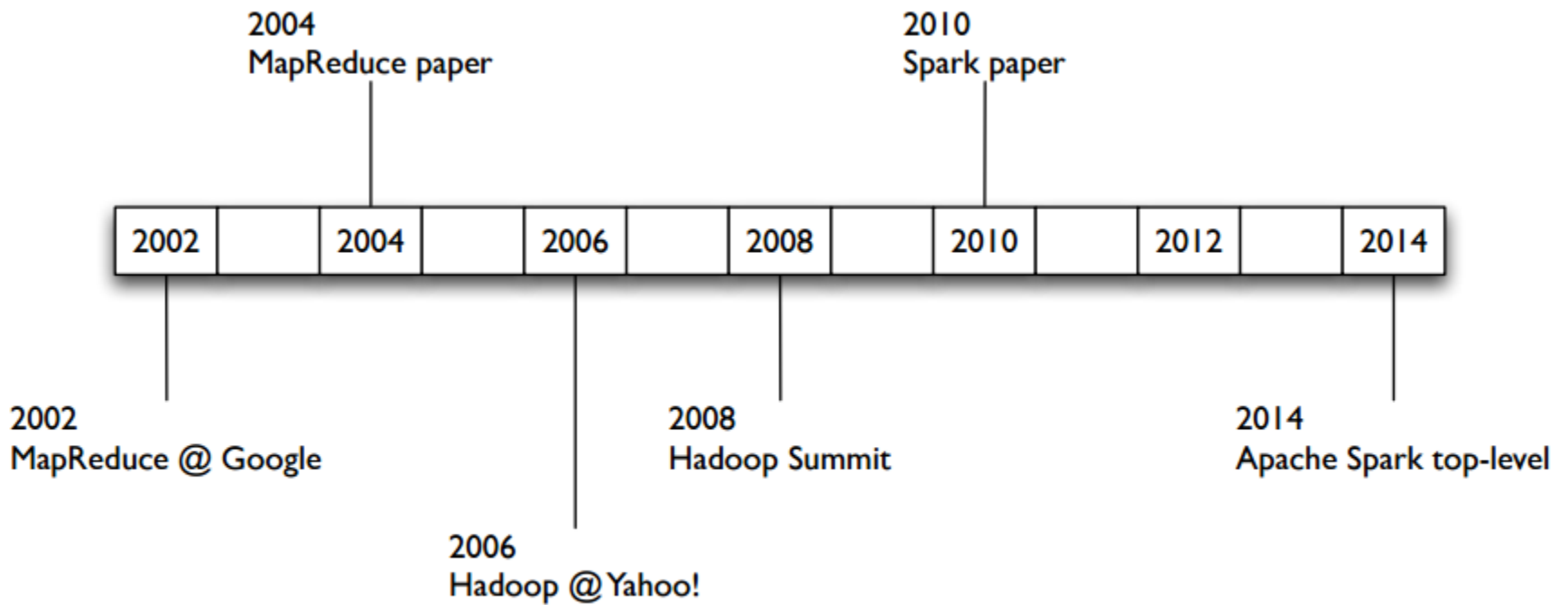


Distributed Computing, Spark

Gombos Gergő

Történelem



Spark Motiváció

- Hadoop MapReduce problémák:
 - Programozás nehézkes
 - Performancia, sok I/O használat
 - Speciális alkalmazásokra nem alkalmas

MapReduce programozás

- Hadoop: 61 sor
- Spark: 1 sor

```
sc.textFile('...').flatMap(lambda x: x.split())  
  .map(lambda x: (x, 1))  
  .reduceByKey(lambda x, y: x+y)  
  .saveAsTextFile('...')
```

```
import java.io.IOException;  
import java.util.StringTokenizer;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.Reducer;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
  
public class WordCount {  
  
    public static class TokenizerMapper  
        extends Mapper<Object, Text, Text, IntWritable>{  
  
        private final static IntWritable one = new IntWritable(1);  
        private Text word = new Text();  
  
        public void map(Object key, Text value, Context context  
            ) throws IOException, InterruptedException {  
            StringTokenizer itr = new StringTokenizer(value.toString());  
            while (itr.hasMoreTokens()) {  
                word.set(itr.nextToken());  
                context.write(word, one);  
            }  
        }  
    }  
  
    public static class IntSumReducer  
        extends Reducer<Text, IntWritable, Text, IntWritable> {  
        private IntWritable result = new IntWritable();  
  
        public void reduce(Text key, Iterable<IntWritable> values,  
            Context context  
            ) throws IOException, InterruptedException {  
  
            int sum = 0;  
            for (IntWritable val : values) {  
                sum += val.get();  
            }  
            result.set(sum);  
            context.write(key, result);  
        }  
    }  
  
    public static void main(String[] args) throws Exception {  
        Configuration conf = new Configuration();  
        Job job = Job.getInstance(conf, "word count");  
        job.setJarByClass(WordCount.class);  
        job.setMapperClass(TokenizerMapper.class);  
        job.setCombinerClass(IntSumReducer.class);  
        job.setReducerClass(IntSumReducer.class);  
        job.setOutputKeyClass(Text.class);  
        job.setOutputValueClass(IntWritable.class);  
        FileInputFormat.addInputPath(job, new Path(args[0]));  
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  
        System.exit(job.waitForCompletion(true) ? 0 : 1);  
    }  
}
```

Performancia

- Hadoop adat kiírása a diskre egy jobban: 3 vagy több
- Hive (SQL tool Hadoopra)
 - Egy tipikus lekérdezés 3-5 MR Job
 - Mindegyik olvassa a HDD-t 3 vagy több alkalommal
 - Mindegyik ír a HDD-re minden olvasás után
 - Összesen 18-30 olvasás egy egyszerű lekérdezésnél

Performancia

- Mit ad a Spark?
 - Lusta kiértékelés
 - Optimalizálja a job-ot mielőtt lefuttatná
 - Memória alapú cache
 - Egyszer olvassa a HDD-t, utána Ram-ot használ
 - Hatékony pipeline
 - Adatok átadása a folyamatok között HDD nélkül

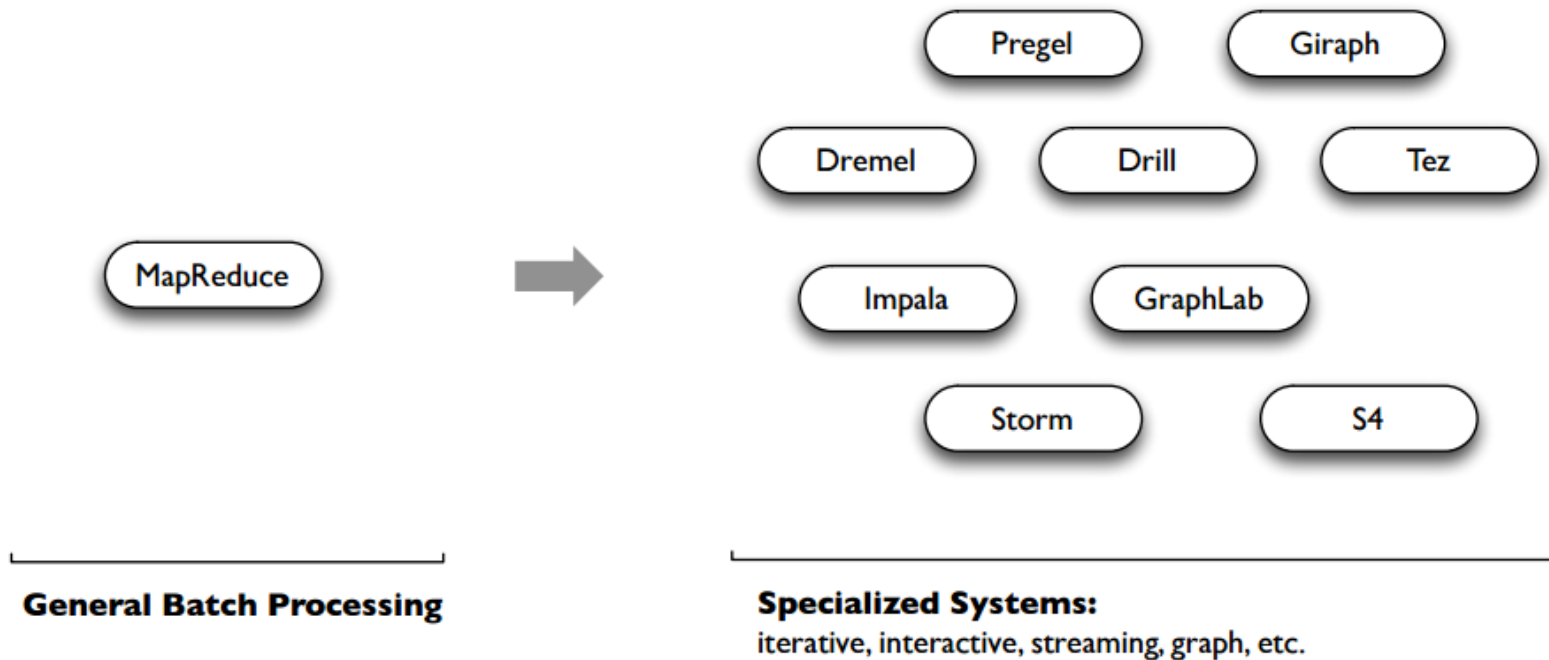
Performancia

Daytona Gray Sort 100TB Benchmark

	Data Size	Time	Nodes	Cores
Hadoop MR (2013)	102.5 TB	72 min	2,100	50,400 physical
Apache Spark (2014)	100 TB	23 min	206	6,592 virtualized

source: <http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html>

Speciális eszközök



Spark alapkövek

- Két fő egység
 - RDD – Resilient Distributed Dataset
 - DAG – Direct Acyclic Graph

RDD

- Egyszerűen:
 - az RDD egy kollekcója az adatpartícióknak, amelyek a worker node-okon vannak tárolva
- Komplexen:
 - egy interface az adat transzformációhoz
 - átmenetileg tárolja az adatokat
 - nem módosítható

RDD Metadata

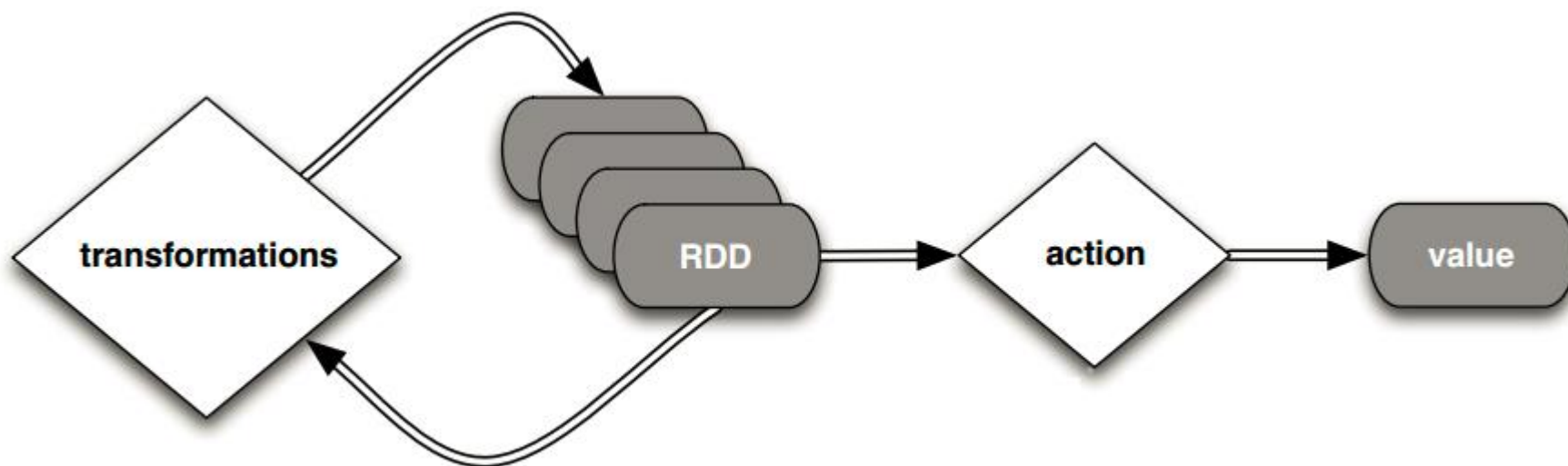
- Metainformációk
 - Partitions – mely adatok kapcsolódnak ehhez az RDD-hez
 - Dependencies – a „szülő” RDD-k listája
 - Compute – a „szülő” RDD-ken végezendő függvény
 - Preferred Locations – hol van a legjobb hely a számítás elvégzésére (data locality)
 - Partitioner – Hogyan vágjuk szét az adatot partíciókba

RDD

- Az egyetlen lehetőség az adatok módosítására Spark-ban
- Két osztály:
 - Transformations
 - Actions

RDD – Lusta kiértékelés

- A transformációk, csak a metadata-t fogják változtatni



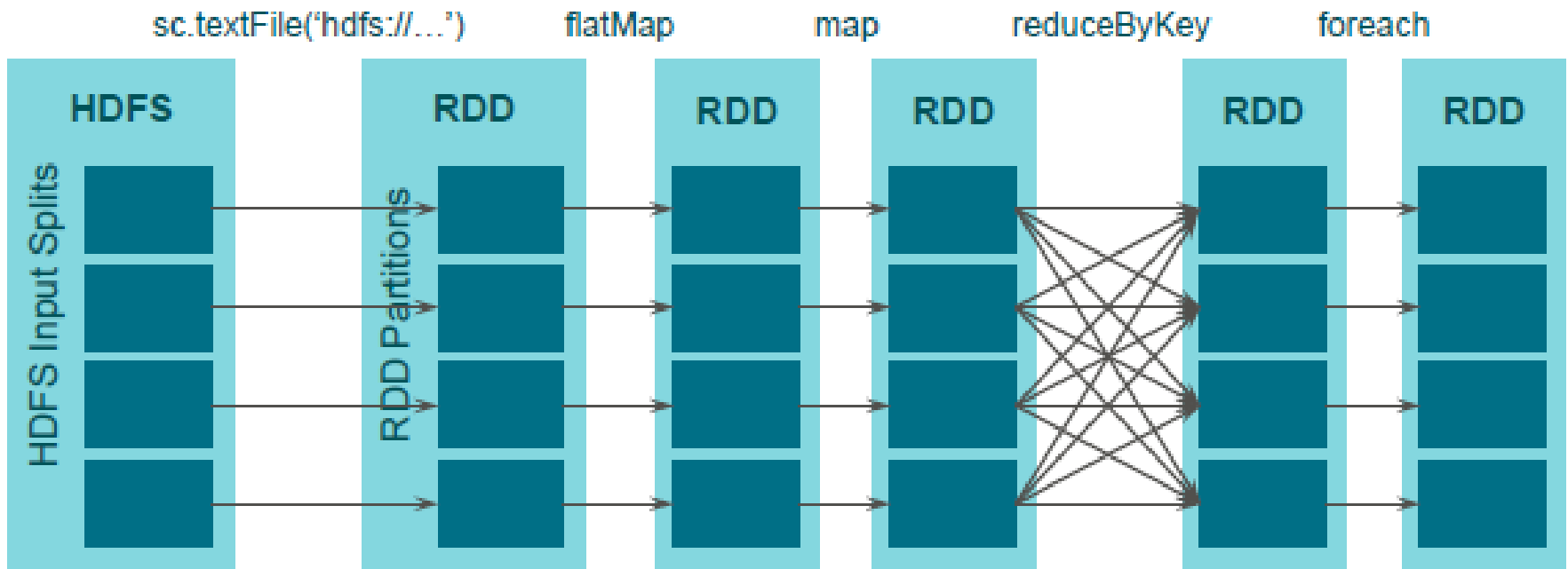
DAG

- Direct Acyclic Graph – az adaton végezendő számítások sorozata
 - Csúcs – RDD partíció
 - Él – átalakítás az adaton
 - Aciklikus – nem térünk vissza régebbi partícióhoz
 - Irányított – Az adat partíciók átvitele egyik állapotból a másikba

WordCount example

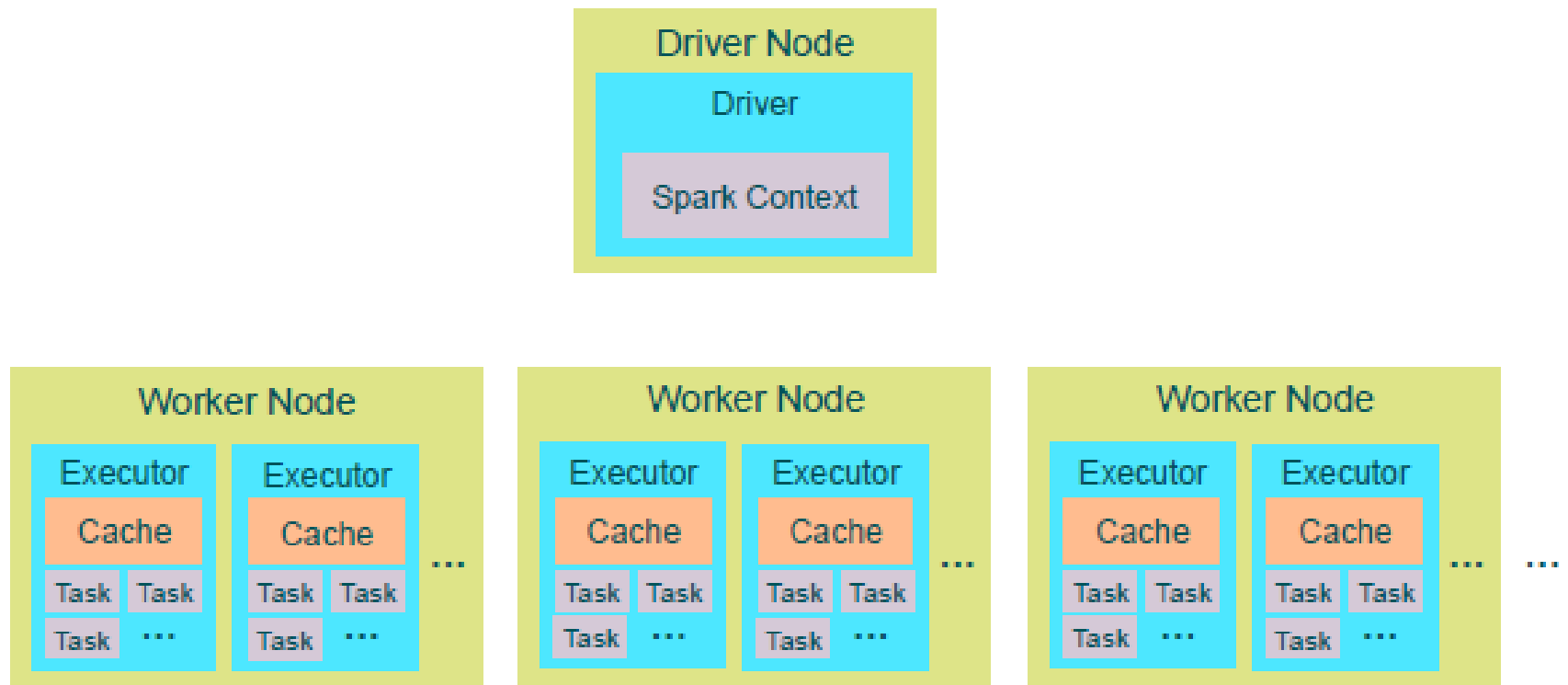
```
def printfunc (x):  
    print 'Word "%s" occurs %d times' % (x[0], x[1])  
  
infile = sc.textFile('hdfs://sparkdemo:8020/sparkdemo/textfiles/README.md', 4)  
rdd1 = infile.flatMap(lambda x: x.split())  
rdd2 = rdd1.map(lambda x: (x, 1)).reduceByKey(lambda x, y: x+y)  
print rdd2.toDebugString()  
rdd2.foreach(printfunc)
```

WordCount example



Spark Architektúra

- Cluster



Spark Cluster

- Driver
 - Spark Shell elérés (Scala, Python, R)
 - A hely ahol elkészül a SparkContext
 - Az RDD-t átalakítja egy lekérdezési gráffá
 - Stage-ekbe pakolja a lépéseket
 - Ütemezi a taskokat és irányítja a futásukat
 - Tárolja a metadata-t az RDD-kről
 - WebUI-t futtat

Spark Cluster

- Executor
 - Tárolja az adatot a cache-ben (JVM Heap, HDD)
 - Olvassa az adatot a külső forrásról
 - Írja az adatot külső forrásra
 - Elvégzi az adat feldolgozásokat

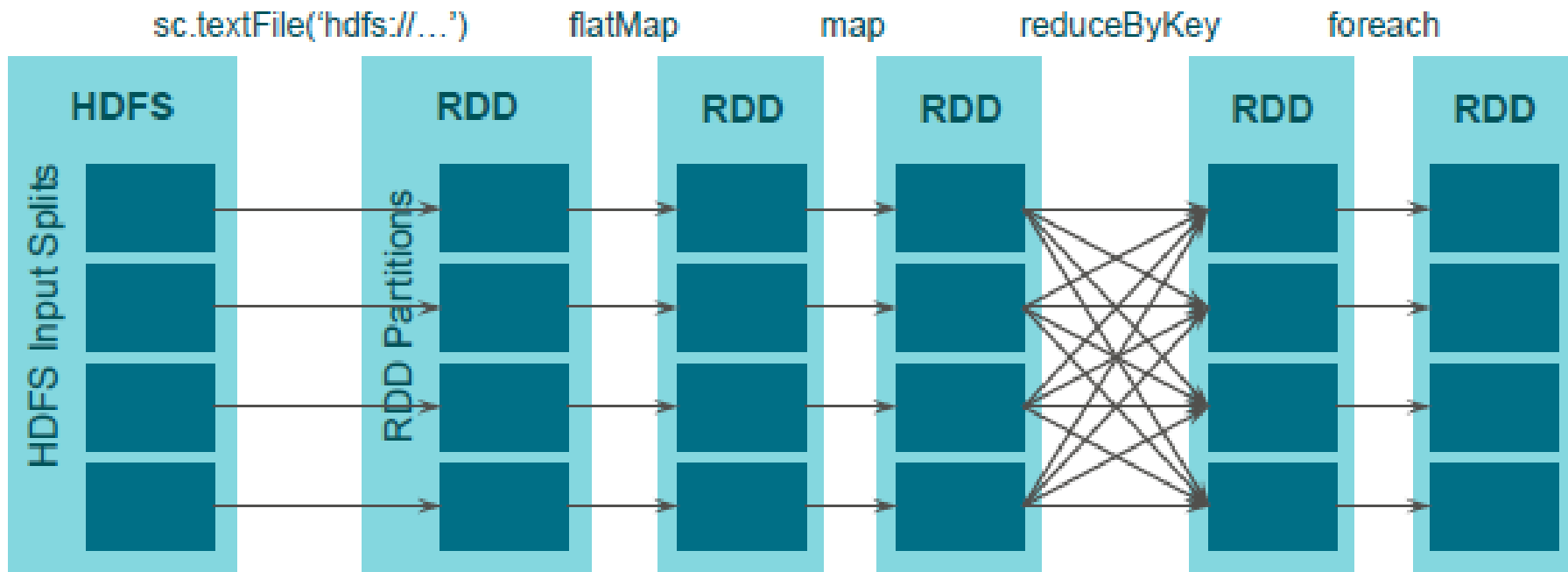
Alkalmazás dekompozíció

- Alkalmazás
 - Egy SparkContext ami tárol adatot a feldolgozáshoz, és ütemezi a jobok sorozatát, párhuzamos is lehet
- Job
 - RDD transzformációk sorozata, amely egy action-nel vagy adat kiírással végződik, driver alkalmazás vezérli

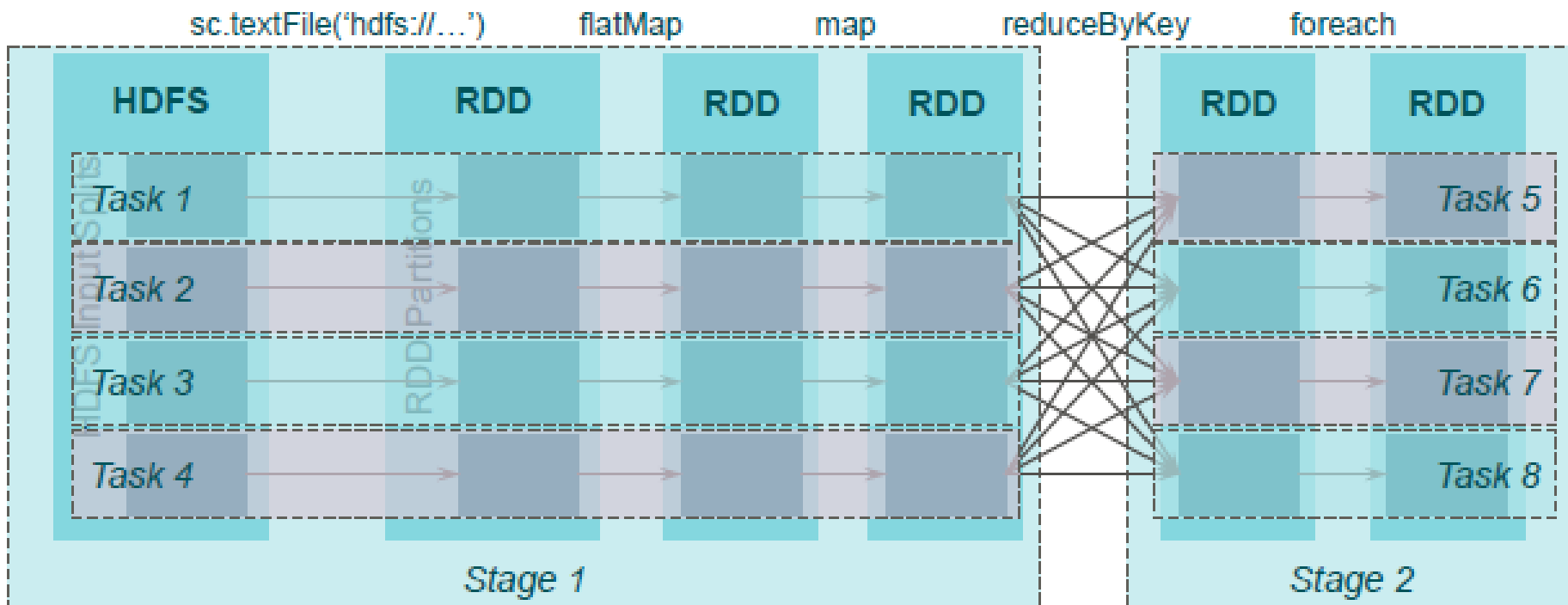
Alkalmazás dekompozíció

- Stage
 - Transzformációk egy olyan sorozata, amelyet egy független worker el tud végezni.
- Task
 - Egy stage futtatása egy adat partíción.

WordCount példa



WordCount példa



Spark perzisztencia

Persistence Level	Leírás
MEMORY_ONLY	Az RDD deszerializált JAVA objektumként tároljuk. Ha nem fér el a memóriában újraszámoljuk.
MEMORY_AND_DISK	Az RDD deszerializált JAVA objektumként tároljuk. Ha nem fér el a memóriában tároljuk a lemezen.
MEMORY_ONLY_SER	Ua. mint MEMORY_ONLY, csak szerializálva tároljuk a memóriában.
MEMORY_AND_DISK_SER	Ua. mint MEMORY_AND_DISK, csak szerializálva tároljuk a memóriában.
DISK_ONLY	Az adatokat a lemezen tároljuk.
MEMORY_ONLY_2, DISK_ONLY_2	Ua. mint feljebb, csak replikáljuk a partíciókat a klaszterben.

Spark eszközök

Spark SQL

Spark
Streaming

MLib
(machine
learning)

GraphX
(graph)

Apache Spark Core

Shark (SQL Api)

- Alapja a DataFrame-k
- DataFrame:
 - Elosztott adatset
 - Nevesített oszlopokkal rendelkezik
- Forrás lehet:
 - Hive
 - Külső adatbázis
 - Meglévő RDDs
 - File

Shark (SQL Api)

```
val sc: SparkContext // An existing SparkContext.  
val sqlContext = new org.apache.spark.sql.SQLContext(sc) // Create the DataFrame  
val df = sqlContext.jsonFile("examples/src/main/resources/people.json")  
  
// Show the content of the DataFrame  
df.show()  
// age name  
// null Michael  
// 30 Andy // 19 Justin  
  
// Print the schema in a tree format  
df.printSchema()  
// root  
// |-- age: long (nullable = true)  
// |-- name: string (nullable = true)
```

Shark (SQL Api)

// Select only the "name" column

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

// name

// Michael

// Andy

// Justin

// Select everybody, but increment the age by 1

```
df.select(df("name"), df("age") + 1).show()
```

// name (age + 1)

// Michael null

// Andy 31

// Justin 20

Shark (SQL Api)

```
// Select people older than 21  
df.filter(df("age") > 21).show()  
// age name  
// 30 Andy
```

```
// Count people by age df.groupBy("age").count().show()  
// age count  
// null 1  
// 19 1  
// 30 1
```

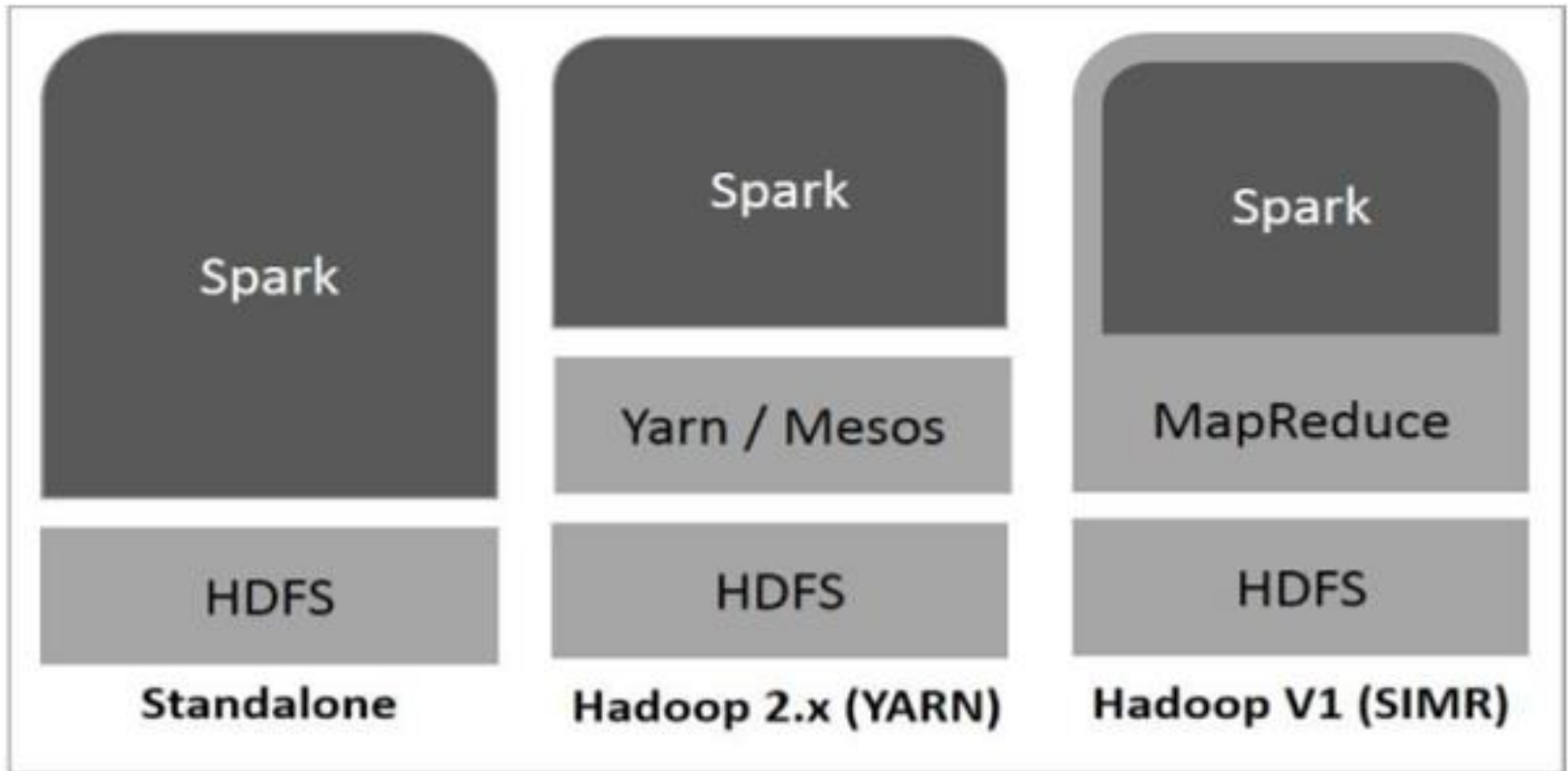
Shark (SQL Api)

SQL Futtatás

```
val sqlContext = ... // An existing SQLContext
```

```
val df = sqlContext.sql("SELECT * FROM table")
```

Futatósi környezetek



Transformations and Actions

Transformations

`map (func)`
`flatMap(func)`
`filter(func)`
`groupByKey()`
`reduceByKey(func)`
`mapValues(func)`
`sample(...)`
`union(other)`
`distinct()`
`sortByKey()`
...

Actions

`reduce(func)`
`collect()`
`count()`
`first()`
`take(n)`
`saveAsTextFile(path)`
`countByKey()`
`foreach(func)`
...

Scala Tutorial

- Lásd [ScalaTutorial.html](#)

Példák

```
scala> var ex = sc.parallelize(List("szoveg 1", "szoveg 2", "szoveg bla bla"))
ex: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[0] at parallelize at
<console>:21
```

```
scala> val lines = sc.textFile("c:/Users/bigdata/valami.txt")
```

```
scala> val pair = (a, b)
```

```
pair._1 // => a
```

```
pair._2 // => b
```

```
scala> object HelloWorld {
  |   def main(args: Array[String]) {
  |     println("Hello, world!")
  |   }
  | }
```

```
defined module HelloWorld
```

Példák

```
scala> HelloWorld.main(null)
```

```
Hello, world!
```

```
scala> ex.map(line => (line, 1))
```

```
res0: org.apache.spark.rdd.RDD[(String, Int)] =  
MapPartitionsRDD[1] at map at <console>:24
```

```
scala> res0.foreach(println)
```

```
[Stage 0:>
```

```
(0 + 0) / 4]
```

```
(szoveg 1,1)
```

```
(szoveg bla bla,1)
```

```
(szoveg 2,1)
```

Példák

```
scala> ex.map(line => line.split(" "))
```

```
res4: org.apache.spark.rdd.RDD[Array[String]] =  
MapPartitionsRDD[2] at map at <console>:24
```

```
scala> res4.foreach(println)
```

```
[Ljava.lang.String;@3cc018c2  
[Ljava.lang.String;@4c3da8ce  
[Ljava.lang.String;@31a73053
```

```
scala> res4.flatMap(a => a)
```

```
res8: org.apache.spark.rdd.RDD[String] =  
MapPartitionsRDD[3] at flatMap at <console>:26
```

Példák

```
scala> res8.foreach(println)
```

```
[Stage 2:> (0 + 0) / 4]
```

```
szoveg
```

```
1
```

```
szoveg
```

```
2
```

```
szoveg
```

```
bla
```

```
bla
```

Példák

```
scala> res8.filter(l => l.contains("ove"))  
res10: org.apache.spark.rdd.RDD[String] =  
MapPartitionsRDD[4] at filter at <console>:28
```

```
scala> res10.foreach(println)
```

```
szoveg
```

```
szoveg
```

```
szoveg
```

Példák

```
scala> res8.map(w => (w,1)).reduceByKey(_ + _)
res12: org.apache.spark.rdd.RDD[(String, Int)] =
ShuffledRDD[6] at reduceByKey at <console>:28
```

```
scala> res12.foreach(println)
```

```
(1,1)
```

```
(2,1)
```

```
(bla,2)
```

```
(szoveg,3)
```

Példák

```
scala> res8.map(w => (w,1)).groupByKey().map(w  
=> (w._1, w._2.reduce(_+_)))
```

```
res17: org.apache.spark.rdd.RDD[(String, Int)] =  
MapPartitionsRDD[11] at map at <console>:28
```

```
scala> res17.foreach(println)
```

```
(1,1)
```

```
(bla,2)
```

```
(2,1)
```

```
(szoveg,3)
```


Köszönöm a figyelmet!

- <http://www.slideshare.net/AGrishchenko/apache-spark-architecture>