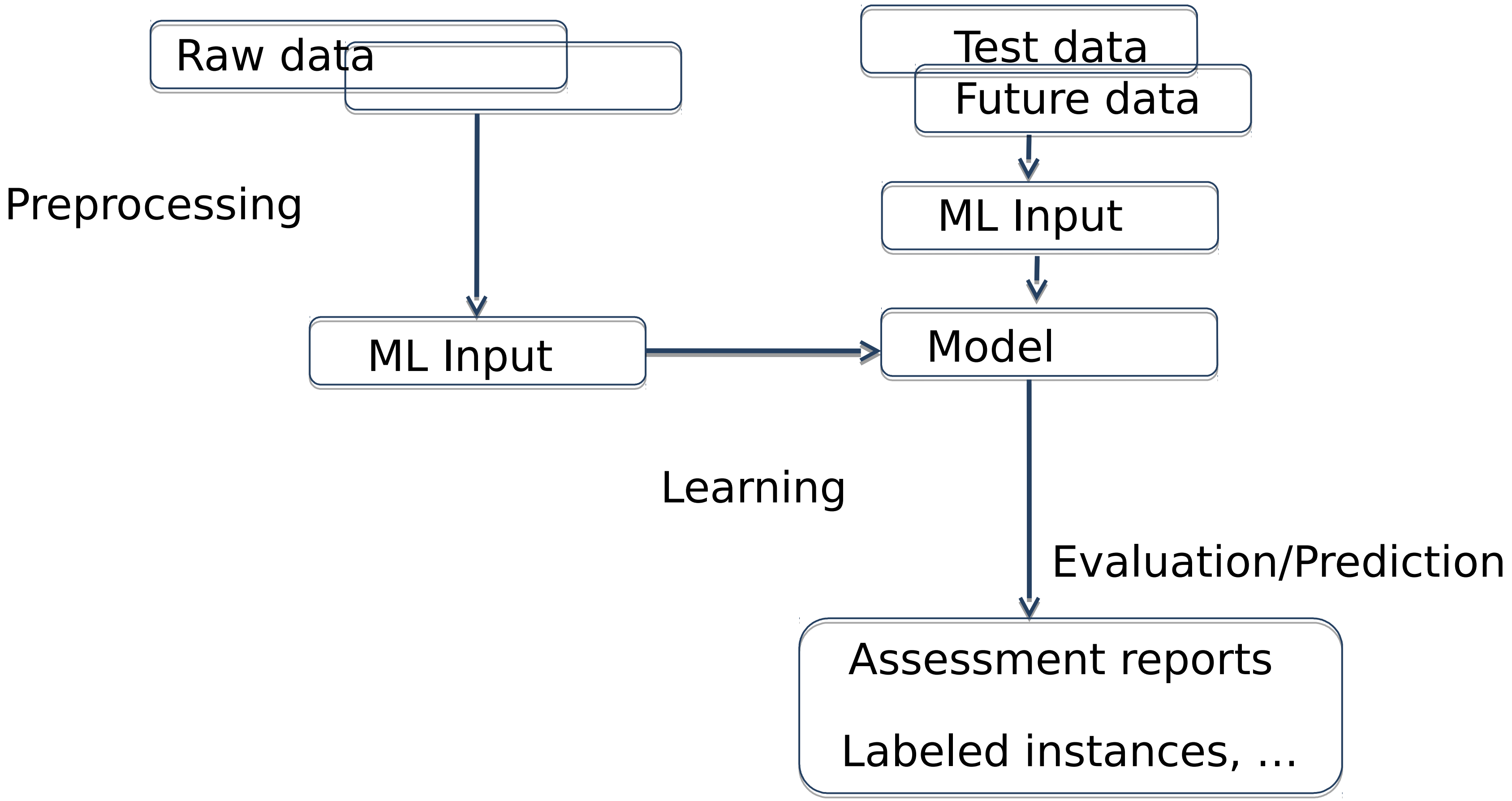
- Apache Mahout:
  - M/R based API is deprecated, more of a matrix computations library now, turns your math expressions into tasks running on Spark, Flink, H2O
- Apache Spark ML Package:
  - Learning library with easy-to-build pipelines: Algorithms for classification, regression, clustering, mixed membership, matrix factorization, etc. with preprocessing, evaluation and prediction capabilities
- Apache SystemML:
  - Part of Apache-Incubator, users write scripts in an R-like declarative ML DSL (similar to new Mahout), running on Spark is automatically handled by the framework. Several algorithms written in the DML readily available

Spark ML Paketi

- Running on distributed collections with schema (DataFrames, or recently Datasets), Spark ML Package includes:
- Preprocessing Transformations:
  - Binarizer/Bucketizer, TF-IDF Representation, Polynomial Expansion, etc.
- Learning Algorithms (Estimators):Logistic Regression, Naïve Bayes, Decision Trees and their Ensembles, …
  - Linear Regression, Regression Tree, …
  - K-means, Gaussian Mixtures, …
  - Latent Dirichlet Allocation
  - Matrix Factorization

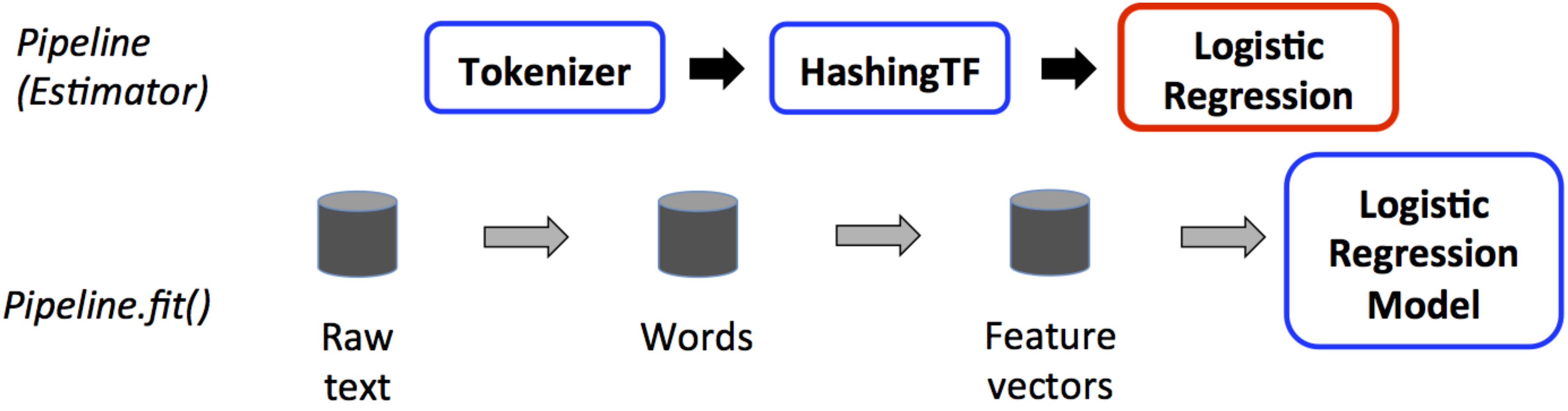
    with several distributed optimization/inference techniques
- Evaluators
- Predictors
- SparkSQL, Streaming ve GraphX ile beraber kullanılabilir

Spark ML Pipelines



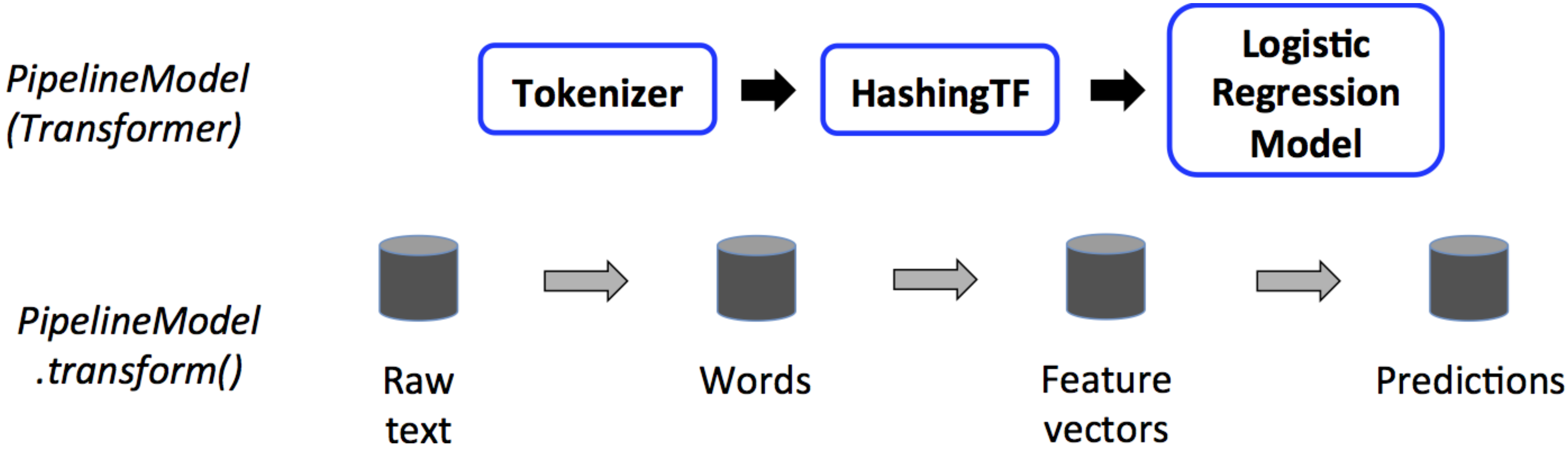
Spark ML Pipelines-2

- Eğitim aşamasında



Spark ML Pipelines-3

- Test aşamasında



Lineer Regresyon

```
val lir = new LinearRegression()
  .setFeaturesCol("features")
  .setLabelCol("label")
  .setRegParam(lambda)
  .setMaxIter(maxIter)

// Train the model, df is the input DataFrame
val lirModel = lir.fit(df)

// Test on new data – needs a “features” column
// Predictions on “prediction” column by default
val predictions = lirModel.transform(testDf)
```

Kümeleme: K-Means

```
val kmeans = new KMeans()
  .setK(20)
  .setSeed(1L)

//holds cluster centers
val cmodel = kmeans.fit(training)

//assigns cluster labels
val predictions = cmodel.transform(test)
```

Öneri Sistemi: Matrix Factorization

```
// Will run on a frame with “user”, “item”, “rating”
columns
val als = new ALS()
  .setRank(20)
  .setMaxIter(100)
  .setRegParam(0.01)
  .setUserCol("user”)
  .setItemCol(”item”)
  .setRatingCol("rating”)

// holds 2 low-rank factor matrices
val recModel = als.fit(training)

// makes predictions for user-item pairs
val recommendations = recModel.transform(test)
```

Demo 3

- JavaRandomForestClassifierExample

Spark and Map Reduce Farkları

	Hadoop Map Reduce	Spark
Storage	Disk only	In-memory or on disk
Operations	Map and Reduce	Map, Reduce, Join, Sample, etc…
Execution model	Batch	Batch, interactive, streaming
Programming environments	Java	Scala, Java, R, and Python

In-Memory Büyük fark olusturuyor

- Two iterative Machine Learning algorithms:

Spark Araştırma makaleleri

- Spark: Cluster Computing with Working Sets
  - Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica
  - USENIX HotCloud (2010)
  - people.csail.mit.edu/matei/papers/2010/hotcloud_spark.pdf!
- Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing
  - Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica
  - NSDI (2012)
  - usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf"

Sonuç

- Neler öğrendik?
  - Apache Spark nedir, RDD ve sürekliliği
  - Spark Uygulaması Geliştirme (Scala, Java)
  - Spark için Shell kullanımı
  - Büyük veri işleme modelleri
  - Spark Kütüphaneleri (Spark SQL, MLlib, Streaming, GraphX)
  - Big Learning, Spark ML algoritmaları
  - Derin öğrenme

Referanslar

- N. Marz & J. Warren, “Big Data: Principles and best practices of scalable real-time data systems”, Manning, 2015
- https://spark-summit.org/2014/training/
- https://cs.stanford.edu/~matei/