An Introduction to SparkR

The Apache Spark Framework

Roland Boubela

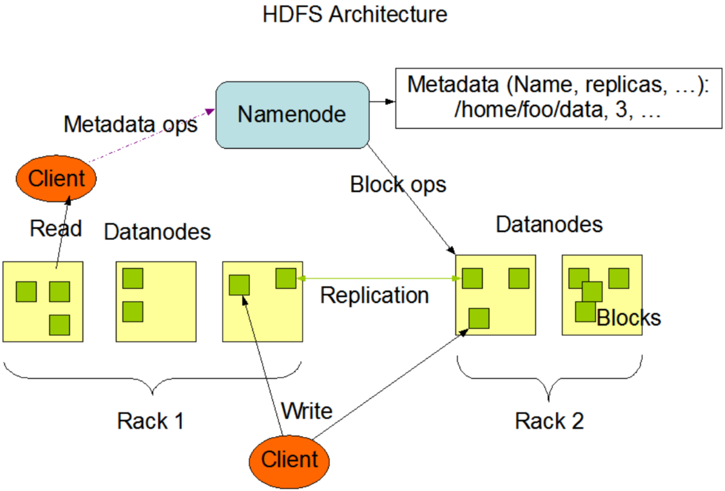
Center for Medical Physics and Biomedical Engineering
Medical University of Vienna

26. Jan. 2016

Hadoop Distributed Filesystem (HDFS)

- Based on Google filesystem
- Fault tolerant (distributed blocks, multiple replicas)
- One client can write, multiple clients read
- Only sequential r/w-operations (no random access)

Hadoop Distributed Filesystem (HDFS)

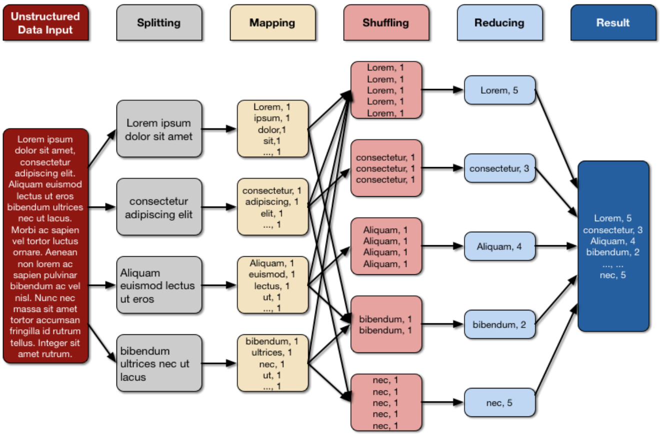


MapReduce Paradigm

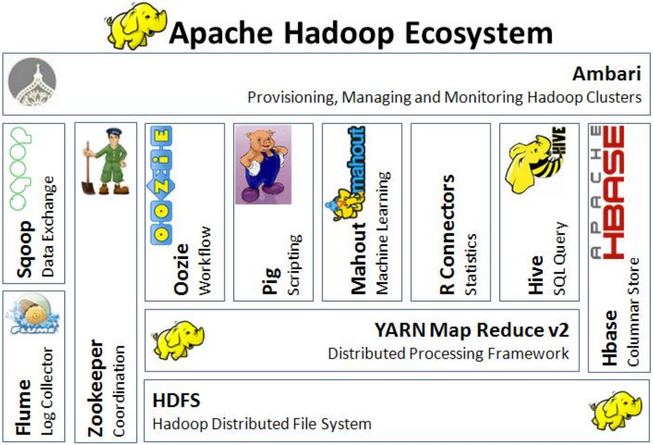
- MapReduce: Simplified Data Processing on Large Clusters (Dean and Gemawat 2004, OSDI)
- Splits computations in *map* and *reduce* phase
- Handles
  - Details of input data partitioning
  - Scheduling program's execution across a set of machines
  - Machine failures
  - Required inter-machine communication

Backgrou

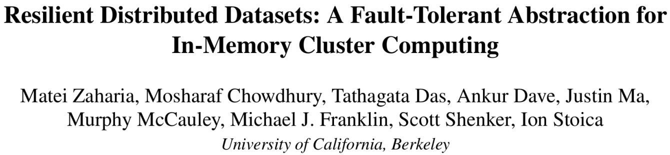
MapReduce Data and Process Flow of Word Count


Unstructured Data Input
Splitting
Mapping
Shuffling
Reducing
Result
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam euismod lectus ut eros bibendum ultrices nec ut lacus. Morbi ac sapien vel tortor luctus ornare. Aenean non lorem ac sapien pulvinar bibendum ac vel nisl. Nunc nec massa sit amet tortor accumsan fringilla id rutrum tellus. Integer sit amet rutrum.
Lorem ipsum dolor sit amet
consectetur adipiscing elit
Aliquam euismod lectus ut eros
bibendum ultrices nec ut lacus
Lorem, 1 ipsum, 1 dolor,1 sit,1 ...., 1
consectetur, 1 adipiscing, 1 elit, 1 ...., 1
Aliquam, 1 euismod, 1 lectus, 1 ut, 1 ...., 1
bibendum, 1 ultrices, 1 nec, 1 ut, 1 ...., 1
Lorem, 1 Lorem, 1 Lorem, 1 Lorem, 1 Lorem, 1
consectetur, 1 consectetur, 1 consectetur, 1
Aliquam, 1 Aliquam, 1 Aliquam, 1 Aliquam, 1
bibendum, 1 bibendum, 1
nec, 1 nec, 1 nec, 1 nec, 1 nec, 1
Lorem, 5
consectetur, 3
Aliquam, 4
bibendum, 2
nec, 5
Lorem, 5 consectetur, 3 Aliquam, 4 bibendum, 2 ...., ... nec, 5

Hadoop Ecosystem



Matei Zaharia et al. Paper



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
University of California, Berkeley

Spark Concept

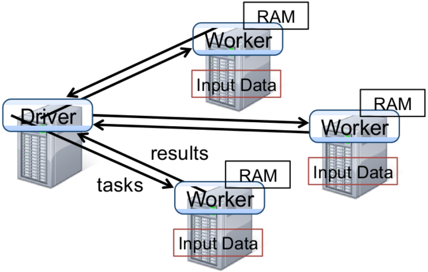


Figure 2: Spark runtime. The user's driver program launches multiple workers, which read data blocks from a distributed file system and can persist computed RDD partitions in memory.

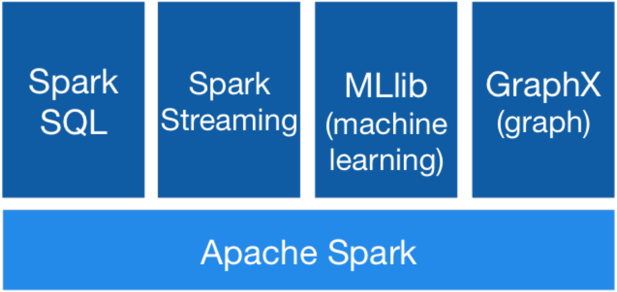
RDD Properties

- Resilient, distributed collections
- Immutable
- Transformations
  - map, filter, reduceByKey, join, ...
- Actions
  - reduce, collect, count, foreach, ...

RDD Example

```
scala> val textFile = sc.textFile("README.md")
textFile: spark.RDD[String] = spark.MappedRDD@2ee9b6e3

scala> textFile.count() // Number of items in this RDD
res0: Long = 126

scala> textFile.first() // First item in this RDD
res1: String = # Apache Spark


scala> val linesWithSpark = textFile.filter(line => line.contains("Spark"))
linesWithSpark: spark.RDD[String] = spark.FilteredRDD@7dd4af09


scala> textFile.filter(line => line.contains("Spark")).count()
res3: Long = 15
```

Spark Libraries



Spark SQL

Spark Streaming

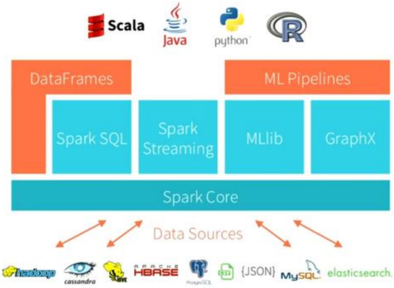
MLlib (machine learning)

GraphX (graph)

Apache Spark

Current Situation

- Current version 1.6.0 (January 2016)
- SparkR included since 1.3.0 (March 2015)



What is SparkR

- Light-weight frontend to Apache Spark from R
- SparkR's development is very *dynamic*
- Main feature: SparkR DataFrames
  - distributed collection of data organized into named columns
  - conceptually equivalent to
    - a table in a relational database or
    - a data frame in R

  ... but on large datasets
- GLM using MLib

R Example

```
df <- createDataFrame(sqlContext, faithful)

# Get basic information about the DataFrame
df
## DataFrame[eruptions:double, waiting:double]

# Displays the content of the DataFrame to stdout
head(df)
##  eruptions waiting
##1     3.600      79
##2     1.800      54
##3     3.333      74
```

R Example

```
# Filter the DataFrame to only retain rows with wait times shorter than 50 mins
head(filter(df, df$waiting < 50))
##  eruptions waiting
##1     1.750      47
##2     1.750      47
##3     1.867      48

# We use the 'n' operator to count the number of times each waiting time appears
head(summarize(groupBy(df, df$waiting), count = n(df$waiting)))
##  waiting count
##1      81    13
##2      60     6
##3      68     1

# We can also sort the output from the aggregation to get the most common waiting times
waiting_counts <- summarize(groupBy(df, df$waiting), count = n(df$waiting))
head(arrange(waiting_counts, desc(waiting_counts$count)))

##   waiting count
##1      78    15
##2      83    14
##3      81    13
```

Reading from Data Sources

```
sc <- sparkR.init(sparkPackages="com.databricks:spark-csv_2.11:1.0.3")
sqlContext <- sparkRSQL.init(sc)

people <- read.df(sqlContext, "./examples/src/main/resources/people.json", "json")
head(people)
##  age    name
##1  NA Michael
##2  30    Andy
##3  19  Justin

# SparkR automatically infers the schema from the JSON file
printSchema(people)
# root
#  |-- age: integer (nullable = true)
#  |-- name: string (nullable = true)

write.df(people, path="people.parquet", source="parquet", mode="overwrite")
```

Cluster setup

- Apache Spark in standalone mode
- 5 Servers
    - 1 master node
    - 4 worker nodes
- Node specifications
    - 2 x 4 core Intel Xeon Processor
    - 48 GB RAM
    - 1 GBit network to storage
    - 20 GBit IP over InfiniBand (interconnect)

Connecting to the Apache Spark Master

```
./bin/sparkR --master spark://10.110.100.120:7077 \
--packages com.databricks:spark-csv_2.10:1.3.0

R version 3.1.1 (2014-07-10) -- "Sock it to Me"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-unknown-linux-gnu (64-bit)

...
```