MapReduce: Esercizi

Fabio Cumbo

Corso: Introduzione ai Big Data – A.A. 2016/2017

Indice

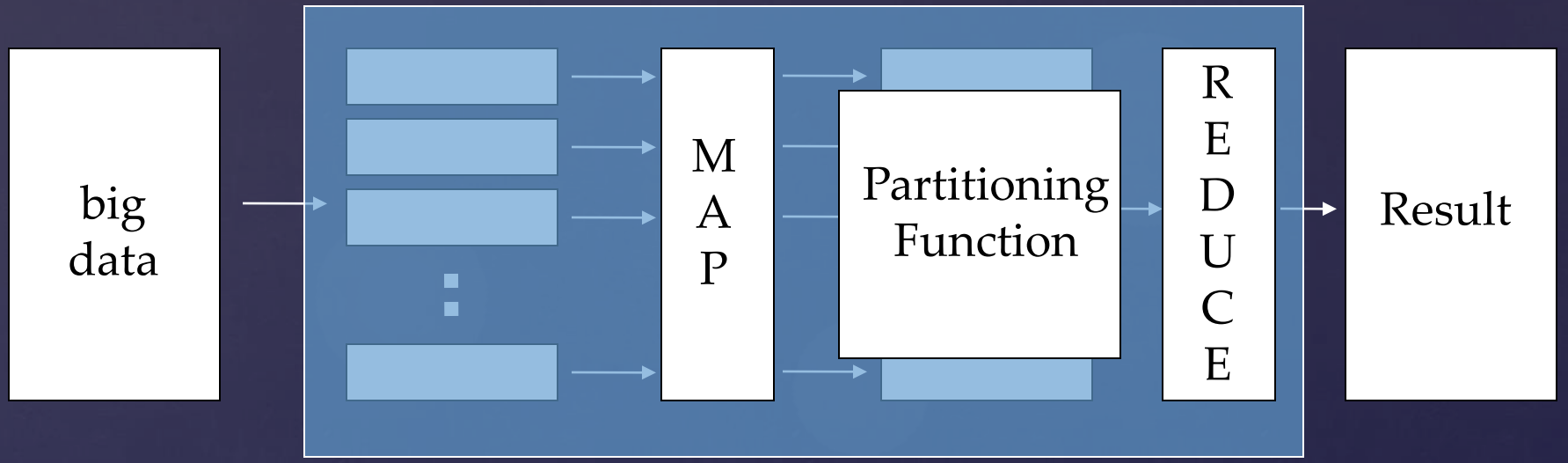
MapReduce: concetti chiave

Esercizi

MapReduce

- Un semplice modello di programmazione
- Modello funzionale
- Creato per il processamento di grandi moli di dati
  - Sfrutta le risorse di grandi insiemi di calcolatori
  - Esegue processi in maniera distribuita
- Motivazioni
  - Cresce la necessità di processare grandi moli di dati
  - Scalabilità

Map + Reduce



- Map:
  - Accetta in input una coppia chiave/valore
  - Restituisce in output un risultato intermedio come coppia chiave/valore
- Reduce :
  - Accetta in input il risultato intermedio come coppia chiave/valore
  - Restituisce in output una coppia chiave/valore

Hadoop

http://hadoop.apache.org/

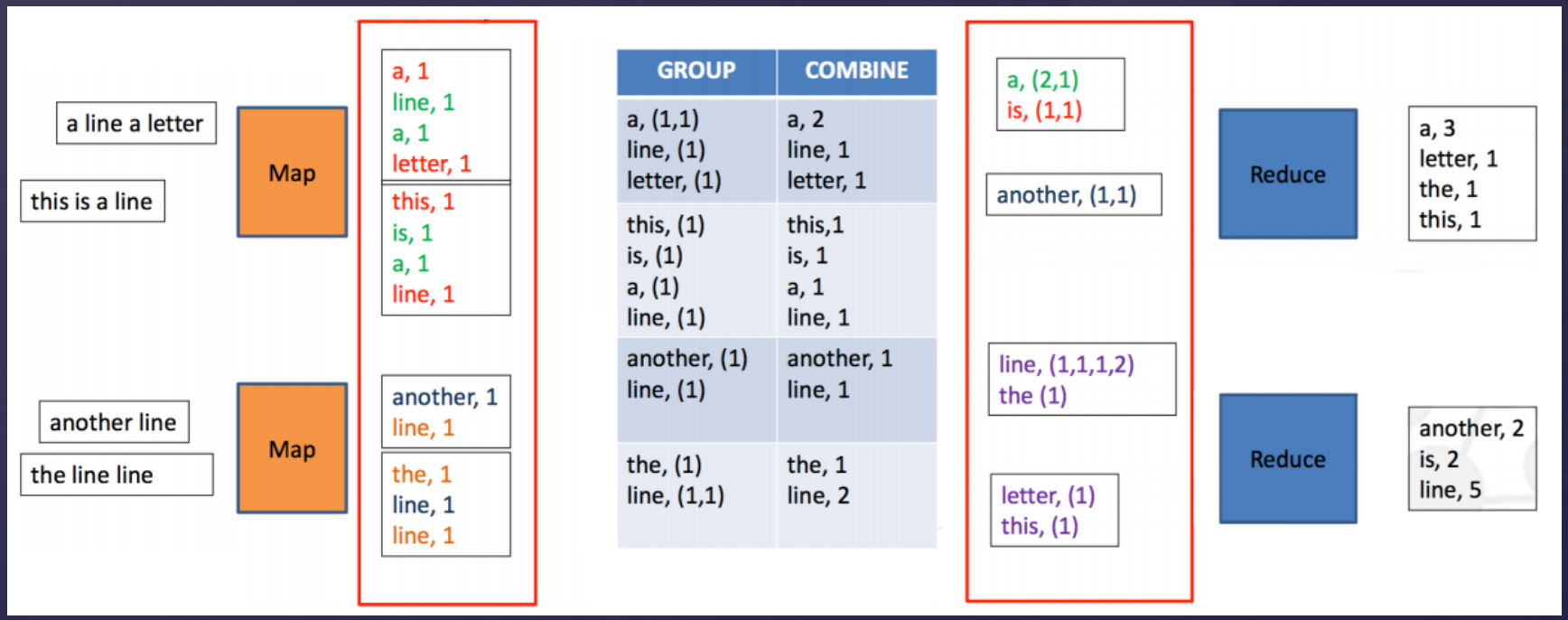
- Open source
- Implementazione Java di MapReduce
- Utilizza HDFS come file system sottostante

RHadoop

https://github.com/RevolutionAnalytics/RHadoop/wiki

Insieme di pacchetti per R che permettono all'utente di gestire e analizzare dati con Hadoop.

Word Count


a line a letter
this is a line
Map
a, 1
line, 1
a, 1
letter, 1
this, 1
is, 1
a, 1
line, 1
another line
the line line
Map
another, 1
line, 1
the, 1
line, 1
line, 1
GROUP
COMBINE
a, (1,1)
line, (1)
letter, (1)
a, 2
line, 1
letter, 1
this, (1)
is, (1)
a, (1)
line, (1)
this,1
is, 1
a, 1
line, 1
another, (1)
line, (1)
another, 1
line, 1
the, (1)
line, (1,1)
the, 1
line, 2
a, (2,1)
is, (1,1)
another, (1,1)
line, (1,1,1,2)
the (1)
letter, (1)
this, (1)
Reduce
Reduce
a, 3
letter, 1
the, 1
this, 1
another, 2
is, 2
line, 5

Word Count

```
library(rmr2)

## map function
map <- function(k,lines) {
      words.list <- strsplit(lines, '\\s')
      words <- unlist(words.list)
      keyval(words, 1)
}

## reduce function
reduce <- function(word, counts) {
      keyval(word, sum(counts))
}

wordcount <- function (input, output=NULL) {
      mapreduce(input=input, output=output,
      input.format="text", map=map, reduce=reduce)
}
```

Word Count

```
## svuota la cartella di output
system("bin/hadoop fs -rmr wordcount/out")

## sottomette il job
hdfs.root <- 'wordcount'
hdfs.data <- file.path(hdfs.root, 'data')
hdfs.out <- file.path(hdfs.root, 'out')
out <- wordcount(hdfs.data, hdfs.out)

## recupera i dati dal file system distribuito (HDFS)
results <- from.dfs(out)

## stampa le prime 30 parole più frequenti
results.df <- as.data.frame(results, stringsAsFactors=F)
colnames(results.df) <- c('word', 'count')
head(results.df[order(results.df$count, decreasing=T), ], 30)
```

Esercizi

- Somma di numeri da 1 fino ad N
- Calcolo della varianza di N numeri
- Re-Implementazione dell'algoritmo K-Means

- Applicazione reale: ritardi aerei + tempo di volo

Somma di N numeri

```
sum_test <- function(N){
      #Move data to HDFS
      hdfs.data <- to.dfs(1:N)

      #Define and run addition function
      result <- mapreduce(input = hdfs.data,
            map = function(k, v) keyval(1, v),
            reduce = function(k, v)
                  keyval(k, sum(v)), combine = T)

      #Retrieve and return the sum
      from.dfs(result)
}
```

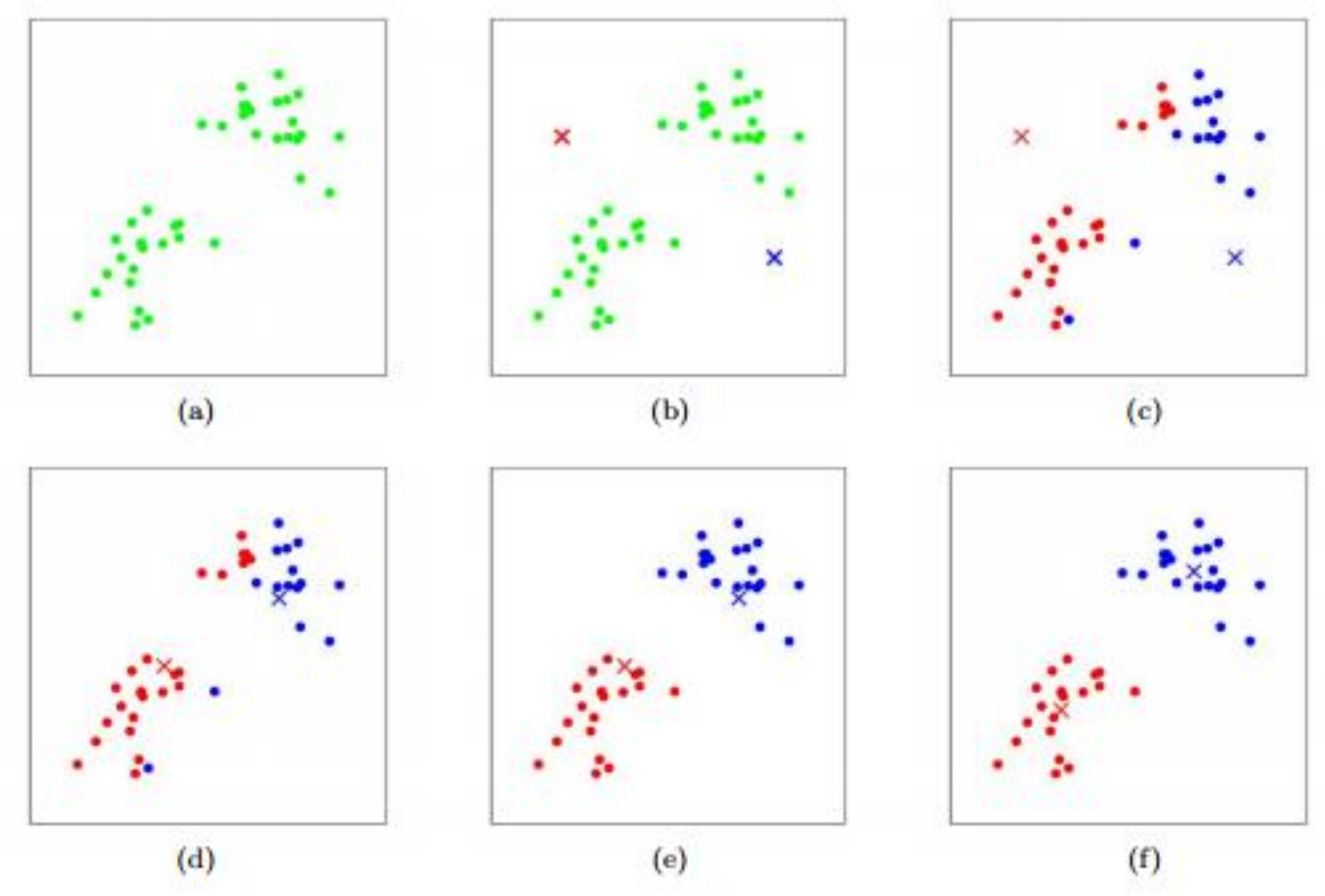
Varianza di N numeri

```
var_test <- function(N){
      #Move data to HDFS
      hdfs.data <- to.dfs(1:N)

      #Run mapreduce job to calculate count, sum,
            and sum of squares
      sum_squares <- mapreduce(input = hdfs.data,
            map = function(k, v)
                  keyval(1, cbind(1, v, v^2)),
            reduce = function(k, v)
                  keyval(k, t(colSums(v))),
                  combine = T)

      #Retrieve result and compute variance
      sums <- from.dfs(sum_squares)$val
      (sums[3] - (sums[2]^2)/sums[1])/(sums[1]-1)
}
```

K-Means



K-Means

- Posizione K punti random sullo spazio, con K uguale al numero di cluster. Questi punti rappresentano i centroidi iniziali.
- Assegna ogni punto al gruppo rappresentato dal centroide più vicino.
- Quando tutti i punti sono stati assegnati, ricalcola le posizioni dei centroidi.
- Ripeti i due step precedenti finchè i centroidi diventino stabili.

L'algoritmo potrebbe non arrivare mai a convergenza. Per questo è necessario specificare il numero massimo di iterazioni da effettuare prima di terminare la procedura.

K-Means

```
kmeans_test <- function(data_size, num_clusters,
                        num_iterations) {

      #Create data and move data to HDFS
      input <- do.call(rbind,rep(
            list(matrix(rnorm(data_size, sd=10),
            ncol=2)), 20)) + matrix(rnorm(200), ncol=2)
      hdfs.data <- to.dfs(input)

      #Helper function: Euclidian distance
      dist.fun <- function(C, P) {
                  apply(C, 1, function(x)
                  colSums((t(P) - x)^2)) }
```

K-Means

```
#Map: Compute distances of points to centroids
kmeans.map <- function(., P) {
        nearest = {
                if (is.null(C))
                        sample(1:num_clusters, nrow(P), replace = T)
                else {
                        distance = dist.fun(C, P)
                        nearest = max.col(-distance)
                }
        }
        keyval(nearest, P)
}

#Reduce: Compute new centroids
kmeans.reduce <- function(k, P)
        t(as.matrix(apply(P, 2, mean)))
```

K-Means

```
    C = NULL

    for (i in 1:num_iterations ) {
          C = values(from.dfs( mapreduce(hdfs.data,
                map = kmeans.map,
                reduce = kmeans.reduce)))
    }

    C
}
```

Applicazione reale

Ritardi aerei

Problema:
Calcolare la media dei ritardi delle partenze aeree per anno e mese, per ogni linea aerea nel dataset

Dataset:
http://stat-computing.org/dataexpo/2009/the-data.html

Requisiti: pacchetto rmr
https://github.com/RevolutionAnalytics/RHadoop/wiki

Applicazione reale

Ritardi aerei

2004,3,25,4,1445,1437,1820,1812,AA,399,N275AA,215,215,197,8,8,BOS,MIA,1258,6,12,0,,0,0,0,0,0,0
2004,3,25,4,728,730,1043,1037,AA,596,N066AA,195,187,170,6,-2,MIA,BOS,1258,7,18,0,,0,0,0,0,0,0
2004,3,25,4,1333,1335,1651,1653,AA,680,N075AA,198,198,168,-2,-2,MIA,BOS,1258,9,21,0,,0,0,0,0,0,0
2004,3,25,4,1051,1055,1410,1414,AA,836,N494AA,199,199,165,-4,-4,MIA,BOS,1258,4,30,0,,0,0,0,0,0,0
2004,3,25,4,558,600,900,924,AA,989,N073AA,182,204,157,-24,-2,BOS,MIA,1258,11,14,0,,0,0,0,0,0,0
2004,3,25,4,1514,1505,1901,1844,AA,1359,N538AA,227,219,176,17,9,BOS,MIA,1258,15,36,0,,0,0,0,15,0,2
2004,3,25,4,1754,1755,2052,2121,AA,1367,N075AA,178,206,158,-29,-1,BOS,MIA,1258,5,15,0,,0,0,0,0,0,0
2004,3,25,4,810,815,1132,1151,AA,1381,N216AA,202,216,180,-19,-5,BOS,MIA,1258,7,15,0,,0,0,0,0,0,0
2004,3,25,4,1708,1710,2031,2033,AA,1636,N523AA,203,203,173,-2,-2,MIA,BOS,1258,4,26,0,,0,0,0,0,0,0
2004,3,25,4,1150,1157,1445,1524,AA,1901,N066AA,175,207,161,-39,-7,BOS,MIA,1258,4,10,0,,0,0,0,0,0,0
2004,3,25,4,2011,1950,2324,2257,AA,1908,N071AA,193,187,163,27,21,MIA,BOS,1258,4,26,0,,0,0,21,6,0,0
2004,3,25,4,1600,1605,1941,1919,AA,2010,N549AA,221,194,196,22,-5,MIA,BOS,1258,10,15,0,,0,0,0,22,0,0

Ritardi aerei

```
library(rmr)

in <- "/data/airline/1987.csv";
out <- "/dept-delay-month";

csvtextinputformat <- function (line)
    keyval(NULL,
    unlist(strsplit(line, "\\,")))
```

Ritardi aerei

```
mapping <- function(k, fields) {
        # Skip header lines and bad records:
        if (!(identical(fields[[1]], "Year")) & length(fields) == 29) {
                deptDelay <- fields[[16]]

                # Skip records where departure dalay is "NA":
                if (!(identical(deptDelay, "NA"))) {
                        # field[9] is carrier, field[1] is year,
                        field[2] is month:

                        keyval(c(fields[[9]], fields[[1]],
                                fields[[2]]), deptDelay)
                }
        }
}
```

Ritardi aerei

```
reducing <- function(keySplit, vv) {
              keyval(keySplit[[2]], c(keySplit[[3]],
                     length(vv), keySplit[[1]],
                     mean(as.numeric (vv))))
              }

deptdelay <- function (input, output) {
              mapreduce(input = input, output = output,
                     textinputformat = csvtextinputformat,
                     map = mapping,
                     reduce = reducing)
              }

from.dfs(deptdelay(in, out))
```

Applicazione reale

Tempo di volo

Problema:
Calcolare il tempo medio di volo considerando tutte le tratte aeree effettuate di anno in anno

Dataset:
http://stat-computing.org/dataexpo/2009/the-data.html

Requisiti: pacchetto rmr
https://github.com/RevolutionAnalytics/RHadoop/wiki

Tempo di volo

```
library(rmr)

in <- “/data/airline/1987.csv”;
out <- “/dept-delay-month”;

asa.csvtextinputformat <- make.input.format( format =
function(line) {
        values = unlist( strsplit(line, "\\,") )
        names(values) = c('Year','Month','DayofMonth','DayOfWeek',
                'DepTime','CRSDepTime','ArrTime','CRSArrTime',
                'UniqueCarrier','FlightNum','TailNum',
                'ActualElapsedTime','CRSElapsedTime','AirTime',
                'ArrDelay','DepDelay','Origin','Dest','Distance',
                'TaxiIn','TaxiOut','Cancelled','CancellationCode',
                'Diverted','CarrierDelay','WeatherDelay',
                'NASDelay','SecurityDelay','LateAircraftDelay')
        return( keyval(NULL, values) )
        }
)
```

Tempo di volo

```
# the mapper gets a key and a value vector generated by the formatter
mapper.year.market.enroute_time = function(key, val) {
        # Skip header lines, cancellations, and diversions:
        if ( !identical(as.character(val['Year']), 'Year') &
                identical(as.numeric(val['Cancelled']), 0) &
                identical(as.numeric(val['Diverted']), 0) ) {

                # We don't care about direction of travel, so construct 'market'
                # with airports ordered alphabetically (LAX to JFK becomes 'JFK-LAX')
                if (val['Origin'] < val['Dest'])
                        market = paste(val['Origin'],
                                val['Dest'], sep='-')
                else
                        market = paste(val['Dest'],
                                val['Origin'], sep='-')

                # key consists of year, market
                output.key = c(val['Year'], market)
                # output time in air
                output.val = val['AirTime']

                return( keyval(output.key, output.val) )
        }
}
```

Tempo di volo

```
# the reducer gets all the values for a given key
# the values (which may be multi-valued as here) come in the form of a list()
reducer.year.market.enroute_time = function(key, val.list) {
        # val.list is a list of row vectors
        # a data.frame is a list of column vectors
        # plyr's ldply() is the easiest way to convert IMHO
        if ( require(plyr) )
                val.df = ldply(val.list, as.numeric)
        else {
                # this is as close as my deficient *apply skills
                # can come w/o plyr
                val.list = lapply(val.list, as.numeric)
                val.df = data.frame( do.call(rbind, val.list) )
        }
        colnames(val.df) = c('actual','crs','air')

        output.key = key
        output.val = mean(val.df$air, na.rm=T)

        return( keyval(output.key, output.val) )
}
```

Tempo di volo

```
mr.year.market.enroute_time = function (input, output) {
	mapreduce(input = input, output = output,
		input.format = asa.csvtextinputformat,
		map = mapper.year.market.enroute_time,
		reduce = reducer.year.market.enroute_time,
		backend.parameters = list(
			hadoop = list(D = "mapred.reduce.tasks=10") ),
		verbose=T)
}

hdfs.output.path = file.path(hdfs.output.root, 'enroute-time')
results = mr.year.market.enroute_time(hdfs.input.path,
	hdfs.output.path)

results.df = from.dfs(results, to.data.frame=T)
colnames(results.df) = c('year', 'market', 'flights',
	'scheduled', 'actual', 'in.air')

save(results.df, file="out/enroute.time.RData")
```

Tempo di volo

```
nrow(results.df)

yearly.mean = ddply(results.df, c('year'), summarise,
      in.air = weighted.mean(in.air, flights))

ggplot(yearly.mean) +
      geom_line(aes(x=year, y=in.air), color='#4689cc')
      + theme_bw() + ylim(c(60, 130)) + ylab('minutes')
```