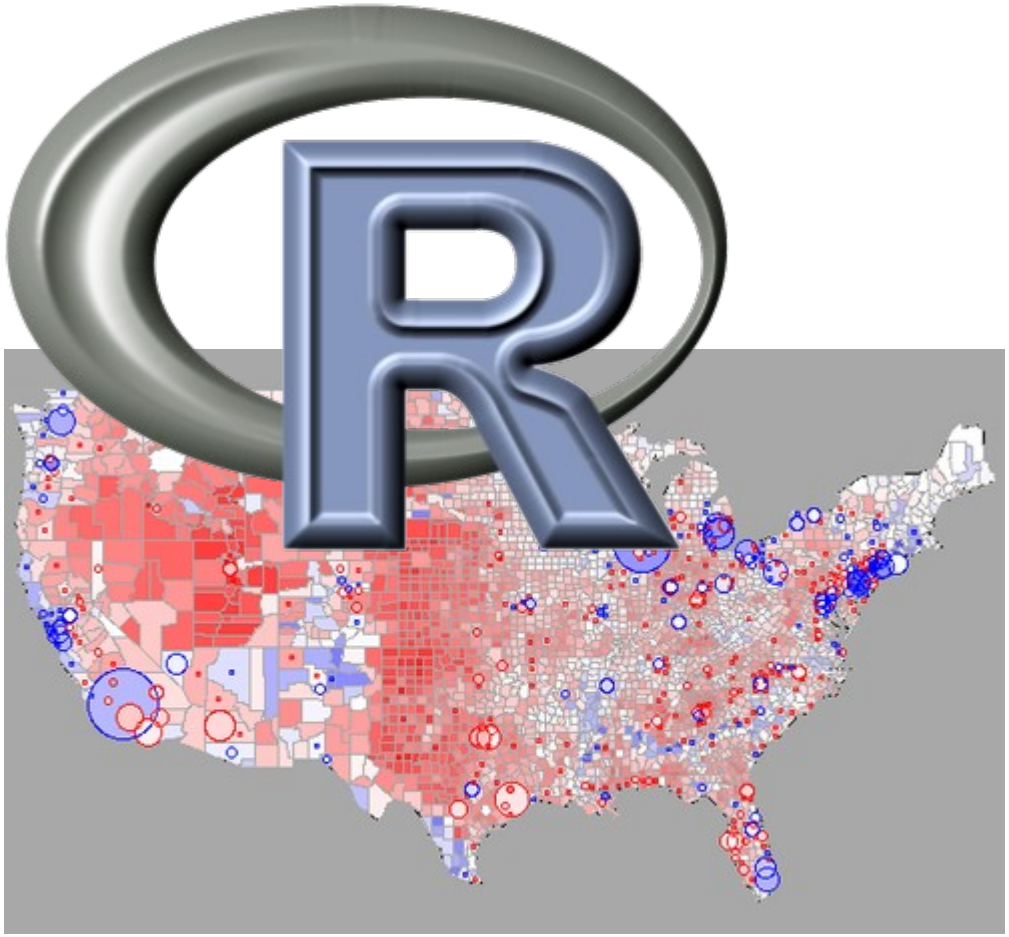
FIAP

PÓS-GRADUAÇÃO



MBA em Big Data

Introdução à Linguagem R

Encontro 4

Prof. Antonio Henrique Pinto Selvatici

antoniohps@gmail.com

Versão 1 – 10/2014

Professor

- **Antonio Henrique Pinto Selvatici**

- É engenheiro eletrônico formado pelo Instituto Tecnológico de Aeronáutica (ITA), com mestrado e doutorado pela Escola Politécnica (USP), e passagem pela Georgia Institute of Technology em Atlanta (EUA). Desde 2002, atua na indústria em projetos nas áreas de robótica, visão computacional e internet das coisas, aliando teoria e prática no desenvolvimento de soluções baseadas em Machine Learning, processamento paralelo e modelos probabilísticos, tendo desenvolvidos projetos para Avibrás, Rede Globo, IPT e Systax. Foi professor do curso de Ciência da Computação da Uninove de 2009 a 2013. Em 2012, tendo ajudado a fundar a Selsantech, participou do desenvolvimento do CatSearch, uma solução para Data Mining preparada para o paradigma MapReduce. É professor do MBA do curso de Big Data da FIAP e trabalha na reformulação do sistema de pagamento on-line eWally.

Resumo

- Processamento paralelo em R
- Apresentação do pacote gputools
- Introdução ao RHadoop programação em R
- Atividade prática

6. Processamento paralelo em R

1 – Por que precisamos de paralelização?

FIAP

- Resolver problemas de performance
  - R é uma linguagem single-thread
  - Chamas interpretadas possuem execução mais lenta
  - Não há ponteiros ou passagem de parâmetros por referência
- Abordagens para melhorar a performance
  - Chamar funções vetorizadas, que tratam os múltiplos dados internamente
  - Chamar funções em C que implementam algoritmos mais custosos
  - Paralelizar o processamento
- Resolver problemas de memória (big data)
  - Volume de dados pode não caber no armazenamento de um único nó de processamento
  - R carrega todos os dados na RAM, ou seja, a RAM deve suportar todos objetos que são carregados e são criados a cada atribuição, passagem de parâmetros, etc.
  - R limita o tamanho de um objeto a 2GB, independentemente da quantidade de memória do computador

2 – Abordagens de paralelização

FIAP

- Multi-threading:
  - Através de pacotes que implementam funções nativas de multi-threading
  - Através de funções do C que implementam multi-threading
- GPU
  - Através de pacotes que implementam operações paralelizadas na GPU
  - `gputools`
- Computação distribuída
  - Através de pacotes que implementam a distribuição do processamento entre diversos nós da rede
  - `snow`, `parallel`
  - Através de pacotes que utilizam infraestrutura de big data, como Hadoop
  - "R+Hadoop" (`rhdfs`, `rmr`, `rhbase`), `rhipe`, `orch`, etc.

3 – Pacote gputools

- Possui funções para paralelização de diversas operações aritméticas e de álgebra linear
  - gpuCor: calcula o coeficiente correlação
  - gpuCrossprod: calcula a multiplicação “cruzada” entre matrizes, ou o produto interno entre vetores
  - gpuDist: calcula a distância entre pontos
  - gpuDistClust: realiza a clusterização de dados com base em distâncias
  - gpuMatMult: calcula a multiplicação entre matrizes
  - gpuSolve: resolve sistemas de equações lineares
- Exemplo
  - Seja a multiplicação de duas matrizes quadradas com 2000 x 2000 elementos.
    - » Com a CPU: 8,0 segundos
    - » Com a GPU: 1,4 segundo

4 – R + Hadoop

- Há vários pacotes que incorporam R à infraestrutura do Hadoop
- Em linhas gerais, o interpretador do R e seus pacotes precisam estar instalados em todos os nós de processamento
- R + Hadoop é, na verdade, um conjunto de pacotes, elaborados pela RevolutionAnalytics:
  - Site: https://github.com/RevolutionAnalytics/RHadoop/wiki
  - `rhdfs`: funções de acesso ao sistema de arquivos HDFS
  - `rmr`: funções para implementar a lógica map-reduce através do streaming do Hadoop
  - `rhbase`: funções para acesso aos dados do HBase
- Aqui vamos instalar rhdfs + rmr para implementar a lógica map-reduce através do R

5 – Tutorial de instalação do R+Hadoop

- Consideramos que o Hadoop já está instalado e configurado em todos os nós
- A primeira coisa é instalar o R em todos os
  - Sistemas da Red Hat: `su -c "yum install R"`
    - » Para CentOS ver www.jason-french.com/blog/2013/03/11/installing-r-in-linux
  - Sistemas com base em Debian: `sudo apt-get install R`
- Após a instalação, devemos configurar a localização do Java em todos os nós (considerar diretório atual em `$HADOOP_PREFIX`)
  - Pegar o diretório Java em `etc/hadoop/hadoop-env.sh`
  - Seja o diretório Java: `/usr/lib/jvm/java-7-oracle`
  - Configurar o Java para todos os usuários do R:
    - » `export JAVA_HOME=/usr/lib/jvm/java-7-oracle`
    - » `sudo –E R CMD javareconf`
  - `Verificar se temos curl-config na máquina`
    - » Em sistemas Debian, temos o pacote `libcurl4-openssl-dev` que provê esse programa
    - » Para Red Hat, instalar `libcurl-devel`

5 – Tutorial de instalação do R+Hadoop

FIAP

- Iniciar R como super-usuário:
  - `sudo –E R`
- Instalar os seguintes pacotes através de:
  `install.packages(<pkgs>,lib="/usr/lib/R/site-library")`
  - `rJava`
  - `RJSONIO`
  - `itertools`
  - `digest`
  - `Rcpp`
  - `httr`
  - `functional`
  - `devtools`
  - `plyr`
  - `reshape2`
  - `caTools`

5 – Tutorial de instalação do R+Hadoop

FIAP

- Baixar e instalar os pacotes `rhdfs` e `rmr2` (opcionalmente, pode instalar `rhbase`)
  - https://github.com/RevolutionAnalytics/RHadoop/wiki/Downloads
- Instalar os pacotes baixados
  - export HADOOP_CMD=$HADOOP_PREFIX/bin/hadoop
  - sudo –E R CMD INSTALL rhdfs1.8.0.tar.gz -l /usr/lib/R/site-library
  - sudo –E R CMD INSTALL rmr2_3.2.0.tar.gz -l /usr/lib/R/site-library
- Instalar RStudio na máquina que roda o cluster Hadoop como master
  - http://www.rstudio.com/products/rstudio/download/
- Testar o Hadoop na máquina para ver se está funcionando
  - $HADOOP_PREFIX/sbin/start-dfs.sh
  - Copiar arquivos de texto para o HDFS (estando `hadoop` no PATH)
    - » hadoop fs -mkdir 1
    - » hadoop fs -put <files> 1
  - hadoop jar $HADOOP_PREFIX/share/hadoop/mapreduce/hadoop-mapreduce-example*.jar wordcount 1 1.out

5 – Tutorial de instalação do R+Hadoop

- Entrar no RStudio no nó master
- Antes de executar qualquer programa do `rmr`, é necessário:
  - `Sys.setenv(HADOOP_CMD="<path-to-hadoop>")`
    - » Para descobrir: `which hadoop`
  - `Sys.setenv(HADOOP_STREAMING="<path-to-hadoop-streaming-jar>")`
    - » Para descobrir: `locate hadoop-streaming`
  - `library('rhdfs')`
  - `library('rmr2')`
  - `hdfs.init()`
- Testando o `rmr`
  - `small.ints = to.dfs(keyval(1:10,seq(10,100,by=10)))`
  - `mapreduce(`
  - `  input = small.ints,`
  - `  map = function(k, v) keyval(k,cbind(v, v^2)))`

6 – O velho exemplo ‘wordcount’ no R

```
wordcount = function( input, output = NULL, pattern = " "){
  #Nosso mapper – cria os pares (key, value)
  wc.map = function(., lines) {
    keyval(
      #strsplit: divide uma string em pedaços
      #unlist: transforma listas em vetores
      unlist( strsplit( x = lines, split = pattern)),
      1)}
  #Nosso reducer
  wc.reduce = function(word, counts) {
    keyval(word, sum(counts))}
  #Chamando o job do MapReduce
       mapreduce(input = input , output = output,
        input.format = "text", map = wc.map,
        reduce = wc.reduce)}
```

6 – O velho exemplo ‘wordcount’ no R

FIAP

- Para executá-lo, basta invocar:
  - `wordcount( hdfs_input, hdfs_output )`
  - hdfs_input tem que ser um objeto big.data.object ou um arquivo no HDFS
  - hdfs_output pode ser NULL ou um diretório novo no HDFS
- Exemplo:
  - Mandar um arquivo de texto para o HDFS e executar o worcount:
    - » `hdfs.put(‘alice.txt’,’1’)`
    - » `wordcount(‘1’, ‘1.out’)`
  - Ler o resultado: `from.dfs(‘1.out’)`
- Nosso `mapper` vai ler o arquivo de texto linha por linha e dividi-las por espaços, transformando as palavras resultantes em `keys`. Para cada `key`, será atribuído um `value` de 1.
- Nosso `reducer` vai somar todos os `values` (1s) correspondentes a cada `key` (palavra)

7 – Como usar `mapreduce`

- Principal função do pacote rmr, permitindo a execução dos jobs MapReduce
- Utilização:

```
– mapreduce(
–   input, #arquivo ou diretório de entrada no HDFS
–   output = NULL, #caso NULL, cria um arq. temporário
–   map = to.map(identity), #função do mapper
–   reduce = NULL, #função do reducer
–   vectorized.reduce = FALSE,
–   combine = NULL, #define o combiner a ser usado
–   in.memory.combine = FALSE,
–   input.format = "native", #”native” ou “text”
–   output.format = "native",# melhor deixar “native”
–   backend.parameters = list(), #Não usar
–   verbose = TRUE)
```

8 – Argumentos de `mapreduce`

- `input`: arquivo de entrada do HDFS. Para gerar um arquivo com o conteúdo de um objeto do R, usar `to.dfs(objeto, output="nome-no-hdfs")`
- `input.format`: formato do arquivo de entrada, geralmente "`text`" (para arquivo de texto) ou "`native`" (para objetos gerados no R), ou ainda um parser específico para arquivos de dados
- `map`: função que recebe uma lista com dois elementos:
  - `$key`: vetor ou lista contendo os valores das keys (pode ser NULL)
  - `$val`: vetor ou lista de valores a serem mapeados possui um valor
  - Deve retornar uma lista no mesmo formato, onde para cada value deve haver uma key (geralmente número ou texto)
  - Função `keyval(keys, values)`: monta a lista com os pares (key,val)
- `reducer`: função que recebe uma lista com dois elementos:
  - `$key`: valor de uma key específica
  - `$values`: vetor ou lista contendo todos os valores associados àquela key
  - Deve retornar pares (key,val) correspondente ao processamento dos valores de entrada

9 – Trabalhando com tabelas CSV

- É possível extrair um arquivo csv do HDFS diretamente como um data frame

```
hdfs.put("ResultadosRTLS.csv",“/1/")
#Criando o parser para leitura
rtlsFmt = make.input.format(  "csv", sep=",",
quot="\"",stringsAsFactors=F, col.names=c("LM", "MCMC.LM",
"LM.FT", "MCMC.LM.FT"))
rtls <- from.dfs("/1/ResultadosRTLS.csv",format=rtlsFmt)
```

- Processando com o map reduce: ver o resultado da fase “map”

```
mapping = mapreduce (  input = "/1/ResultadosRTLS2.csv",
  input.format = rtlsFmt,  map = function(k, v) {
    colunas_empilhadas <- numeric()
    for(i in 1:ncol(v)) {
      colunas_empilhadas <- rbind(colunas_empilhadas,v[,i])
    }
    keyval(names(v),colunas_empilhadas)
})
```

9 – Trabalhando com tabelas CSV

- `mapreduce` divide o arquivo em pedaços antes da fase “map”.
- É possível configurar formatadores para outros tipos de arquivo, veja em
  - `https://github.com/RevolutionAnalytics/rmr2/blob/master/docs/getting-data-in-and-out.md`
- Não há opção para excluir o cabeçalho. Por isso, o arquivo CSV deve conter apenas os dados (a opção `skip` irá pular linhas a cada vez que o formatador for chamado): usar write.table() com opções para salvar o csv sem cabeçalho
- Reducer para fazer a média das colunas (todos os `values` correspondentes à `key` são concatenados em `values2`):

```
reduce = function(key,values2) {
  keyval(key,mean(values2))
}
```

10 – Correção da lista de exercícios 3

- Faça um script do R que implemente a função fib(n), que retorna um vetor com a sequência de Fibonacci de 1 até n. A sequência de Fibonnaci é definida como f(1) = 1, f(2) = 1 e f(n) = f(n-1) + f(n-2) para n > 2

```
fib <- function(n) {
  n <- as.integer(n)
  if(n <= 0) stop('O argumento n deve ser estritamente positivo');

  r <- integer(n)
  r[1] = 1
  if(n > 1) r[2] = 1
  for(i in seq(3,n,by=1)) {
    r[i] <- r[i-1]+r[i-2]
  }
  return(r)
}
```

- Modele o seguinte problema na forma de um sistema de equações e resolva através da função solve do R. Escreva a solução na forma de um Markdown do R. **Problema:** Zezinho tem 5 pedras a mais do que Huguinho. Luisinho tinha 10 pedras antes de dar a Zezinho metade do que este tem hoje. Huguinho tem 3 pedras a menos do que Luisinho. Quantas pedras tem cada um?

- Faça um script do R analisador da bolsa de valores. Faça uma função que receba um data frame com as mesmas colunas daquele gerado por COTAHIST.A1997 e um vetor de códigos de ação. A função deve gerar a plotagem dos papeis ao longo dos dias, a plotagem das correlações dos dados históricos de preço de fechamento (PREULT) dos papéis, e o gráfico de pizza dos volumes negociados no ano daqueles papeis.

11 – Atividade

- Processar o arquivo COTAHIST.A1997 para processamento no R, separando, os dados do arquivo em vários arquivos data frames, cada um contendo os dados de apenas os dados de uma ação
  1. Transformar de FWF para CSV:
     a. Ler COTAHIST.A1997 em um data frame
     b. Salvar o data frame, a menos da última linha, como CSV, através de:
        `write.table( <data>, <arquivo>, sep=",",row.names=F,col.names=F)`
  2. Enviar o arquivo gerado para o HDFS usando `hdfs.put()`
  3. Criar o formatador para ler COTAHIST.A1997.csv como no exemplo
  4. Processar os dados de entrada, gerando o arquivo "/COTAHIST/1997/medias" no HDFS
     a. Fase "map": para cada data frame de entrada, gerar os pares `<key,val>`, onde `key` é o nome (CODNEG) da ação e `val` é a linha do data frame contendo os dados desejados da ação (DATA, PREABE, PREMAX, PREMIN, PREMED, PREULT, PREOFC, PREOFV), restrito a CODBDI=2
     b. Fase "reduce": gerar um par `<key,val>` onde `key` é o nome da ação e `val` é a lista com as médias anuais dos valores de preço da ação (PREABE, PREMAX, PREMIN, PREMED, PREULT, PREOFC, PREOFV)

Referências

- Norman Matloff. *The Art Of R Programming*. No Starch Press, São Francisco, CA, 2011.
- Joseph Adler. *R in a Nutshell*. O'Reilly Media, Inc., Sebastopol, CA, 2012
- Mark Gardener. Beginning R: The Statistical Programming Language. John Wiley & Sons, Indiana, IN, 2012.
- Vignesh Prajapati. Big Data Analytics with R and Hadoop, Packt Publishing Ltd, 2013

FIAP

A MELHOR FACULDADE DE TECNOLOGIA