INSTITUTO FEDERAL
Sudeste de Minas Gerais

MapReduce e Hadoop Básico

Curso: Tecnologia em Gestão da Tecnologia da Informação

Prof.: Jean Henrique de Sousa Câmara

Contato: jean.camara@ifsudestemg.edu.br

MapReduce

- Processar dados requer grande capacidade de armazenamento
  - Mas outros fatores críticos, como **desempenho e disponibilidade**, também são fundamentais para que se possa atender às requisições da melhor forma
- O Map Reduce é um modelo de programação utilizado para gerar **processamento distribuído**

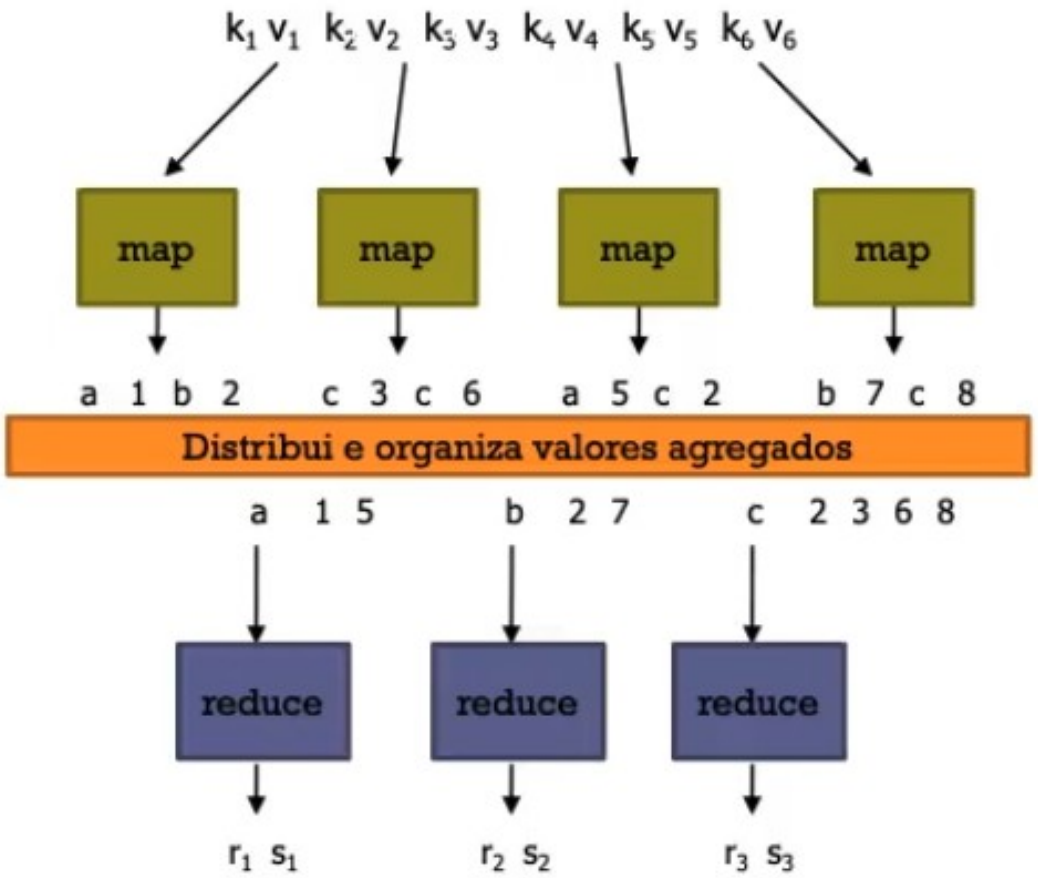
MapReduce

- **Processa** grande quantidades de dados **de forma paralela**, com uso de **hardware de propósito geral** e tecnologia de **cluster**
- Os dados **são fragmentados** por diversos computadores para serem processados simultaneamente
- A sua ideia consiste em aplicar funções nas entradas de valores, de forma a **reduzir a saída em um único valor**

MapReduce

- Se resume em duas funções:
  - **map** $(k_1, v_1) \rightarrow [<k_2, v_2>]$
  - **reduce** $(k_2, [v_2]) \rightarrow [<k_3, v_3>]$
- Todos os valores v_2 com a mesma chave k_2 são garantidamente enviados para o mesmo nó do cluster para sofrerem o reduce
- O arcabouço de execução cuida de todo o resto

MapReduce


k1 v1 k2 v2 k3 v3 k4 v4 k5 v5 k6 v6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
Distribui e organiza valores agregados
a 1 5
b 2 7
c 2 3 6 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

Etapas do MapReduce

1. Os dados de entrada são divididos em segmentos
2. O programa é inicializado no conjunto de máquinas que executam as funções atribuídas para a aplicação do MapReduce
3. Uma das máquinas é selecionada como **mestre** e as outras máquinas são selecionadas como trabalhadores (**escravos**). O mestre escolhe trabalhadores livres e lhes envia tarefas de forma distribuída

Etapas do MapReduce

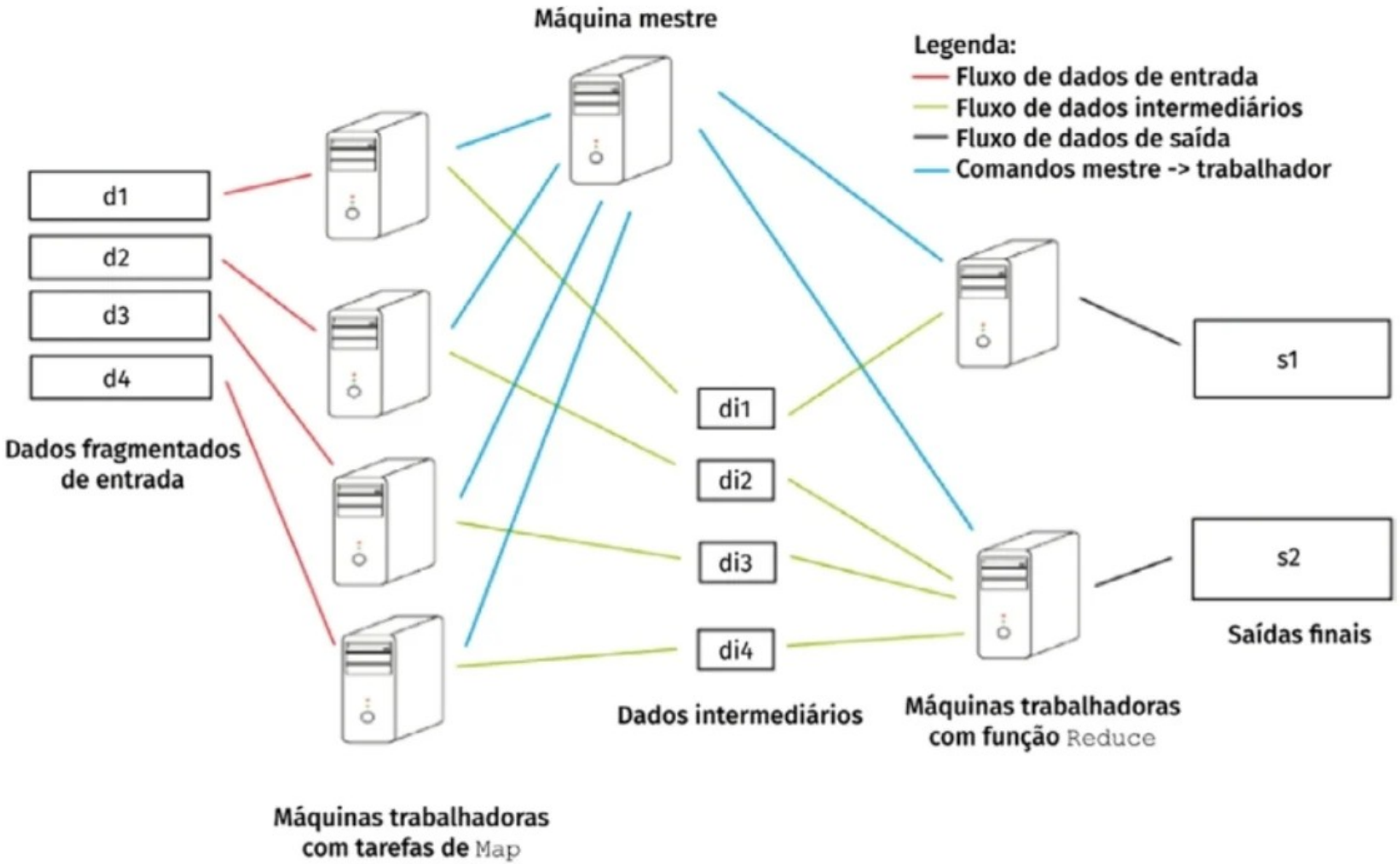
4. Os escravos que executam a função Map mantêm os valores intermediários na memória e **os gravam periodicamente no disco local**, notificando assim o computador principal (mestre) a respeito da localização dos pares intermediários de chave e valor

5. Os escravos que executam a função Reduce obtêm a autorização para iniciar a tarefa por meio do mestre e também recebem informações sobre o local onde está escrito o valor relacionado à tarefa de redução

Etapas do MapReduce

6. A máquina de trabalho responsável pela redução de dados recebe os parâmetros de entrada e escreve seu valor de saída no final do arquivo de saída da redução
7. O mestre desbloqueia o programa do usuário, retornando com o resultado final

Etapas do MapReduce



MapReduce - JobTracker

- É o gerenciador de processamento do MapReduce
- Funções
  - Designar o nó que deve gerenciar determinado dado
  - Verificar falhas
  - Reenviar os dados em caso de falha
  - Reiniciar um nó
  - Trocar o nó que deve processar os dados

MapReduce - TaskTracker

- Responsável por executar as tarefas do MapReduce dentro dos nós escravos
- É executada em máquina virtual
  - é possível criar mais de uma máquina virtual em um mesmo servidor

Contador de Palavras

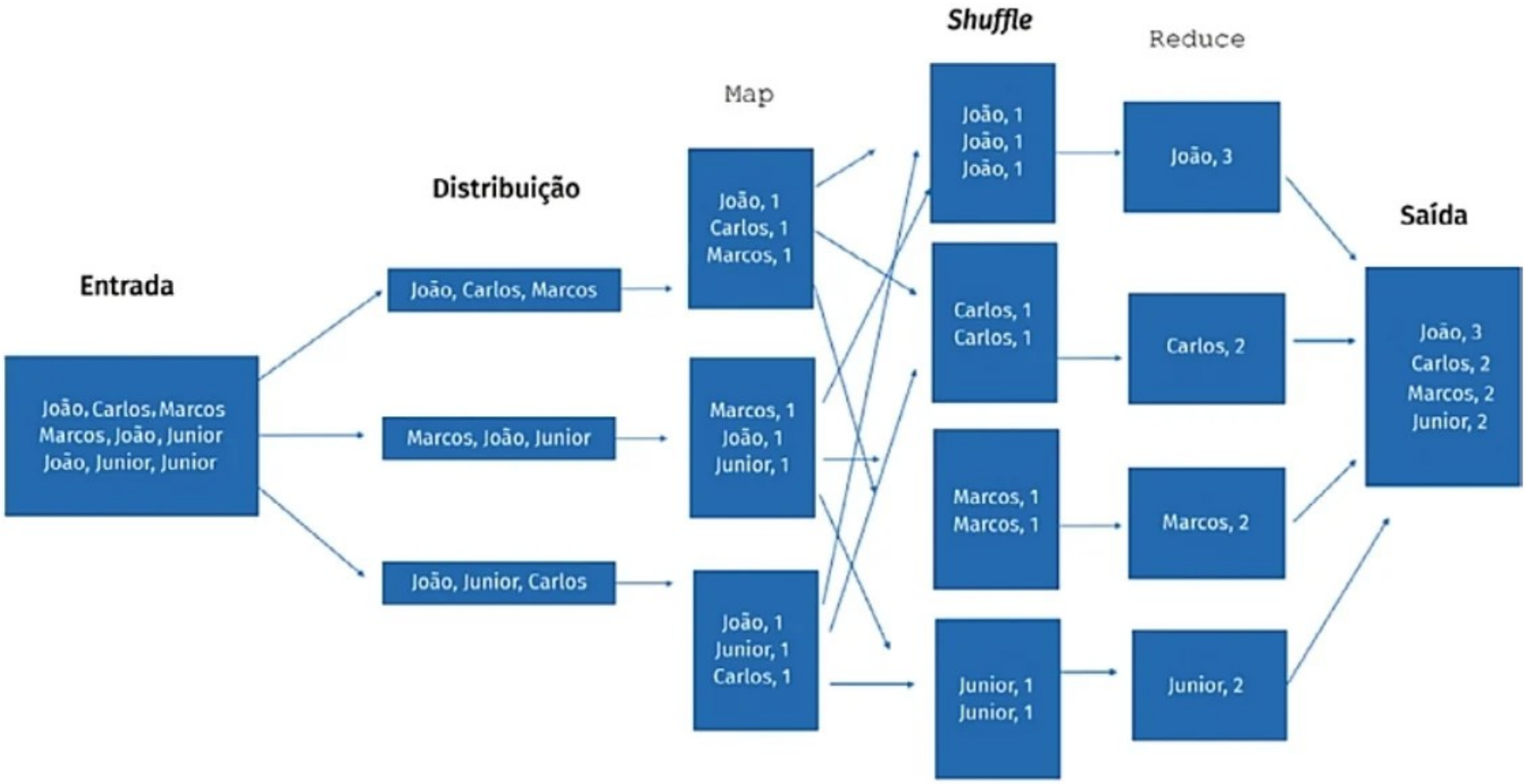

```
Map(String docid, String text):
      for each word w in text:
            Emit(w,1);
```

```
Reduce(String term, Iterator <Int> values):
      int sum = 0;
      for each v in values:
            sum += v;
      Emit(term, sum);
```

Contador de Palavras - Rastreio

- João se matriculou em Big Data. Big Data é um termo que descreve um grande volume de dados. João quer ser um grande cientista de dados.

Contador de Palavras - Rastreio



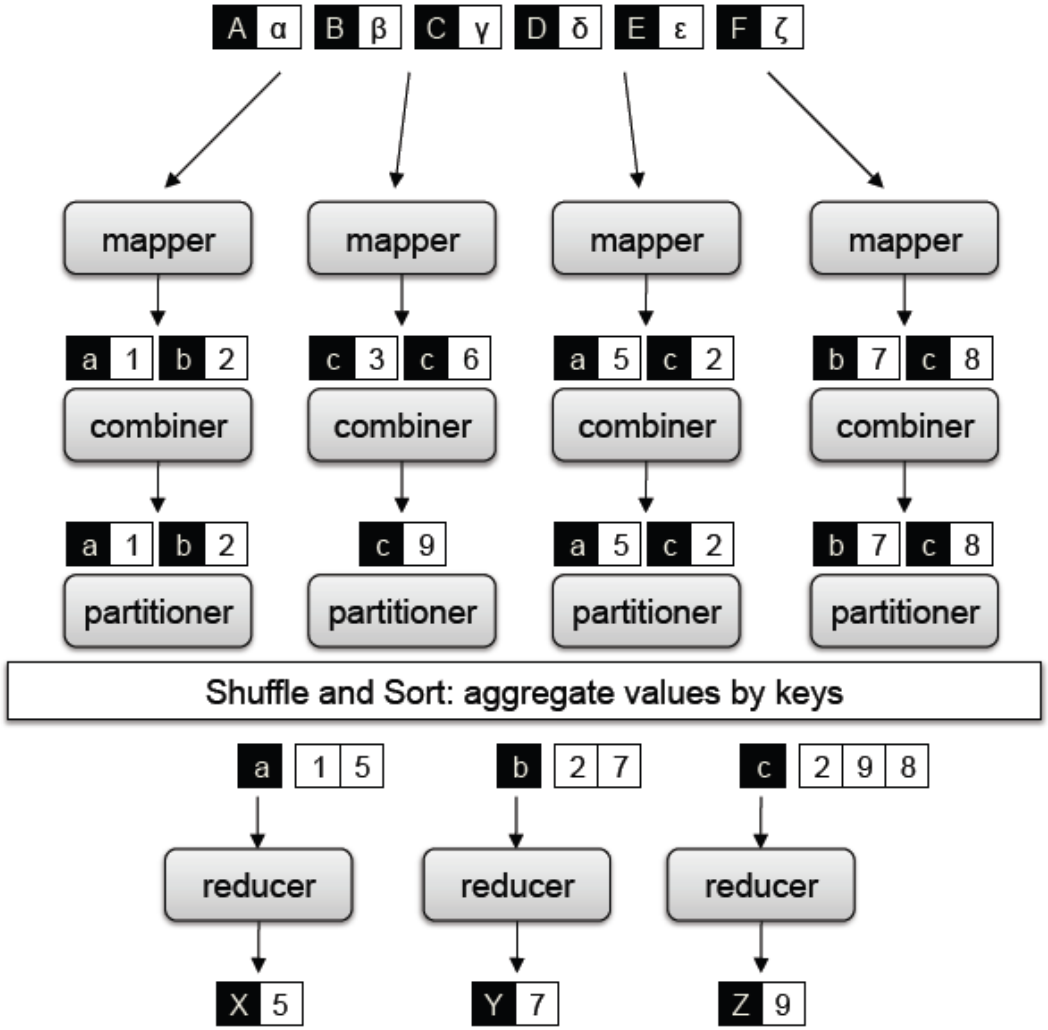
MapReduce - Shuffle

- O objetivo dessa etapa é **agrupar chaves equivalentes** para que seus valores possam ser iterados facilmente na tarefa de redução, gerando a tupla de chave e listando os valores recebidos por ela
- Garante que todos os dados pertencentes a uma mesma chave estejam localizados no mesmo nó

MapReduce - Shuffle

- Eventualmente o programador pode especificar
  - **partition** (k', nº de partições) → partição para k'
    - Divide o espaço de chaves para operações reduce paralelas
    - Ex: hash(k') mod n
  - **combine** (k', v') → <k', v'>*
    - "Mini-reducers" que rodam em memória depois de uma fase de map
    - Otimização para reduzir o tráfego na rede

MapReduce


A α
B β
C γ
D δ
E ε
F ζ
mapper
mapper
mapper
mapper
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
combiner
combiner
combiner
combiner
a 1 b 2
c 9
a 5 c 2
b 7 c 8
partitioner
partitioner
partitioner
partitioner
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 8
reducer
reducer
reducer
X 5
Y 7
Z 9

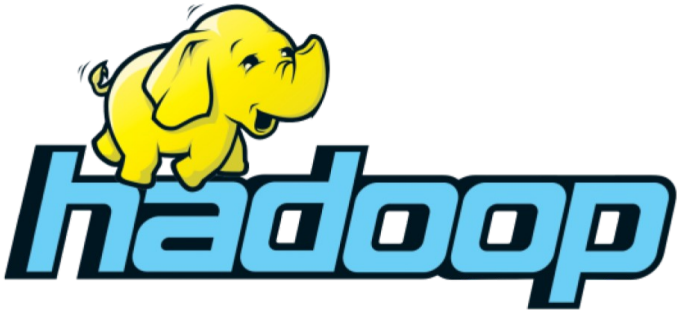
MapReduce

- Na fase do reduce é possível que os dados sejam agregados, filtrados ou combinados de várias maneiras para sumarizar um grande volume de dados
  - Pode-se aplicar algumas funções como mínimo, máximo, média, contador...

Hadoop Básico

Hadoop

- É um framework open source, escalável e tolerante a falhas escrito em Java
- Curiosidade
  - Hadoop é o nome do elefante de pelúcia do filho do criador do projeto Doug Cutting



Hadoop

- O Hadoop é formado por diversos componentes.

  Três desses componentes são os principais

  - **Hadoop YARN**: camada de gerenciamento de recursos
  - **Hadoop Distributed File System (HDFS)**: camada de armazenamento. Altamente tolerante a falhas
  - **Hadoop MapReduce**: camada de processamento de dados

Sistema de Arquivos Distribuído

- Workers são movidos para onde estão os dados
  - Dados são armazenados nos discos locais dos nodos do cluster
  - Inicia cada worker no nó que contém o dado local
- Por que?
  - Assume se que os dados não cabem na RAM
  - Acesso ao disco é lento, mas a taxa de transferência é razoável

Sistema de Arquivos Distribuído

- GFS (Google File System)
  - Sistema proprietário
  - Criado em 2003
- HDFS (Hadoop Distributed File System)
  - Open source
  - Hadoop

GFS (Google File System)

- Arquivos armazenados em chunks (pedaços)
  - Tamanho fixo (64MB)
- Confiabilidade por replicação
  - Cada chunk replicado em 3+ chunkservers
- Nó master único
  - Coordena acesso e mantém metadados

GFS vs HDFS

- Mesmas características básicas
- Diferenças na terminologia
  - GFS master = Hadoop namenode
  - GFS chunkservers = Hadoop datanodes
- Modelo de consistência para appends em arquivo
- Implementação e performance similares

Arquitetura HDFS

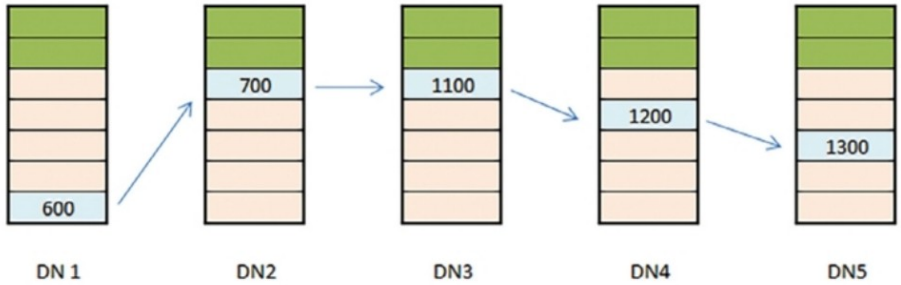
- Armazena os dados em blocos de 64 MB por padrão
  - Pode ser configurado na instalação
- Os blocos são replicados em 3 máquinas por padrão

Arquitetura HDFS

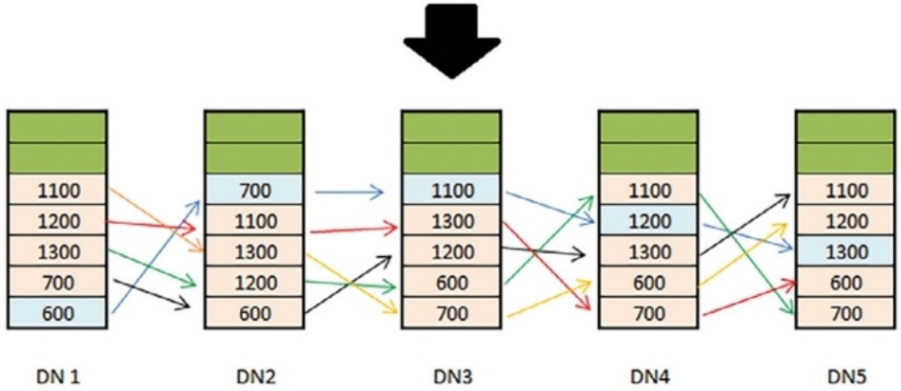
Divisão em blocos arquivo 320 MB

Arquivo 320 MB:

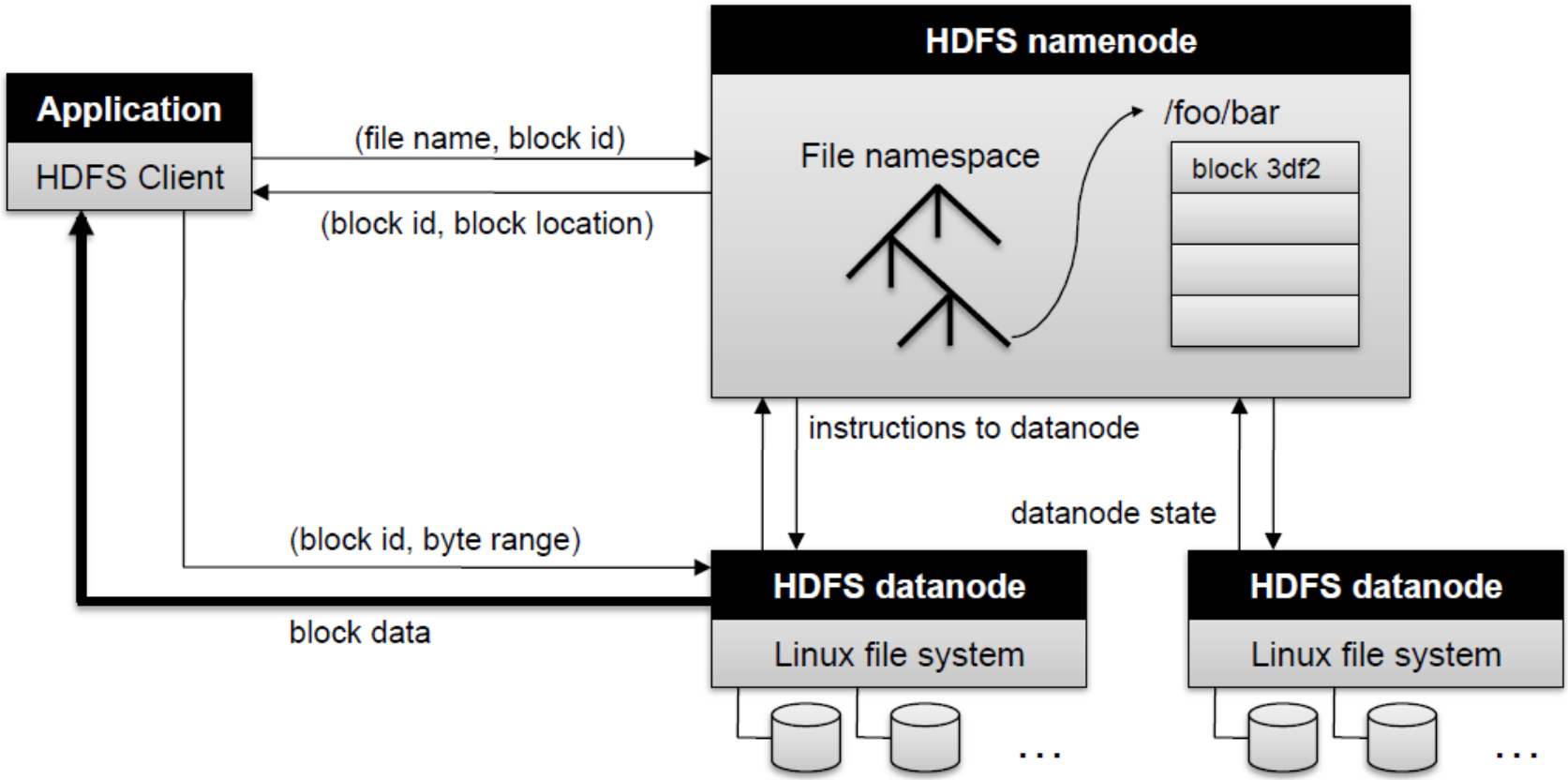
- NameNode verifica espaços em branco
- Divide o arquivo em cinco pedaços de 64 MB e nomeia os blocos com os seguintes IDs (600, 700, 1100, 1200, 1300)
- Define local onde devem ser gravados os blocos originais (em azul)



Esquema de replicação dos blocos



Arquitetura HDFS



Arquitetura HDFS - NameNode

- Também conhecido como master
- Armazena os metadados necessários para a estrutura total do sistema de arquivos
- Funções:
  - Dividir os arquivos em blocos
  - Mapear a localização dos blocos
  - Encaminhar os dados aos nós escravos
  - Gerenciar a localização das réplicas dos dados

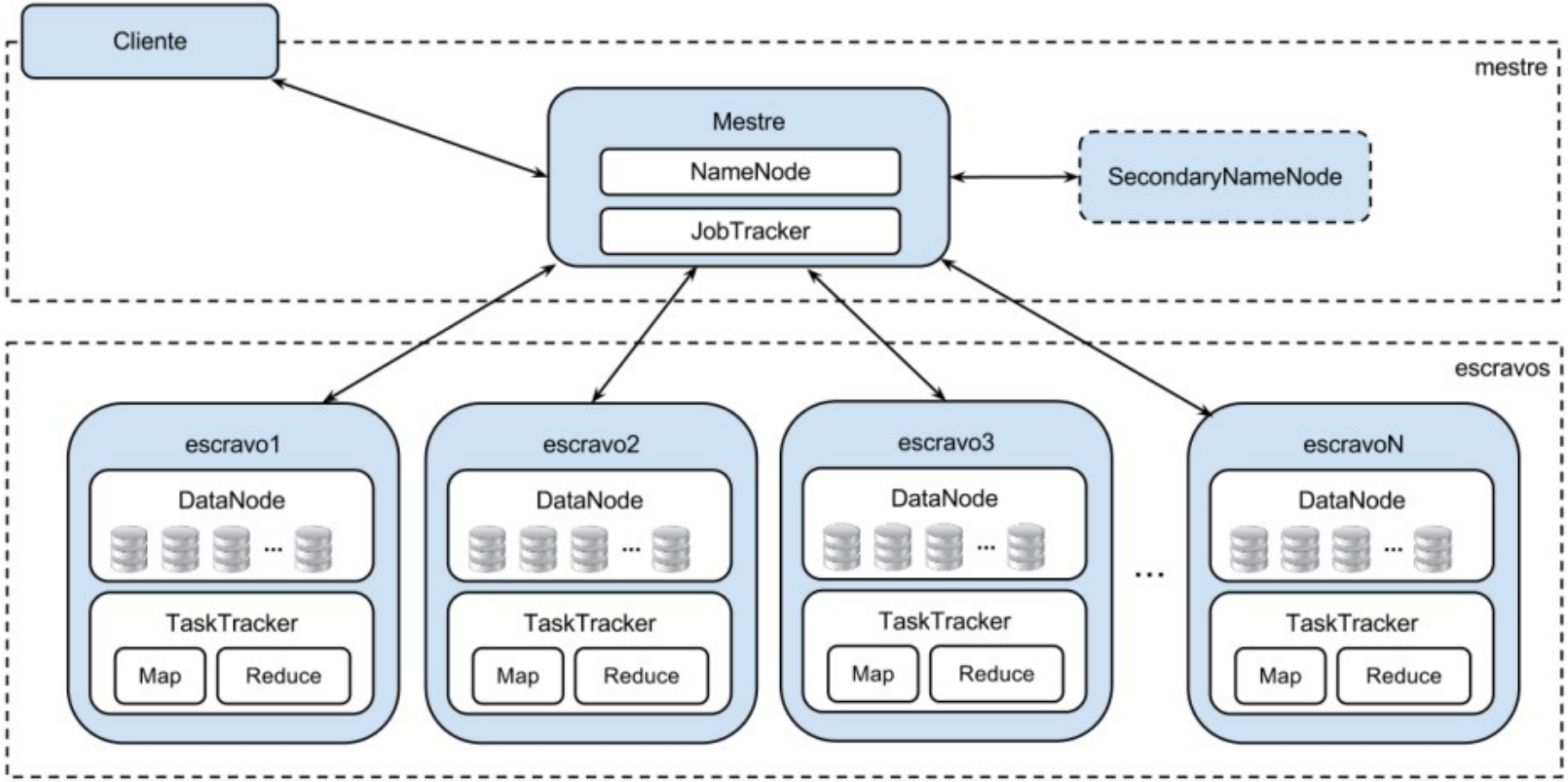
Arquitetura HDFS - DataNode

- Armazena o conteúdo dos arquivos em blocos
- A sua função é transmitir informações constantemente ao NameNode, informando o status dos blocos

Arquitetura HDFS - SecondaryNameNode

- Armazena uma cópia do sistema de arquivos (FSimage) e os arquivos de logs
- Não é uma cópia do NameNode
- Auxilia o NameNode em seu funcionamento, fazendo as checagens e garantindo a sua recuperação em caso de falhas

Em resumo


Cliente
mestre
Mestre
NameNode
JobTracker
SecondaryNameNode
escravos
escravo1
DataNode
TaskTracker
Map
Reduce
escravo2
DataNode
TaskTracker
Map
Reduce
escravo3
DataNode
TaskTracker
Map
Reduce
escravoN
DataNode
TaskTracker
Map
Reduce

Exercícios

- Valor: **2 pontos**
- Deverá ser entregue pelo **SIGAA**
- Pode ser feito no computador ou manuscrito. Se for manuscrito deve-se digitalizar para enviar
- O arquivo a ser enviado deve ser **PDF**

Exercícios

1. O que é um cluster computacional e como é o seu funcionamento? Destaque as vantagens e desvantagens dos cluster computacionais.
2. Cite e explique 4 benefícios de utilizar o framework Hadoop.
3. Explique com suas palavras como funciona o modelo de programação MapReduce para processar dados de forma distribuída.
4. Por que o Hadoop divide seus arquivos em blocos? E por que replica esses blocos ?
5. Descreva brevemente o funcionamento do HDFS do Hadoop.

Instalação Hadoop

Instalação Hadoop

- Existem três formas para se utilizar o Hadoop, sendo elas:
  - **Modo local** (*localhost* ou *standalone mode*)
    - Mais adequado para o desenvolvimento e os testes
  - **Modo pseudodistribuído** (*pseudo-distributed mode*)
    - Vai funcionar como um cluster de uma máquina só
    - Nós escravos são máquinas simuladas
  - **Modo distribuído** (*distributed mode*)
    - Ter um bom conhecimento com o Haddop e habilidades relacionadas a redes de computadores
    - O processo de instalação deve ocorrer em todas as máquinas que vão compor o cluster

Instalação Hadoop

- Distribuição Hadoop: Cloudera QuickStart
- Plataforma: Virtual Box
- Requisitos do Sistema
  - SO 64 bits
  - RAM para a VM: 4GB
  - HD: 20GB

CLOUDERA

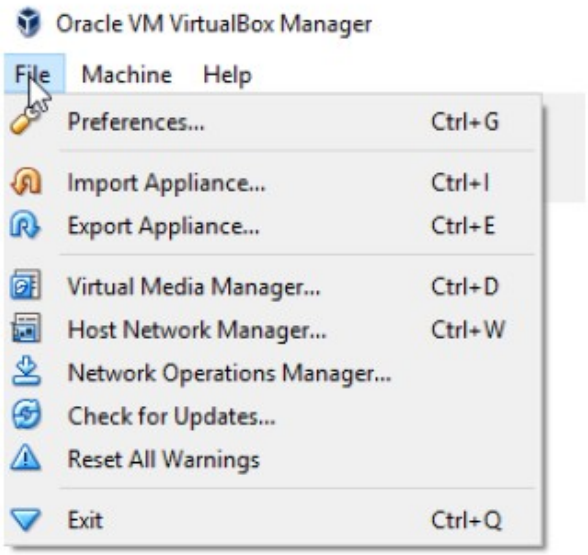
Instalação Cloudera QuickStart

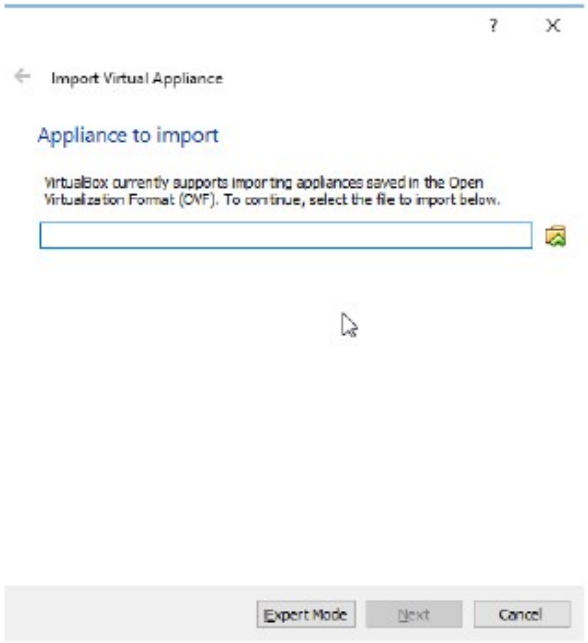
- Tamanho do Download: ~5.5 GB
- Links
  - https://www.virtualbox.org/wiki/Downloads
  - https://downloads.cloudera.com/demo_vm/virtualbox/cloudera-quickstart-vm-5.13.0-0-virtualbox.zip

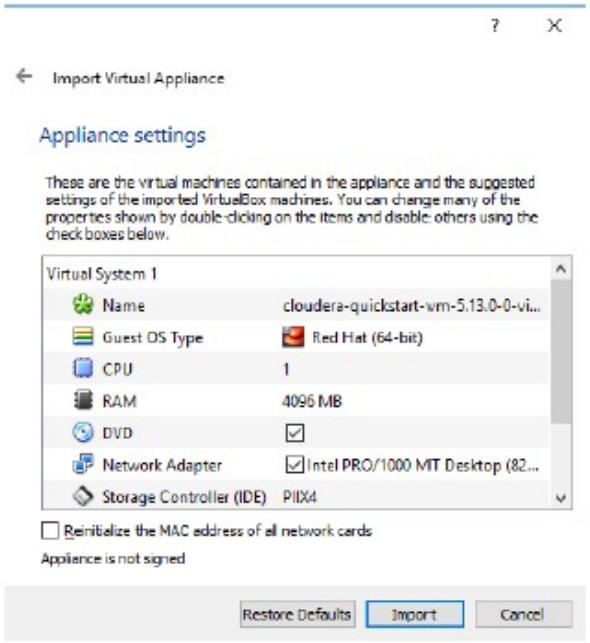
Instalação Cloudera QuickStart

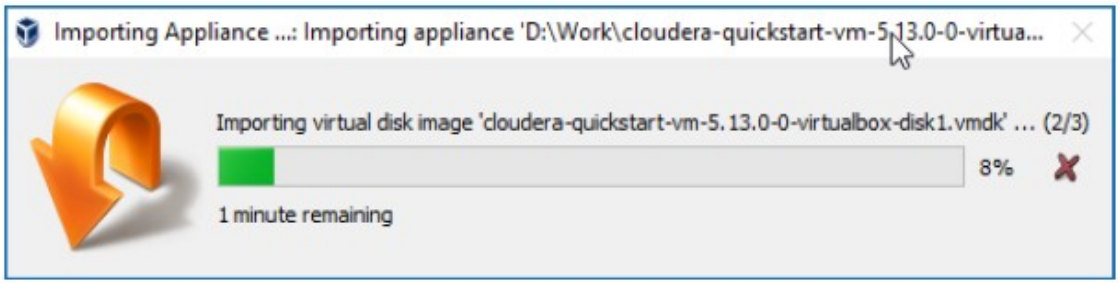
- Instale a VirtualBox (Next, next, finish)
- Descompacte a VM da Cloudera
- Inicie a VirtualBox
- Importe a VM da Claudera (Arquivo → Importe Appliance

Importe a VM da Claudera

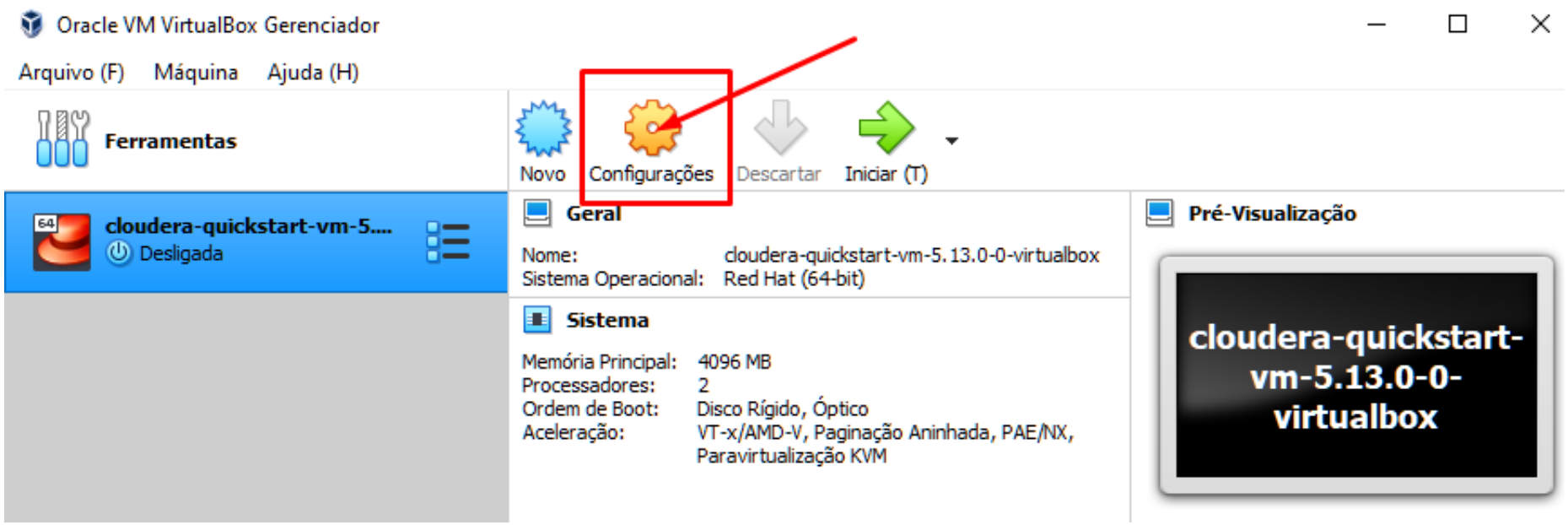




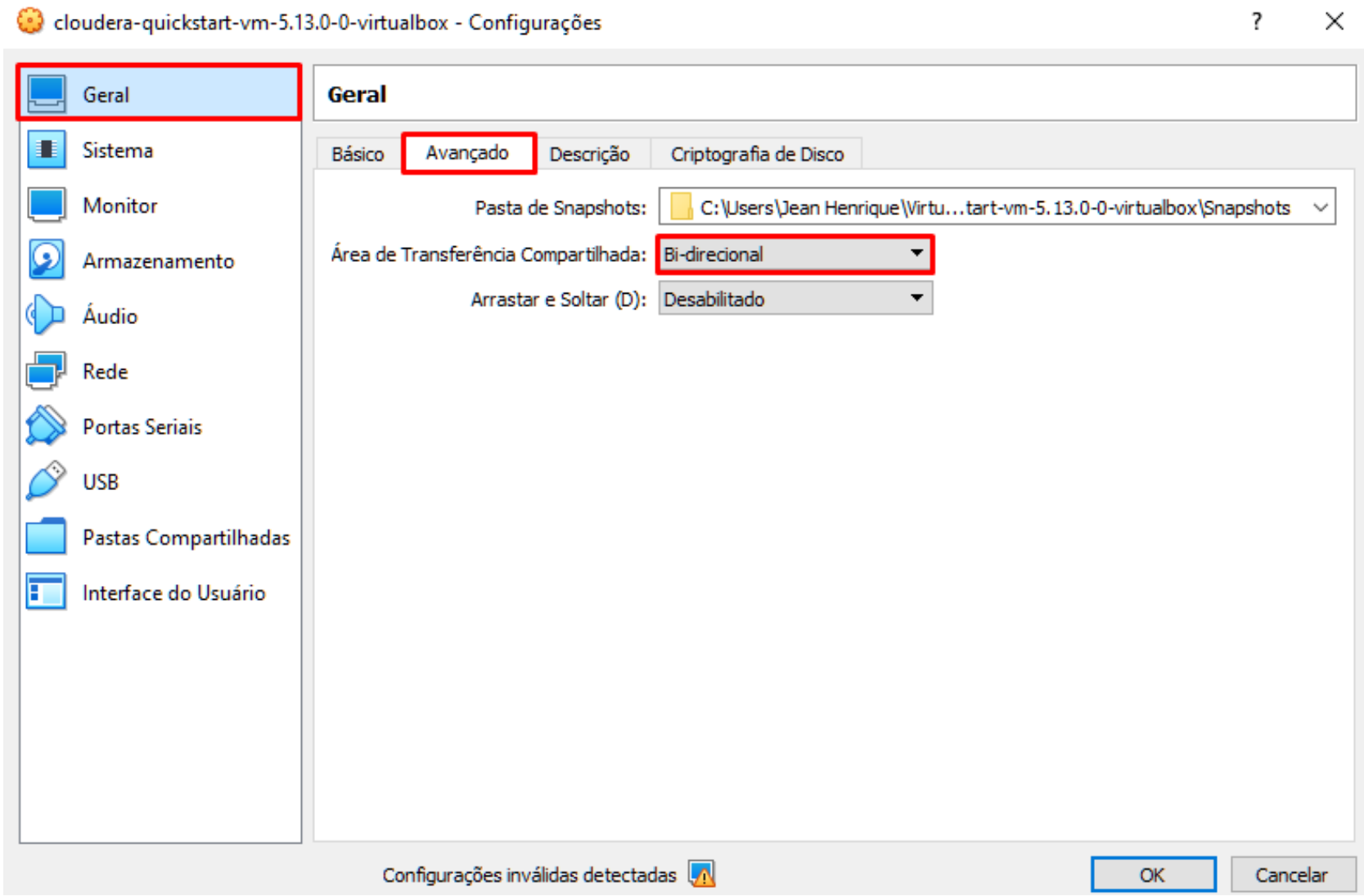




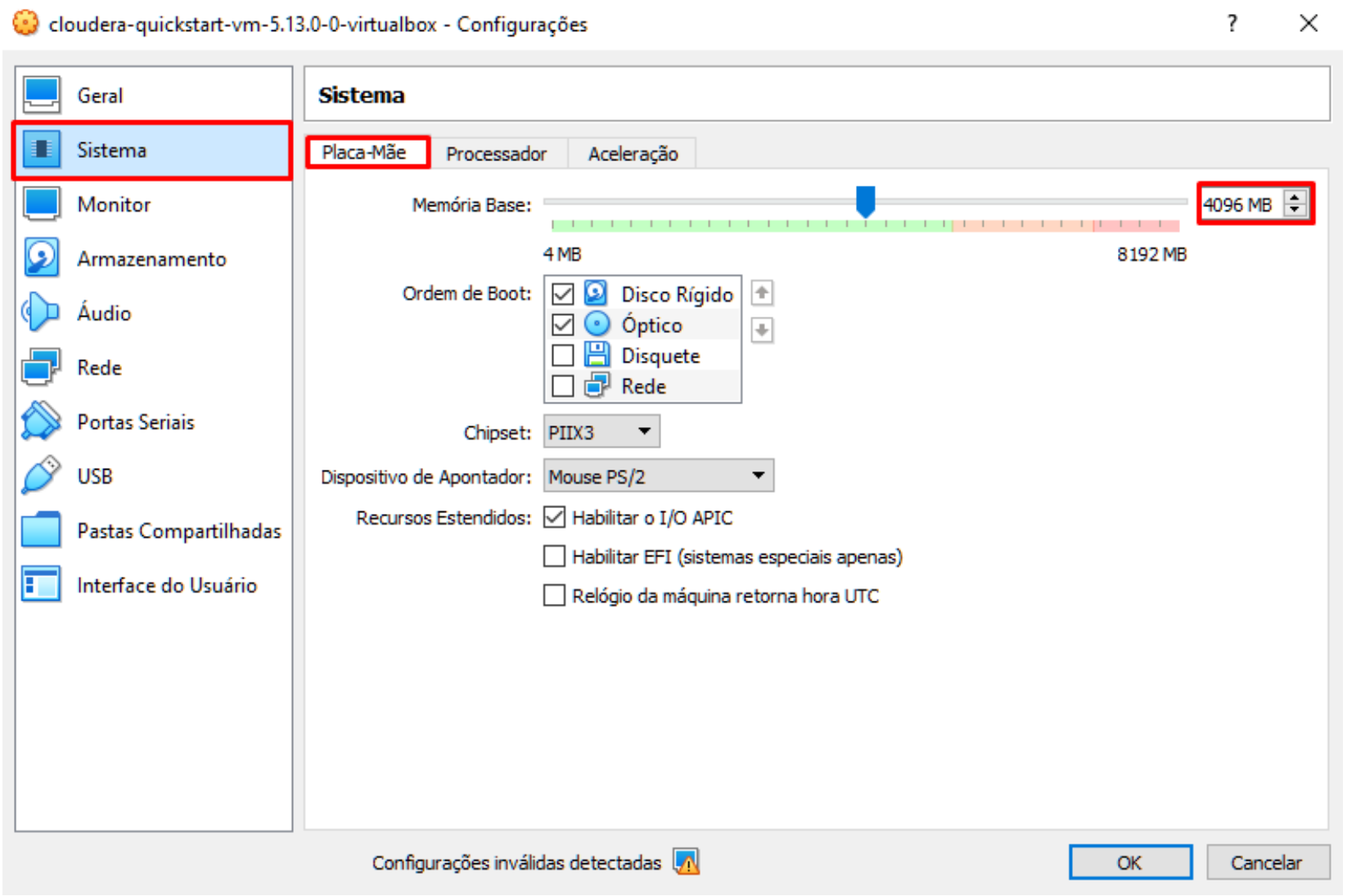
Configurando a VM



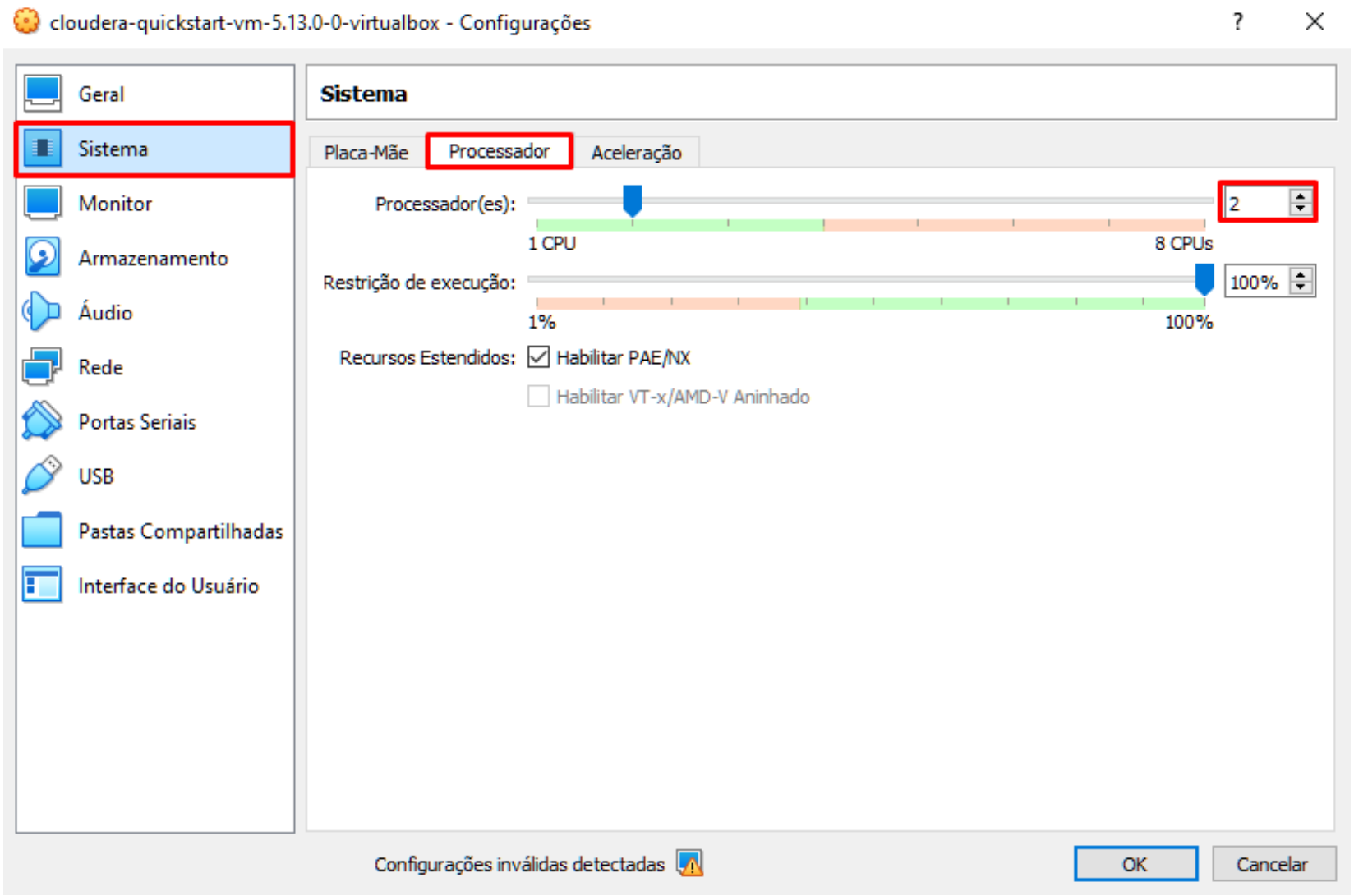
Configurando a VM



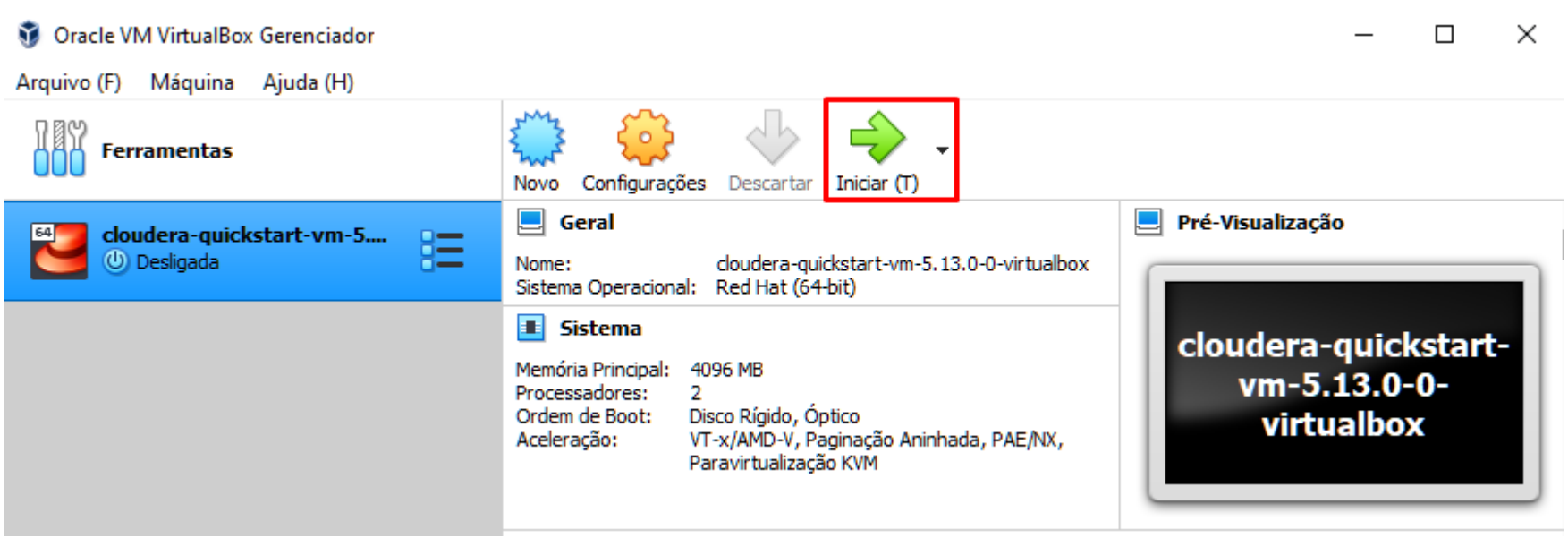
Configurando a VM



Configurando a VM



Inicie a VM da Cloudera

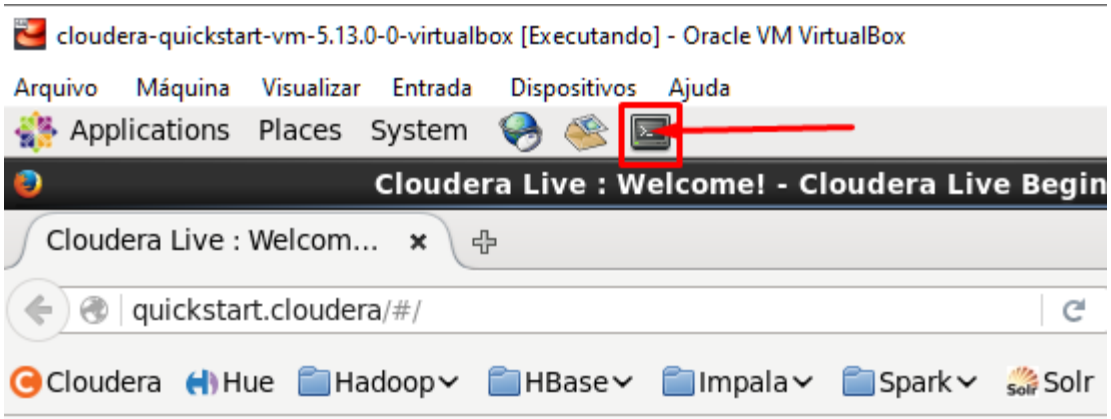


Login: cloudera

Senha: cloudera

Verifique se podemos rodar o Hadoop

- Abra o terminal



- Digite o seguinte comando

  hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar

- Deve exibir uma lista de comandos disponíveis

Contador de Palavras

- Execute o seguinte comando

  hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar **wordcount**

- Resultado

```
[cloudera@quickstart ~]$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-e
xamples.jar wordcount
Usage: wordcount <in> [<in>...] <out>
```

- Deve se informar um ou vários arquivos de entrada e um local para saída

Arquivos com palavras

- O trabalho completo de William Shakespeare

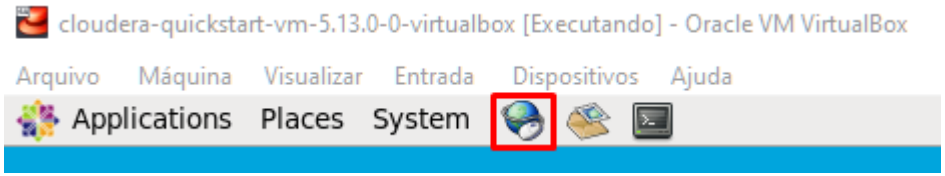
  https://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/files/t8.shakespeare.txt

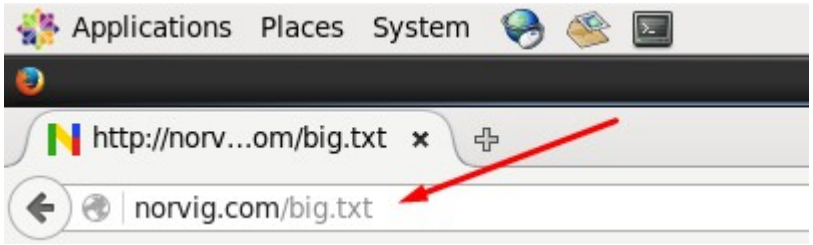
- O Projeto Gutenberg – Ebook das Aventuras de Sherlock Holmes

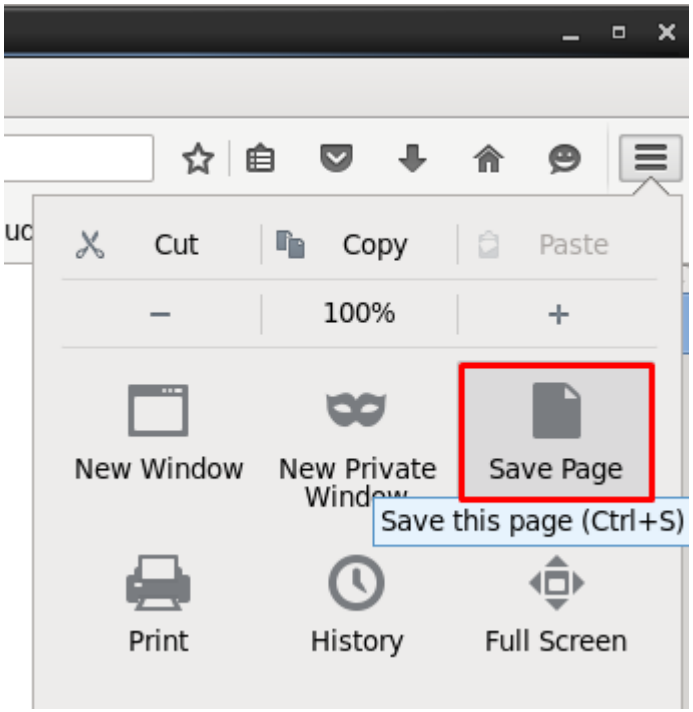
  http://norvig.com/big.txt

- Baixe e salve esses arquivos na pasta Downloads da VM

Arquivos com palavras







Vamos contar as palavras

- Abra o terminal e digite

  hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar **wordcount big.txt out**

- Vai dar um erro

```
InvalidInputException: Input path
does not exist:
```

- Isso aconteceu pois o arquivo não está no HDFS ainda

Copiar os dados para o HDFS

- Acesse a pasta Downloads

  cd Downloads

- Liste os arquivos

  ls

  ls -al

```
[cloudera@quickstart Downloads]$ ls
big.txt  t8.shakespeare.txt
[cloudera@quickstart Downloads]$ ls -al
total 11680
drwxr-xr-x  2 cloudera cloudera    4096 Oct  1 09:40 .
drwxrwxr-x 27 cloudera cloudera    4096 Oct  1 09:14 ..
-rw-rw-r--  1 cloudera cloudera 6488666 Sep 29 16:37 big.txt
-rw-rw-r--  1 cloudera cloudera 5458199 Sep 29 16:53 t8.shakespeare.txt
```

Copiar os dados para o HDFS

- Copie o arquivo do sistema de arquivos local para o HDFS

  hadoop fs -copyFromLocal big.txt

- Verifique se o arquivo foi copiado corretamente

  hadoop fs -ls

- Agora tente copiar o arquivo big.txt para o HDFS novamente

Outras opções de comandos HDFS

- Lista os arquivos da pasta atual

  hadoop fs -ls

- Cria uma cópia do arquivo

  hadoop fs -cp big.txt big2.txt

- Copia os arquivos do HDFS para o sistema de arquivo local

  hadoop fs -copyToLocal big2.txt

- Remove arquivos no HDFS

  hadoop fs -rm big2.txt

- Mostra todas opções de comandos

  hadoop fs

Vamos contar as palavras

- Digite o comando

  hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar **wordcount big.txt out**

- Dessa vez irá funcionar

- Enquanto ele está executando, o Hadoop mostrará o progresso

```
INFO mapreduce.Job:  map 0% reduce 0%
INFO mapreduce.Job:  map 100% reduce 0%
INFO mapreduce.Job:  map 100% reduce 100%
```

Copie o resultado para o FS local

- A saída está armazenada na pasta out no HDFS
- Você pode listar o conteúdo dessa pasta com o comando

  hadoop fs -ls out

- Copie o resultado para o FS local

  hadoop fs -copyToLocal out/part-r-00000

- Agora veja o conteúdo do resultado

  more part-r-00000

Código do contador de palavras


```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
…

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 {
                  …
      }
            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 {
                        …
                  }
                  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");
```

Código do contador de palavras

- Um job Hadoop consiste de uma Mapper e um Reducer
- Quando nós iniciamos o job, temos que especificar o mapper e o reducer
- Na classe WordCount nós temos duas classes aninhadas
  - TokenizerMapper extends Mapper
  - IntSumReducer extends Reducer
- Há um método main que inicia a tarefa
  - Também conhecido como Driver

Função main (Driver)


```
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);
}
```

Função main (Driver)


```
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);
}
```

Nome do Job

Especifica o JAR correspondente ao Job

Classes de saída Chave-Valor

Submissão do Job

Caminhos de entrada e saída

Função main (Driver)

- O método main especifica os atributos do Job
  - Mapper, Reducer e Combiner (opcional)
  - Classes de entrada e saída
  - Local de entrada e saída
- Também submete o job ao framework
  - waitForCompletion

Tokenizer Mapper


```
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);
        }
    }
}
```

Tokenizer Mapper

- A classe Mapper é um tipo genérico com quatro parâmetros:
  - Chave de entrada
  - Valor de entrada
  - Chave de saída
  - Valor de saída
- Um determinado par de entrada pode mapear para zero ou muitos pares de saída
- Dentro da classe Mapper também há um método map para executar a tarefa de map

Tokenizer Mapper

- No WordCount, nós definimos o mapeador como

```
public class TokenCounterMapper
     extends Mapper<Object, Text, Text,
IntWritable>
```

- Isso significa que a classe TokenCounterMapper é uma subclasse de Mapper
- Map processa a entrada linha por linha

Tokenizer Mapper


```
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);
        }
    }
}
```

Valor de saída (1)

Chave de saída

Dado de entrada

Define a chave de saída

Escreve uma linha de saída no buffer de saída

Chave = word
Valor = one

Combiner

- Ás vezes, os resultados do map são grandes
- Os resultados do map podem ser **combinados** antes de serem enviados para o reducer
  - A combinação pode ser feita localmente
  - É um processo opcional de otimização
  - O Hadoop não garante que o combiner será executado
- Combiner não tem interface
  - Deve ter a mesma interface que o reducer
  - No nosso caso, a combinação tem a mesma função de redução

```
job.setCombinerClass(IntSumReducer.class);
```

Integer Sum Reducer


```
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);
   }
}
```

Classe Reducer

- A classe Reducer é um tipo genérico com quatro parâmetros:
  - Chave de entrada
  - Valor de entrada
  - Chave de saída
  - Valor de saída
- A entrada do reducer é uma chave e uma lista correpondente de valores
- Dentro da classe Reducer também há um método reduce para executar a tarefa de reduce

Classe Reducer

- No WordCount nós definimos o reducer como

```
public static class IntSumReducer
extends Reducer
<Text,IntWritable,Text,IntWritable>
```

- Isso significa que a classe IntSumReducer é uma subclasse de Reducer

Preparando o Ambiente de Compilação

- Muitas variáveis de ambiente já existem na VM da Cloudera QuickStart, para vê-las digite:

  printenv

- As seguintes variáveis devem estar lá

```
JAVA_HOME=/usr/java/jdk1.7.0_67-cloudera
PATH=/usr/java/jdk1.7.0_67-cloudera/bin
```

- O que nós temos que fazer é definir isto:

  export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar

Compilando o contador de palavras

- Para compilar

  hadoop com.sun.tools.javac.Main WordCount.java

- O resultado será vários arquivos classes

- Nós temos que colocá-los dentro de um único arquivo JAR

  jar cf wc.jar WordCount*.class

- O resultado será um arquivo: wc.jar

Rodando o contador de palavras

- Contando palavras do arquivo big.txt

  hadoop jar wc.jar WordCount big.txt out2

- Você tem que ter o mesmo resultado do exemplo anterior

- O resultado está armazenado na pasta out2

- Faça uma cópia para o sistema de arquivos local

  hadoop fs -copyToLocal out2

Exercícios

- Valor: **2 pontos**
- Deverá ser entregue pelo **SIGAA**
- Pode ser feito no computador ou manuscrito. Se for manuscrito deve-se digitalizar para enviar
- O arquivo a ser enviado deve ser **PDF**

Exercícios

1. Edite o mapeador para contar linhas ao invés de palavras
2. Edite o mapeador para contar o número de todas as palavras (não de cada palavra)
3. Edite o mapeador para contar linhas e palavras
4. Limpe o resultado do contador de palavras

   Atualmente os resultados da contagem de palavras são muito redundantes porque as palavras podem conter caracteres especiais (Ex.: “Caesars., Caesars’, “Caesars,”)

   Remova esses caracteres especiais

   https://www.alura.com.br/artigos/trocando-caracteres-de-uma-string-no-java

5. Maiúsculas e minúsculas

   Algumas palavras podem conter caracteres maiúsculos e/ou minúsculos

   Conte as palavras ignorando esses casos

   https://www.devmedia.com.br/forum/converter-string-para-maisculo-e-minusculo/566327

6. Faça a média de palavras por linha

Dúvidas?

jean.camara@ifsudestemg.edu.br