



Módulo 3 - Solução de Dados utilizando Ecossistema Hadoop

Capítulo 1. Modelo MapReduce

Prof. João Paulo Barbosa Nascimento



Aula 1.1. História e motivação



Nesta aula

- ❑ Modelo MapReduce.
- ❑ Motivação para criação.

O modelo MapReduce.

- Desafio: a grande quantidade de dados.
- Dificuldade em escrever programas distribuídos.
- Programas:
 - Não escaláveis.
 - Não tolerantes a falhas.

O modelo MapReduce

- Quem é responsável por toda essa produção de dados?
 - Redes sociais (Facebook, Twitter, etc).
 - Experimentos científicos (LHC, Projeto Genoma, etc).
 - Blogs, notícias, fotos, vídeos, etc.
 - Sensores, câmeras de monitoramento, etc.
 - Aumento massivo de espaço nos Hard Disks.
 - Aumento da velocidade de acesso aos dados.
 - Queda no custo do armazenamento.



O modelo MapReduce

- Grande quantidade de dados (Big Data).
 - Necessário uma infraestrutura para:
 - Gerenciar e manipular esses dados.
 - Tolerante a falhas.
 - Adaptável aos diversos problemas rotineiros.
 - Com alta disponibilidade e escalável.
 - Fácil de programar.
 - Paralelizado e distribuído.



O modelo MapReduce

- Em 2003 o Google propôs uma ferramenta:
 - Altamente escalável e tolerante a falhas.
 - Com sistema próprio de arquivos distribuído (Google File System – GFS).
- Composto por duas funções principais (Map e Reduce).
- Trazendo simplicidade no desenvolvimento.
- Modelo MapReduce.

O modelo MapReduce

- “Novo paradigma de programação”.
- O usuário programa duas funções: Map e Reduce.
- O sistema paralelizará a execução dessas funções.
- Com tolerância a falhas e balanceamento de carga.
- Com sistema próprio de armazenamento distribuído.
- Necessário modelar o problema nesse formato.

Conclusão

- Criação do modelo MapReduce.
- Solução para a distribuição, escalabilidade e tolerância a falhas.

Próxima aula

- ❑ Funcionamento do modelo MapReduce.



Aula 1.2. Exemplo de funcionamento



Nesta aula

- ❑ Funcionamento do modelo MapReduce.
- ❑ Exemplo de execução.
- ❑ Funções Map e Reduce.

Exemplo de funcionamento do MapReduce

- Contexto do exemplo:
 - Rede de sensores.
 - Monitora o nível de CO₂ (dióxido de carbono) na atmosfera.
 - Sensores em diversos pontos do globo terrestre.
 - Coletando dados o tempo todo.
 - Gerando grande massa de dados.
 - Com o crescimento dos dados a pesquisa sequencial fica inviável.



Exemplo de funcionamento do MapReduce

- Modelo MapReduce torna-se um aliado para auxiliar nesse processamento.
- Sensores espalhados fazem as medições e armazenam os resultados em arquivos texto.
- Esse problema é candidato a ser resolvido pelo modelo MR.
- Dados estruturados dentro do arquivo em posições específicas.
- Cada medição consiste em uma linha do arquivo.

Exemplo de funcionamento do MapReduce

- Dados do site CO2 Now (<http://www.co2now.org>).
- Focaremos na data da medição e no nível do CO2.

```
1988 04121402 35707
1988 04251405 35727
1988 05031922 35823
1988 05091357 35719
1987 02231400 35412
1987 03021349 35267
1987 03091400 35350
1987 03161359 35418
1986 01201523 35129
1986 01271518 35520
1986 02031455 35328
1986 02101622 35257
1985 05061943 35234
1985 06241908 35056
1985 07011900 34967
1985 08261528 33733
```

Exemplo de funcionamento do MapReduce

- Queremos saber o maior nível de CO₂ por ano.
- Função Map recebe as linhas e extrai ano e nível de CO₂.

(1988, 357.07)

(1988, 357.27)

(1988, 358.23)

(1988, 357.19)

(1987, 354.12)

(1987, 352.67)

(1987, 353.50)

(1987, 354.18)

(1986, 351.29)

(1986, 355.20)

(1986, 353.28)

(1986, 352.57)

(1985, 352.34)

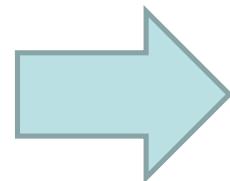
(1985, 350.56)

(1985, 349.67)

(1985, 337.33)

Exemplo de funcionamento do MapReduce

```
1988 04121402 35707  
1988 04251405 35727  
1988 05031922 35823  
1988 05091357 35719  
1987 02231400 35412  
1987 03021349 35267  
1987 03091400 35350  
1987 03161359 35418  
1986 01201523 35129  
1986 01271518 35520  
1986 02031455 35328  
1986 02101622 35257  
1985 05061943 35234  
1985 06241908 35056  
1985 07011900 34967  
1985 08261528 33733
```



(1988, 357.07)
(1988, 357.27)
(1988, 358.23)
(1988, 357.19)
(1987, 354.12)
(1987, 352.67)
(1987, 353.50)
(1987, 354.18)
(1986, 351.29)
(1986, 355.20)
(1986, 353.28)
(1986, 352.57)
(1985, 352.34)
(1985, 350.56)
(1985, 349.67)
(1985, 337.33)

Resultado da função Map.

Exemplo de funcionamento do MapReduce

- Map ordena os dados pelo par chave-valor.
- A função Reduce receberá o seguinte conjunto de dados:

(1985, [337.33, 349.67, 350.56, 352.34])

(1986, [351.29, 352.57, 353.28, 355.20])

(1987, [352.67, 353.50, 354.12, 354.18])

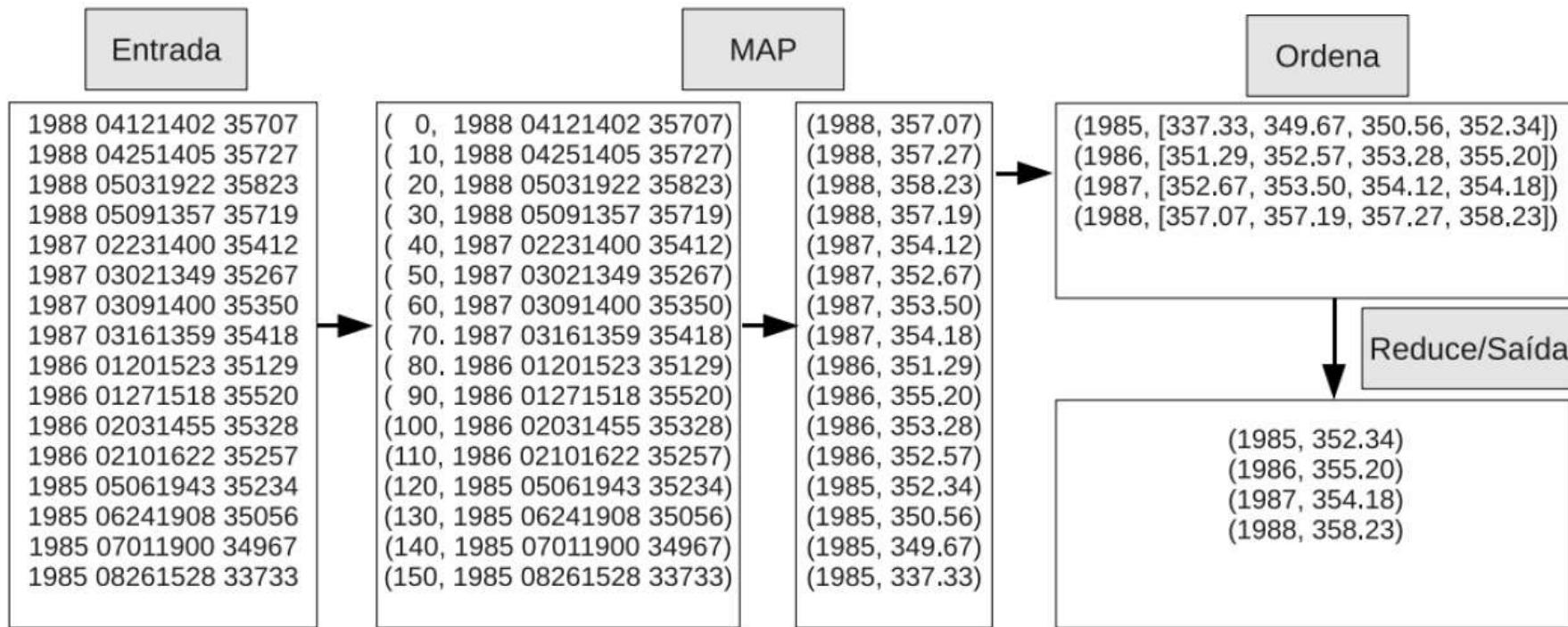
(1988, [357.07, 357.19, 357.27, 358.23])

Exemplo de funcionamento do Hadoop

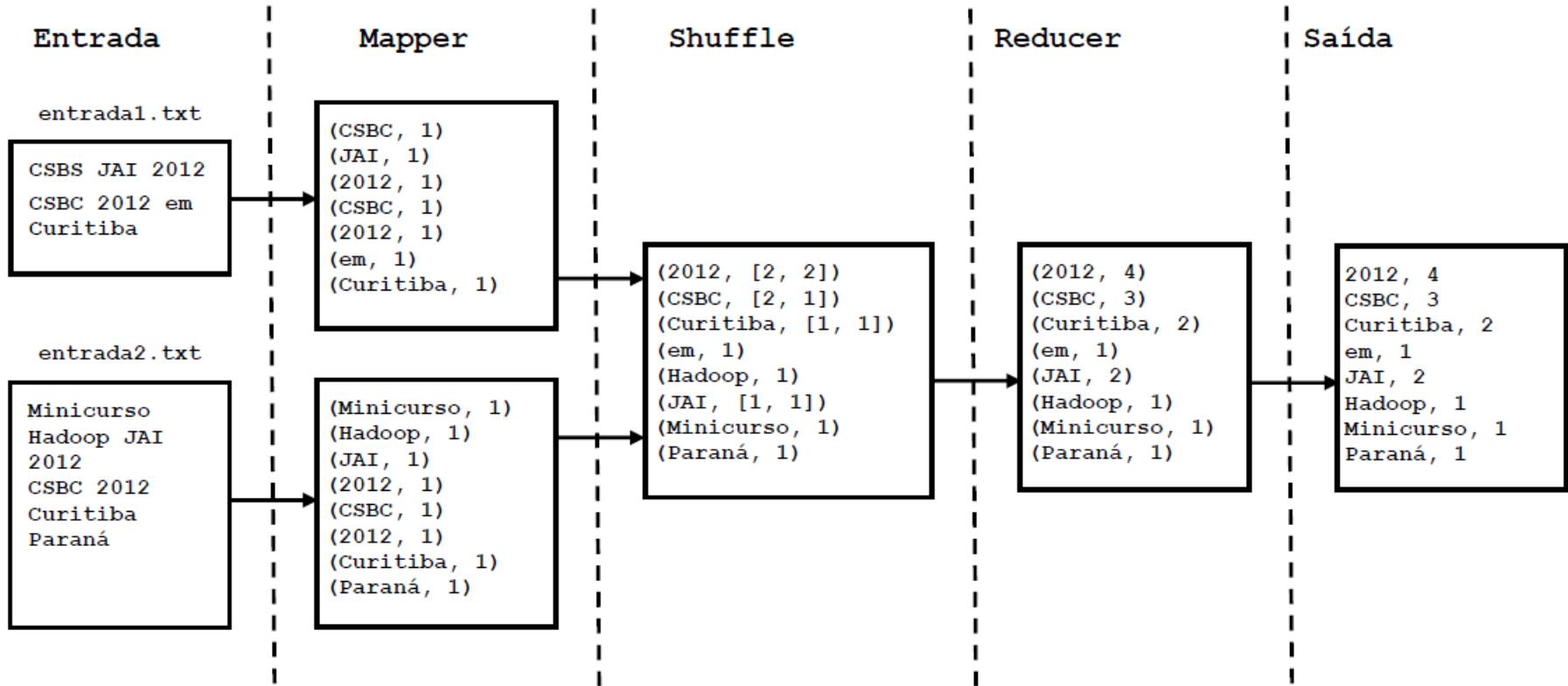
- Cada ano pode ser processado por uma função Reduce.
- Ao final, a seguinte saída será produzida pela função Reduce:

(1985, 352.34)
(1986, 355.20)
(1987, 354.18)
(1988, 358.23)

Exemplo de funcionamento do MapReduce



Exemplo de funcionamento do Hadoop



Conclusão

- Funcionamento do modelo.
- Entrada / Saída.
- Funções Map e Reduce.



Módulo 3 - Solução de Dados utilizando Ecossistema Hadoop

Capítulo 2. Conhecendo o Hadoop

Prof. João Paulo Barbosa Nascimento



Aula 2.1. História



Nesta aula

- Criação do Hadoop.

- Foi criado por Doug Cutting, criador do Apache Lucene.
- Lucene: uma biblioteca de busca textual amplamente utilizada.
- O Hadoop teve suas origens no Apache Nutch.
- Um motor de busca na web open-source.
- O Apache Nutch é uma parte do Apache Lucene.



- O Apache Nutch começou em 2002, juntamente com um crawler e um sistema de busca.
- Percebeu-se que a arquitetura implementada não escalaria para bilhões de páginas.
- A ajuda para isso estava em um artigo publicado em 2003, pelo Google.



- Esse artigo descrevia a arquitetura do Google's Distributed FileSystem (GFS).
- GFS resolveria a necessidade de armazenamento de arquivos muito grandes, gerado pela coleta e indexação da web.
- Em 2004, começaram a escrever uma implementação em código aberto do GFS.



- Surgiu aí o Nutch Distributed FileSystem (NDFS).
- Em 2004 o Google publicou o documento que apresentou o modelo MapReduce ao mundo.
- Em 2005, os desenvolvedores implementaram o modelo MapReduce no Nutch.



- No meio de 2005, todos os principais algoritmos do Nutch foram portados para executar usando o MapReduce e o NDFS.
- NDFS e o MapReduce no Nutch foram além do ambiente acadêmico.
- Em 2006, eles saíram do Nutch para formarem um projeto independente.
- Surgiu o projeto Hadoop.



- Na mesma época, Doug Cutting foi para o Yahoo.
- O Yahoo forneceu uma equipe para transformar o Hadoop em um sistema, para funcionar na escala da web.
- Em 2008 o Yahoo anunciou que sua pesquisa contava com um cluster Hadoop com 10.000 núcleos.



- Ainda em 2008, o Hadoop ganhou um projeto de alto nível na Apache.
- Muitas outras empresas passaram a utilizar.
- Yahoo!, Facebook e New York Times.
- Em abril de 2008, o Hadoop quebrou o recorde mundial, se tornando o sistema mais rápido para classificar 1TB de dados.
- 910 nós em 209 segundos.



- Em novembro de 2008, o Google informou que a sua implementação do MapReduce ordenou 1TB em 68 segundos.
- Em abril de 2009, foi anunciado pelo Yahoo a ordenação de 1TB em 62 segundos.
- Em uma competição em 2014, uma equipe da Databricks usou um cluster Spark com 207 nós para classificar 1TB em 1406 minutos.
- 4,27 TB por minuto.



- Hoje, o Hadoop é amplamente utilizado por diversas organizações.
- Ferramenta para análise e armazenamento de Big Data.
- Grandes empresas e produtos dão suporte ao Hadoop.
- IBM, Oracle, Microsoft, etc.
- Cloudera, Hortonworks, etc.



Conclusão

- ✓ Surgimento e evolução do Hadoop.
- ✓ Armazenamento e processamento distribuído.
- ✓ Atualmente é amplamente utilizado.

Próxima aula

- ❑ Ecossistema Hadoop.



Aula 2.2. Ecossistema



- Ecossistema.
- Componentes.

- Arcabouço (framework) que permite processamento:
 - Paralelo.
 - Distribuído.
- Grandes massas de dados.
- Utiliza clusters de computadores.



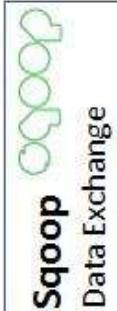
- Proporciona escalabilidade.
- Modelo de programação simplificado:
 - Paralelismo e distribuição automática (por conta do framework).
 - Desenvolvedor pode focar seus esforços no negócio.



- Hadoop Common:
 - Aplicativos comuns à todos os outros módulos do ambiente.
- Hadoop Distributed File System (HDFS):
 - Sistema de arquivos distribuído.



Apache Hadoop Ecosystem



Scoop
Data Exchange



Zookeeper
Coordination



Oozie
Workflow



Pig
Scripting



Mahout
Machine Learning

R Connectors
Statistics



Hive
SQL Query



Hbase
Columnar Store



YARN Map Reduce v2
Distributed Processing Framework

Flume
Log Collector

HDFS

Hadoop Distributed File System



- Ambari: gerenciamento e monitoramento do cluster.
- Flume: coletor de logs.
- Sqoop: transferência de dados.
- PIG: Script.
- Hive: Data Warehouse.
- Hbase: Banco de dados não-relacional, distribuído e orientado a colunas.
- Yarn: gerenciador de execução.
- Oozie: agendamento de tarefas Hadoop.

- Hadoop Yarn:
 - Framework para distribuição de tarefas.
 - Gerenciamento dos recursos do cluster.
 - Inserido a partir da versão 2.
- Hadoop MapReduce:
 - Framework para escrita de aplicações para processamento de grandes massas de dados.
 - Aplicações paralelas, distribuídas e tolerantes a falhas.

- Ambari:
 - Ferramenta para gerenciamento e monitoramento do cluster.
 - Consumo de CPU, memória e rede.
 - Máquina por máquina.



Apache Ambari

- Hbase:
 - Banco de dados não-relacional e distribuído.
 - Suporta o armazenamento estruturado em grandes tabelas (BigTable).



Ecossistema

- Hive:
 - Estrutura para Data Warehouse.
 - Suporta consultas SQL, indexação e grandes conjuntos de dados.
- Mahout:
 - Biblioteca que fornece algoritmos de aprendizado de máquina.
 - Mineração de dados (classificação, agrupamento, etc.).



- Spark:
 - Estrutura do Hadoop que prioriza o processamento em memória.
 - Possui algoritmos de aprendizado de máquina, grafos e streaming.
- Zookeeper:
 - Coordenador de serviços altamente distribuído.
 - Coordena locks, sincronização e grupos de serviços para aplicações distribuídas.



Apache ZooKeeper™

Conclusão

- Componentes.
- Ecossistema Hadoop.

Próxima aula

YARN.



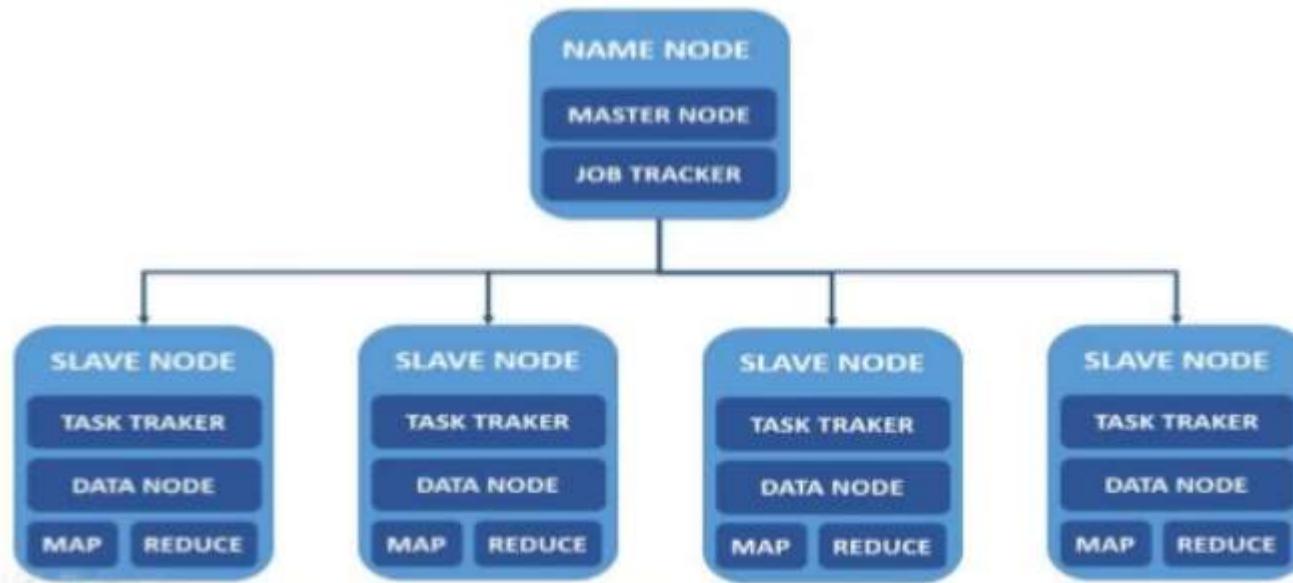
Aula 2.3. YARN



Nesta aula

- ❑ YARN.
- ❑ Arquitetura, importância e funcionamento.

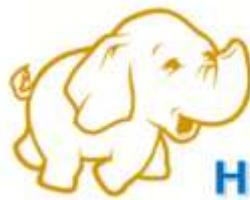
- Arquitetura mestre/escravo.



- YARN (**Y**et **A**nother **R**esource **N**avigator).
- No Hadoop versão 1.0 (versão 1 do MapReduce), o MapReduce desempenhava as funções de processamento e gerenciamento de recursos.
- O mestre alocava os recursos, agendava e monitorava o processamento.
- Esse desenho resultou em um gargalo de escalabilidade.
- O Yahoo! Encontrou o limite em 5000 nós e 40.000 tarefas simultâneas.

- A utilização de recursos computacionais era ineficiente.
- Para superar esses problemas, o YARN foi introduzido na versão 2.0 do Hadoop (2012).
- A ideia por trás do YARN é aliviar o MapReduce.
- O YARN assume a responsabilidade do gerenciamento de recursos e agendamento de tarefas.

- O YARN começou a fornecer ao Hadoop a capacidade de executar tarefas não MapReduce dentro da estrutura.
- Revolucionou o ecossistema Hadoop.
- O Hadoop tornou-se muito mais flexível, eficiente e escalável.
- Em 2013, o Yahoo implantou o YARN.
- Reduziu o tamanho de seu cluster de 40.000 para 32.000 nós.
- O número de trabalhos dobrou para 26 milhões por mês.



Hadoop v1.0

MapReduce

Data Processing
& Resource Management

HDFS

Distributed File Storage



Hadoop v2.0

MapReduce

Other Data
Processing
Frameworks

YARN

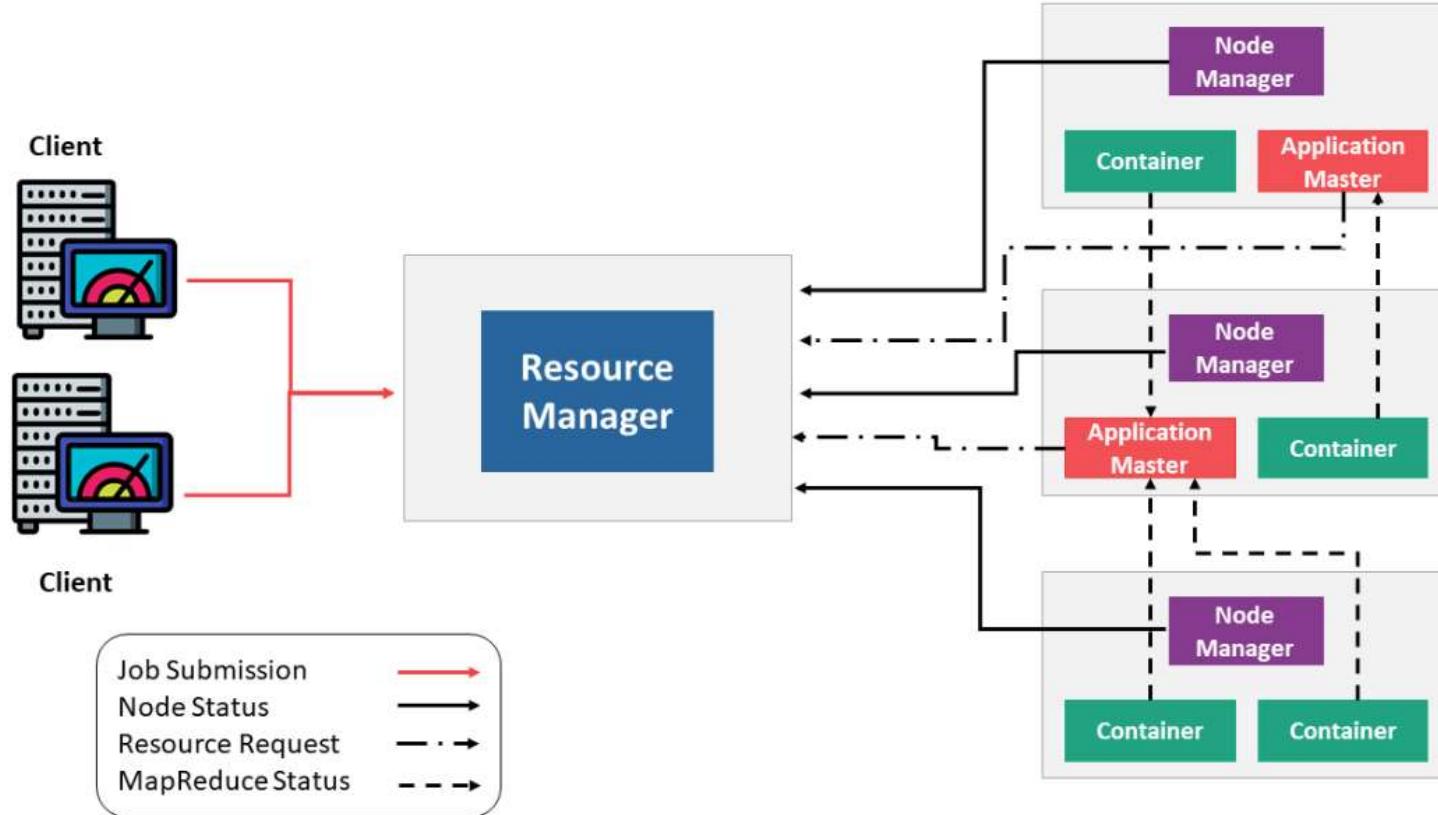
Resource Management

HDFS

Distributed File Storage

- O YARN permitiu a execução de operações usando uma maior variedade de ferramentas:
 - Spark (processamento em tempo real).
 - Hive (para consultas SQL).
 - Hbase (para noSQL).

- A arquitetura YARN consiste nos seguintes componentes principais:
 - **Resource Manager**: executa no mestre e gerencia a alocação de recursos no cluster.
 - **Node Manager**: executa nos escravos e são responsáveis pela execução das tarefas em cada um dos nós.
 - **Application Master**: gerencia o ciclo de vida do job e os recursos necessários a cada aplicação. Monitora as execuções das tarefas.
 - **Container**: pacote de recursos, incluindo memória RAM, CPU, rede, etc. (em um nó)



Conclusão

- ✓ YARN – Gerenciador de recursos.
- ✓ MapReduce x YARN.

Próxima aula

- ❑ Funcionamento dos componentes do YARN.



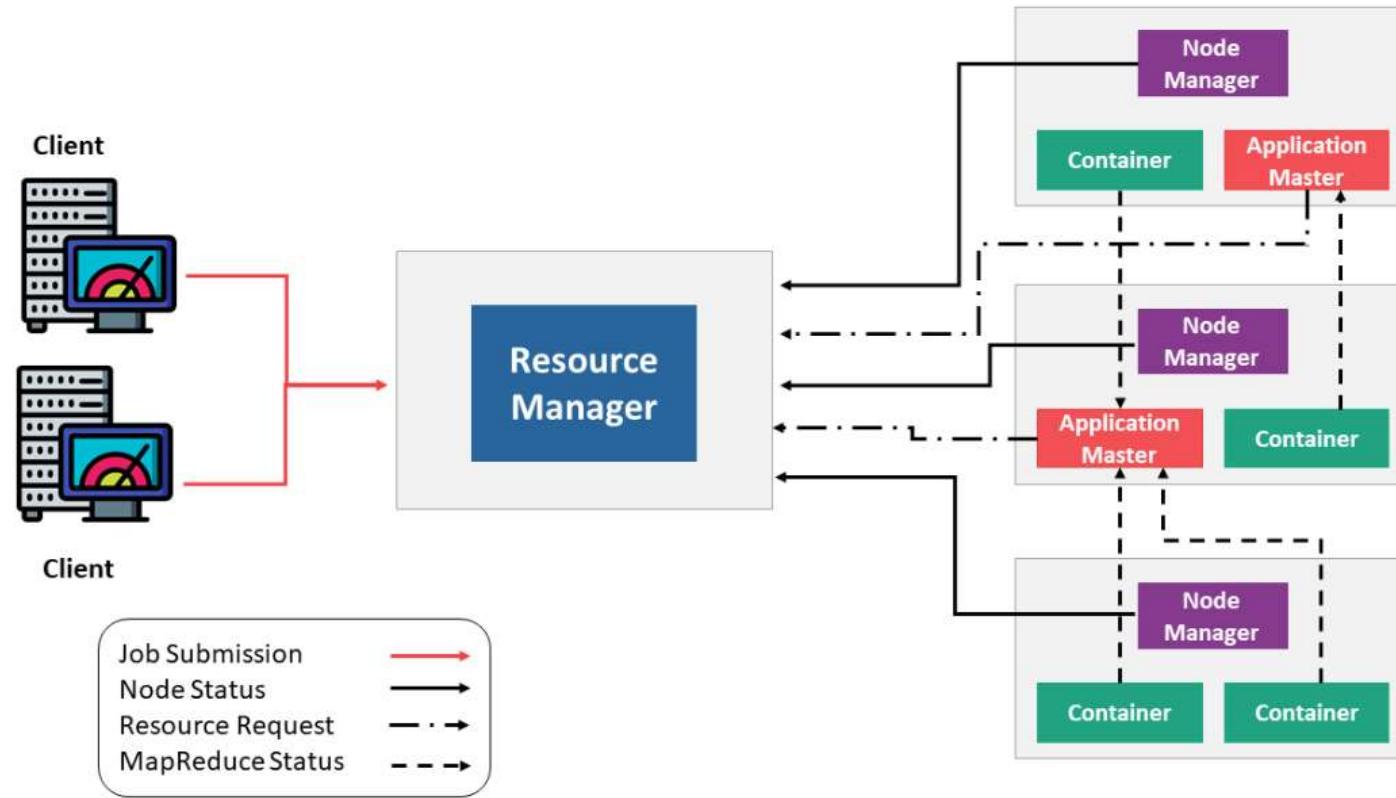
Aula 2.4. Componentes do YARN



Nesta aula

- ❑ Arquitetura YARN.
- ❑ Detalhe dos componentes do YARN.

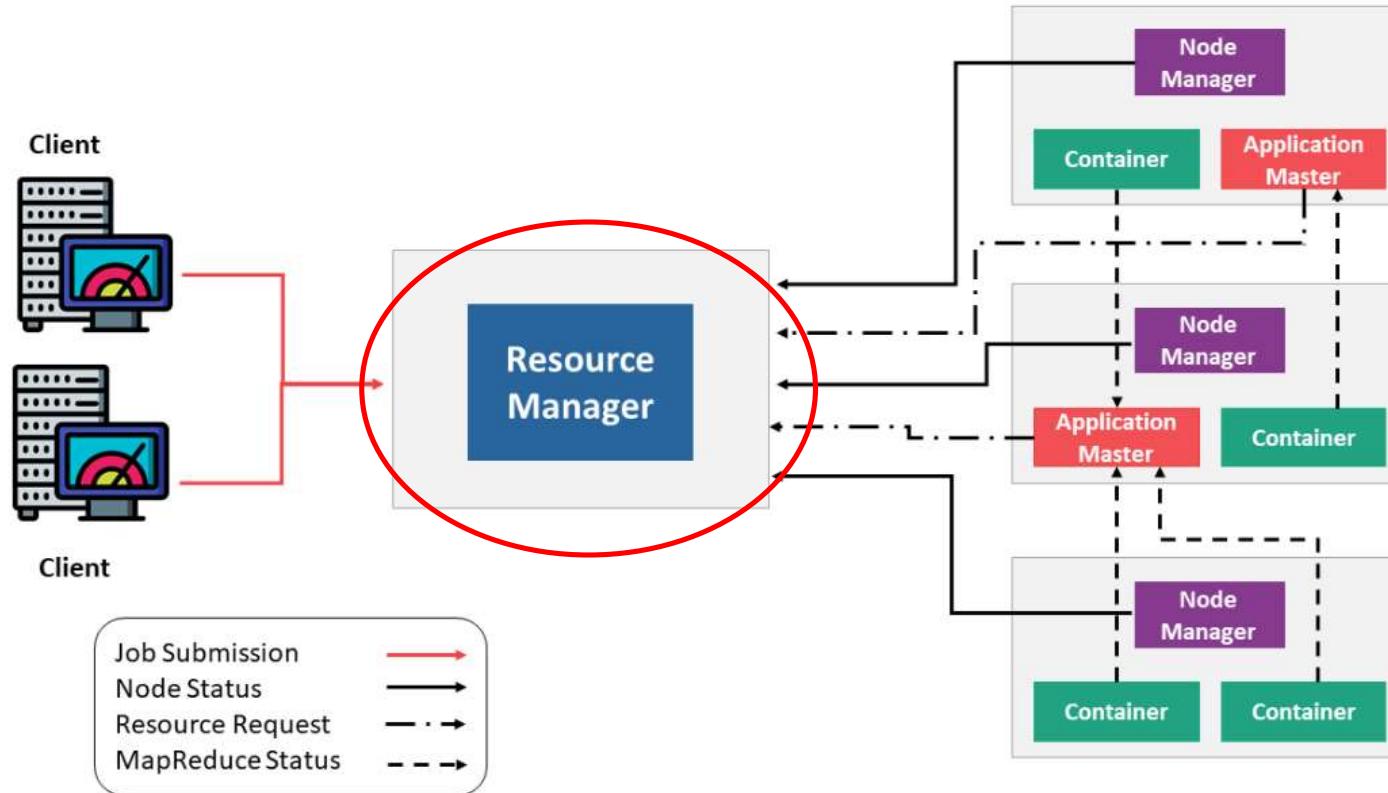
Componentes do YARN



Componentes do YARN

- Resource Manager:
 - Autoridade final na alocação de recursos.
 - Árbitro dos recursos do cluster. Decide pela alocação dos recursos.
 - Otimiza a utilização do cluster.
 - Possui dois componentes: o Agendador e o Gerenciador de Aplicativos.

Componentes do YARN



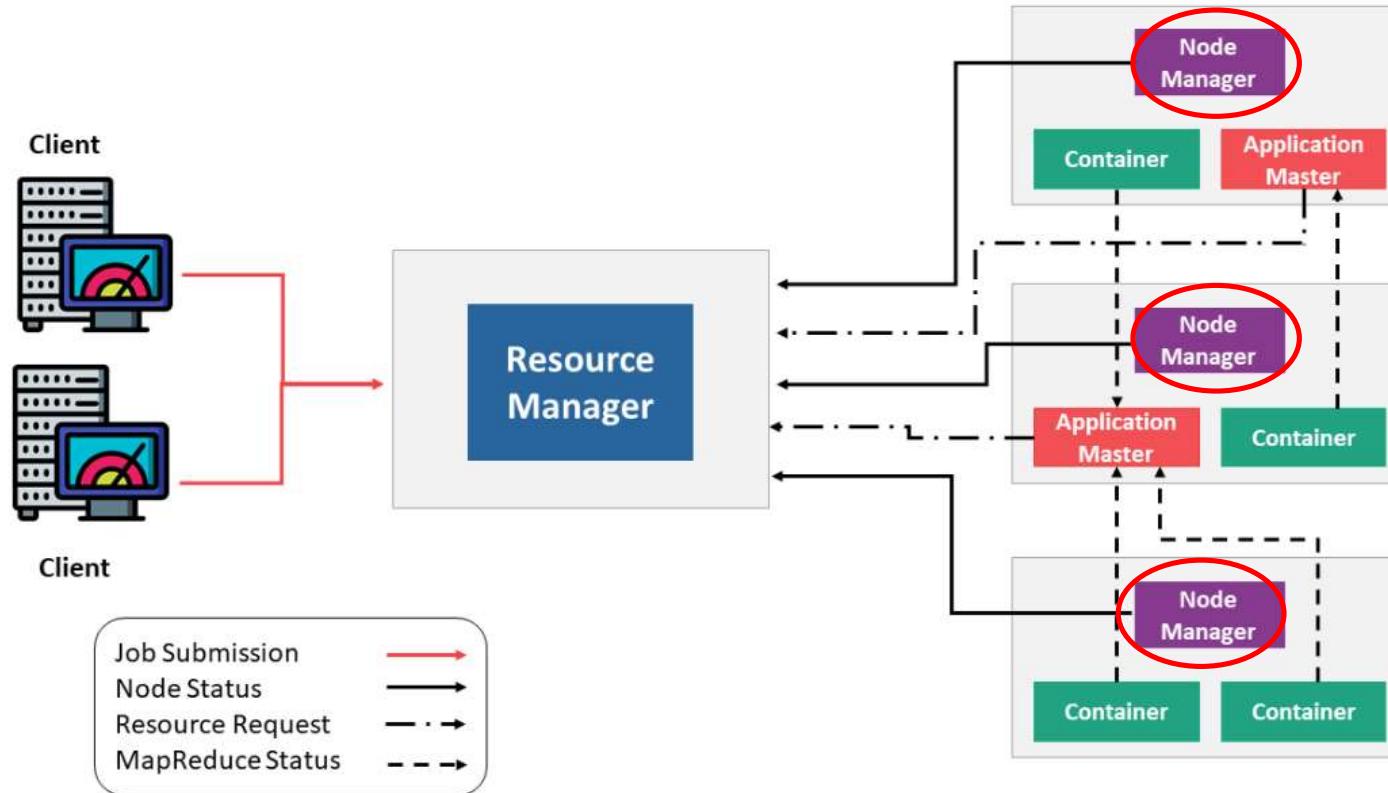
Componentes do YARN

- Resource Manager – Agendador:
 - Responsável por alocar recursos para os vários aplicativos em execução, sujeito a restrições de capacidades, filas, etc.
 - Não executa nenhum monitoramento ou rastreamento de status dos aplicativos (Agendador Puro).
 - Não garante a reinicialização das tarefas com falhas.

Componentes do YARN

- Resource Manager - Gerenciador de Aplicativos:
 - Responsável por aceitar os trabalhos.
 - Negocia a execução.

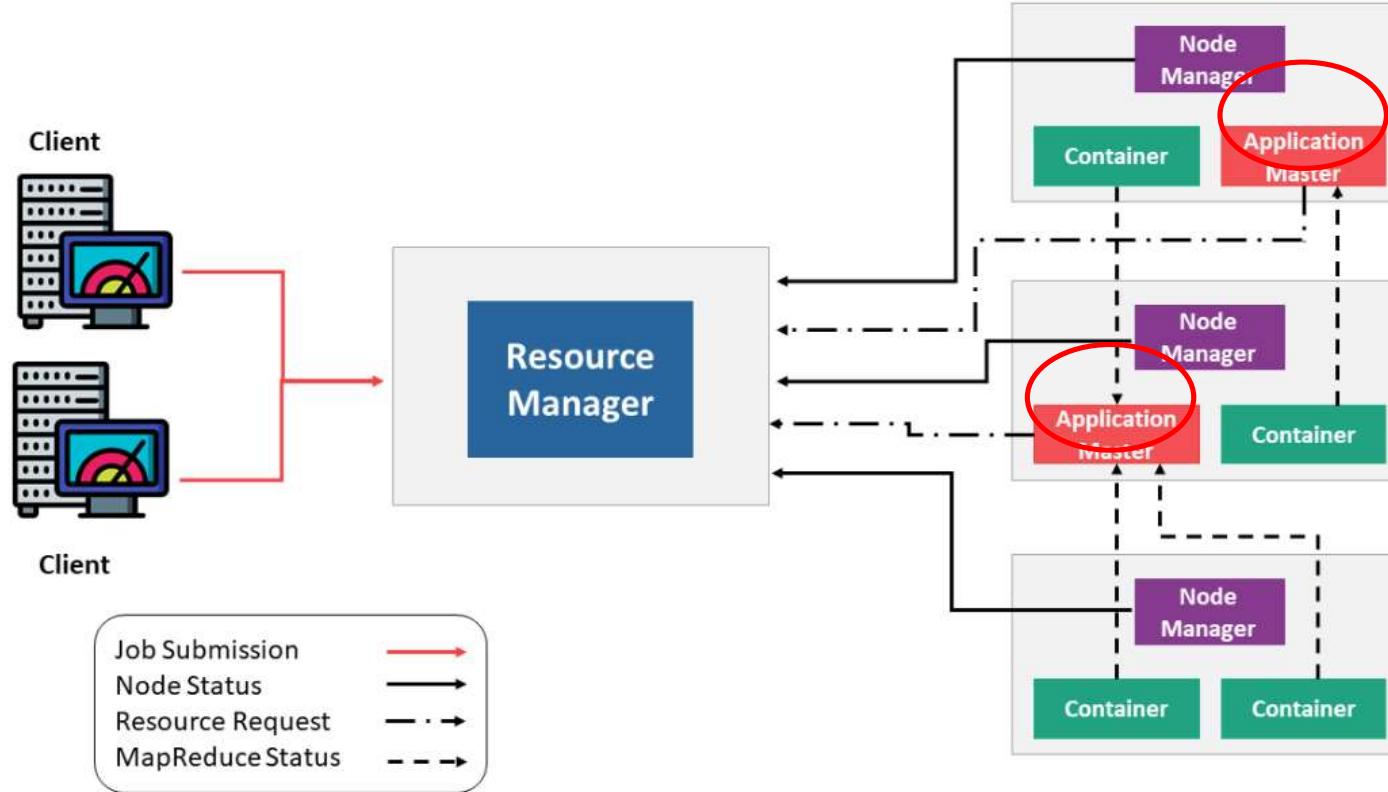
Componentes do YARN



Componentes do YARN

- Node Manager:
 - Cuida dos nós individualmente no cluster.
 - Envia “pulsos” sobre a integridade do nó.
 - Seu objetivo principal é gerenciar contêineres de aplicativos.
 - Executa o gerenciamento de logs.
 - Monitora o uso de recursos (CPU e memória) de contêineres individuais.

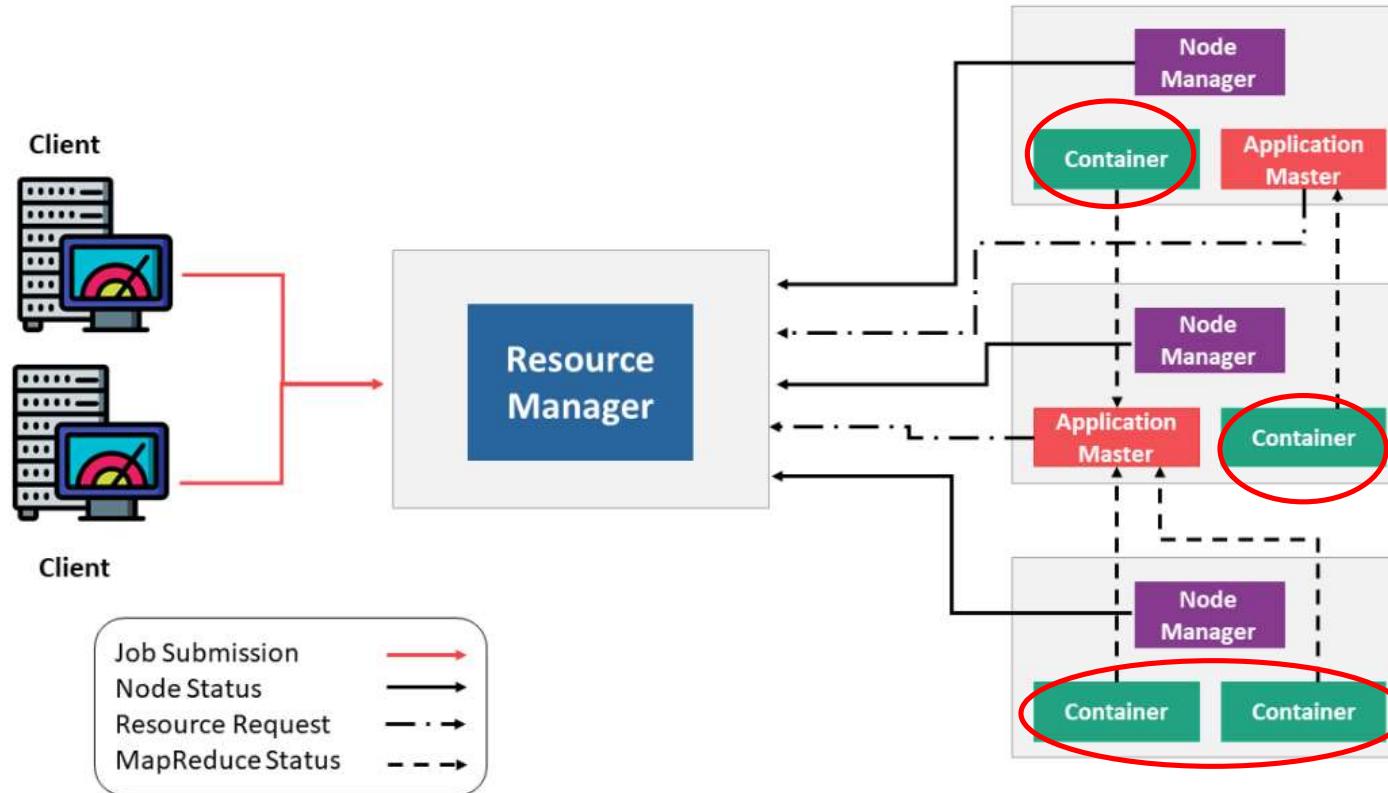
Componentes do YARN



Componentes do YARN

- Application Master:
 - Cada trabalho enviado ao framework possui um único AM associado a ele.
 - É o processo que coordena a execução de um aplicativo no cluster e também gerencia as falhas.
 - Sua tarefa é negociar recursos com o Resource Manager.
 - Responsável por negociar os contêineres de recursos apropriados.

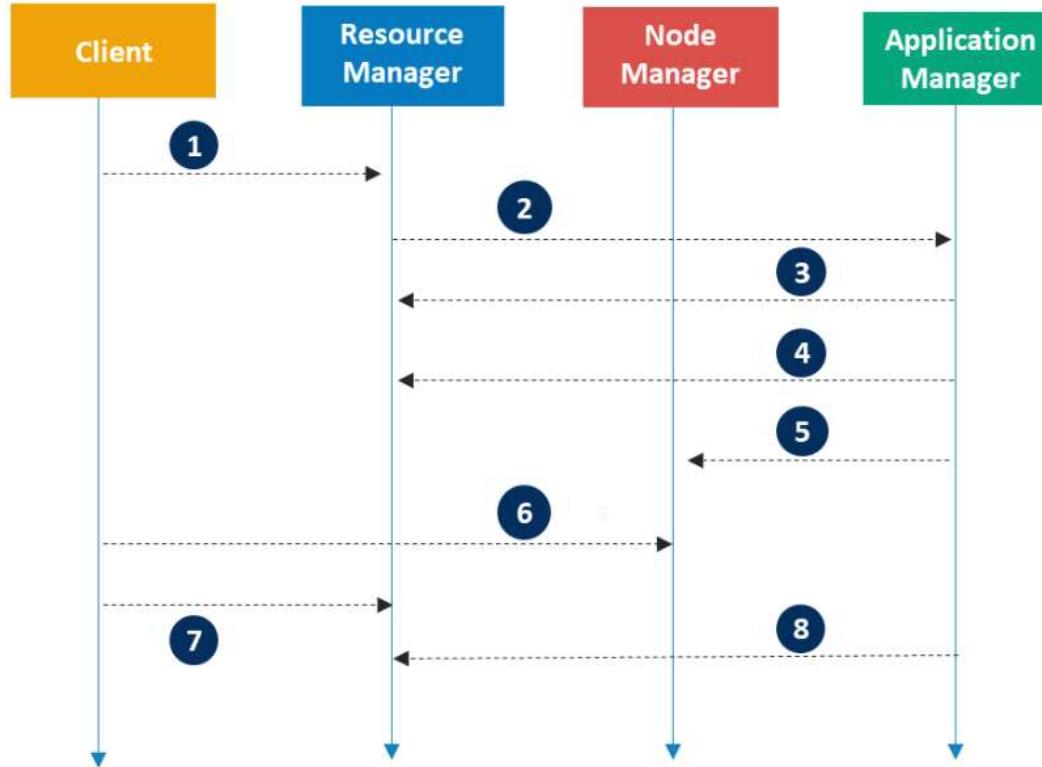
Componentes do YARN



Componentes do YARN

- Contêiner:
 - É uma coleção de recursos físicos (RAM, núcleos de CPU e discos em um único nó).
 - Gerenciado pelo ciclo de vida do contêiner (CLC).

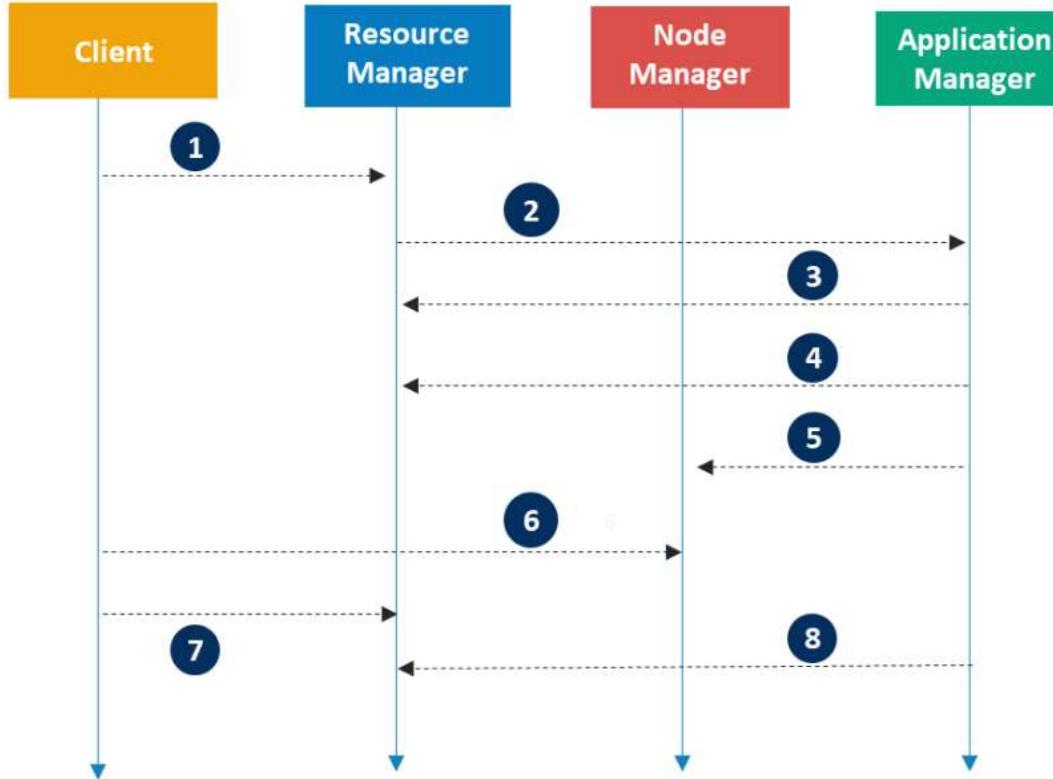
Componentes do YARN



Fluxo de execução:

1. O Client submete uma aplicação.
2. O Resource Manager aloca um contêiner para iniciar o Application Manager.
3. O Application Manager registra-se no Resource Manager.
4. O Application Manager solicita contêiners ao Resource Manager.

Componentes do YARN



Fluxo de execução:

- O Application Manager notifica o Node Manager para que os contêineres sejam disponibilizados.
- O código da aplicação é executado nos contêineres.
- O Client entra em contato com o Resource Manager para que ele monitore o status da aplicação.
- O Application Manager encerra seu registro no Resource Manager.

Conclusão

- ✓ Detalhes sobre os componentes do YARN.
- ✓ Fluxo.

Próxima aula

- Instalação do Hadoop.



Aula 2.5. Instalação



Nesta aula

- Instalação do Hadoop.

Instalação do Hadoop

- Primeiro passo:
 - Pré-requisitos.



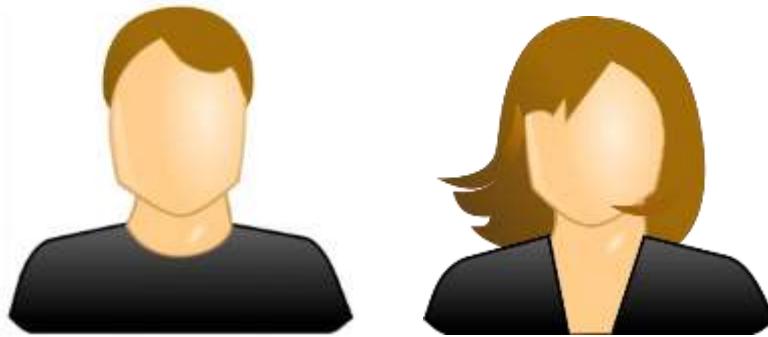
Hadoop – Instalação

- Pré-requisitos para a instalação:
 - Máquina Virtual JAVA.
 - Use o comando java: –version.
 - SSH.



Instalação do Hadoop

- Segundo passo:
 - Usuário hduser.



Hadoop – Instalação

- Criar um usuário exclusivo para trabalhar com o Hadoop.
- Adicionando o usuário hduser ao grupo hadoop.
- Essa etapa é opcional.

```
1 $ sudo addgroup hadoop  
2 $ sudo adduser --ingroup hadoop hduser
```

Instalação do Hadoop

- Terceiro passo:
 - Configurando o SSH.



Hadoop – Instalação

- Hadoop utiliza SSH para gerenciar os nós.
- Mesmo para localhost.
- Devemos configurar o acesso ao localhost para o usuário hduser sem senha.
- Iremos considerar que o ssh está executando na máquina.

Hadoop – Instalação

- Conecte com o usuário hduser. (1)
- Gere uma chave SSH para o usuário hduser. (2)
- Senha vazia.

```
1 user@ubuntu:~$ su - hduser (1)
2 hduser@ubuntu:~$ ssh-keygen -t rsa -P ""
3 Generating public/private rsa key pair.
4 Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
5 Created directory '/home/hduser/.ssh'.
6 Your identification has been saved in /home/hduser/.ssh/id_rsa.
7 Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
8 The key fingerprint is:
9 9b:82:ea:58:b4:e0:35:d7:ff:19:66:a6:ef:ae:0e:d2 hduser@ubuntu
10 The key's randomart image is:
11 [...snipp...]
12 hduser@ubuntu:~$
```

Hadoop – Instalação

- Habilitar o SSH para acessar a máquina local com a chave recentemente criada.

```
1 hduser@ubuntu:~$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

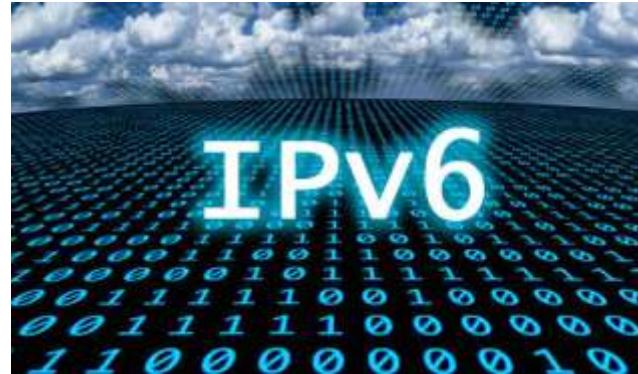
Hadoop – Instalação

- Vamos testar a conexão para a máquina local com o usuário hduser!

```
1 hduser@ubuntu:~$ ssh localhost
2 The authenticity of host 'localhost (::1)' can't be established.
3 RSA key fingerprint is d7:87:25:47:ae:02:00:eb:1d:75:4f:bb:44:f9:36:26.
4 Are you sure you want to continue connecting (yes/no)? yes
5 Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
6 Linux ubuntu 2.6.32-22-generic #33-Ubuntu SMP Wed Apr 28 13:27:30 UTC 2010 i686 GNU/Linux
7 Ubuntu 10.04 LTS
8 [...snipp...]
9 hduser@ubuntu:~$
```

Instalação do Hadoop

- Quarto passo:
 - Desabilitar o IPv6.



Hadoop – Instalação

- Abra o arquivo /etc/sysctl.conf em um editor de texto e adicione as seguintes linhas no fim do arquivo:

/etc/sysctl.conf

```
1 # disable ipv6
2 net.ipv6.conf.all.disable_ipv6 = 1
3 net.ipv6.conf.default.disable_ipv6 = 1
4 net.ipv6.conf.lo.disable_ipv6 = 1
```

- Reinicie a máquina.

Hadoop – Instalação

- Verifique se o IPv6 está realmente desabilitado:

```
1 $ cat /proc/sys/net/ipv6/conf/all/disable_ipv6
```

- Se o comando retornar 0 (zero), o IPv6 está habilitado.
- **Se retornar 1, o IPv6 está desabilitado (é o que queremos).**

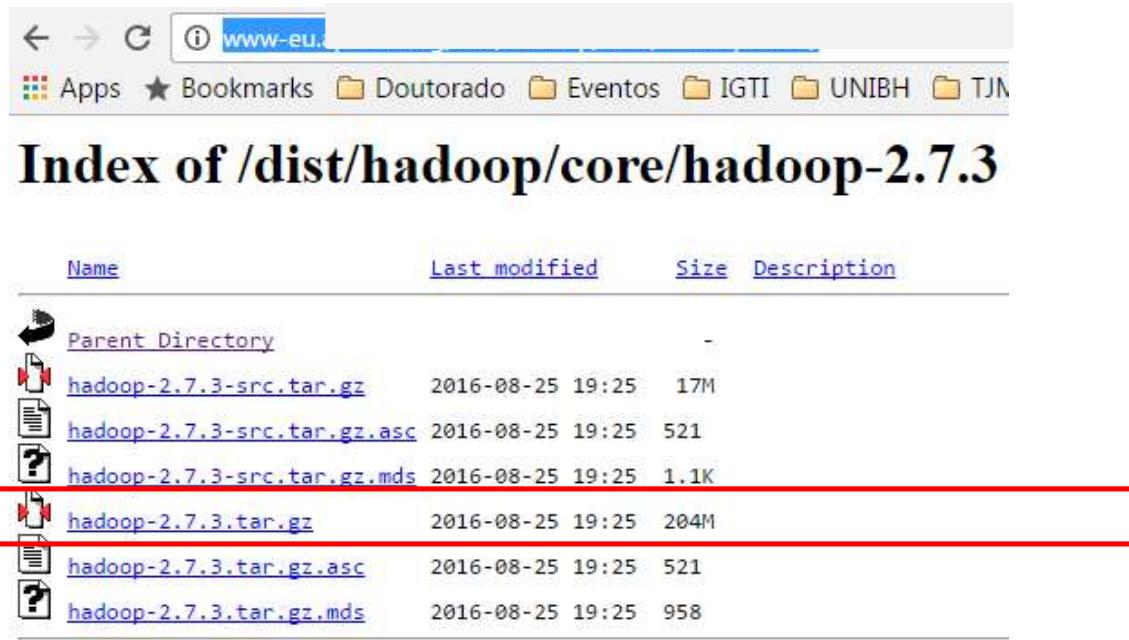
Instalação do Hadoop

- Quinto passo:
 - Hadoop.



Hadoop – Instalação

- Faça o download do Hadoop:
 - <http://www-eu.apache.org/dist/hadoop/core/hadoop-2.7.3/>



Name	Last modified	Size	Description
Parent Directory		-	
hadoop-2.7.3-src.tar.gz	2016-08-25 19:25	17M	
hadoop-2.7.3-src.tar.gz.asc	2016-08-25 19:25	521	
hadoop-2.7.3-src.tar.gz.mds	2016-08-25 19:25	1.1K	
hadoop-2.7.3.tar.gz	2016-08-25 19:25	204M	
hadoop-2.7.3.tar.gz.asc	2016-08-25 19:25	521	
hadoop-2.7.3.tar.gz.mds	2016-08-25 19:25	958	

Hadoop – Instalação

- Sugestão: descompactar no diretório /usr/local.

```
1 $ cd /usr/local  
2 $ sudo tar xzf hadoop- 2.7.3 .tar.gz  
3 $ sudo mv hadoop- 2.7.3 hadoop  
4 $ sudo chown -R hduser:hadoop hadoop
```

Instalação do Hadoop

- Sexto passo:
 - Variáveis de ambiente.



Hadoop – Instalação

- Precisamos editar o arquivo `.bashrc` para o usuário `hduser`.
- Incluir variáveis de ambiente.
- `HADOOP_HOME`.
- `JAVA_HOME`.

```
$HOME/.bashrc
1 # Set Hadoop-related environment variables
2 export HADOOP_HOME=/usr/local/hadoop ← yellow arrow
3
4 # Set JAVA_HOME (we will also configure JAVA_HOME directly for Hadoop later on)
5 export JAVA_HOME=/usr/lib/jvm/java-6-sun ← blue arrow
6
7 # Some convenient aliases and functions for running Hadoop-related commands
8 unalias fs &> /dev/null
9 alias fs="hadoop fs"
10 unalias hls &> /dev/null
11 alias hls="fs -ls"
12
13 # If you have LZOP compression enabled in your Hadoop cluster and
14 # compress job outputs with LZOP (not covered in this tutorial):
15 # Conveniently inspect an LZOP compressed file from the command
16 # line; run via:
17 #
18 # $ lzohead /hdfs/path/to/lzop/compressed/file.lzo
19 #
20 # Requires installed 'lzohead' command.
21 #
22 lzohead () {
23     hadoop fs -cat $1 | lzop -dc | head -1000 | less
24 }
25
26 # Add Hadoop bin/ directory to PATH
27 export PATH=$PATH:$HADOOP_HOME/bin ← red arrow
```

Instalação do Hadoop

- Sétimo passo:
 - Configurando o Hadoop.



Hadoop – Instalação

- Crie um diretório para armazenamento de dados temporários do Hadoop.
- Sugestão: criar no endereço: /usr/local/tmp
`sudo mkdir -p /usr/local/tmp`
- Esse diretório vai ser utilizado posteriormente.
- Atribua permissões ao novo diretório:
`sudo chown hduser:hadoop /usr/local/hadoop/tmp`
`sudo chmod /usr/local/hadoop/tmp`

Hadoop – Instalação

- Altere o arquivo core-site.xml.
- /usr/local/hadoop/etc/hadoop/core-site.xml
- Inserir entre a tag: <configuration> </configuration>:

```
conf/core-site.xml

1 <property>
2   <name>fs.default.name</name>
3   <value>hdfs://localhost:54310</value>
4   <description>The name of the default file system. A URI whose
5   scheme and authority determine the FileSystem implementation. The
6   uri's scheme determines the config property (fs.SCHEME.impl) naming
7   the FileSystem implementation class. The uri's authority is used to
8   determine the host, port, etc. for a filesystem.</description>
9 
10 </property>
11 
12 <property>
13   <name>fs.default.name</name>
14   <value>hdfs://localhost:54310</value>
15   <description>A base for other temporary directories.</description>
```

Hadoop – Instalação

- Altere o arquivo mapred-site.xml.
- /usr/local/hadoop/etc/hadoop/mapred-site.xml.
- Inserir entre a tag: <configuration> </configuration>:

conf/mapred-site.xml

```
1 <property>
2   <name>mapred.job.tracker</name>
3   <value>localhost:54311</value>
4   <description>The host and port that the MapReduce job tracker runs
5     at. If "local", then jobs are run in-process as a single map
6     and reduce task.
7   </description>
8 </property>
```

Hadoop – Instalação

- Altere o arquivo hdfs-site.xml.
- /usr/local/hadoop/etc/hadoop/hdfs-site.xml.
- Inserir entre as tags: <configuration> <\configuration>:

conf/hdfs-site.xml

```
1 <property>
2   <name>dfs.replication</name>
3   <value>1</value>
4   <description>Default block replication.
5   The actual number of replications can be specified when the file is created.
6   The default is used if replication is not specified in create time.
7   </description>
8 </property>
```

Instalação do Hadoop

- Oitavo passo:
 - Formatando o HDFS.



Hadoop – Instalação

- Formatando o sistema de arquivos distribuído – HDFS:

```
1 hduser@ubuntu:~$ /usr/local/hadoop/bin/hadoop namenode -format
```

- A saída do comando acima deve ser a seguinte:

```
1 hduser@ubuntu:/usr/local/hadoop$ bin/hadoop namenode -format
2 10/05/08 16:59:56 INFO namenode.Namenode: STARTUP_MSG:
3 ****
4 STARTUP_MSG: Starting NameNode
5 STARTUP_MSG:   host = ubuntu/127.0.1.1
6 STARTUP_MSG:   args = [-format]
7 STARTUP_MSG:   version = 0.20.2
8 STARTUP_MSG:   build = https://svn.apache.org/repos/asf/hadoop/common/branches/branch-0.20 -i
9 ****
10 10/05/08 16:59:56 INFO namenode.FSNamesystem: fsOwner=hduser,hadoop
11 10/05/08 16:59:56 INFO namenode.FSNamesystem: supergroup=supergroup
12 10/05/08 16:59:56 INFO namenode.FSNamesystem: isPermissionEnabled=true
13 10/05/08 16:59:56 INFO common.Storage: Image file of size 96 saved in 0 seconds.
14 10/05/08 16:59:57 INFO common.Storage: Storage directory .../hadoop-hduser/dfs/name has been
15 10/05/08 16:59:57 INFO namenode.Namenode: SHUTDOWN_MSG:
16 ****
17 SHUTDOWN_MSG: Shutting down NameNode at ubuntu/127.0.1.1
18 ****
19 hduser@ubuntu:/usr/local/hadoop$
```

Instalação do Hadoop

- Nono passo:
 - Iniciando os serviços Hadoop.



Hadoop – Instalação

- Execute o comando:
 - hduser@ubuntu:~\$ /usr/local/hadoop/sbin/start-all.sh
- A saída do comando acima deve ser a seguinte:

```
1  hduser@ubuntu:~$ /usr/local/hadoop/sbin/start-all.sh
2  starting namenode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduser-namenode-ubuntu.out
3  localhost: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduser-datanode-
4  localhost: starting secondarynamenode, logging to /usr/local/hadoop/bin/../logs/hadoop-hduser-
5  starting jobtracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hduser-jobtracker-ubuntu.
6  localhost: starting tasktracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hduser-tasktr
7  hduser@ubuntu:/usr/local/hadoop$
```

Hadoop – Instalação

- Por meio do comando jps, verifique se os serviços estão ativos:
 - Datanode.
 - ResourceManager.
 - NameNode.
 - SecondaryNameNode.
 - NodeManager.

Instalação do Hadoop

- Décimo passo:
 - Executando um teste.



Hadoop – Instalação

- Executando o cálculo do PI.
- Dentro da pasta /usr/local/hadoop.

```
bin/hadoop      jar      /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-  
2.7.3.jar pi 16 1000
```

Hadoop – Instalação

```
HDFS: Number of write operations=309
Map-Reduce Framework
    Map input records=16
    Map output records=32
    Map output bytes=288
    Map output materialized bytes=448
    Input split bytes=2390
    Combine input records=0
    Combine output records=0
    Reduce input groups=2
    Reduce shuffle bytes=448
    Reduce input records=32
    Reduce output records=0
    Spilled Records=64
    Shuffled Maps =16
    Failed Shuffles=0
    Merged Map outputs=16
    GC time elapsed (ms)=828
    Total committed heap usage (bytes)=2527027200
Shuffle Errors
    BAD_ID=0
    CONNECTION=0
    IO_ERROR=0
    WRONG_LENGTH=0
    WRONG_MAP=0
    WRONG_REDUCE=0
File Input Format Counters
    Bytes Read=1888
File Output Format Counters
    Bytes Written=97
Job Finished in 9.169 seconds
Estimated value of Pi is 3.14250000000000000000
hduser@Inovaz-hadoop1$
```

Conclusão

- ✓ Instalação do Hadoop.

Próxima aula

- Mecanismo e Estrutura da Ferramenta.



Aula 2.6 Mecanismo e estrutura da ferramenta

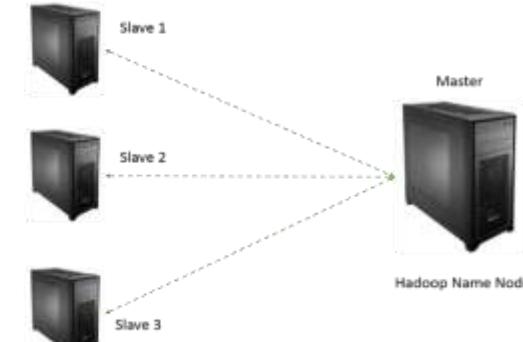
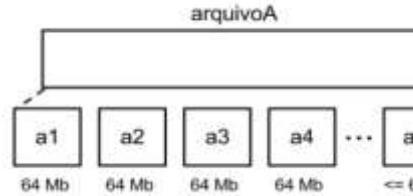


Nesta aula

- Mecanismos e estrutura da ferramenta.
- Arquivos de configuração.

Mecanismo e estrutura da ferramenta

- O processo começa com o arquivo de entrada.
- MapReduce divide o(s) arquivo(s) de entrada em M partições.
- Esse processo inicia cópias do programa no cluster.
- Uma das cópias é o Master e o resto Workers.



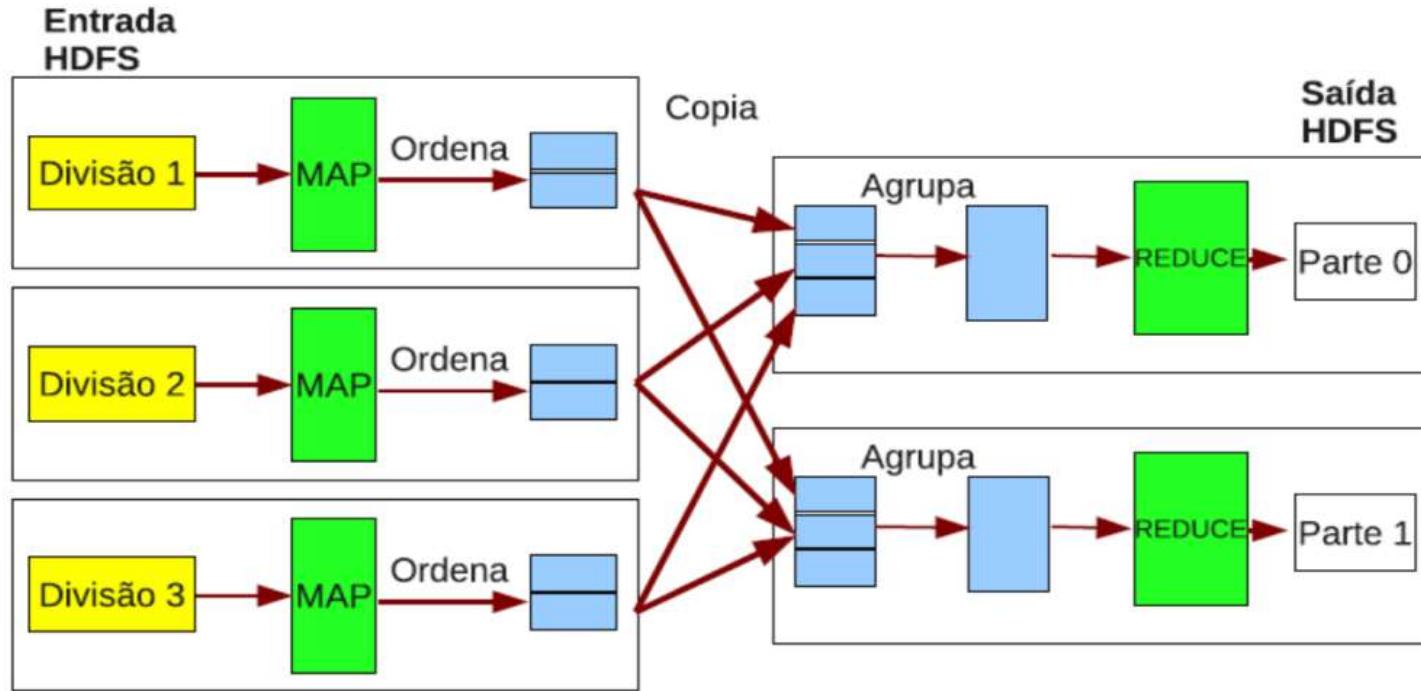
Mecanismo e estrutura da ferramenta

- Master escolhe Workers inativos e atribui tarefas.
- O Worker analisa os dados de entrada e envia à função Map.
- O resultado da função Map é armazenado em buffer.
- Periodicamente os dados do buffer são escritos em disco.

Mecanismo e estrutura da ferramenta

- A localização do resultado do Map é enviada ao master.
- Os dados são ordenados e agrupados.
- O Worker Reduce interage sobre os dados.
- Quando todas as tarefas Map e Reduce são completadas, o master é notificado.
- O resultado fica disponível em arquivos de saída.

Mecanismo e estrutura da ferramenta



Arquivos de configuração

- Hadoop possui diversos arquivos de configuração.
- XML.
- Hadoop-env.sh: variáveis de ambiente do Hadoop.
- Core-site.xml: parâmetros importantes para o núcleo do Hadoop.
- Hdfs-site.xml: configurações do sistema de arquivos (HDFS).
- Mapred-site.xml: configurações do MapReduce.



Arquivos de configuração

- Slaves:
 - Lista de máquinas do cluster (não é XML).
 - Cada linha é uma máquina Worker (Datanode).
- Masters:
 - Lista de máquinas Master do cluster (não é XML).
 - Cada linha é uma máquina Master (Namenode).

Arquivos de configuração



The screenshot shows a Windows Notepad window titled "Hdfs-site.xml - Bloco de notas". The content of the file is an XML configuration for HDFS. It includes a license notice from the Apache License, Version 2.0, followed by specific site-specific overrides for replication, namenode, datanode, and blocksize.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

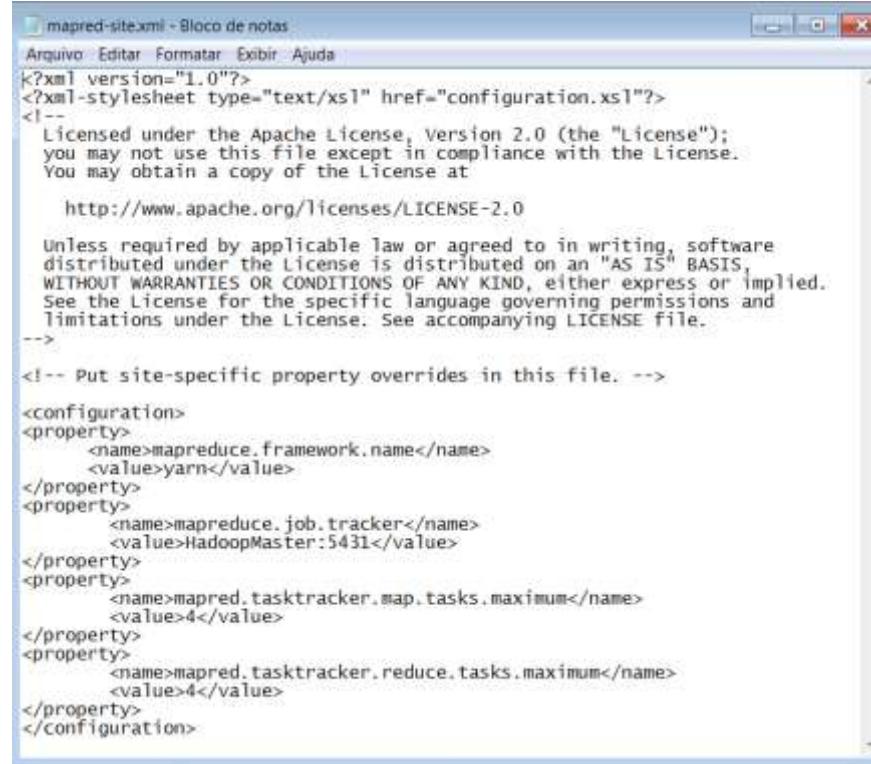
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
    <name>dfs.replication</name>
    <value>3</value>
</property>
<property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/mnt/hd2/hdfs/namenode</value>
</property>
<property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/mnt/hd2/hdfs/datanode</value>
</property>
<property>
    <name>dfs.blocksize</name>
    <value>67108864</value>
</property>
</configuration>
```

Arquivos de configuração



```
mapred-site.xml - Bloco de notas
Arquivo Editar Formatar Exibir Ajuda
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
<property>
    <name>mapreduce.job.tracker</name>
    <value>HadoopMaster:5431</value>
</property>
<property>
    <name>mapred.tasktracker.map.tasks.maximum</name>
    <value>4</value>
</property>
<property>
    <name>mapred.tasktracker.reduce.tasks.maximum</name>
    <value>4</value>
</property>
</configuration>
```

Conclusão

- ✓ Mecanismos e estrutura da ferramenta:
 - Map, reduce, workers e master.
- ✓ Arquivos de configuração:
 - mapred-site.xml, mapred-site.xml, core-site.xml, slaves e masters.

□ Funções:

- Map.
- Reduce.
- Combine.



Aula 2.7 Funções map, reduce e combine



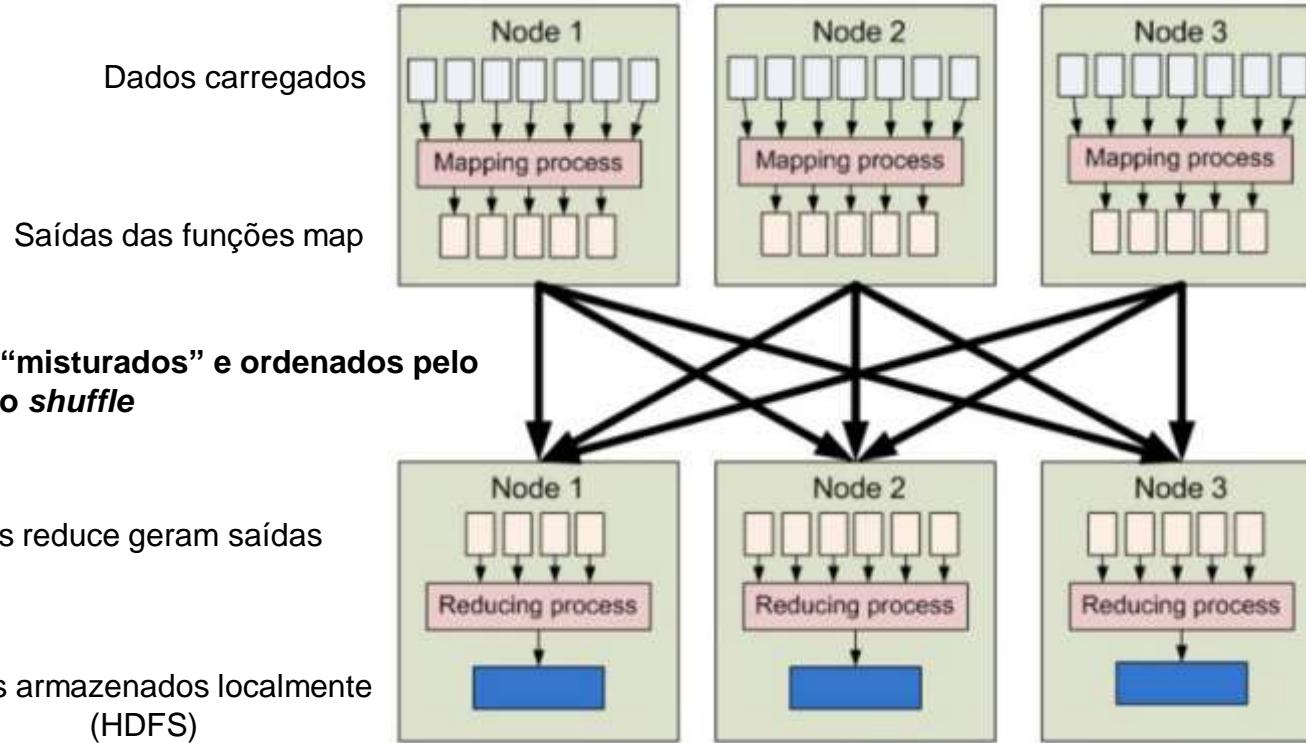
□ Funções:

- Map.
- Reduce.
- Combine.

Funções map, reduce e combine

- Hadoop trabalha dividindo os trabalhos em duas funções distintas e principais: map e reduce.
- A forma de trabalho de cada função é definida pelo programador.
- Recebem como entrada pares chave-valor.
- Os tipos de dados desses pares são definidos pelo programador.
- O modelo fornece uma função intermediária: combine.

Funções map, reduce e combine



Funções map, reduce e combine

- Dado um arquivo.
- Grande volume.
- Com vendas realizadas por uma empresa.

Funções map, reduce e combine

- Dados de entrada. Arquivo(s) no HDFS:

```
0505345642014032016454645640000846652021142052130378974351200000500004546654564564564564631231230003586453210
0505345642014032016454645645640000846652021142052130378974351200000302004546654564564564564631231230003586453210
0505345642005072016454645645640000846652021142052130378974351200000507754546654564564564564564631231230003586453210
0505345642007052016454645645640000846652021142052130378974351200000599244546654564564564564564631231230003586453210
0505345642014032016454645645640000846652021142052130378974351200000512234546654564564564564564631231230003586453210
0505345642008042016454645645640000846652021142052130378974351200000637224546654564564564564631231230003586453210
0505345642020012016454645645640000846652021142052130378974351200001419254546654564564564564631231230003586453210
0505345642014032016454645645640000846652021142052130378974351200002329224546654564564564564631231230003586453210
0505345640508042016454645645640000846652021142052130378974351200006258804546654564564564564631231230003586453210
0505345642007052016454645645640000846652021142052130378974351200000080244546654564564564564631231230003586453210
0505345642005072016454645645640000846652021142052130378974351200000637884546654564564564564631231230003586453210
0505345642008042016454645645640000846652021142052130378974351200001255594546654564564564564631231230003586453210
0505345642014032016454645645640000846652021142052130378974351200000832694546654564564564631231230003586453210
0505345640208042016454645645640000846652021142052130378974351200000369694546654564564564631231230003586453210
0505345642013052016454645645640000846652021142052130378974351200000333694546654564564564564631231230003586453210
0505345642007052016454645645640000846652021142052130378974351200000635854546654564564564564631231230003586453210
050534564200804201645464564564000084665202114205213037897435120000120004546654564564564564631231230003586453210
0505345642005072016454645645640000846652021142052130378974351200000825504546654564564564564631231230003586453210
0505345642014032016454645645640000846652021142052130378974351200000637504546654564564564564631231230003586453210
0505345642007052016454645645640000846652021142052130378974351200000612124546654564564564564631231230003586453210
0505345642013052016454645645640000846652021142052130378974351200000137694546654564564564564631231230003586453210
0505345642008112016454645645640000846652021142052130378974351200001819364546654564564564631231230003586453210
0505345642008112016454645645640000846652021142052130378974351200001369524546654564564564564631231230003586453210
0505345642008112016454645645640000846652021142052130378974351200008520694546654564564564564631231230003586453210
0505345642008112016454645645640000846652021142052130378974351200006352524546654564564564564631231230003586453210
050534564201403201645464564564000084665202114205213037897435120000368854546654564564564564631231230003586453210
050534564202001201645464564564000084665202114205213037897435120000237854546654564564564564631231230003586453210
050534564202001201645464564564000084665202114205213037897435120000100254546654564564564564631231230003586453210
```

Funções map, reduce e combine

- Posições 12 até 19 referem-se às datas das vendas.
- Posições 63 até 72 referem-se aos valores das vendas.

Funções map, reduce e combine

- Map:

05053456420**14032016**4546456456400008466520211420521303789743512**0000050000**454665456456456456464631231230003586453210
05053456420**14032016**4546456456400008466520211420521303789743512**0000030200**4546654564564564564631231230003586453210
05053456420**05072016**4546456456400008466520211420521303789743512**0000050775**4546654564564564564631231230003586453210
05053456420**07052016**4546456456400008466520211420521303789743512**0000059924**4546654564564564564631231230003586453210
05053456420**14032016**4546456456400008466520211420521303789743512**0000051223**4546654564564564564631231230003586453210
05053456420**08042016**4546456456400008466520211420521303789743512**0000063722**4546654564564564631231230003586453210
05053456420**20012016**4546456456400008466520211420521303789743512**0000141925**4546654564564564631231230003586453210
05053456420**14032016**4546456456400008466520211420521303789743512**0000232922**4546654564564564631231230003586453210
05053456405**08042016**4546456456400008466520211420521303789743512**0000625880**4546654564564564631231230003586453210
05053456420**07052016**4546456456400008466520211420521303789743512**0000008024**4546654564564564631231230003586453210
05053456420**05072016**4546456456400008466520211420521303789743512**000063788**4546654564564564631231230003586453210
05053456420**08042016**4546456456400008466520211420521303789743512**0000125559**4546654564564564631231230003586453210
05053456420**14032016**4546456456400008466520211420521303789743512**0000083269**4546654564564564631231230003586453210
05053456420**08042016**4546456456400008466520211420521303789743512**0000036969**4546654564564564631231230003586453210
05053456420**13052016**4546456456400008466520211420521303789743512**0000033369**4546654564564564631231230003586453210
05053456420**07052016**4546456456400008466520211420521303789743512**0000063585**4546654564564564631231230003586453210
05053456420**08042016**4546456456400008466520211420521303789743512**0000012000**4546654564564564631231230003586453210
05053456420**05072016**4546456456400008466520211420521303789743512**0000082550**4546654564564564631231230003586453210
05053456420**14032016**4546456456400008466520211420521303789743512**0000063750**4546654564564564564631231230003586453210
05053456420**07052016**4546456456400008466520211420521303789743512**0000061212**4546654564564564631231230003586453210
05053456420**13052016**4546456456400008466520211420521303789743512**0000013769**4546654564564564631231230003586453210
05053456420**08112016**4546456456400008466520211420521303789743512**0000181936**4546654564564564631231230003586453210
05053456420**08112016**4546456456400008466520211420521303789743512**0000136952**4546654564564564631231230003586453210
05053456420**08112016**4546456456400008466520211420521303789743512**0000852069**4546654564564564631231230003586453210
05053456420**08112016**4546456456400008466520211420521303789743512**0000635252**4546654564564564631231230003586453210
05053456420**14032016**4546456456400008466520211420521303789743512**0000036885**4546654564564564631231230003586453210
05053456420**20012016**4546456456400008466520211420521303789743512**0000023785**4546654564564564631231230003586453210
05053456420**20012016**4546456456400008466520211420521303789743512**0000010025**4546654564564564631231230003586453210

05053456420**07052016**4546456456400008466520211420521303789743512**0000088896**4546654564564564631231230003586453210

Funções map, reduce e combine

- Map.
- Combine.
- Agrupamento e ordenação (Shuffle).
- Reduce.

Funções map, reduce e combine

- Map:

05053456420**14032016**4546456456400008466520211420521303789743512**000005000**4546654564564564564631231230003586453210
05053456420**14032016**4546456456400008466520211420521303789743512**0000030200**4546654564564564564631231230003586453210
05053456420**05072016**4546456456400008466520211420521303789743512**0000050775**4546654564564564564631231230003586453210
05053456420**07052016**4546456456400008466520211420521303789743512**0000059924**4546654564564564564631231230003586453210
05053456420**14032016**4546456456400008466520211420521303789743512**0000051223**4546654564564564564631231230003586453210
05053456420**08042016**4546456456400008466520211420521303789743512**0000063722**4546654564564564564631231230003586453210
05053456420**20012016**4546456456400008466520211420521303789743512**0000141925**4546654564564564564631231230003586453210
05053456420**14032016**4546456456400008466520211420521303789743512**0000232922**4546654564564564564631231230003586453210
05053456405**08042016**4546456456400008466520211420521303789743512**0000625880**4546654564564564631231230003586453210
05053456420**07052016**4546456456400008466520211420521303789743512**0000008024**4546654564564564564631231230003586453210
05053456420**05072016**4546456456400008466520211420521303789743512**0000063788**4546654564564564564631231230003586453210
05053456420**08042016**4546456456400008466520211420521303789743512**0000125559**4546654564564564564631231230003586453210
05053456420**14032016**4546456456400008466520211420521303789743512**0000083269**4546654564564564564631231230003586453210
05053456402**08042016**4546456456400008466520211420521303789743512**0000036969**4546654564564564564631231230003586453210
05053456420**13052016**4546456456400008466520211420521303789743512**0000033369**4546654564564564564631231230003586453210
05053456420**07052016**4546456456400008466520211420521303789743512**0000063585**4546654564564564564631231230003586453210
05053456420**08042016**4546456456400008466520211420521303789743512**0000012000**4546654564564564564631231230003586453210
05053456420**05072016**4546456456400008466520211420521303789743512**0000082550**4546654564564564564631231230003586453210
05053456420**14032016**4546456456400008466520211420521303789743512**0000063750**4546654564564564564631231230003586453210
05053456420**07052016**4546456456400008466520211420521303789743512**0000061212**4546654564564564564631231230003586453210
05053456420**13052016**4546456456400008466520211420521303789743512**0000013769**4546654564564564564631231230003586453210
05053456420**08112016**4546456456400008466520211420521303789743512**00000181936**4546654564564564631231230003586453210
05053456420**08112016**4546456456400008466520211420521303789743512**00000136952**4546654564564564631231230003586453210
05053456420**08112016**4546456456400008466520211420521303789743512**00000852069**4546654564564564631231230003586453210
05053456420**08112016**4546456456400008466520211420521303789743512**0000635252**4546654564564564631231230003586453210
05053456420**14032016**4546456456400008466520211420521303789743512**0000036885**4546654564564564564631231230003586453210
05053456420**20012016**4546456456400008466520211420521303789743512**0000023785**4546654564564564564631231230003586453210
05053456420**20012016**4546456456400008466520211420521303789743512**0000010025**4546654564564564564631231230003586453210

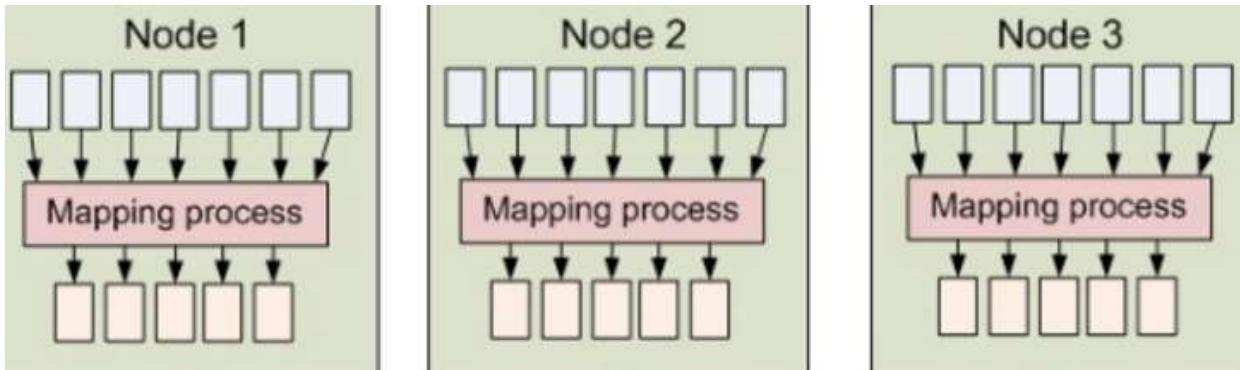
Funções map, reduce e combine

Entrada para os Mappers

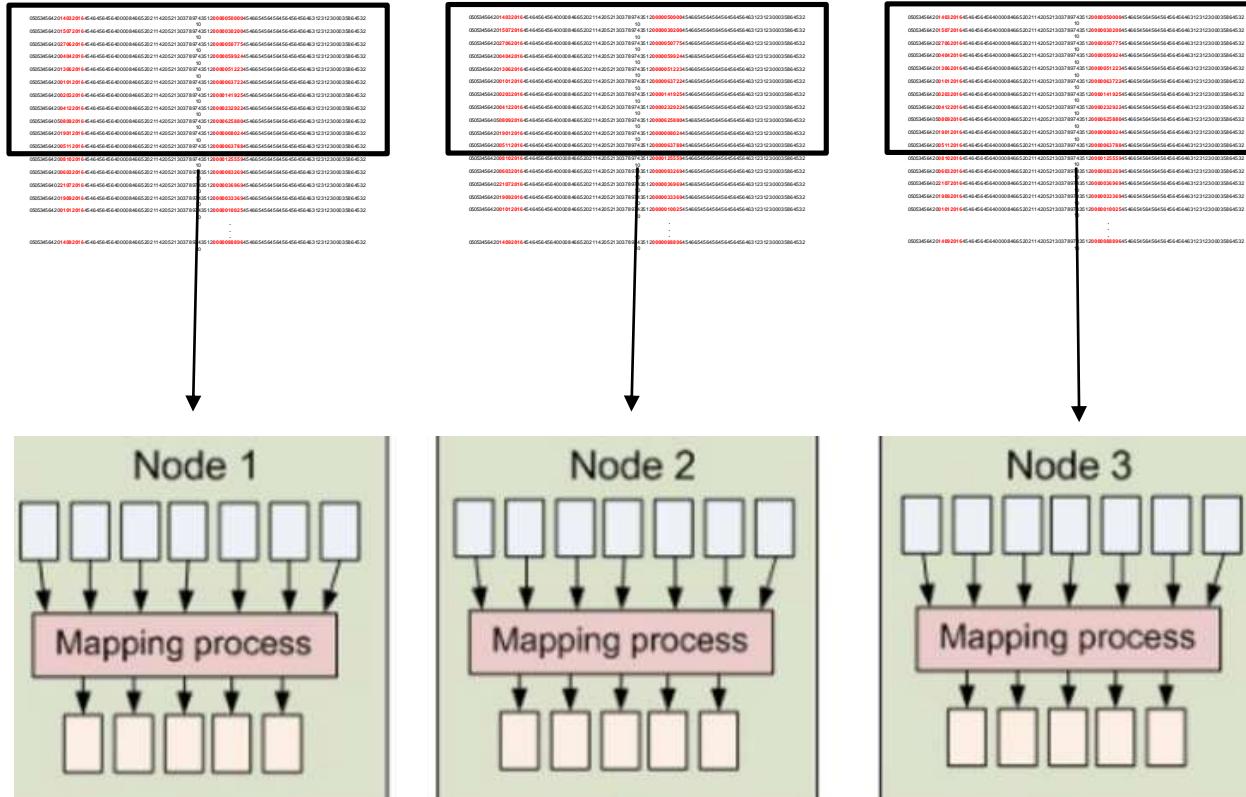
Dataset particionado

```
05053456420140320164546456456400084665202114205213037897435120000050000454665456456456456456463123123003586453210  
050534564201403201645464564564000846652021142052130378974351200003020045466545645645645456456463123123003586453210  
0505345642005072016454645645640008466520211420521303789743512000005975454665456456456456456463123123003586453210  
0505345642007052016454645645640008466520211420521303789743512000005992454665456456456456456463123123003586453210  
0505345642014032016454645645640008466520211420521303789743512000051223454665456456456456456463123123003586453210  
0505345642009042016454645645640008466520211420521303789743512000063722454665456456456456456463123123003586453210  
05053456420200420164546456456400084665202114205213037897435120000141925454665456456456456456463123123003586453210  
0505345642014032016454645645640008466520211420521303789743512000002322454665456456456456456463123123003586453210  
0505345642009042016454645645640008466520211420521303789743512000005598454665456456456456456463123123003586453210  
05053456420070520164546456456400084665202114205213037897435120000024454665456456456456456456456463123123003586453210  
0505345642009042016454645645640008466520211420521303789743512000063788454665456456456456456456463123123003586453210  
0505345642009042016454645645640008466520211420521303789743512000012559454665456456456456456456463123123003586453210  
0505345642009042016454645645640008466520211420521303789743512000083269454665456456456456456456463123123003586453210  
050534564200904201645464564564000846652021142052130378974351200003699454665456456456456456456463123123003586453210  
0505345642013052016454645645640008466520211420521303789743512000003369454665456456456456456456463123123003586453210  
0505345642007052016454645645640008466520211420521303789743512000063585454665456456456456456456463123123003586453210  
050534564200904201645464564564000846652021142052130378974351200001200454665456456456456456456463123123003586453210  
050534564200507201645464564564000846652021142052130378974351200008250454665456456456456456456463123123003586453210  
05053456420140320164546456456400084665202114205213037897435120000063750454665456456456456456456463123123003586453210  
0505345642007052016454645645640008466520211420521303789743512000006121454665456456456456456456463123123003586453210  
0505345642009042016454645645640008466520211420521303789743512000013769454665456456456456456456463123123003586453210  
050534564200801120164546456456400084665202114205213037897435120000018936454665456456456456456456463123123003586453210  
05053456420080112016454645645640008466520211420521303789743512000002098454665456456456456456456463123123003586453210  
050534564200904201645464564564000846652021142052130378974351200000515164665456456456456456456456463123123003586453210  
0505345642009042016454645645640008466520211420521303789743512000008269454665456456456456456456456463123123003586453210  
0505345642009042016454645645640008466520211420521303789743512000008269454665456456456456456456456463123123003586453210  
050534564200801120164546456456400084665202114205213037897435120000036525454665456456456456456456456463123123003586453210  
05053456420090420164546456456400084665202114205213037897435120000036885454665456456456456456456456463123123003586453210  
050534564200200120164546456456400084665202114205213037897435120000023785454665456456456456456456463123123003586453210  
0505345642020012016454645645640008466520211420521303789743512000001025454665456456456456456456463123123003586453210
```

⋮
050534564200705201645464564564000846652021142052130378974351200008886454665456456456456456463123123003586453210



Funções map, reduce e combine



Funções map, reduce e combine

- O que a função **map** vai fazer?
- Cada linha do arquivo é uma execução do Map.
- Retirar de cada linha somente o que é útil.

Funções map, reduce e combine

<CHAVE, VALOR>

14032016	0000050000
14032016	0000030200
05072016	0000050775
07052016	0000059924
14032016	0000051223
08042016	0000063722
20012016	0000141925
14032016	0000232922
08042016	0000625880
07052016	0000008024
05072016	0000063788
08042016	0000125559
14032016	0000083269
08042016	0000036969
13052016	0000033369
07052016	0000063585
08042016	0000012000
05072016	0000082550
14032016	0000063750
07052016	0000061212
13052016	0000013769
08112016	0000181936
08112016	0000136952
08112016	0000852069
08112016	0000635252
14032016	0000036885
20012016	0000023785
20012016	0000010025

07052016

0000088896

Funções map, reduce e combine

- Parâmetros da função Map:
 - **Key**: gerada automaticamente pelo framework.
 - **Value**: toda a linha do arquivo:
 - 05053456420**14032016**4546456456400008466520211420521303789743512**0000050000**4546654564564564564631231230003586453210

```
public static class MapStageIGTI extends MapReduceBase implements Mapper<LongWritable, Text, Text, Text> {  
    public void map(LongWritable key, Text value, OutputCollector<Text, Text> output, Reporter reporter) throws  
    IOException  
    {  
        Text txtKey = new Text();  
        Text txtValue = new Text();  
  
        String Data = value.substring(12, 19);  
        txtKey.set(Data);  
  
        String Valor = value.substring(63,72);  
        txtValue.set(Valor);  
  
        output.collect(txtKey, txtValue);  
    }  
}
```

Funções map, reduce e combine

- Parâmetros da função Map:

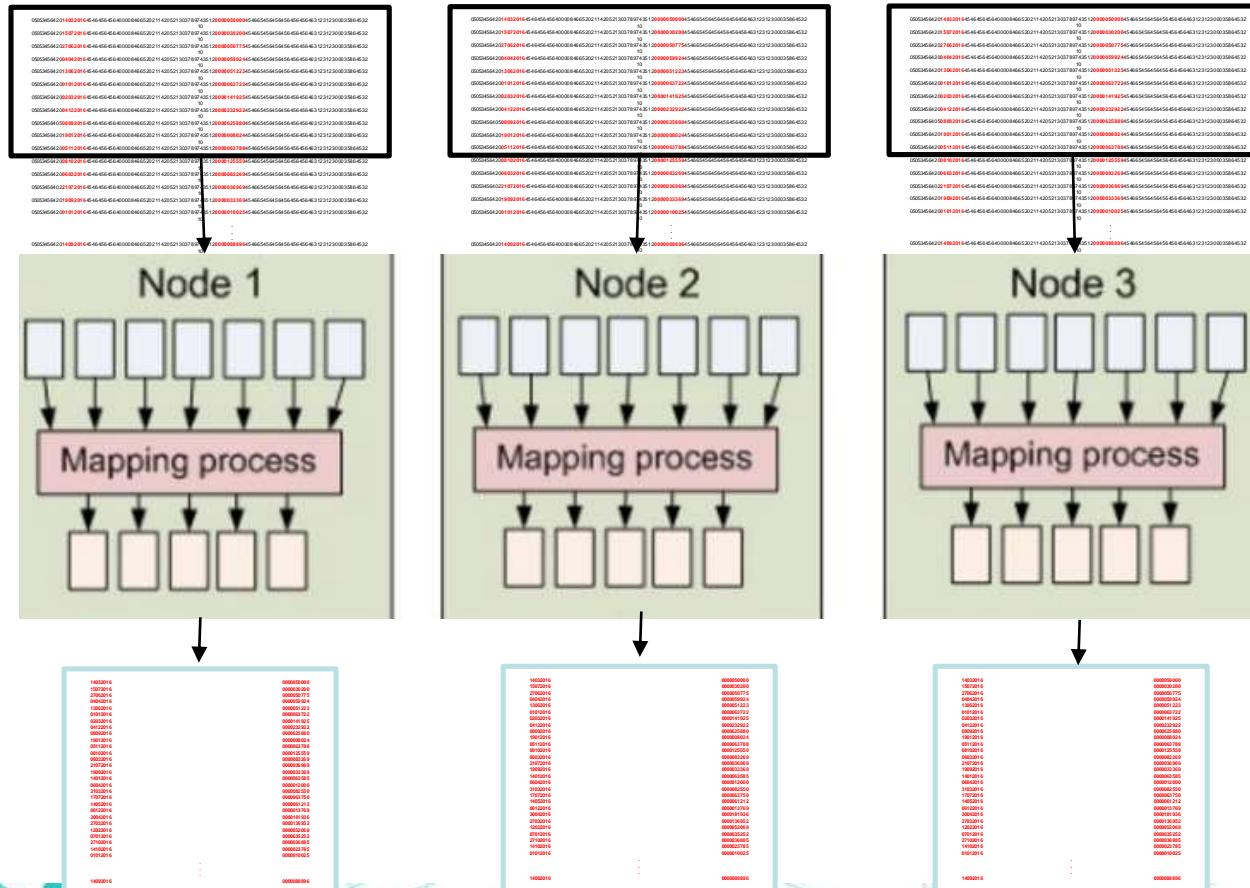
- **Key**: gerada automaticamente pelo framework.

- **Value**: toda a linha do arquivo:

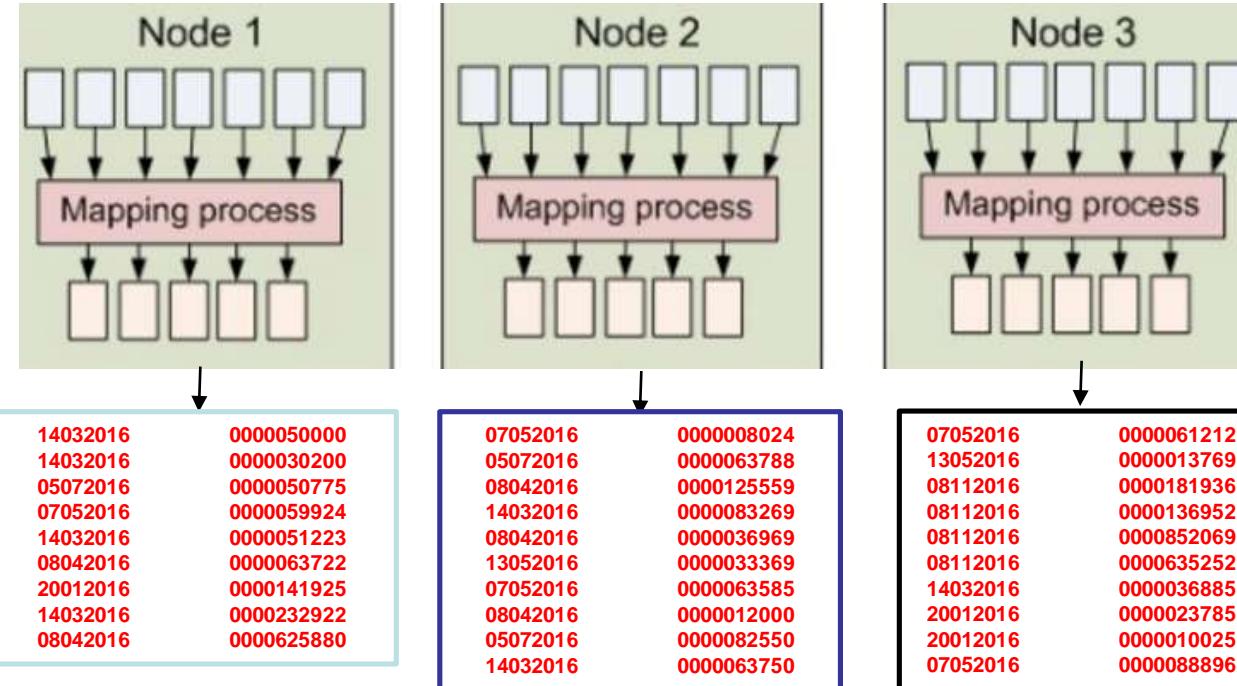
- 05053456420**14032016**4546456456400008466520211420521303789743512**0000050000**4546654564564564564631231230003
586453210

```
public static class MapStageIGTI extends MapReduceBase implements Mapper<LongWritable, Text, Text, Text> {  
    public void map(LongWritable key, Text value, OutputCollector<Text, Text> output, Reporter reporter) throws  
    IOException  
    {  
        Text txtKey = new Text();  
        Text txtValue = new Text();  
  
        String Data = value.substring(12, 19);  
        txtKey.set(Data); 14032016  
  
        String Valor = value.substring(63,72);  
        txtValue.set(Valor); 0000050000  
  
        output.collect(txtKey, txtValue);  
    }  
    14032016 0000050000  
}
```

Funções map, reduce e combine



Funções map, reduce e combine

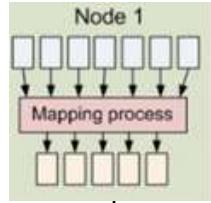


Saídas das funções map

Funções map, reduce e combine

- Map.
- **Combine.**
- Agrupamento e ordenação (Shuffle).
- Reduce.

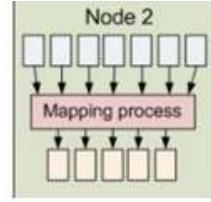
Funções map, reduce e combine



14032016	0000050000
14032016	0000030200
05072016	0000050775
07052016	0000059924
14032016	0000051223
08042016	0000063722
20012016	0000141925
14032016	0000232922
08042016	0000625880

Combine

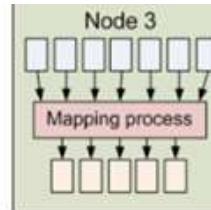
05072016	0000050775
07052016	0000059924
20012016	0000141925
14032016	0000232922
08042016	0000625880



07052016	0000008024
05072016	0000063788
08042016	0000125559
14032016	0000083269
08042016	0000036969
13052016	0000033369
07052016	0000063585
08042016	0000012000
05072016	0000082550
14032016	0000063750

Combine

08042016	0000125559
14032016	0000083269
13052016	0000033369
07052016	0000063585
05072016	0000082550



07052016	0000061212
13052016	0000013769
08112016	0000181936
08112016	0000136952
08112016	0000852069
08112016	0000635252
14032016	0000036885
20012016	0000023785
20012016	0000010025
07052016	0000088896

Combine

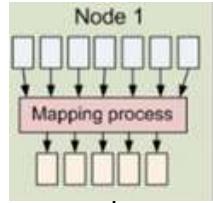
13052016	0000013769
08112016	0000852069
14032016	0000036885
20012016	0000023785
07052016	0000088896

Cada função combine é executada no próprio datanode

Funções map, reduce e combine

- Map.
- Combine.
- Agrupamento e ordenação (Shuffle).
- Reduce.

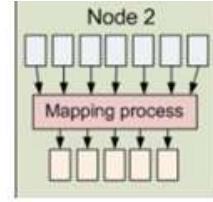
Funções map, reduce e combine



05072016	0000050775
07052016	0000059924
20012016	0000141925
14032016	0000232922
08042016	0000625880

Ordenação

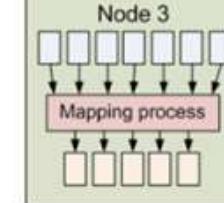
20012016	0000141925
14032016	0000232922
08042016	0000625880
07052016	0000059924
05072016	0000050775



08042016	0000125559
14032016	0000083269
13052016	0000033369
07052016	0000063585
05072016	0000082550

Ordenação

14032016	0000083269
08042016	0000125559
07052016	0000063585
13052016	0000033369
05072016	0000082550

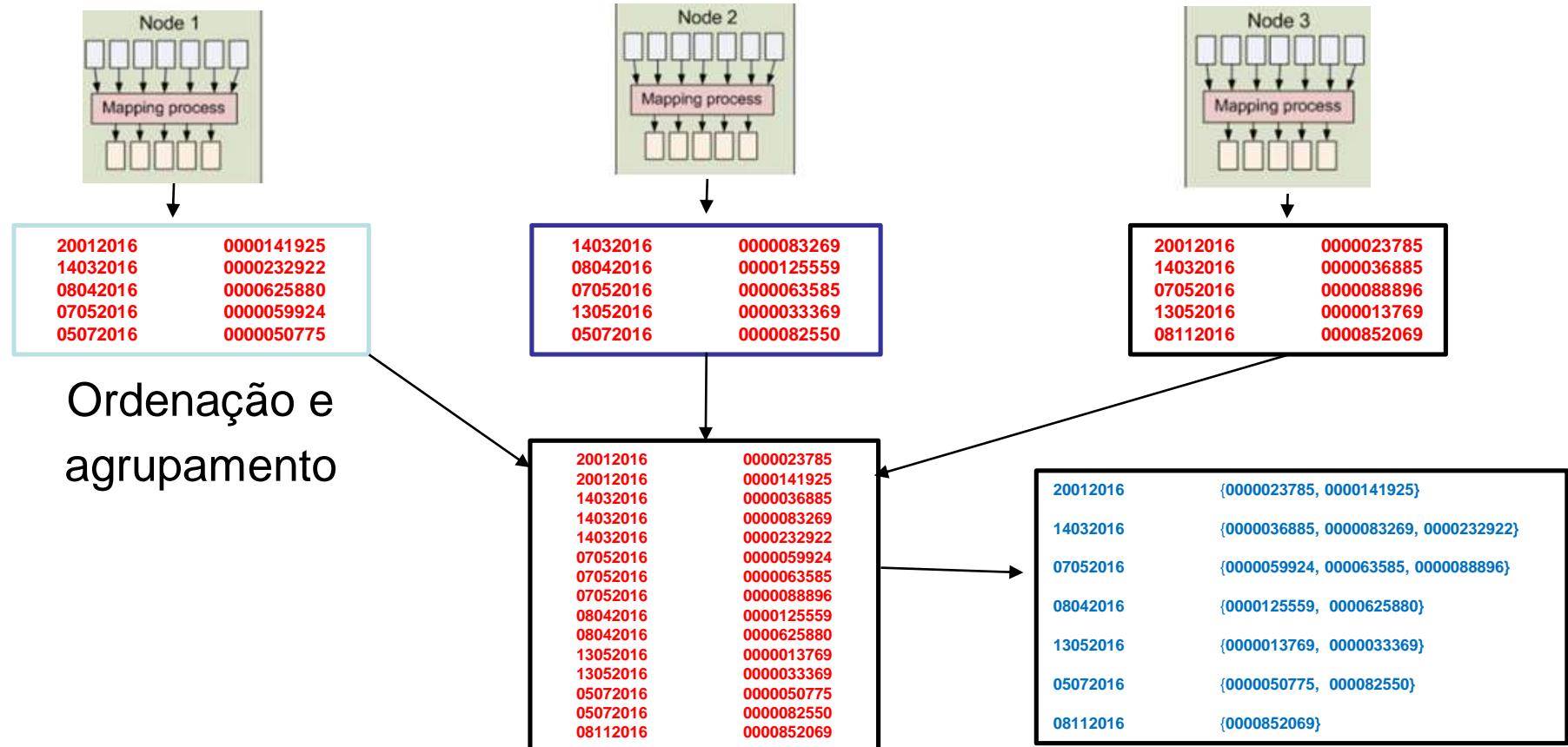


13052016	0000013769
08112016	0000852069
14032016	0000036885
20012016	0000023785
07052016	0000088896

Ordenação

20012016	0000023785
14032016	0000036885
07052016	0000088896
13052016	0000013769
08112016	0000852069

Funções map, reduce e combine



Funções map, reduce e combine

- Map.
- Combine.
- Agrupamento e ordenação (Shuffle).
- Reduce.

Funções map, reduce e combine

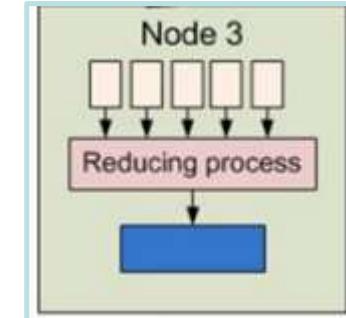
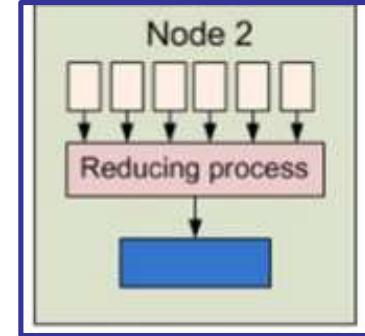
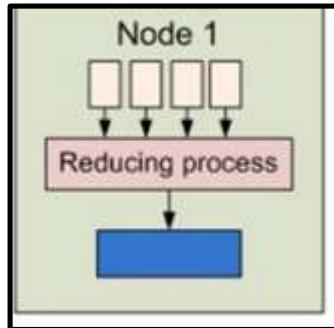
20012016	{0000023785, 0000141925}
14032016	{0000036885, 0000083269, 0000232922}
07052016	{0000059924, 000063585, 0000088896}
08042016	{0000125559, 0000625880}
13052016	{0000013769, 0000033369}
05072016	{0000050775, 000082550}
08112016	{0000852069}

Entrada de cada fase reduce

05072016	{0000050775, 000082550}
08112016	{0000852069}

08042016	{0000125559, 0000625880}
13052016	{0000013769, 0000033369}

20012016	{0000023785, 0000141925}
14032016	{0000036885, 0000083269, 0000232922}
07052016	{0000059924, 000063585, 0000088896}



Funções map, reduce e combine

- Como encontrar a maior venda do dia?

Funções map, reduce e combine

- Parâmetros da função Reduce:
 - Key: **14032016 (A Data)**
 - Value: Uma lista com todos os values (vendas)
 - **{0000036885, 0000083269, 0000232922}**

```
public static class RedStageIGTI extends MapReduceBase implements Reducer<Text, Text, Text, Text>{  
    public void reduce (Text key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {  
        Text vendaCorrente = new Text();  
        Double maiorVenda = 0;  
        Text txtMaiorVenda = new Text();  
  
        while (values.hasNext()) {  
            vendaCorrente = values.next();  
            if (vendaCorrente.toDouble() > maiorVenda )  
                maiorVenda = vendaCorrente;  
        }  
  
        txtMaiorVenda.set(maiorVenda.toString());  
        output.collect(key, txtMaiorVenda);  
    }  
}
```

Funções map, reduce e combine

- Parâmetros da função Reduce:
 - Key: 14032016 (A Data)
 - Value: Uma lista com todos os values (vendas)
 - {0000036885, 0000083269, 0000232922}

```
public static class RedStageIGTI extends MapReduceBase implements Reducer<Text, Text, Text, Text>{  
    public void reduce (Text key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {  
        Text vendaCorrente = new Text();  
        Double maiorVenda = 0;  
        Text txtValue = new Text();  
  
        {0000036885, 0000083269, 0000232922}  
        while (values.hasNext()) {  
            vendaCorrente = values.next();  
            if (vendaCorrente.toDouble() > maiorVenda )  
                maiorVenda = vendaCorrente;  
        }  
  
        txtValue.set(maiorVenda.toString());  
        output.collect(key, txtValue);  
    }  
    14032016  
}
```



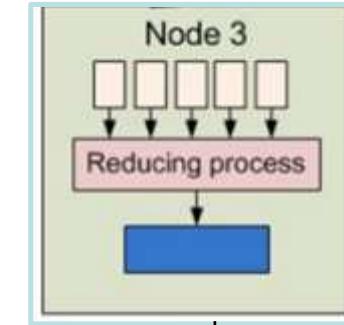
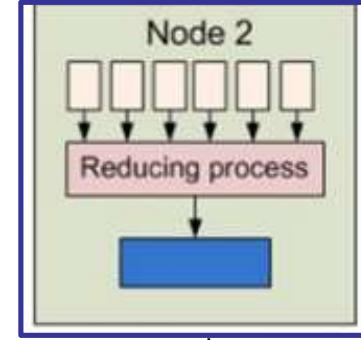
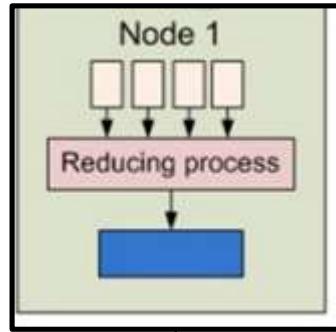
Funções map, reduce e combine

Entrada de cada fase reduce

05072016 {0000050775, 000082550}
08112016 {0000852069}

08042016 {0000125559, 0000625880}
13052016 {0000013769, 0000033369}

20012016 {0000023785, 0000141925}
14032016 {0000036885, 0000083269, 0000232922}
07052016 {0000059924, 000063585, 0000088896}



05072016 {000082550}
08112016 {0000852069}

08042016 {0000625880}
13052016 {0000033369}

20012016 {0000141925}
14032016 {0000232922}
07052016 {0000088896}

Saída das fases reduce para o HDFS

Conclusão

- Map.
- Reduce.
- Combine.

Próxima aula

- Tolerância a falhas.



Aula 2.8 Tolerância a falhas



Nesta aula

- Tolerância a falhas.

Tolerância a falhas

- Aplicações podem se deparar com problemas:
 - Erros no código.
 - Falhas de máquinas (memória, disco, etc.).
 - Processos inesperadamente encerrados.
 - Falhas na rede.
- Um dos maiores benefícios do Hadoop é sua habilidade em lidar com falhas.
- Habilidade para garantir a conclusão dos trabalhos já iniciados.

- Quando um erro acontece a JVM comunica o fato ao TaskTracker (processo responsável pela execução de tarefas MapReduce).
- TaskTracker executa uma tarefa Map ou Reduce atribuída.
- O erro é registrado em um arquivo de log.
- O TaskTracker marca a tarefa com o status de falha.
- Libera o slot para execução de nova tarefa.



Tolerância a falhas

- Quando o TaskTracker fica muito tempo sem receber informações de um nó, a tarefa é considerada nula.
- Será identificado que aquele nó falhou.
- Nesse momento a tarefa será novamente agendada.
- TaskTracker tenta evitar novo agendamento para o mesmo nó.

Tolerância a falhas

- Caso uma tarefa falhe sucessivas vezes (parâmetro).
- Isso poderá resultar no cancelamento da tarefa.
- Nem sempre do trabalho total.
- Existe um parâmetro para definir qual o percentual aceito.

Tolerância a falhas

- Um dos maiores problemas em um cluster Hadoop é a falha do nó mestre.
- Por se tratar do nó controlador do cluster.
- Se não existir redundância do nó mestre, o Hadoop não conseguirá se recuperar dessa falha.

✓ Tolerância a falhas:

- Falhas no código ou estrutura.
- Recuperação de falhas.
- Nó mestre e nó escravo.

Próxima aula

- Formatos de entrada e saída.



Aula 2.9 Formatos de entrada e saída



- Formatos de entrada e saída do Hadoop.

Formatos de entrada e saída

- Formatos mais comuns: csv e txt.
- Mas não são somente esses.
- O Hadoop pode processar dados não-estruturados, como vídeo, áudio, texto, etc.
- A maioria dos dados são estruturados ou semiestruturados.

Formatos de entrada e saída

- A importância do formato do arquivo para a operação de split.
- Os dados que o Hadoop processa são divididos em blocos.
- Precisamos estar disponíveis para ler os dados em qualquer ponto do arquivo, mesmo fora da ordem. O processamento distribuído exige isso.

Formatos de entrada e saída

- Os arquivos csv têm facilidade para a realização do processo de split.
- Podemos começar a ler os csv's de qualquer ponto.
- Um arquivo XML é diferente. Não podemos começar a ler o arquivo no meio de uma tag.
- Se nossos arquivos forem menores que o bloco (blocksize) do HDFS, pode ser que não tenhamos problemas com o Split.
- Entretanto, arquivos pequenos são a exceção no Hadoop.

Formatos de entrada e saída

- Muitos pequenos arquivos pode ocasionar problemas de desempenho.
- Arquivos csv são muito comuns para troca de informações entre o Hadoop e sistemas externos.
- Registros JSON são diferentes de arquivos JSON. Eles tornam os arquivos divisíveis.

Formatos de entrada e saída

- Sequence files.
- Armazenam dados em formato binário.
- Utilizado para armazenamento de dados complexos.
- Muito utilizado para processamento de áudio e vídeo.

Conclusão

- ✓ Formatos de entrada e saída.



Módulo 3 - Solução de Dados utilizando Ecossistema Hadoop

Capítulo 3. O HDFS (Hadoop Distributed Filesystem)

Prof. João Paulo Barbosa Nascimento



Aula 3.1. Formato, conceitos e comandos básicos do HDFS



Nesta aula

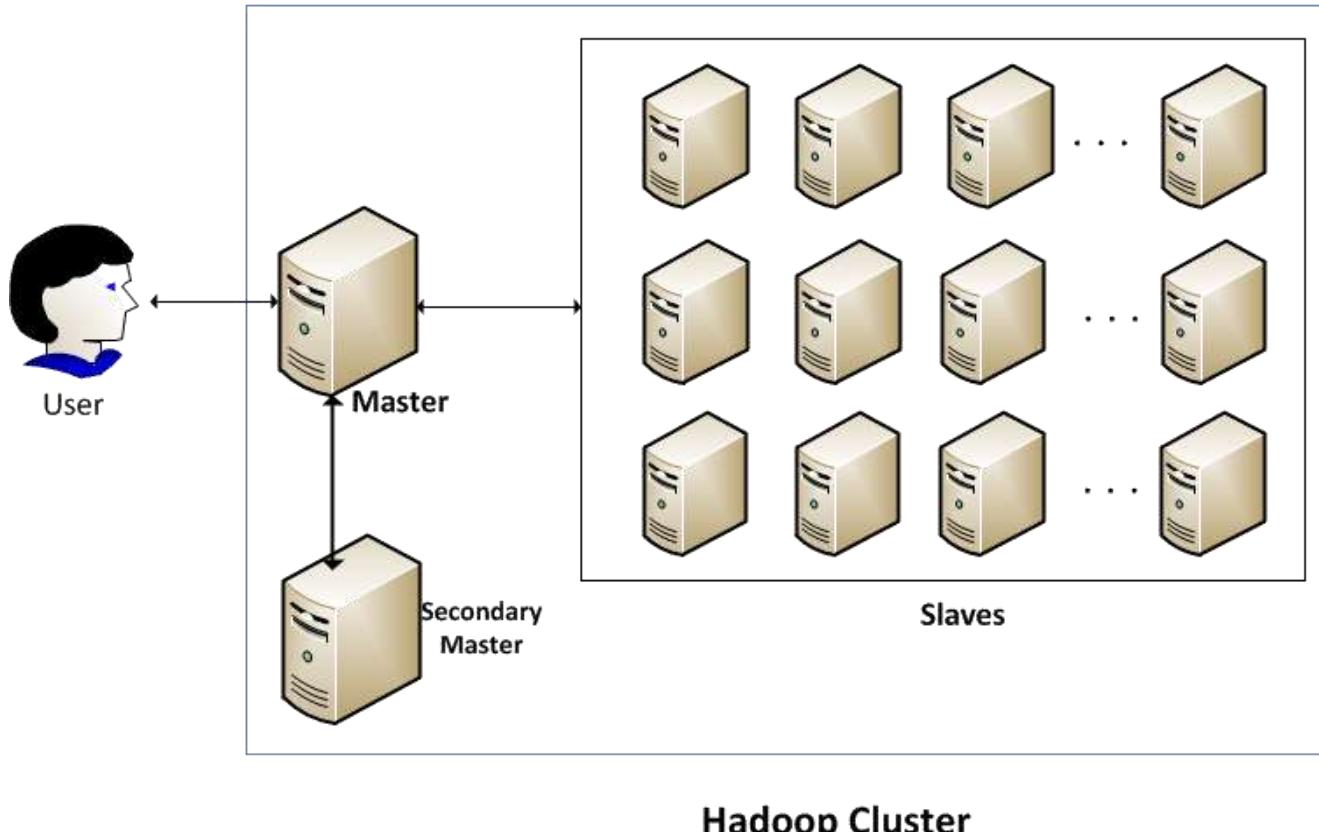
- ❑ Formato do HDFS.
- ❑ Conceitos e Comandos Básicos do HDFS.

Formato do HDFS

- Sistema de arquivos distribuídos:
 - Conjunto de dados (*dataset*) extrapola a capacidade de armazenamento de uma máquina.
 - Necessário realizar o particionamento desse *dataset* através de outras máquinas.
 - Interligadas em cluster.
 - Sistemas de arquivos distribuídos: gerenciam e armazenam dados através de uma rede.
 - Sistema de arquivos mais complexo que sistemas que atuam localmente.
 - Maior desafio: gerenciar e tolerar falhas. Evitar a perda de dados.
 - Hadoop possui seu próprio sistema de arquivos distribuído.



Formato do HDFS



Formato do HDFS

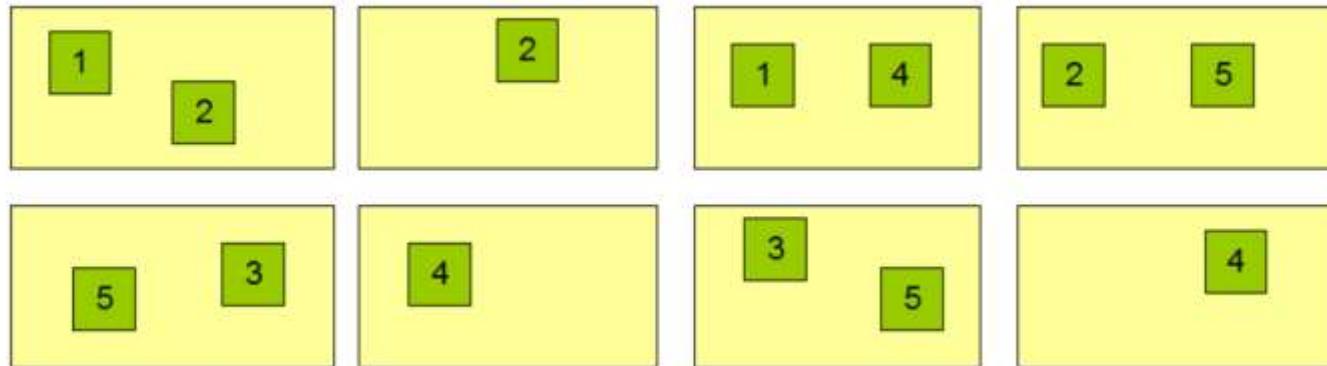
- HDFS:
 - Sistema de arquivos distribuídos.
 - Projetado para armazenar arquivos grandes.
 - Distribuído em grandes clusters.
- Grandes Arquivos:
 - Arquivos com centenas de Megabytes, Gigabytes ou Terabytes de tamanho.
 - Alguns clusters Hadoop já trabalham com Petabytes.
 - HDFS está preparado também para trabalhar com vários pequenos arquivos.
 - Diversos estudos compararam o comportamento da ferramenta nas duas situações.
 - Granularidade.

Formato do HDFS

Block Replication

```
Namenode (Filename, numReplicas, block-ids, ...)  
/users/sameerp/data/part-0, r:2, {1,3}, ...  
/users/sameerp/data/part-1, r:3, {2,4,5}, ...
```

Datanodes



Formato do HDFS

- Acesso aos Dados
 - Baseado em um padrão eficiente de processamento.
 - Escreva uma vez e leia muitas vezes.
 - Dados copiados de uma fonte e diversos tipos de análises são feitas.
 - Essas análises podem envolver todo o conjunto ou apenas uma parte.
- Hardware
 - HDFS não exige um grande e caro cluster.
 - Hadoop foi projetado para executar em clusters homogêneos ou heterogêneos.



Conceitos e comandos básicos do HDFS



- Para interagir com o HDFS o usuário utiliza linhas de comando.
- Existem outras formas de interação.
- Linhas de comando é a forma mais eficiente.
- Mais familiar aos desenvolvedores.

```
[root@sandbox ~]# hadoop fs -help
Usage: hadoop fs [generic options]
[-appendToFile <localsrc> ... <dst>]
[-cat [-ignoreCrc] <src> ...]
[-checksum <src> ...]
[-chgrp [-R] GROUP PATH...]
[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
[-chown [-R] [OWNER][:GROUP]] PATH...
[-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
[-copyToLocal [-p] [-ignorecrc] <src> ... <localdst>]
[-count [-q] [-h] [-v] [-t [<storage type>]] <path> ...]
[-cp [-f] [-p | -p[topax]] <src> ... <dst>]
[-createSnapshot <snapshotDir> [<snapshotName>]
[-deleteSnapshot <snapshotDir> <snapshotName>]
[-df [-h] [<path> ...]]
[-du [-s] [-h] <path> ...]
[-expunge]
[-find <path> ... <expression> ...]
```

Conceitos e comandos básicos do HDFS

- HDFS fornece comandos para
 - Criar diretórios
 - Adicionar, mover ou excluir arquivos
 - Listar o conteúdo de diretórios
 - Formatar o sistema de arquivos
- Todos os comandos podem ser consultados usando:
 - `hadoop fs -help`



- Sintaxe dos comandos:
 - Similar aos comandos do Linux.
 - Iniciados por “hadoop dfs” ou “hdfs dfs”.
 - `hadoop dfs -comando [argumentos]`.

Conceitos e Comandos Básicos do HDFS



- Alguns comandos úteis:
 - Formatação do HDFS: `bin/hdfs namenode -format`
 - Criação de diretórios: `hdfs dfs -mkdir arquivos_hadoop`
 - Inserir arquivos: `hdfs dfs -put meuarquivo.txt /user/hadoop_user`
 - Listar arquivos: `hdfs dfs -ls`
 - Remover arquivos: `hdfs dfs rm <path>`
 - Remover diretórios: `hdfs dfs rmr <path>`



Conceitos e comandos básicos do HDFS



```
% hadoop fs -mkdir books
% hadoop fs -ls .
Found 2 items
drwxr-xr-x    - tom supergroup          0 2009-04-02 22:41 /user/tom/books
-rw-r--r--    1 tom supergroup      118 2009-04-02 22:29 /user/tom/quangle.txt
```

- Informação retornada similar ao ls – l do Linux.
- Primeira coluna: permissões do arquivo ou diretório.
- Segunda coluna: fator de replicação.
- Terceira e quarta colunas: proprietário do arquivo (usuário e grupo).
- Quinta coluna: tamanho do arquivo.
- Sexta e sétima colunas: data e hora de última modificação.
- Última coluna: caminho e nome completo do arquivo ou diretório.

Conceitos e comandos básicos do HDFS



- Linhas de comando podem ser usadas sem muita dificuldade.
- Principalmente para conhecedores de Linux.
- Existem outras formas de acesso e manipulação.
- Providas pelo framework.

Conclusão

- Sistema de arquivos distribuídos.
- Sistema de arquivos distribuídos do Hadoop.
- Comandos úteis.
 - Linhas de comando.

Próxima aula

- Namenodes.
- Datanodes.



Aula 3.2. Namenodes e Datanodes



- Namenodes.
- Datanodes.

Namenodes e Datanodes

- Um cluster Hadoop possui dois tipos de nós:
 - Namenode.
 - Datanode.
- Realizam operações no padrão mestre-escravo:
 - Namenode (nó mestre) – geralmente 1. Algumas vezes 2.
 - Datanode (nó escravo) – número indefinido.



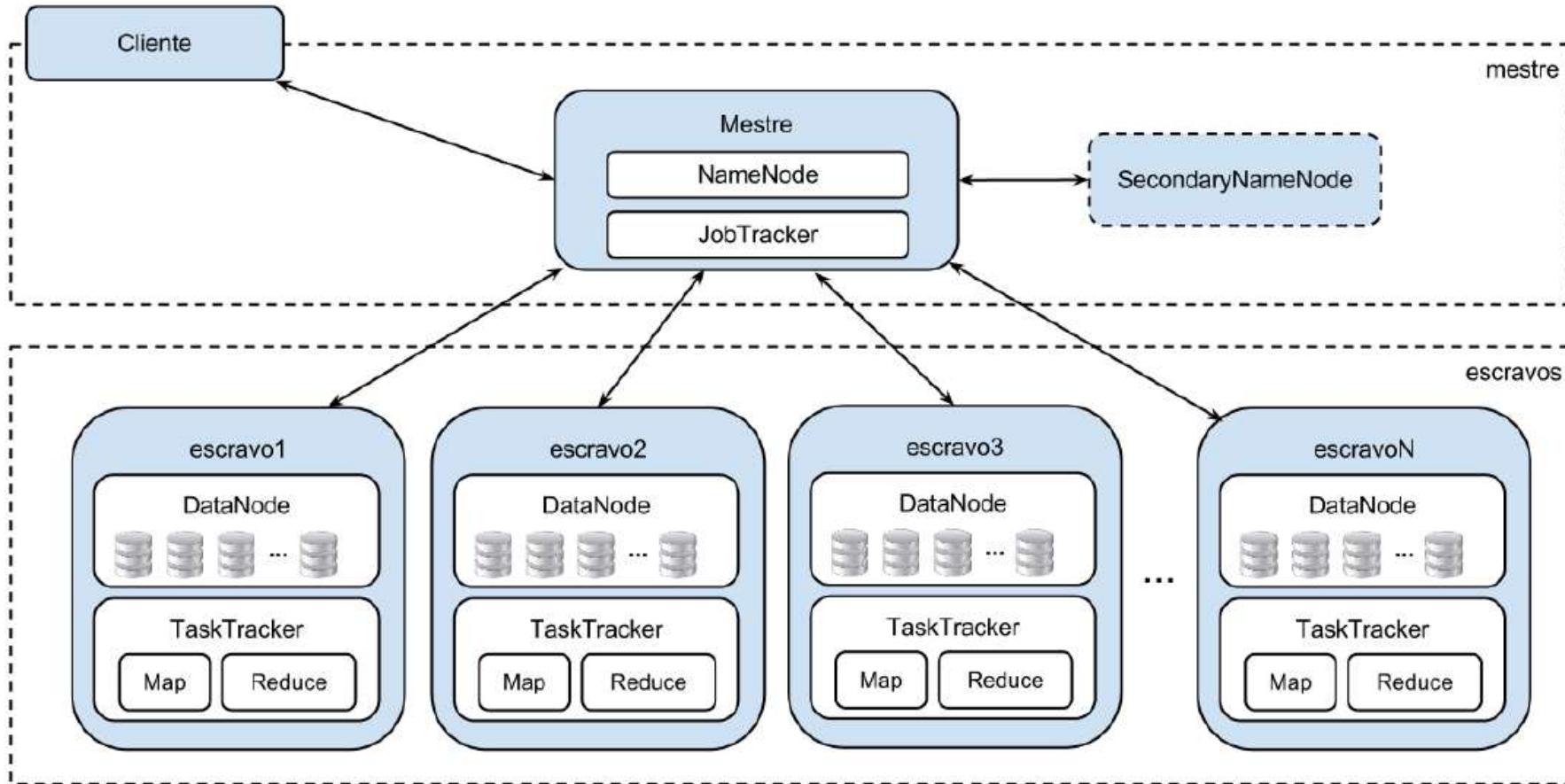
- O Namenode gerencia o sistema de arquivos (HDFS).
- Mantém a árvore de diretórios e arquivos.
- Possui acesso a todos os Datanodes do cluster, onde os blocos de dados serão enviados.
- Sem o Namenode não há como usar o sistema de arquivos.

- Funções:
 - Realizar a divisão dos arquivos em blocos.
 - Mapear a localização dos blocos de dados.
 - Encaminhar blocos aos nós escravos.
 - Fica localizado no nó-mestre da aplicação.

- Se o Namenode sofrer algum problema grave, todos os arquivos do HDFS serão perdidos.
- Caso as informações para sua reconstrução não esteja nos Datanodes.
- Dois tipos de recuperação:
 - Backup dos dados nos Datanodes.
 - Segundo Namenode.

- Os Datanodes são os “trabalhadores” do sistema.
- São instruídos pelo Namenode.
- Armazenam e recuperam blocos de dados.
- Periodicamente enviam ao Namenode uma lista com os blocos de dados que estão armazenando.

Datanode



Conclusão

Namenode:

- Responsável pelo HDFS.
- Ligado ao nó mestre.
- Distribui as tarefas.

Datanode:

- Nó escravo.
- Executa os trabalhos.
- Armazena blocos de dados do HDFS.

Próxima aula

- ❑ Operações básicas do HDFS.



Aula 3.3. Operações básicas do HDFS



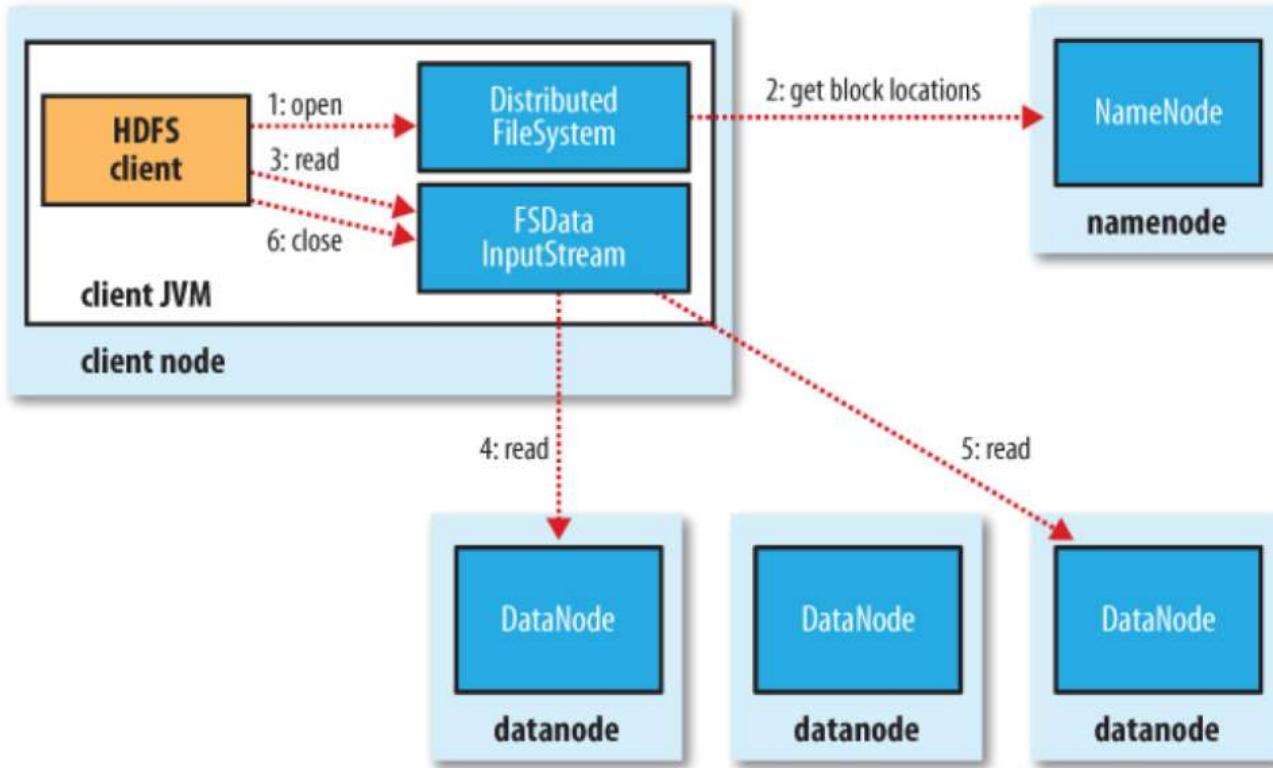
Nesta aula

- Fluxo de dados entre nó-mestre e nó-escravo.
- Processo de leitura.
- Processo de escrita.

Operações Básicas do HDFS

- Processo de leitura e processo de escrita são efetuadas o tempo todo no HDFS.
- Grande troca de dados entre os nós mestre e escravos.
- Processo definido.
- Começaremos pelo processo de leitura.

Processo de Leitura



1 - O HDFS Client abre o arquivo desejado chamando a função `open()` no objeto `FileSystem`

1.1 – `FileSystem` é um objeto do tipo `DistributedFileSystem`

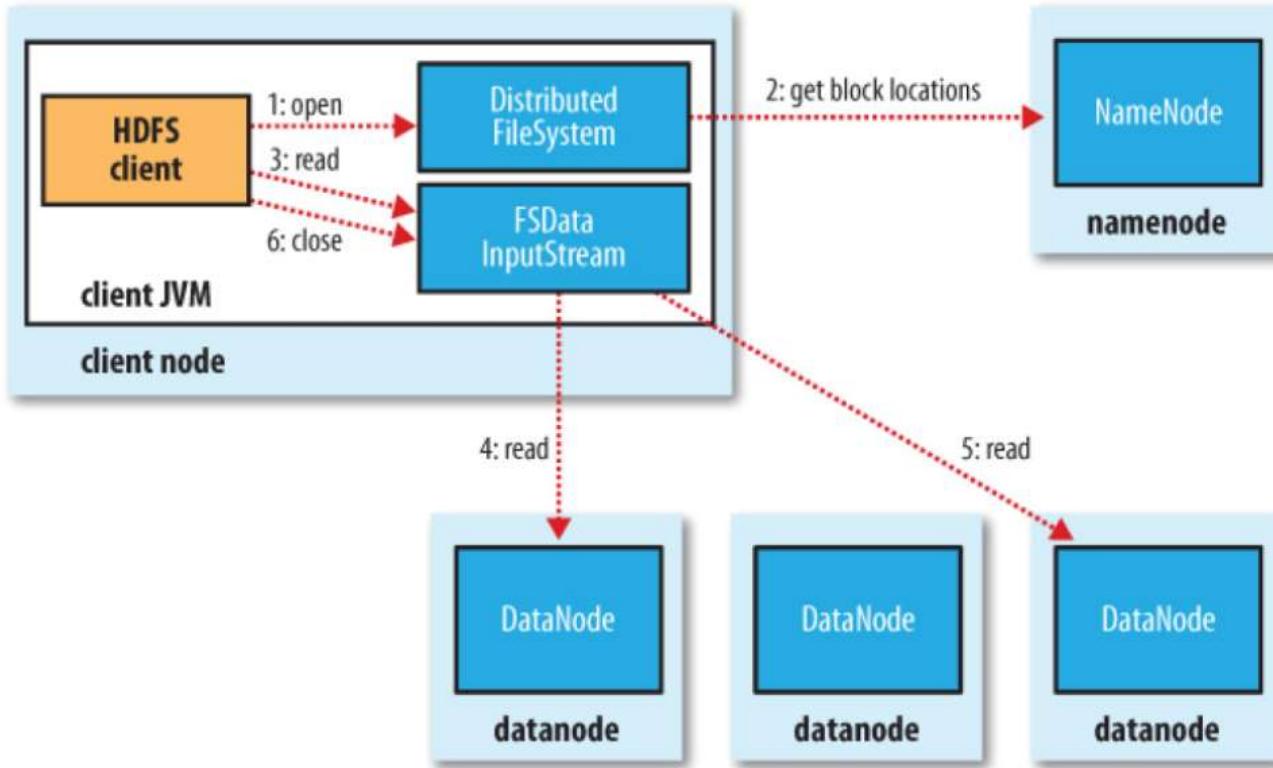
2 – `DistributedFileSystem` chama o `Namenode` e solicita informações sobre a localização dos arquivos

2.1 – O `Namenode` retorna os nomes dos `Datanodes` que possuem cópias do bloco

2.2 – Os `Datanodes` estão ordenados de acordo com sua proximidade com o cliente

3.3 – `DistributedFileSystem` retorna um `FSDataInputStream` para o HDFS Cliente para que ele possa ler os dados

Processo de Leitura



3 – O cliente HDFS chama a função Read() para a leitura dos dados

4 – O DFSInputStream, que possui armazenado os endereços dos Datanodes, conecta-se ao Datanode mais próximo

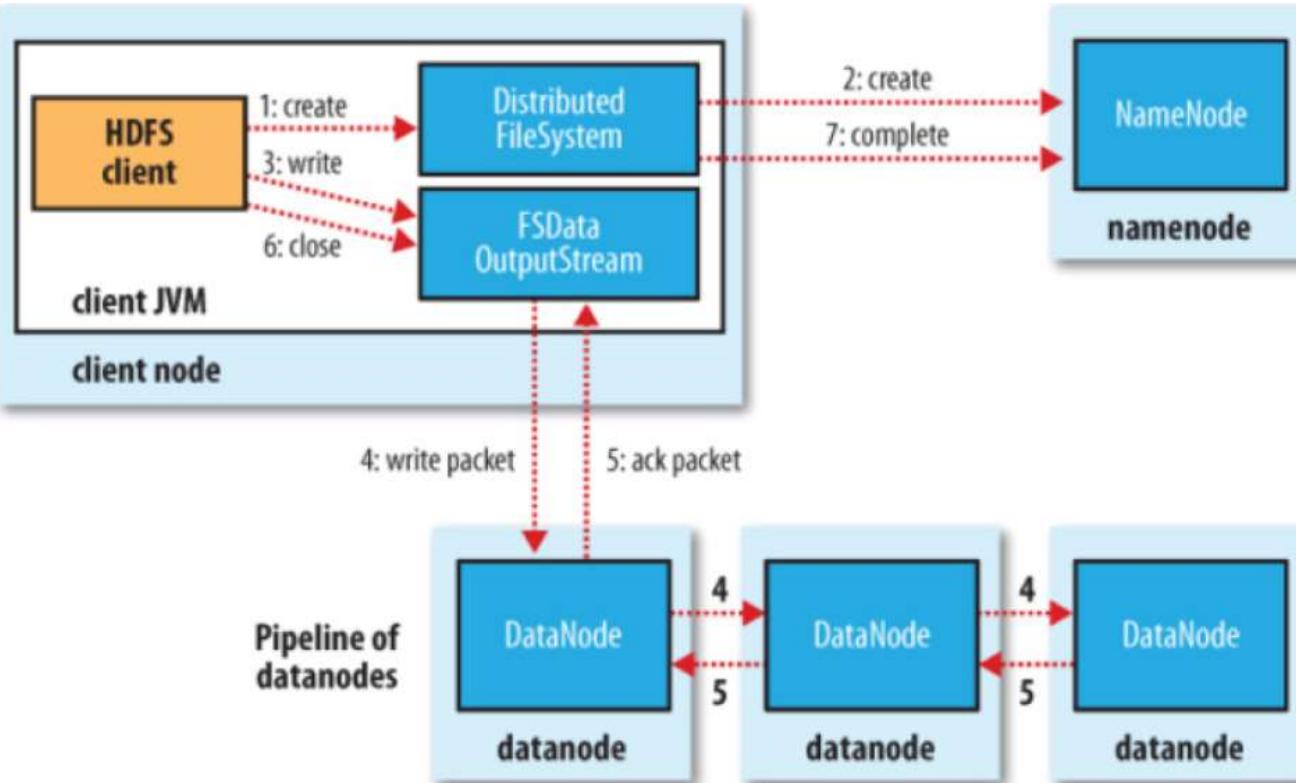
4.1 – Os dados são enviados do Datanode de volta ao cliente chamando repetidas vezes a função Read()

5 – Quando o fim do bloco é alcançado, DFSInputStream fechará a conexão com o Datanode

Operações Básicas do HDFS

- Em seguida temos o processo de escrita de arquivos no HDFS.

Processo de Escrita



1 – O HDFS Cliente solicita a criação do arquivo chamando a função Create() do DistributedFileSystem

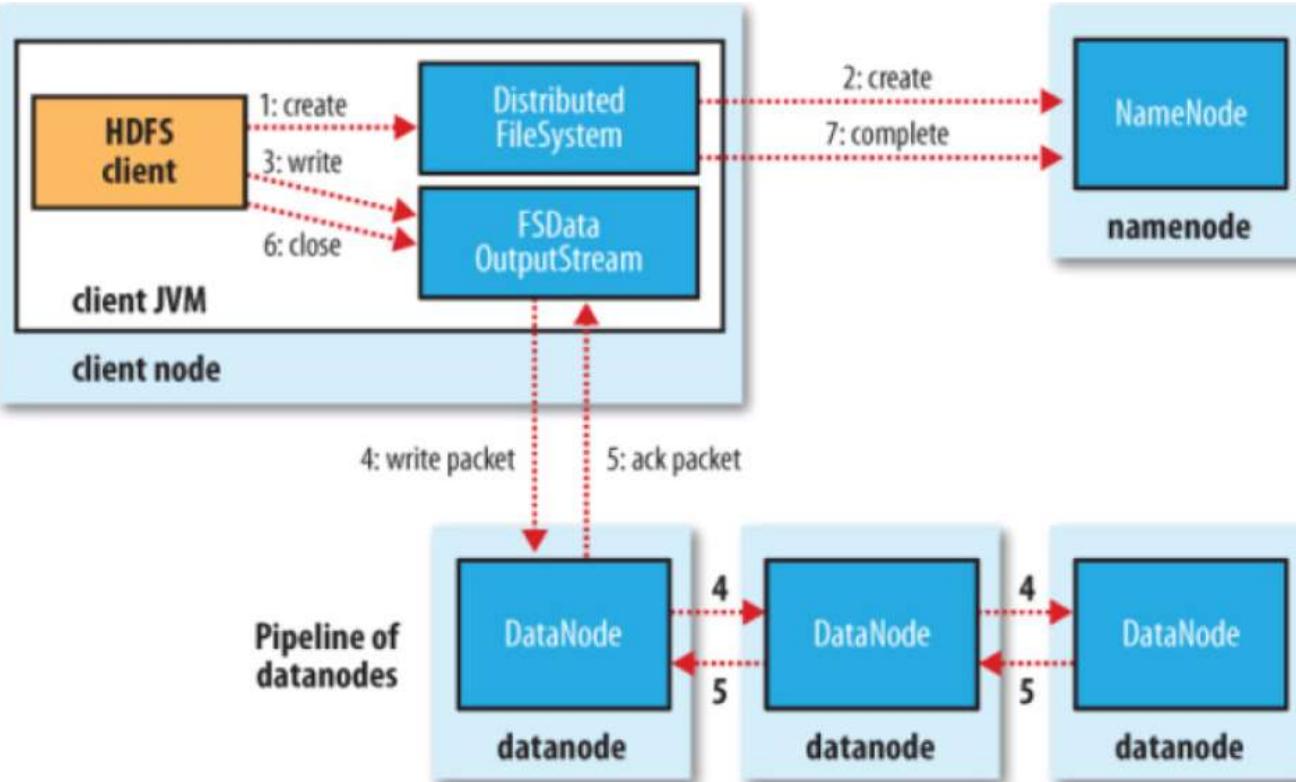
2 – DistributedFileSystem faz uma chamada ao Namenode para que o arquivo seja criado

2.1 – Namenode faz várias verificações para descobrir se o arquivo já existe e se existe permissão para a criação

2.3 – Se existir permissão o Namenode cria o arquivo, senão a criação falha e o cliente recebe uma exceção

2.4 – DistributedFileSystem retorna um FsDataOutputStream para o início da escrita

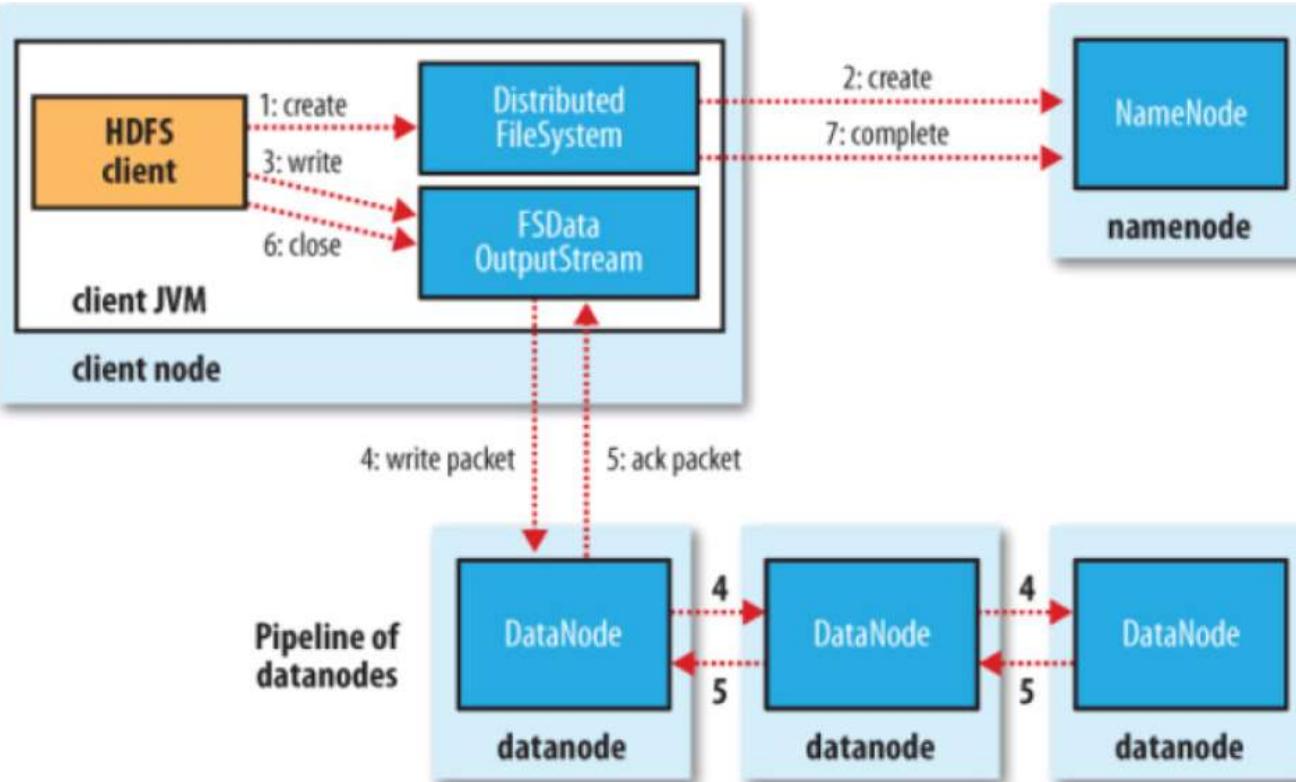
Processo de Escrita



3 – Cliente começa a escrever os dados e o DFSOutputStream particiona esses dados em uma fila interna (Data Queue)

3.1 – Objeto DataStreamer consome a fila, solicitando ao Namenode a alocação de novos blocos

Processo de Escrita

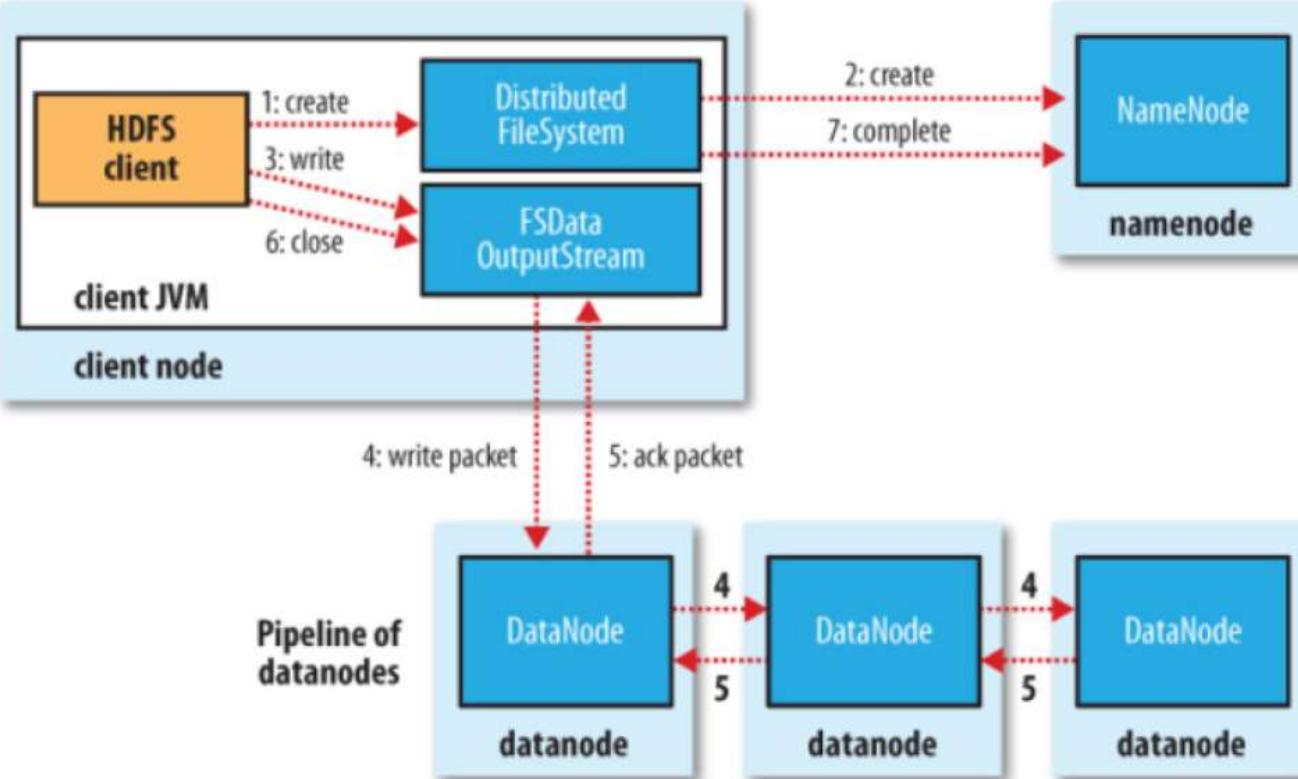


4 – DataStreamer escolhe em uma lista de Datanodes os mais adequados para armazenar as réplicas

4.1 – A lista de Datanodes forma um pipeline. Caso o número de réplicas seja 3, teremos 3 nós na pipeline

4.2 – DataStreamer divide os pacotes para o primeiro Datanode e esse Datanode repassa aos demais

Processo de Escrita



5 – O DFSOutputStream mantém uma fila interna de pacotes que está aguardando para serem conhecidos (Ack Queue)

5.1 – Um pacote é removido da Ack Queue apenas quando ele foi conhecido por todos os Datanodes

6 – Quando o cliente finaliza a escrita dos dados ele chama a função Close()

6.1 – Essa ação descarrega todos os pacotes restantes para a fila e aguarda que os Datanodes reconheçam

7 – Entra em contato com o Namenode e informa que a escrita do arquivo foi completa

Conclusão

- Operações de Escrita.
- Operações de Leitura.

Próxima aula

- ❑ Interfaces gráficas para acesso ao HDFS.



Aula 3.4. Interface gráfica para gerenciamento do HDFS



Nesta aula

- ❑ Interfaces do Hadoop.
- ❑ Consultando o HDFS.
- ❑ Consultando os Jobs.

Interface gráfica para acesso ao HDFS



- Hadoop traz algumas interfaces gráficas.
- Utilizadas para gerenciamento dos serviços.
- Podem ser acessadas pelo *browser*.
- Todos os passos de um trabalho podem ser acompanhados pela interface gráfica.

Interface gráfica para acesso ao HDFS



- Falhas podem ser acompanhadas.
- Uma das interfaces pode ser acessada pela porta 9870.
- <http://localhost:9870>
- Fornece informações sobre o HDFS e sobre os Datanodes.

Overview 'HadoopMaster:9000' (active)

Started:	Wed May 18 12:37:44 UTC 2016
Version:	2.7.1, r15ecc87ccf4a0228f35af08fc56de536e6ce657a
Compiled:	2015-06-29T06:04Z by jenkins from (detached from 15ecc87)
Cluster ID:	CID-ed310e79-b1a1-4c78-93b8-d35821a97577
Block Pool ID:	BP-652330868-10.54.8.12-1463575059827

Summary

Security is off.

Safemode is off.

102 files and directories, 86 blocks = 188 total filesystem object(s).

Heap Memory used 162.23 MB of 332.5 MB Heap Memory. Max Heap Memory is 889 MB.

Non Heap Memory used 61.08 MB of 63.27 MB Committed Non Heap Memory. Max Non Heap Memory is -1 B.

[Hadoop](#)[Overview](#)[Datanodes](#)[Datanode Volume Failures](#)[Snapshot](#)[Startup Progress](#)[Utilities](#) ▾

Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
metrica1-7:50010 (10.54.8.18:50010)	0	In Service	393.6 GB	11.56 MB	20.07 GB	373.52 GB	29	11.56 MB (0%)	0	2.7.1
metrica1-4:50010 (10.54.8.13:50010)	0	In Service	393.6 GB	11.47 MB	20.07 GB	373.52 GB	37	11.47 MB (0%)	0	2.7.1
metrica1-8:50010 (10.54.8.16:50010)	0	In Service	393.6 GB	11.29 MB	20.07 GB	373.52 GB	26	11.29 MB (0%)	0	2.7.1
metrica1-2:50010 (10.54.8.17:50010)	0	In Service	393.6 GB	13.93 MB	20.07 GB	373.52 GB	28	13.93 MB (0%)	0	2.7.1
metrica1-5:50010 (10.54.8.15:50010)	0	In Service	393.6 GB	12.96 MB	20.07 GB	373.52 GB	36	12.96 MB (0%)	0	2.7.1
metrica1-3:50010 (10.54.8.14:50010)	2	In Service	393.6 GB	6.98 MB	20.07 GB	373.52 GB	24	6.98 MB (0%)	0	2.7.1
metrica1-6:50010 (10.54.8.19:50010)	2	In Service	393.6 GB	15.16 MB	20.07 GB	373.52 GB	32	15.16 MB (0%)	0	2.7.1
HadoopMaster:50010 (10.54.8.12:50010)	0	In Service	393.6 GB	18.9 MB	20.07 GB	373.51 GB	46	18.9 MB (0%)	0	2.7.1

[Hadoop](#)[Overview](#)[Datanodes](#)[Datanode Volume Failures](#)[Snapshot](#)[Startup Progress](#)[Utilities ▾](#)

Datanode Volume Failures

There are no reported volume failures.

Hadoop, 2015.

Interface gráfica para acesso ao HDFS



Hadoop Overview Datanodes Snapshot Startup Progress Utilities ▾

Browse the file system
Logs

Browse Directory

/ Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwx-----	hduser	supergroup	0 B	18/05/2016 09:38:21	0	0 B	tmp
drwxr-xr-x	hduser	supergroup	0 B	18/05/2016 09:38:06	0	0 B	user

Hadoop, 2015.

Interface gráfica para acesso ao HDFS



Hadoop Overview Datanodes Snapshot Startup Progress Utilities ▾

Browse Directory

/user/hduser Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	supergroup	684.65 KB	18/05/2016 09:38:10	3	128 MB	Grafo_Internet
-rw-r--r--	hduser	supergroup	2.16 KB	18/05/2016 09:38:12	3	128 MB	Grafo_Internet200
-rw-r--r--	hduser	supergroup	532 B	18/05/2016 09:38:19	3	128 MB	Grafo_Internet50
-rw-r--r--	hduser	supergroup	15.33 KB	18/05/2016 09:38:14	3	128 MB	Grafo_Twitter
-rw-r--r--	hduser	supergroup	2.97 KB	18/05/2016 09:38:17	3	128 MB	Grafo_Twitter200
drwxr-xr-x	hduser	supergroup	0 B	18/05/2016 10:47:12	0	0 B	excentricidade
drwxr-xr-x	hduser	supergroup	0 B	18/05/2016 10:40:47	0	0 B	parada
drwxr-xr-x	hduser	supergroup	0 B	18/05/2016 10:47:31	0	0 B	resultado

Directory: /logs/

[SecurityAuth-hduser.audit](#)

[hadoop-hduser-datanode-HadoopMaster.log](#)

[hadoop-hduser-datanode-HadoopMaster.out](#)

[hadoop-hduser-namenode-HadoopMaster.log](#)

[hadoop-hduser-namenode-HadoopMaster.out](#)

[hadoop-hduser-secondarynamenode-HadoopMaster.log](#)

[hadoop-hduser-secondarynamenode-HadoopMaster.out](#)

[userlogs/](#)

[yarn-hduser-nodemanager-HadoopMaster.log](#)

[yarn-hduser-nodemanager-HadoopMaster.out](#)

[yarn-hduser-resourcemanager-HadoopMaster.log](#)

[yarn-hduser-resourcemanager-HadoopMaster.out](#)

0 bytes May 18, 2016 12:37:43 PM

295935 bytes May 18, 2016 3:29:20 PM

718 bytes May 18, 2016 12:37:48 PM

1199096 bytes May 18, 2016 3:37:20 PM

4960 bytes May 18, 2016 1:51:32 PM

26986 bytes May 18, 2016 2:38:57 PM

718 bytes May 18, 2016 12:37:54 PM

4096 bytes May 18, 2016 3:36:03 PM

557962 bytes May 18, 2016 1:47:39 PM

702 bytes May 18, 2016 12:38:02 PM

1574389 bytes May 18, 2016 1:47:39 PM

2086 bytes May 18, 2016 12:41:34 PM

Interface gráfica para acesso ao HDFS

```
2016-05-18 12:37:43,379 INFO org.apache.hadoop.hdfs.server.namenode.NameNode: STARTUP_MSG:  
*****  
STARTUP_MSG: Starting NameNode  
STARTUP_MSG:   host = HadoopMaster/18.54.8.12  
STARTUP_MSG:   args = {}  
STARTUP_MSG:   version = 2.7.1  
STARTUP_MSG:   classpath = /usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/hadoop/common/lib/paranamer-2.3.jar:/usr/local/hadoop/share/hadoop/common/lib/mockito-all-1.8.5.jar:/usr/local/hadoop/share/hadoop/common/lib/httpcore-4.2.5.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-beanutils-1.7.0.jar:/usr/local/hadoop/share/hadoop/common/lib/servlet-api-2.5.jar:/usr/local/hadoop/share/hadoop/common/lib/httpclient-4.2.5.jar:/usr/local/hadoop/share/hadoop/common/lib/api-asni-api-1.8.0-M10.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-logging-1.1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-configuration-1.6.jar:/usr/local/hadoop/share/hadoop/common/lib/stax-api-1.0-2.jar:/usr/local/hadoop/share/hadoop/common/lib/jettison-1.1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/jackson-mapper-asl-1.9.19.jar:/usr/local/hadoop/share/hadoop/common/lib/jets3t-0.9.0.jar:/usr/local/hadoop/share/hadoop/common/lib/gson-2.2.4.jar:/usr/local/hadoop/share/hadoop/common/lib/jetty-util-6.1.26.jar:/usr/local/hadoop/share/hadoop/common/lib/api-util-1.0.0-M28.jar:/usr/local/hadoop/share/hadoop/common/lib/jersey-json-1.9.jar:/usr/local/hadoop/share/hadoop/common/lib/protobuf-java-2.5.0.jar:/usr/local/hadoop/share/hadoop/common/lib/jsp-api-2.1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar:/usr/local/hadoop/share/hadoop/common/lib/zookeeper-3.4.6.jar:/usr/local/hadoop/share/hadoop/common/lib/snappy-java-1.0.4.1.jar:/usr/local/hadoop/share/hadoop/common/lib/hadoop-auth-1.7.1.jar:/usr/local/hadoop/share/hadoop/common/lib/avro-1.7.4.jar:/usr/local/hadoop/share/hadoop/common/lib/apacheds-kerberos-codec-2.0.0-M15.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-digester-1.8.jar:/usr/local/hadoop/share/hadoop/common/lib/jaxb-api-2.2.2.jar:/usr/local/hadoop/share/hadoop/common/lib/jersey-server-1.9.jar:/usr/local/hadoop/share/hadoop/common/lib/xmlenc-0.52.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-cl-1.2.jar:/usr/local/hadoop/share/hadoop/common/lib/guava-11.0.2.jar:/usr/local/hadoop/share/hadoop/common/lib/junit-4.11.jar:/usr/local/hadoop/share/hadoop/common/lib/slf4j-api-1.7.10.jar:/usr/local/hadoop/share/hadoop/common/lib/apacheds-l18n-2.0.0-M15.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-lang-2.6.jar:/usr/local/hadoop/share/hadoop/common/lib/jsr305-3.0.0.jar:/usr/local/hadoop/share/hadoop/common/lib/jaxb-impl-2.2.3-1.jar:/usr/local/hadoop/share/hadoop/common/lib/jackson-jaxrs-1.9.13.jar:/usr/local/hadoop/share/hadoop/common/lib/curator-recipes-2.7.1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-compress-1.4.1.jar:/usr/local/hadoop/share/hadoop/common/lib/htrace-core-3.1.0-incubating.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-collections-3.2.1.jar:/usr/local/hadoop/share/hadoop/common/lib/jersey-core-1.9.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-net-3.1.jar:/usr/local/hadoop/share/hadoop/common/lib/hmcrest-core-1.3.jar:/usr/local/hadoop/share/hadoop/common/lib/jscn-0.1.42.jar:/usr/local/hadoop/share/hadoop/common/lib/netty-3.6.2.Final.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-beanutils-core-1.8.8.jar:/usr/local/hadoop/share/hadoop/common/lib/curator-framework-2.7.1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-math3-3.1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/activation-1.1.jar:/usr/local/hadoop/share/hadoop/common/lib/java-xmllibuilder-0.4.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-io-2.4.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-codec-1.4.jar:/usr/local/hadoop/share/hadoop/common/lib/curator-client-2.7.1.jar:/usr/local/hadoop/share/hadoop/common/lib/xz-1.0.jar:/usr/local/hadoop/share/hadoop/common/lib/log4j-1.2.17.jar:/usr/local/hadoop/share/hadoop/common/lib/jetty-6.1.26.jar:/usr/local/hadoop/share/hadoop/common/lib/asm-3.2.jar:/usr/local/hadoop/share/hadoop/common/lib/jackson-asl-1.9.19.jar:/usr/local/hadoop/share/hadoop/common/lib/jackson-xc-1.9.13.jar:/usr/local/hadoop/share/hadoop/common/lib/hadoop-annotations-2.7.1.jar:/usr/local/hadoop/share/hadoop/common/lib/commons-httpclient-3.1.jar:/usr/local/hadoop/share/hadoop/common/hadoop-common-2.7.1.jar:/usr/local/hadoop/share/hadoop/nfs-2.7.1.jar:/usr/local/hadoop/share/hadoop/common/hadoop-common-2.7.1-tests.jar:/usr/local/hadoop/share/hadoop/hdfs:/usr/local/hadoop/share/hadoop/hdfs/lib/xercesImpl-2.9.1.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/jetty-util-6.1.26.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/protobuf-java-2.5.0.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/commons-daemon-1.0.13.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/commons-dao-1.9.13.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/jersey-server-1.9.13.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/guava-11.0.2.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/xmlenc-0.52.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/commons-cl-1.2.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/guava-11.0.2.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/commons-lang-2.6.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/jsr305-3.0.0.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/htrace-core-3.1.0-incubating.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/jersey-core-1.9.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/xml-apis-1.3.0-04.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/netty-3.6.2.Final.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/commons-io-2.4.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/commons-codec-1.4.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/log4j-1.2.17.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/jetty-6.1.26.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/leveldbjni-all-1.8.jar:/usr/local/hadoop/share/hadoop/hdfs/lib/netty-all-4.0.23.Final.jar:/usr/local/hadoop/share/hadoop/hdfs/hadoop-hdfs-2.7.1.jar:/usr/local/hadoop/share/hadoop/hdfs/hadoop-hdfs-2.7.1-tests.jar:/usr/local/hadoop/share/hadoop/hdfs/hadoop-hdfs-nfs-2.7.1.jar:/usr/local/hadoop/share/hadoop/yarn/lib/awalis-1.0.jar:/usr/local/hadoop/share/hadoop/yarn/lib/guice-3.0.jar:/usr/local/hadoop/share/hadoop/yarn/lib/jettison-1.1.jar:/usr/local/hadoop/share/hadoop/yarn/lib/commons-logging-1.1.1.jar:/usr/local/hadoop/share/hadoop/yarn/lib/stax-api-1.0-2.jar:/usr/local/hadoop/share/hadoop/yarn/lib/jettison-1.1.jar:/usr/local/hadoop/share/hadoop/yarn/lib/jackson-mapper-asl-1.9.19.jar:/usr/local/hadoop/share/hadoop/yarn/lib/jetty-util-6.1.26.jar:/usr/local/hadoop/share/hadoop/yarn/lib/jersey-json-1.9.jar:/usr/local/hadoop/share/hadoop/yarn/lib/protobuf-java-2.5.0.jar:/usr/local/hadoop/share/hadoop/yarn/lib/guice-servlet-3.0.jar:/usr/local/hadoop/share/hadoop/yarn/lib/zookeeper-3.4.6.jar:/usr/local/hadoop/share/hadoop/yarn/lib/jersey-client-1.9.jar:/usr/local/hadoop/share/hadoop/yarn/lib/jaxb-api-2.2.2.jar:/usr/local/hadoop/share/hadoop/yarn/lib/commons-lang-1.9.jar:/usr/local/hadoop/share/hadoop/yarn/lib/commons-cli-1.2.jar:/usr/local/hadoop/share/hadoop/yarn/lib/guava-11.0.2.jar:/usr/local/hadoop/share/hadoop/yarn/lib/commons-lang-2.6.jar:/usr/local/hadoop/share/hadoop/yarn/lib/jsr305-3.0.0.jar:/usr/local/hadoop/share/hadoop/yarn/lib/jaxb-impl-2.2.3-1.jar:/usr/local/hadoop/share/hadoop/yarn/lib/jackson-jaxrs-1.9.13.jar:/usr/local/hadoop/share/hadoop/yarn/lib/javax.inject-1.jar:/usr/local/hadoop/share/hadoop/yarn/lib/commons-compress-1.4.1.jar:/usr/local/hadoop/share/hadoop/yarn/lib/commons-collections-
```

Interface gráfica para acesso ao HDFS



- Os Jobs podem ser acompanhados por outra interface.
- Acessada pela porta 8088.
- <http://localhost:8088>

Interface gráfica para acesso ao HDFS



Logged in as: dr.who

All Applications

Cluster Metrics	Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
26	0	0	26	0	0 B	64 GB	0 B	0	64	0	8	0	0	0	0	0
Scheduler Metrics	Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation												
Capacity Scheduler [MEMORY] <memory 1024 vCores 1> <memory 8192 vCores 8>																
Show 20 entries	Search															
application_1463575081879_0026	hduser	Resultado	MAPREDUCE	default	Wed May 18 10:47:15 -0300 2016	Wed May 18 10:47:31 -0300 2016	FINISHED	SUCCEEDED	History							
application_1463575081879_0025	hduser	Excentricidade	MAPREDUCE	default	Wed May 18 10:44:23 -0300 2016	Wed May 18 10:47:12 -0300 2016	FINISHED	SUCCEEDED	History							
application_1463575081879_0024	hduser	AUX	MAPREDUCE	default	Wed May 18 10:40:49 -0300 2016	Wed May 18 10:44:21 -0300 2016	FINISHED	SUCCEEDED	History							
application_1463575081879_0023	hduser	HEDA	MAPREDUCE	default	Wed May 18 10:38:04 -0300 2016	Wed May 18 10:40:48 -0300 2016	FINISHED	SUCCEEDED	History							
application_1463575081879_0022	hduser	HEDA	MAPREDUCE	default	Wed May 18 10:34:29 -0300 2016	Wed May 18 10:38:02 -0300 2016	FINISHED	SUCCEEDED	History							
application_1463575081879_0021	hduser	HEDA	MAPREDUCE	default	Wed May 18 10:31:47 -0300 2016	Wed May 18 10:34:27 -0300 2016	FINISHED	SUCCEEDED	History							
application_1463575081879_0020	hduser	HEDA	MAPREDUCE	default	Wed May 18 10:29:05 -0300 2016	Wed May 18 10:31:45 -0300 2016	FINISHED	SUCCEEDED	History							

Interface gráfica para acesso ao HDFS



Logged in as: dr.who



Nodes of the cluster

Cluster	
About Nodes	
Node Labels	
Applications	
NEW NEW_SAVING SUBMITTED ACCEPTED RUNNING FINISHED FAILED KILLED	
Scheduler	

Cluster Metrics																
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	
26	0	0	26	0	0 B	64 GB	0 B	0	64	0	8	0	0	0	0	0
Scheduler Metrics																
Scheduler Type				Scheduling Resource Type				Minimum Allocation				Maximum Allocation				
Capacity Scheduler				[MEMORY]				<memory:1024, vCores:1>				<memory:8192, vCores:8>				
Show 20 ▾ entries																
Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used	VCores Avail	Decommissioned	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Version
/default-rack	RUNNING	metrica1-7-50706	metrica1-7-8042	metrica1-7-8042	Wed May 18 15:44:03 +0000 2016		0	0 B	8 GB	0	8	0	0	0	0	2.7.1
/default-rack	RUNNING	metrica1-8-60808	metrica1-8-8042	metrica1-8-8042	Wed May 18 15:44:03 +0000 2016		0	0 B	8 GB	0	8	0	0	0	0	2.7.1
/default-rack	RUNNING	HadoopMaster-59952	HadoopMaster-8042	HadoopMaster-8042	Wed May 18 15:44:03 +0000 2016		0	0 B	8 GB	0	8	0	0	0	0	2.7.1
/default-rack	RUNNING	metrica1-5-36177	metrica1-5-8042	metrica1-5-8042	Wed May 18 15:44:03 +0000 2016		0	0 B	8 GB	0	8	0	0	0	0	2.7.1
/default-rack	RUNNING	metrica1-6-39401	metrica1-6-8042	metrica1-6-8042	Wed May 18 15:44:03 +0000 2016		0	0 B	8 GB	0	8	0	0	0	0	2.7.1
/default-rack	RUNNING	metrica1-2-33592	metrica1-2-8042	metrica1-2-8042	Wed May 18 15:44:03 +0000 2016		0	0 B	8 GB	0	8	0	0	0	0	2.7.1
/default-rack	RUNNING	metrica1-3-42671	metrica1-3-8042	metrica1-3-8042	Wed May 18 15:44:03 +0000 2016		0	0 B	8 GB	0	8	0	0	0	0	2.7.1
/default-rack	RUNNING	metrica1-4-33093	metrica1-4-8042	metrica1-4-8042	Wed May 18 15:44:03 +0000 2016		0	0 B	8 GB	0	8	0	0	0	0	2.7.1

Showing 1 to 8 of 8 entries

First Previous ▾ Next Last

Interface gráfica para acesso ao HDFS

- Além disso é possível consultar:
 - Configurações (XML)
 - Logs



Interface gráfica para acesso ao HDFS

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<><configuration>
  <><property>
    <><name>mapreduce.job.ubertask.enable</name>
    <><value>false</value>
    <><source>mapred-default.xml</source>
  </property>
  <><property>
    <><name>yarn.resourcemanager.max-completed-applications</name>
    <><value>10000</value>
    <><source>yarn-default.xml</source>
  </property>
  <><property>
    <><name>yarn.resourcemanager.delayed.delegation-token.removal-interval-ms
    </name>
    <><value>30000</value>
    <><source>yarn-default.xml</source>
  </property>
  <><property>
    <><name>io.bytes.per.checksum</name>
    <><value>512</value>
    <><source>core-default.xml</source>
  </property>
  <><property>
    <><name>yarn.timeline-service.leveldb-timeline-store.read-cache-size
    </name>
    <><value>104857600</value>
    <><source>yarn-default.xml</source>
  </property>
  <><property>
    <><name>fs.s3a.connection.timeout</name>
    <><value>50000</value>
    <><source>core-default.xml</source>
  </property>
  <><property>
    <><name>mapreduce.client.submit.file.replication</name>
    <><value>10</value>
    <><source>mapred-default.xml</source>
  </property>
  <><property>
```

Conclusão

- Interfaces Gráficas.
- HDFS.
- Jobs.
- Status do cluster.*



INSTITUTO DE GESTÃO EM
TECNOLOGIA DA INFORMAÇÃO

Aula 3.6. Apache Sqoop

Nesta aula

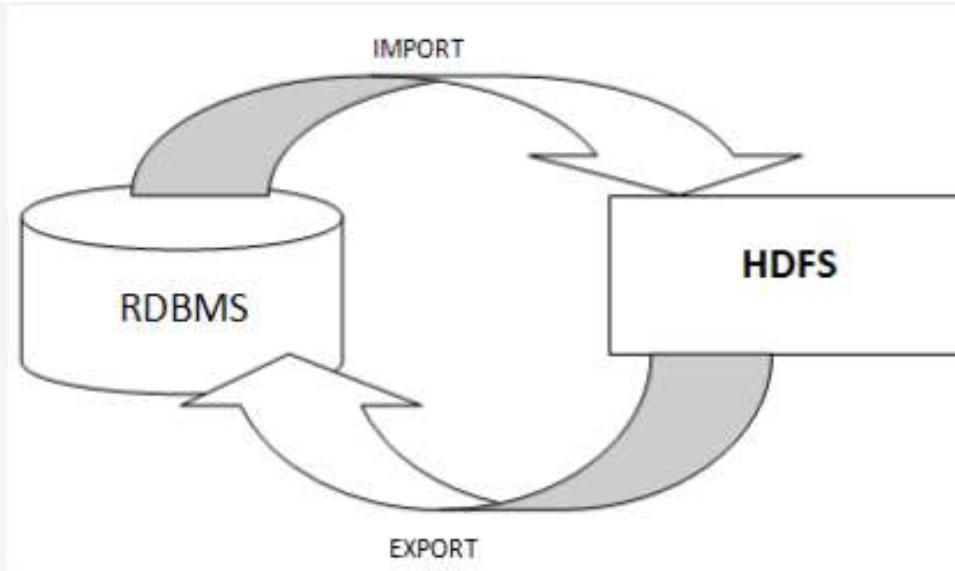
- ❑ Apresentação do Apache Sqoop.
- ❑ Características e funcionamento.

- SQL-to-Hadoop (abreviação):
 - Ferramenta de linha de comando.
 - Desenvolvida para transferir dados entre cluster Hadoop e banco de dados de armazenamento estruturado – relacionais (Oracle, DB2, MySQL, etc.).
 - Projeto iniciado em 2009 pela Cloudera.
 - Virou top-level na ASF (Apache Software Foundation) em 2012.



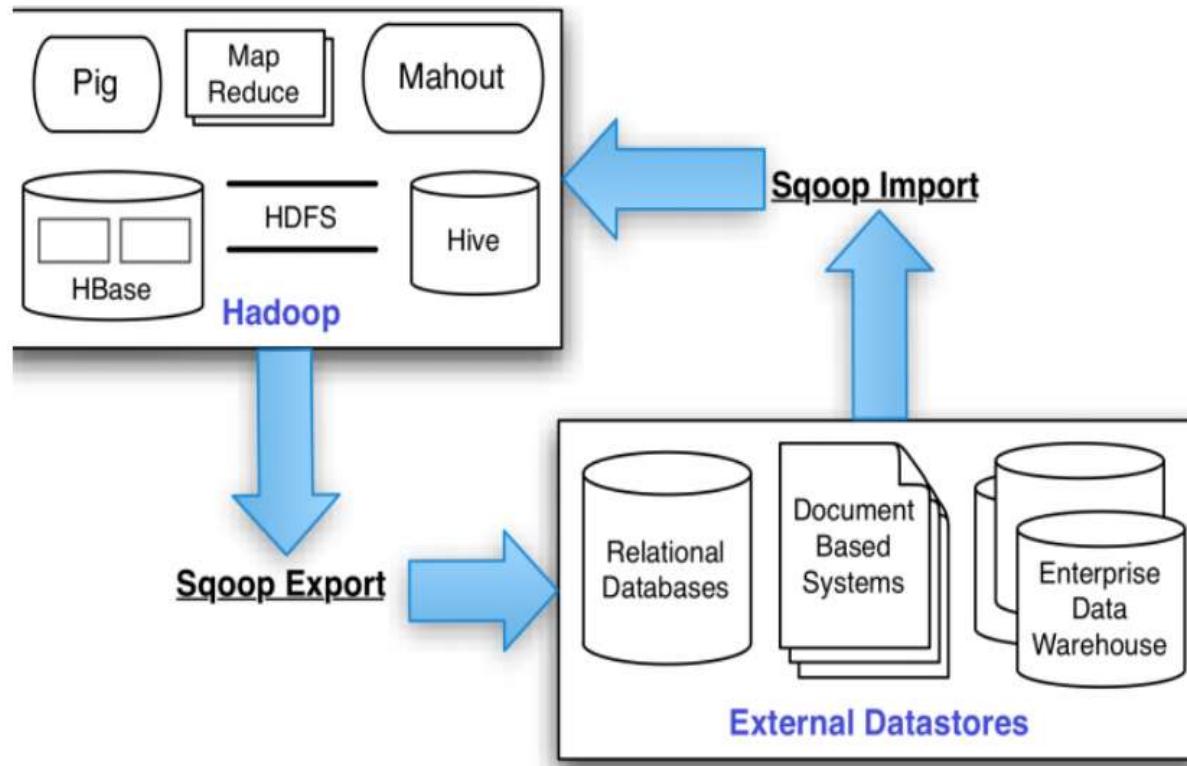
Apache Sqoop

IGTi



Apache Sqoop

IGTI



Apache Sqoop

IGTI



Apache Sqoop

- Ferramenta para ingestão de dados.
- Necessidade de mover/transferir grandes quantidades de dados.
- Transferir dados por script é ineficiente e consome muito tempo.
- Existe a possibilidade de transformar dados.

Apache Sqoop



- O Sqoop permite exportar e importar facilmente dados de armazenamentos estruturados.
- Possibilidade de transferir dados e armazenar no HDFS.
- Povoar tabelas Hive e HBase.
- Executa em cluster Hadoop.

- Realiza a conversão de tipos de campos.
- Aceita a conexão com diversos bancos, via JDBC.
- Utiliza o MapReduce nas atividades de importação e exportação, fornecendo processamento paralelo, distribuído e tolerante a falhas.
- Faz a leitura linha por linha da tabela ao escrever o arquivo no HDFS.
- MySQL, PostGreSQL, Oracle, SQL Server e DB2.
- Possui um conector JDBC genérico.

- Fornece modo de importação incremental.
- Pode ser usado para recuperar apenas novas linhas.
- Sqoop fornece dois tipos de importação incremental:
 - **Append:**
 - Usado para novas linhas.
 - Especifica-se uma coluna contendo um ID (check column).
 - O Sqoop importará linhas onde a check column tem um valor maior que o valor especificado em last value.

- **Last Modified:**

- Pode ser usado quando as linhas da tabela fonte precisam ser atualizados na tabela destino.
- Atualiza um timestamp.
- Colunas no destino que possuem timestamp mais antigo, que na tabela origem serão atualizadas.

Conclusão

- Apresentação do Sqoop.
- Ferramenta ETL.
- Movimentação de grandes quantidades de dados.



Módulo 3 - Solução de Dados utilizando Ecossistema Hadoop

Capítulo 4. Criando Programas Apache Hadoop

Prof. João Paulo Barbosa Nascimento



Aula 4.1. Estrutura de um programa Hadoop e a classe JobConf



- Estrutura do capítulo.
- Estruturando um programa no Apache Hadoop.
- Detalhes da classe JobConf.
- Exemplo de uso da classe JobConf.

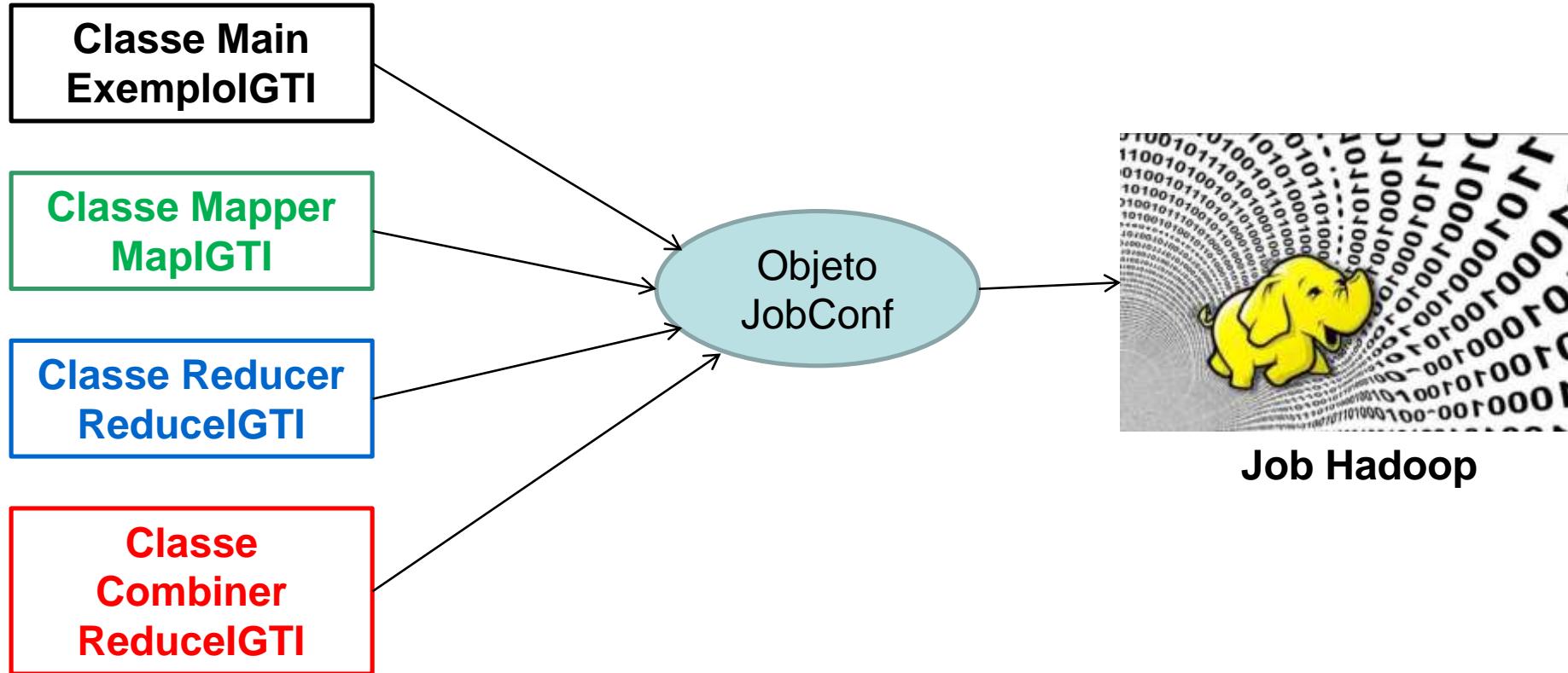
Estruturando um programa no Hadoop



- Estrutura de um programa Java comum.
- Método main().
- Uma classe para o Mapper.
- Uma classe para o Reducer/Combiner.
- Um objeto para unir tudo.

Estrutura básica de um programa Hadoop

IGTI



A classe JobConf

- A classe JobConf é a interface primária onde o usuário descreve um trabalho MapReduce (job).
- Para ser enviado ao framework Hadoop.
- Para que esse job possa ser realizado.
- O framework tenta executar o job fielmente ao que está definido em JobConf.

A classe JobConf

- Alguns parâmetros podem ser atribuídos utilizando a classe JobConf, outros não.
- Especifica o Mapper, Combiner (se houver), Reducer, etc.
- Pode ser usada para especificar outras características avançadas do Job.
- Visível dentro das funções Map e Reduce.

A classe JobConf

- Alguns métodos:
 - **setStrings**: enviar dados para as fases Map e Reduce.
 - **setNumTasks**: atribuir o número de tarefas Map para cada job.
 - **setNumReduceTasks**: atribuir o número de tarefas Reduce para cada job.
 - **setJobName**: atribuir o nome do job.

A classe JobConf

- Alguns métodos:
 - **setOutputKeyClass ([Class](#)<?> theClass)**: atribui a classe para a saída dos dados do *job* (dados da chave – Key).
 - **setOutputValueClass([Class](#)<?> theClass)**: atribui a classe para a saída dos dados do *job* (dados do valor - value).
 - **setMapperClass**: atribui uma classe *Mapper* para o *job* (minha classe).
 - **setReducerClass**: atribui uma classe *Reducer* para o *job* (minha classe).
 - **setCombinerClass**: atribui uma classe *Combiner* para o *job*.
 - Geralmente a mesma classe *Reducer*

A classe JobConf

- Alguns métodos:
 - setMaxMapAttempts(int n): atribui o número máximo de tentativas que serão feitas para executar uma tarefa Map.
 - setMaxReduceAttempts(int n): atribui o número máximo de tentativas que serão feitas para executar uma tarefa Reduce.

Exemplo de utilização da classe JobConf

```
1 package exemploigt;
2 import org.apache.hadoop.conf.*;
3 ...
4 public class ExemploIGTI extends Configured implements Tool {
5     public static void main (final String[] args) throws Exception {
6         int res = ToolRunner.run(new Configuration(), new ExemploIGTI, args);
7     }
8
9     public int run (final String[] args) throws Exception {
10        ...
11        JobConf conf = new JobConf(getConf(), ExemploIGTI.class);
12        conf.setJobName("Nome do Job");
13
14        conf.setOutputKeyClass(Text.class);
15        conf.setOutputValueClass(Text.class);
16
17        conf.setMapperClass(MapIGTI.class);
18        conf.setReducerClass(ReduceIGTI.class);
19        conf.setCombinerClass(ReduceIGTI.class);
20        ...
21    }
22}
```

Conclusão

- Estrutura básica de um programa Hadoop.
- Estrutura Java.
- Classe principal, Mapper, Reducer e Combiner.
- Objeto JobConf.

Próxima aula

- ❑ A classe MapReduceBase.
- ❑ Definições e utilidade.



Aula 4.2. A classe MapReduceBase



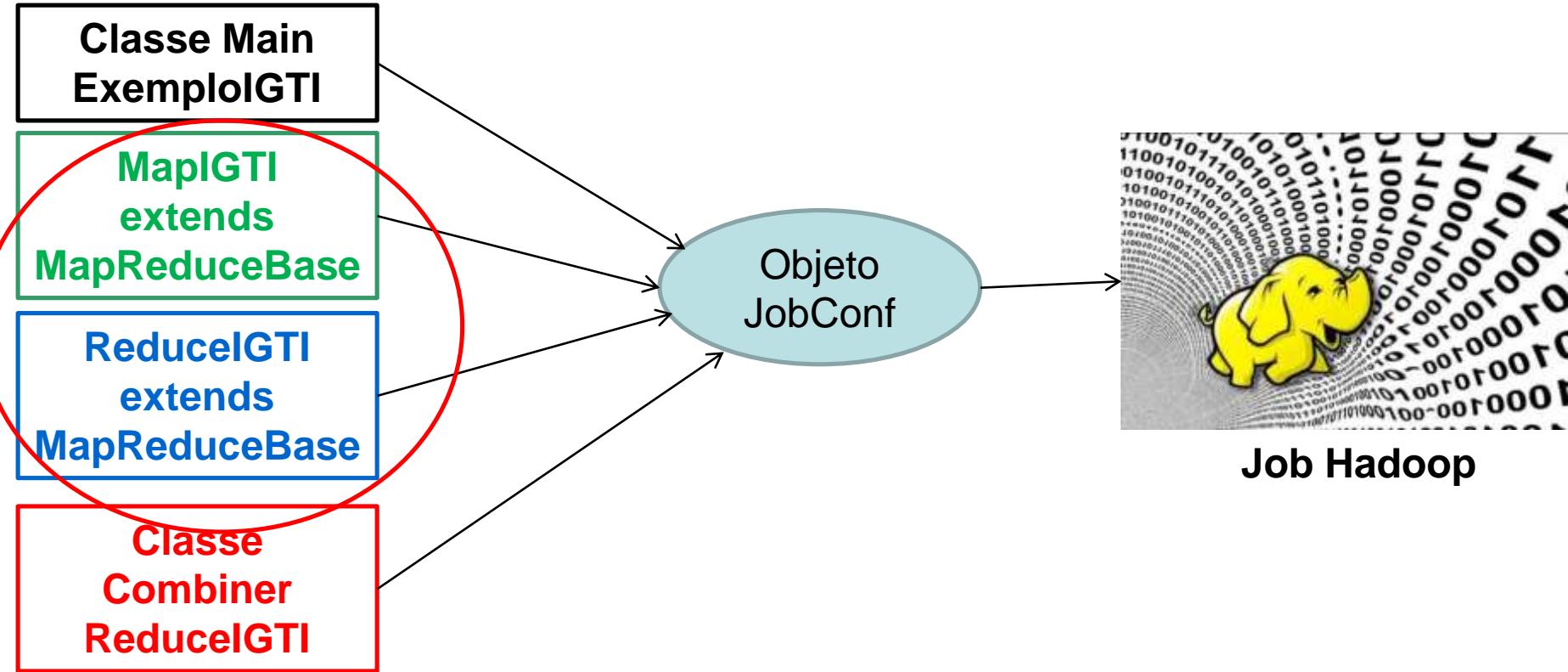
Nesta aula

- ❑ Descrição da classe MapReduceBase.
- ❑ Exemplo de uso.

A classe MapReduceBase

- É a classe base para implementação de Mappers e Reducers.
- Fica em org.apache.hadoop.mapred.MapReduceBase.
- Métodos principais:
 - Close.
 - Configure.

A classe MapReduceBase



Exemplo de utilização da classe MapReduceBase

```
1 package exemploigti;
2 import org.apache.hadoop.conf.*;
3 ...
4 public class ExemploIGTI extends Configured implements Tool {
5     public static void main (final String[] args) throws Exception {
6         int res = ToolRunner.run(new Configuration(), new ExemploIGTI, args);
7     }
8
9     public int run (final String[] args) throws Exception {
10        ...
11        JobConf conf = new JobConf(getConf(), ExemploIGTI.class);
12        conf.setJobName("Nome do Job");
13
14        conf.setOutputKeyClass(Text.class);
15        conf.setOutputValueClass(Text.class);
16
17        conf.setMapperClass(MapIGTI.class);
18        conf.setReducerClass(ReduceIGTI.class);
19        conf.setCombinerClass(ReduceIGTI.class);
20        ...
21    }
22}
```

Exemplo de utilização da classe MapReduceBase - Mapper

```
27 ...
28 public static class MapIGTI extends MapReduceBase implements Mapper<LongWritable, Text, Text, Text> {
29     public void configure(JobConf job) {
30         ... //sua implementação
31     }
32
33     public void map(LongWritable key, Text value, OutputCollector<Text, Text> output, Reporter reporter)
34     throws IOException {
35         ... //sua implementação
36     }
37
38     public void close() {
39         ... //sua implementação
40     }
41 }
```



A classe MapReduceBase

- Mapper possui o método Map.
- Mapeia o key/value de entrada em key/value intermediário.
- Um key/value pode gerar mais de um key/value intermediário
 - Key: a chave de entrada;
 - Value: o valor de entrada;
 - Output: coleta as chaves e valores mapeados;
 - Reporter: facilita a tarefa de reportar o progresso do job (logs).

Exemplo de utilização da classe MapReduceBase - Reducer

```
44 public static class ReduceIGTI extends MapReduceBase implements Reducer<Text, Text, Text, Text>
45     public void configure (JobConf job) {
46         ... //sua implementação
47     }
48
49     public void reduce (Text key, Iterator<Text> values, OutputCollector<Text, Text> output,
50     Reporter reporter) throws IOException {
51         ... //sua implementação
52     }
53
54     public void close(){
55         ... //sua implementação
56     }
57 }
```

A classe MapReduceBase

- Interface Reducer possui o método Reduce.
- Reduz um conjunto de valores intermediários que possuem uma chave e um conjunto de valores.
- Fases primárias:
 - Shuffle:
 - Sort
- Parâmetros:
 - Key: a chave;
 - Values: a lista de valores intermediários para reduzir;
 - Output e Reporter.

Conclusão

✓ Classe MapReduceBase.

✓ Interfaces:

- Mapper;
- Reducer.

Próxima aula

- ❑ Principais comandos de manipulação de diretórios no HDFS via aplicação.



Aula 4.3. Criando, consultando e excluindo diretórios no HDFS (via aplicação)



Nesta aula

- ❑ Criando, consultando e excluindo diretórios no HDFS (via aplicação).

Criando e excluindo diretórios no HDFS



- O Hadoop permite manipular o HDFS via aplicação.
- Para isso utilizamos a classe *FileSystem*.
- Permite manipular sistemas de arquivos distribuídos ou local.
- Possui dezenas de métodos para manipulação do HDFS.
- Para saber mais (documentação da classe):
<https://hadoop.apache.org/docs/r2.7.1/api/index.html?org/apache/hadoop/fs/FileSystem.html>

Criando e excluindo diretórios no HDFS

- FileSystem.
- Fornece diversos métodos para manipulação do HDFS:
 - exists(Path f): verifica se um determinado caminho existe.
 - getHomeDirectory(): retorna o diretório raiz (home) no HDFS.
 - isDirectory (Path p): verifica se o caminho p é um diretório.
 - isFile(Path f): verifica se p é um arquivo.
 - mkdirs(Path p): permite a criação de um diretório.
 - delete(Path p): permite a exclusão de um diretório.

Criando e excluindo diretórios no HDFS



```
package IGTI;

import java.io.*;
import java.util.*;
import java.util.Random;
import java.text.*;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;


public class ComandosHDFS extends Configured implements Tool {

    public static void main (final String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new ComandosHDFS(), args);
        System.exit(res);
    }

    public int run (final String[] args) throws Exception {
        try{
            JobConf conf = new JobConf(getConf(), ComandosHDFS.class);
            conf.setJobName("Olá Mundo");

            final FileSystem fs = FileSystem.get(conf);

            Path p = new Path("IGTI"); /* caminho no HDFS: /user/hduser/IGTI */

            if (fs.exists(p)) { /* verifica se o caminho p existe */
                System.out.println("Diretório IGTI encontrado");
                fs.delete(p); /* como o diretório já existe, vamos exclui-lo */
            }
            else {
                System.out.println("Diretório inexistente");
                fs.mkdirs(p); /* como o diretório não existe, vamos criá-lo */
            }
        } catch ( Exception e ) {
            throw e;
        }
        return 0;
    }
}
```

Criando e excluindo diretórios no HDFS



- O resultados dos métodos podem ser consultados na ferramenta de gerenciamento do Hadoop.
- <http://localhost:9870>
- Ou via linha de comando.

Criando e excluindo diretórios no HDFS



Hadoop Overview Datanodes Snapshot Startup Progress Utilities

Browse Directory

/user/hduser

Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	hduser	supergroup	0 B	19/03/2017 22:52:00	0	0 B	IGTI

Hadoop, 2016.

Conclusão

- Criando, consultando e excluindo diretórios no HDFS.
- Via aplicação.
- Importância para aplicações dinâmicas.

- ❑ Lendo e gravando dados no HDFS via aplicação.
- ❑ Atribuindo os diretórios padrão de entrada e saída de um job MapReduce.



Aula 4.4. Lendo e gravando dados no HDFS (via aplicação)



Nesta aula

- ❑ Lendo e gravando dados no HDFS via aplicação.
- ❑ Atribuindo os diretórios padrão de entrada e saída de um job MapReduce.

Lendo e gravando dados no HDFS (via aplicação)



- Iteração com o HDFS é necessária durante a execução do job.
- Ler e gravar arquivos manipular diretórios.
- Classe FileSystem fornece vários métodos para manipulações de arquivos.
- FileInputFormat e FileOutputFormat: classes utilizadas para manipular arquivos atribuídos às funções Map e Reduce.

Lendo e gravando dados no HDFS (via aplicação)

- **FileInputFormat:**
 - **static setInputPaths(JobConf conf, Path inputPaths):** atribui caminhos como entrada para o job MapReduce
 - **static setInputPaths(JobConf conf, String commaSeparatedPaths):** lista de entradas para o job MapReduce
 - **static getInputPaths(JobConf conf):** retorna uma lista de Paths para o job MapReduce

Lendo e gravando dados no HDFS (via aplicação)



- **FileOutputFormat:**
 - **static setOutputPath(JobConf conf, Path outputDir):** atribui um caminho para o diretório de saída do *job* MapReduce.
 - Diretório onde os dados serão gravados.
 - **setCompressOutput(JobConf conf, boolean compress):** define se os dados de saída do *job* serão compactados.

Lendo e gravando dados no HDFS (via aplicação)



```
import java.io.*;
import java.util.*;
import java.util.Random;
import java.text.*;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class ExemploIGTI extends Configured implements Tool {
    public static void main (final String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new ExemploIGTI(), args);
        System.exit(res);
    }

    public int run (final String[] args) throws Exception {
        try{
            JobConf conf = new JobConf(getConf(), ExemploIGTI.class);
            conf.setJobName("Ola Mundo");

            String caminhoEntrada = "entrada";
            FileInputFormat.setInputPaths(conf, caminhoEntrada);
            FileOutputFormat.setOutputPath(conf, new Path("saída"));

            conf.setOutputKeyClass(Text.class);
            conf.setOutputValueClass(Text.class);
            conf.setMapperClass(MapIGTI.class);
            conf.setReducerClass(ReduceIGTI.class);
            JobClient.runJob(conf);
        }
        catch ( Exception e ) {
            throw e;
        }
        return 0;
    }
}
```

Conclusão

- Atribuindo entradas e saídas para o job.
- FileInputFormat.
- FileOutputFormat.

Próxima aula

- ❑ Um “Olá mundo” com Hadoop.
- ❑ Aplicar todos os conceitos vistos até agora.



Aula 4.5. Um “Olá mundo!” utilizando o Hadoop



- Um programa Hadoop utilizando todos os conceitos apresentados até aqui.

Um “Olá mundo!” utilizando o Hadoop

- Etapas de execução do programa:
 - Configurar um objeto JobConf;
 - Criar um diretório de entrada com a classe FileSystem;
 - Incluir um arquivo para processamento (do sistema de arquivos do Linux para o HDFS), com o método copyFromLocalFile;
 - Atribuir um diretório de entrada para o Job;
 - Atribuir um diretório de saída para o Job (output);
 - Atribuir as classes Mapper e Reducer;
 - Apresentar a implementação dos métodos Map e Reduce;
 - Executar o Job;
 - Apresentar e analisar os resultados.

Um “Olá mundo!” utilizando o Hadoop



- Etapas de execução do programa:
 - Configurar um objeto JobConf.

```
package IGTI;

import java.io.*;
import java.util.*;
import java.util.Random;
import java.text.*;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class ExemploIGTI extends Configured implements Tool
{
    public static void main (final String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new ExemploIGTI(), args);
        System.exit(res);
    }

    public int run (final String[] args) throws Exception {
        try{
            JobConf conf = new JobConf(getConf(), ExemploIGTI.class);
            conf.setJobName("Olá Mundo");
        }
    }
}
```

Um “Olá mundo!” utilizando o Hadoop

- Etapas de execução do programa:
 - Criar um diretório de entrada com a classe FileSystem.
 - Copiar um arquivo para dentro desse diretório.

```
public int run (final String[] args) throws Exception {
    try{
        JobConf conf = new JobConf(getConf(), ExemploIGTI.class);
        conf.setJobName("Olá Mundo");

        final FileSystem fs = FileSystem.get(conf);
        Path diretorioEntrada = new Path("Entrada"), diretorioSaida = new Path("Saida");

        /* Criar um diretório de entrada no HDFS */
        if (!fs.exists(diretorioEntrada))
            fs.mkdirs(diretorioEntrada);

        /* Adicionar um arquivo para ser processado */
        fs.copyFromLocalFile(new Path("/usr/local/hadoop/Dados/arquivoBigData.txt"), diretorioEntrada);
```

Um “Olá mundo!” utilizando o Hadoop



- Etapas de execução do programa:
 - Arquivo a ser processado:

```
fsdffsrrrewrejwrwerkjlkj3423423k4j23kl4j23k4ljk23l4j001 fsfdssdfsdfsdfs00020090 sfsdfsd fdsfa  
44j23lkj4k23lj4k23lj4k23lj4kl23j4k23lj4kl32j4kl2j4ui007 fsdfsda fsdfsde00010020 dsfsdfdsfsdf  
4i234j32oi4jio23j4io32j4i23o j4io23j4i23j4io23j4i23j4io234001 erwerewrewrew4300067800 dsfsdfsd  
4i3o2p4iop23i4o23p4io23p4io23pi4o23i4o2p3i423poi4o23pi423p009 fsfsdfsdfsdfdc00004343 d  
o4pi23po4i23op4ik23o4ik23ç4k23l4kl23çk4lç32k4ç23lk4lç23k4l001 asdfsdfsdfdsfc00004578  
h4jl32h4j23h4jkh234jkh234kjh32jk4h32kjh423jk4h2j3k4h32jh44009 fsfsdfsdfsdfdc00003333  
432423423432k4jk32lj4kl23j4kl2j34lkj234lkj234kl32j4kl23j44009 fsfsdfsdfsdfds00033210  
jk32j423k14j23lk4jk23lj432kj4kl23j4kl32j4k3l2j43lk2jj23kl007 fsdafsdfsdfdc00034567  
23432423432423423k4j23klj423lk4jk2l34432ftf4ty32ft32t4f34009 fsfsdfsdfsdfdc00043321  
86887686786kjkjlwejrkjlwejrklwj32432432432423432423001 fsdfsafsfss00034567 sdfsdfsd  
423i4ui23ou4i32u4io32u4io3u4io3u4iou4io23u4io32u4io32u4oi23u4ic001 fsfsdfsdfsdfdc0000234  
k4k34j23lkj4k32lj4kl23j4kl23j4kl32j4kl32j4kl32j4kl32001 fsfsdfsdfsdfds00003259  
j432j4kl32j4kl32j4kl23j432kj432lkj432lkj4k23l4j32lkj4k32l4007 fsdfsda fsr00092345  
l4j32lkj432kl4j3k2l4j23kl4j3k2l4j32lkj4kl234j3kl2j44004 vvcbcvbcvbcvbcv00000211  
j4ç3k2k423çl4kl23k4l32k4lç23k4lç23k4l32çk4lç32k4l3ç2k4344008 vvcbcvbcvbcvvv00002235  
k43k4lç32k4l23çk4lç23k4lç23k42ioriewopri weopri weopirw004 jfdgfdgdfgfdgfc00002113  
4jop34k23o4io234io32p4i32po4i32po4i3poi432poi432poi432po4004 jdfgdfgfdgfdgff00022119  
4ç324324k23çl4k3l2k4lç2k4lç32k4lç32k4lç32k4lç32k4lç4k1004 jfdgrtqqqqqqqqqc00002135  
4kç23k4lç23k4lç32k4l4k23çlk423çlk4l23çlk432çlk44k333001 ewewqewqewqewv00000123  
43123210909109109209330123k2j32k1çl3k21çlk3l21k3l222007 popopopopo iyy00012999  
iuiuiuiuaaaaaaaa
```

Um “Olá mundo!” utilizando o Hadoop

- Etapas de execução do programa:
 - Atribuir diretórios de entrada e saída:

```
public int run (final String[] args) throws Exception {  
    try{  
        JobConf conf = new JobConf(getConf(), ExemploIGTI.class);  
        conf.setJobName("Olá Mundo");  
  
        final FileSystem fs = FileSystem.get(conf);  
        Path diretorioEntrada = new Path("Entrada"), diretorioSaida = new Path("Saída");  
  
        /* Criar um diretório de entrada no HDFS */  
        if (!fs.exists(diretorioEntrada))  
            fs.mkdirs(diretorioEntrada);  
  
        /* Adicionar um arquivo para ser processado */  
        fs.copyFromLocalFile(new Path("/usr/local/hadoop/Dados/arquivoBigData.txt"), diretorioEntrada);  
  
        /* Atribuindo os diretórios de Entrada e Saída para o Job */  
        FileInputFormat.setInputPaths(conf, diretorioEntrada);  
        FileOutputFormat.setOutputPath(conf, diretorioSaida);  
    }  
}
```



Um “Olá mundo!” utilizando o Hadoop



- Etapas de execução do programa:
 - Atribuir as classes Mapper e Reducer:

```
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(Text.class);
conf.setMapperClass(MapIGTI.class);
conf.setReducerClass(ReduceIGTI.class);
JobClient.runJob(conf);
```

Um “Olá mundo!” utilizando o Hadoop



- Etapas de execução do programa:
 - Implementação do método Map:

```
public static class MapIGTI extends MapReduceBase implements Mapper<LongWritable, Text, Text, Text> {  
    public void map(LongWritable key, Text value, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {  
        Text txtChave = new Text();  
        Text txtValor = new Text();  
  
        String codigoCliente = value.toString().substring(58, 61);  
        String qtdeItens = value.toString().substring(76, 84);  
  
        txtChave.set(codigoCliente);  
        txtValor.set(qtdeItens);  
  
        output.collect(txtChave, txtValor);  
    }  
}
```

Um “Olá mundo!” utilizando o Hadoop

- Etapas de execução do programa:
 - Implementação do método Reduce:

```
public static class ReduceIGTI extends MapReduceBase implements Reducer<Text, Text, Text, Text> {  
    public void reduce (Text key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {  
  
        double media = 0.0;  
        int acumuladorItens = 0, contaVendas = 0;  
        Text value = new Text();  
  
        while (values.hasNext()) {  
            value = values.next();  
            contaVendas++;  
            acumuladorItens += Integer.parseInt(value.toString());  
        }  
  
        media = acumuladorItens / new Double(contaVendas);  
        value.set(String.valueOf(media));  
        output.collect(key, value);  
    }  
}
```

Um “Olá mundo!” utilizando o Hadoop

- Etapas de execução do programa:
 - Execução do Job:

```
17/03/23 19:35:57 INFO mapreduce.JobSubmitter: number of splits:2
17/03/23 19:35:57 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1490307791670_0001
17/03/23 19:35:58 INFO impl.YarnClientImpl: Submitted application application_1490307791670_0001
17/03/23 19:35:58 INFO mapreduce.Job: The url to track the job: http://JOAOLINUX:8088/proxy/application_1490307791670_0001/
17/03/23 19:35:58 INFO mapreduce.Job: Running job: job_1490307791670_0001
17/03/23 19:36:06 INFO mapreduce.Job: Job job_1490307791670_0001 running in uber mode : false
17/03/23 19:36:06 INFO mapreduce.Job: map 0% reduce 0%
17/03/23 19:36:14 INFO mapreduce.Job: map 100% reduce 0%
17/03/23 19:36:30 INFO mapreduce.Job: map 100% reduce 100%
17/03/23 19:36:30 INFO mapreduce.Job: Job job_1490307791670_0001 completed successfully
17/03/23 19:36:30 INFO mapreduce.Job: Counters: 49
    File System Counters
        FILE: Number of bytes read=300
        FILE: Number of bytes written=355853
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=3310
        HDFS: Number of bytes written=71
        HDFS: Number of read operations=9
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
    Job Counters
        Launched map tasks=2
        Launched reduce tasks=1
        Data-local map tasks=2
        Total time spent by all maps in occupied slots (ms)=9793
        Total time spent by all reduces in occupied slots (ms)=12540
        Total time spent by all map tasks (ms)=9793
        Total time spent by all reduce tasks (ms)=12540
        Total vcore-milliseconds taken by all map tasks=9793
        Total vcore-milliseconds taken by all reduce tasks=12540
        Total megabyte-milliseconds taken by all map tasks=10028032
        Total megabyte-milliseconds taken by all reduce tasks=12840960
```

Um “Olá mundo!” utilizando o Hadoop

- Etapas de execução do programa:
 - Analisando os resultados (ferramentas de gerenciamento do Hadoop):

Screenshot of the Hadoop Cluster Metrics UI showing the "All Applications" page.

The UI includes a sidebar with cluster metrics and a scheduler metrics table. The main area displays application details for "Ola Mundo".

Cluster Metrics

	Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes
1	0	0	1	0	0 B	8 GB	0 B	0	8	0	0	1	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>

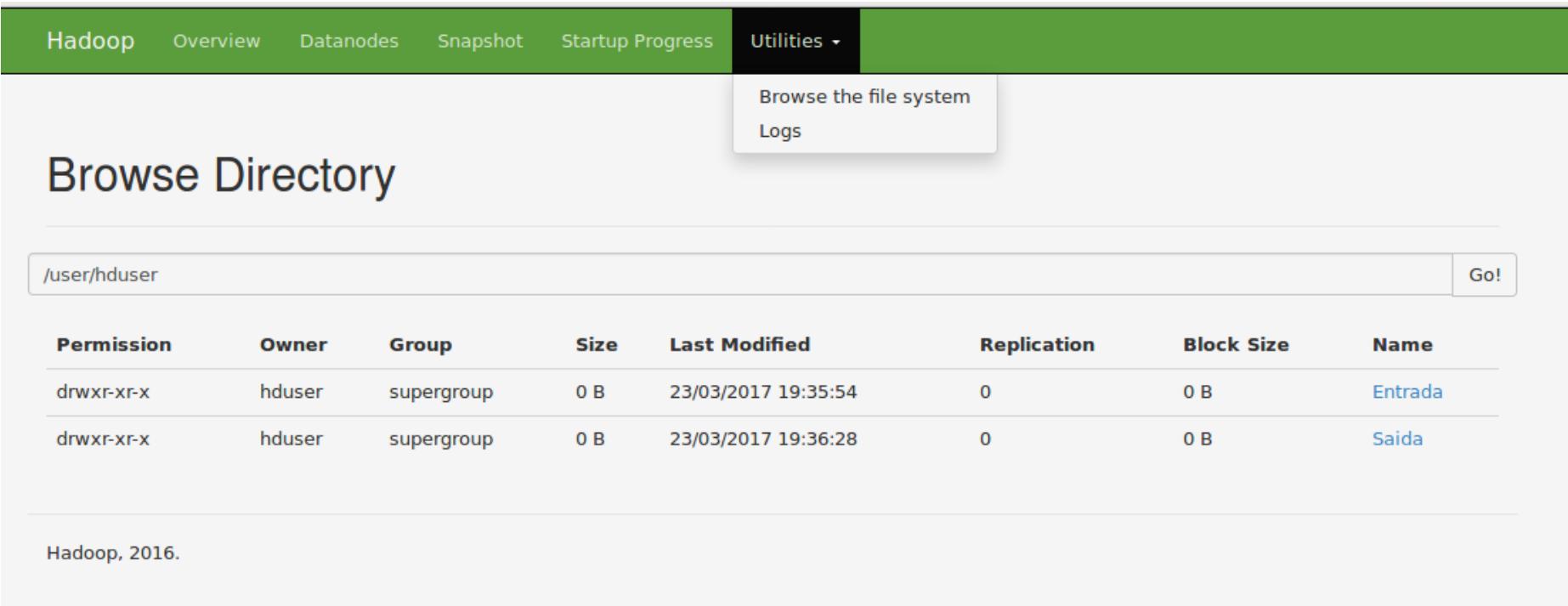
All Applications

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1490307791670_0001	hduser	Ola Mundo	MAPREDUCE	default	Thu Mar 23 19:35:58 -0300 2017	Thu Mar 23 19:36:28 -0300 2017	FINISHED	SUCCEEDED		History

Showing 1 to 1 of 1 entries

Um “Olá mundo!” utilizando o Hadoop

- Etapas de execução do programa:
 - Analisando os resultados (HDFS – localhost:9870):



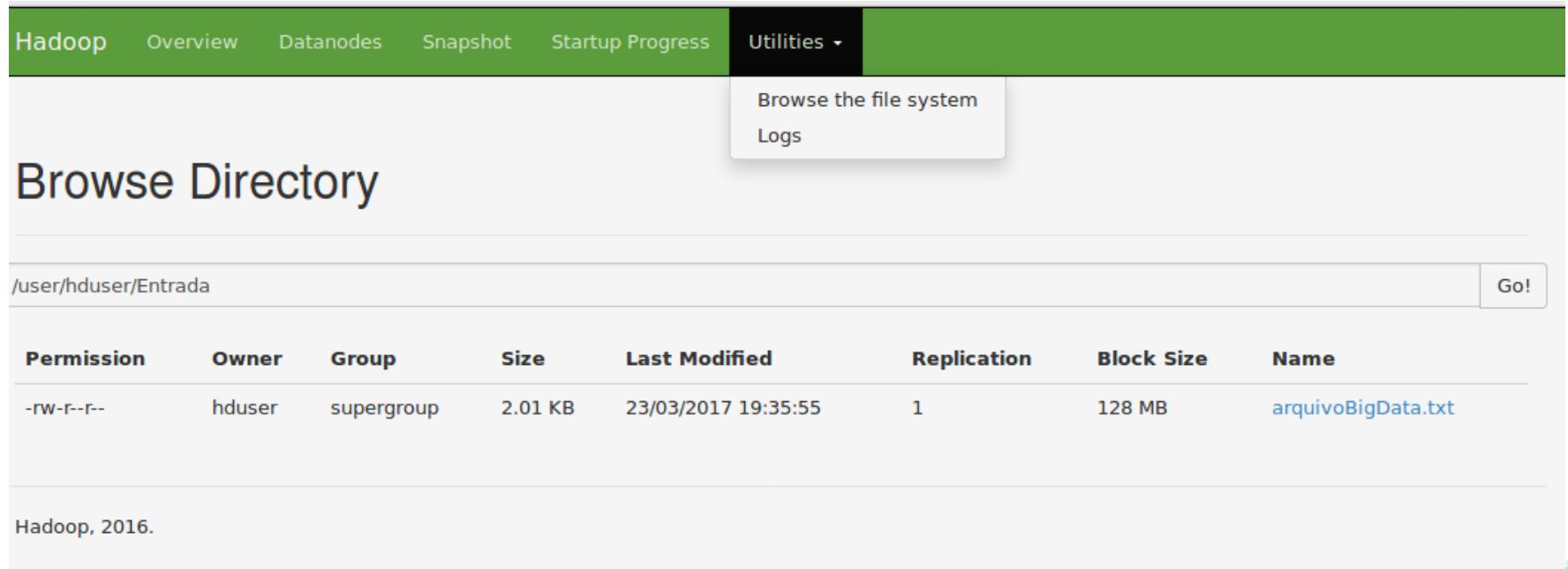
The screenshot shows the HDFS Web Interface. The top navigation bar includes links for Hadoop, Overview, Datanodes, Snapshot, Startup Progress, and Utilities (with a dropdown menu for Browse the file system and Logs). The main content area is titled "Browse Directory" and shows the path "/user/hduser". A "Go!" button is located at the end of the path input field. Below the path, a table lists two files:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	hduser	supergroup	0 B	23/03/2017 19:35:54	0	0 B	Entrada
drwxr-xr-x	hduser	supergroup	0 B	23/03/2017 19:36:28	0	0 B	Saida

At the bottom left, the text "Hadoop, 2016." is visible.

Um “Olá mundo!” utilizando o Hadoop

- Etapas de execução do programa:
 - Analisando os resultados (HDFS – Conteúdo do diretório entrada):



Hadoop Overview Datanodes Snapshot Startup Progress Utilities ▾

Browse the file system
Logs

Browse Directory

/user/hduser/Entrada Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	supergroup	2.01 KB	23/03/2017 19:35:55	1	128 MB	arquivoBigData.txt

Hadoop, 2016.

Um “Olá mundo!” utilizando o Hadoop



- Etapas de execução do programa:
 - Analisando os resultados (HDFS – Conteúdo do diretório saída):

Hadoop Overview Datanodes Snapshot Startup Progress Utilities ▾

Browse Directory

/user/hduser/Saida

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	supergroup	0 B	23/03/2017 19:36:28	1	128 MB	_SUCCESS
-rw-r--r--	hduser	supergroup	71 B	23/03/2017 19:36:28	1	128 MB	part-00000

Hadoop, 2016.

Um “Olá mundo!” utilizando o Hadoop

- Etapas de execução do programa:
 - Analisando os resultados (HDFS – Conteúdo do arquivo de saída):



A screenshot of a file browser window titled "Abrir". The window displays a list of files in a table format. The columns are labeled "001" and "18664.428". The table contains five rows of data:

001	18664.428
004	6644.5
007	37482.75
008	2235.0
009	21051.75

Conclusão

- Primeiro programa completo.
- Manipulação de arquivos no HDFS e processamento.

- ❑ Métodos configure e close (MapReduceBase).



Aula 4.6. Métodos configure e close



- Métodos MapReduceBase:
 - Configure.
 - Close.

Métodos configure e close

- Métodos presentes na classe MapReduceBase:
 - close() – Não possui parâmetros.
 - configure(JobConf job) – JobConf do job MapReduce.
- Na classe MapReduceBase esses métodos não possuem nenhum código.

Métodos configure e close

- Método configure(...):
 - É a única maneira de acessar o JobConf dentro de um Mapper ou Reducer.
 - É uma forma de realizar um setup.
 - É chamado pelo framework antes de iniciar os métodos Map ou Reduce.
 - Usado muitas vezes para:
 - Instanciar objetos.
 - Atribuir valores iniciais para variáveis.
 - Registrar log.

Métodos configure e close

- Pelo JobConf, que é passado como parâmetro, é possível enviar informações do método main() da classe principal para os métodos Map ou Reduce.
- SetStrings.

Métodos configure e close

```
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class ExemploIGTI extends Configured implements Tool
{
    public static void main (final String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new ExemploIGTI(), args);
        System.exit(res);
    }

    public int run (final String[] args) throws Exception {
        try{
            JobConf conf = new JobConf(getConf(), ExemploIGTI.class);
            conf.setJobName("Ola Mundo");
            conf.setStrings("codigoExecucao", Integer.toString(337)); ←
        }
    }
}
```

Métodos configure e close

```
public static class ReduceIGTI extends MapReduceBase implements Reducer<Text, Text, Text, Text> {  
  
    Text value;  
    public void configure(JobConf job) { ←  
        value = new Text();  
        System.out.println(job.getStrings("codigoExecucao")); ←  
    }  
  
    public void reduce (Text key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {  
        double media = 0.0;  
        int acumuladorItens = 0, contaVendas = 0;  
  
        while (values.hasNext()) {  
            value = values.next();  
            contaVendas++;  
            acumuladorItens += Integer.parseInt(value.toString());  
        }  
        media = acumuladorItens / new Double(contaVendas);  
        value.set(String.valueOf(media));  
        output.collect(key, value);  
    }  
}
```

Métodos configure e close

- Método close():
 - Não possui parâmetros.
 - Chamado pelo framework quando todas as entradas tiverem sido processadas pelos métodos Map ou Reduce.
 - Pode ser usado para arrematar o processamento.
 - Fechar algum arquivo suplementar que foi aberto.
 - Efetivar alterações.
 - Gravar algum dado ou log adicional/final.

Métodos configure e close

```
public static class ReduceIGTI extends MapReduceBase implements Reducer<Text, Text, Text> {  
  
    Text value;  
    public void configure(JobConf job) {  
        value = new Text();  
        System.out.println(job.getStrings("codigoExecucao"));  
    }  
  
    public void reduce (Text key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {  
        double media = 0.0;  
        int acumuladorItens = 0, contaVendas = 0;  
  
        while (values.hasNext()) {  
            value = values.next();  
            contaVendas++;  
            acumuladorItens += Integer.parseInt(value.toString());  
        }  
        media = acumuladorItens / new Double(contaVendas);  
        value.set(String.valueOf(media));  
        output.collect(key, value);  
    }  
  
    public void close() {  
        System.out.println("Processamento encerrado ");  
    }  
}
```

Conclusão

- Métodos configure e close.
- setString.
- Enviar dados da classe main() para os métodos Map e Reduce.

Próxima aula

- ❑ Definindo uma função combine.



Aula 4.7. Definindo uma função Combine



Nesta aula

- ❑ Função Combine.
- ❑ Utilidade e formas de uso.

Definindo uma função Combine

- Jobs MapReduce são limitadas pela largura de banda disponível no cluster.
- Para minimizar a transferência de dados entre tarefas Map e Reduce, existe a função Combine.
- Recebe como entrada a saída da função Map.
- A utilização da função Combine não deve alterar o resultado.

Definindo uma função Combine

- Combiner não implementa uma interface específica.
- Combiner implementa a interface Reducer e o método reduce().
- Alguns apelidam o Combiner de semi-reducer ou pré-reducer.
- É totalmente opcional.
- Pode apresentar comportamento diferenciado quando executado em cluster ou localmente.

Definindo uma função Combine

Node 1

(1950, 0)
(1950, 20)
(1950, 10)

Node 2

(1950, 25)
(1950, 15)

Entrada para Reduce

(1950, [0, 20, 10, 25, 15])

Saída de Reduce

(1950, 25)

Entrada para Reduce com Combine

(1950, [20, 25])

- Mais sucintamente podemos expressar a função Reduce na seguinte forma:

$$\max(0, 20, 10, 25, 15) = \max(\max(0, 20, 10), \max(25, 15)) = \max(20, 25) = 25$$

Definindo uma função Combine

- Cuidado! Nem todos os jobs podem utilizar a função Combine e produzir resultados corretos.
- Se ao invés do maior valor quiséssemos a média dos valores?

$$\text{mean}(0, 20, 10, 25, 15) = 14$$

- Não poderíamos usar a nossa função Combine, pois:

$$\text{mean}(\text{mean}(0, 20, 10), \text{mean}(25, 15)) = \text{mean}(10, 20) = 15$$

Definindo uma função Combine

- A função Combine não substitui a função Reduce, pois os dados são processados apenas localmente e não globalmente.
- Podem existir dados com a mesma chave em diferentes Datanodes.
- Combine pode auxiliar a diminuir a quantidade de dados mesclados entre diferentes Mappers e Reducers.
- `setCombinerClass`.

Definindo uma função Combine

```
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(Text.class);
conf.setMapperClass(MapIGTI.class);
conf.setCombinerClass(ReduceIGTI.class); ←
conf.setReducerClass(ReduceIGTI.class);
JobClient.runJob(conf);
```

Definindo uma função Combine

```
Map input records=20
Map output records=20
Map output bytes=260
Map output materialized bytes=108
Input split bytes=226
Combine input records=7
Combine output records=7
Spilled Records=7
Failed Shuffles=0
Merged Map outputs=0
GC time elapsed (ms)=379
CPU time spent (ms)=2180
Physical memory (bytes) snapshot=485355520
Virtual memory (bytes) snapshot=3825672192
Total committed heap usage (bytes)=391118848
File Input Format Counters
Bytes Read=3084
```



Combiners:

- Quando usar.
- Forma de usar.
- Benefícios.

Próxima aula

- ❑ Depurando um *job* e os *logs* do Hadoop.



Aula 4.8. Depurando um job e os logs do Hadoop



- ❑ Hadoop:
 - Depuração.
 - Logs.

Depurando um job e os logs do Hadoop



- A depuração de programas Hadoop pode ser feita de diversas maneiras:
 - Registro de logs;
 - Counters;
 - Plugins.

Depurando um job e os logs do Hadoop



- Registro de logs:
 - System.out.println.
 - Registro nos logs do Hadoop.

Depurando um job e os logs do Hadoop

```
public static class ReduceIGTI extends MapReduceBase implements Reducer<Text, Text, Text, Text> {  
    public void reduce (Text key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {  
        double media = 0.0;  
        int acumuladorItens = 0, contaVendas = 0;  
        Text value = new Text();  
  
        while (values.hasNext()) {  
            value = values.next();  
            contaVendas++;  
            acumuladorItens += Integer.parseInt(value.toString());  
        }  
        media = acumuladorItens / new Double(contaVendas);  
  
        if (media > 15000)  
            System.out.println("Chave " + key.toString() + " possui mais de 15000 itens ( " + String.valueOf(media) + " ) ");  
        value.set(String.valueOf(media));  
        output.collect(key, value);  
    }  
}
```



Depurando um job e os logs do Hadoop

IGTI

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities ▾

Startup Progress

Elapsed Time: 1 sec, Percent Complete: 100%

Phase	Completion	Elapsed Time
Loading fsimage /usr/local/hadoop/tmp/dfs/name/current/fsimage_00000000000000000000000000000000 353 B	100%	0 sec
inodes (0/0)	100%	
delegation tokens (0/0)	100%	
cache pools (0/0)	100%	
Loading edits	100%	0 sec
Saving checkpoint	100%	0 sec
Safe mode	100%	0 sec
awaiting reported blocks (0/0)	100%	

Logs

Hadoop, 2016.

Depurando um job e os logs do Hadoop

IGTI

Directory: /logs/

SecurityAuth-hduser.audit	0 bytes	19/03/2017 19:45:49
hadoop-hduser-datanode-JOAOLINUX.log	609813 bytes	26/03/2017 18:04:16
hadoop-hduser-datanode-JOAOLINUX.out	718 bytes	26/03/2017 16:46:39
hadoop-hduser-datanode-JOAOLINUX.out.1	718 bytes	23/03/2017 19:22:58
hadoop-hduser-datanode-JOAOLINUX.out.2	718 bytes	21/03/2017 23:50:03
hadoop-hduser-datanode-JOAOLINUX.out.3	718 bytes	19/03/2017 21:38:12
hadoop-hduser-datanode-JOAOLINUX.out.4	716 bytes	19/03/2017 21:23:11
hadoop-hduser-datanode-JOAOLINUX.out.5	716 bytes	19/03/2017 21:17:54
hadoop-hduser-namenode-JOAOLINUX.log	702092 bytes	26/03/2017 18:04:15
hadoop-hduser-namenode-JOAOLINUX.out	5001 bytes	26/03/2017 17:37:08
hadoop-hduser-namenode-JOAOLINUX.out.1	5001 bytes	23/03/2017 20:20:24
hadoop-hduser-namenode-JOAOLINUX.out.2	5006 bytes	21/03/2017 23:59:15
hadoop-hduser-namenode-JOAOLINUX.out.3	5006 bytes	19/03/2017 22:28:31
hadoop-hduser-namenode-JOAOLINUX.out.4	716 bytes	19/03/2017 21:23:07
hadoop-hduser-namenode-JOAOLINUX.out.5	716 bytes	19/03/2017 21:17:50
hadoop-hduser-secondarynamenode-JOAOLINUX.log	244436 bytes	26/03/2017 17:47:51
hadoop-hduser-secondarynamenode-JOAOLINUX.out	718 bytes	26/03/2017 16:46:45
hadoop-hduser-secondarynamenode-JOAOLINUX.out.1	718 bytes	23/03/2017 19:23:03
hadoop-hduser-secondarynamenode-JOAOLINUX.out.2	718 bytes	21/03/2017 23:50:09
hadoop-hduser-secondarynamenode-JOAOLINUX.out.3	718 bytes	19/03/2017 21:38:18
hadoop-hduser-secondarynamenode-JOAOLINUX.out.4	716 bytes	19/03/2017 21:23:20
hadoop-hduser-secondarynamenode-JOAOLINUX.out.5	716 bytes	19/03/2017 21:18:03
userlogs/	4096 bytes	26/03/2017 18:16:55
yarn-hduser-nodemanager-JOAOLINUX.log	1521399 bytes	26/03/2017 18:04:22
yarn-hduser-nodemanager-JOAOLINUX.out	1571 bytes	26/03/2017 16:46:59
yarn-hduser-nodemanager-JOAOLINUX.out.1	1571 bytes	23/03/2017 19:23:15
yarn-hduser-nodemanager-JOAOLINUX.out.2	1578 bytes	21/03/2017 23:50:25
yarn-hduser-nodemanager-JOAOLINUX.out.3	1571 bytes	19/03/2017 21:38:55
yarn-hduser-nodemanager-JOAOLINUX.out.4	1571 bytes	19/03/2017 21:23:46
yarn-hduser-nodemanager-JOAOLINUX.out.5	1571 bytes	19/03/2017 21:18:31
yarn-hduser-resourcemanager-JOAOLINUX.log	1461861 bytes	26/03/2017 18:04:20



Depurando um job e os logs do Hadoop

IGTI



All Applications

Cluster Metrics									
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Vcores Used	
5	0	0	5	0	0 B	8 GB	0 B	0	
Scheduler Metrics									
Scheduler Type: Capacity Scheduler Scheduling Resource Type: [MEMORY] <memory:1024, vCores:1>									
Show 20 entries									
ID	User	Name	Application Type	Queue	StartTime	FinishTime	Duration	Progress	State
application_1490557613925_0005	hduser	Ola Mundo	MAPREDUCE	default	Sun Mar 26 18:03:35 -0300 2017	Sun Mar 26 18:03:35 -0300 2017	00:00:00	100%	FINISHED
application_1490557613925_0004	hduser	Ola Mundo	MAPREDUCE	default	Sun Mar 26 17:55:42 -0300 2017	Sun Mar 26 17:55:42 -0300 2017	00:00:00	100%	FINISHED
application_1490557613925_0003	hduser	Ola Mundo	MAPREDUCE	default	Sun Mar 26 17:49:11 -0300 2017	Sun Mar 26 17:49:11 -0300 2017	00:00:00	100%	FINISHED
application_1490557613925_0002	hduser	Ola Mundo	MAPREDUCE	default	Sun Mar 26 17:42:29 -0300 2017	Sun Mar 26 17:42:29 -0300 2017	00:00:00	100%	FINISHED
application_1490557613925_0001	hduser	Ola Mundo	MAPREDUCE	default	Sun Mar 26 16:47:41 -0300 2017	Sun Mar 26 16:47:41 -0300 2017	00:00:00	100%	FINISHED

Showing 1 to 5 of 5 entries

Directory: /logs/userlogs/

Parent Directory

application_1489968623877_0001/	4096 bytes	19/03/2017 21:11:48
application_1489969102190_0001/	4096 bytes	19/03/2017 21:19:50
application_1489969416842_0001/	4096 bytes	19/03/2017 21:25:23
application_1489970317324_0001/	4096 bytes	19/03/2017 21:43:56
application_1490151019328_0001/	4096 bytes	22/03/2017 21:49:02
application_1490151019328_0002/	4096 bytes	22/03/2017 21:52:55
application_1490151019328_0003/	4096 bytes	22/03/2017 21:56:57
application_1490151019328_0004/	4096 bytes	22/03/2017 22:02:24
application_1490151019328_0005/	4096 bytes	22/03/2017 22:06:58
application_1490151019328_0006/	4096 bytes	22/03/2017 22:13:00
application_1490151019328_0007/	4096 bytes	22/03/2017 22:18:31
application_1490151019328_0008/	4096 bytes	22/03/2017 22:31:39
application_1490151019328_0009/	4096 bytes	22/03/2017 22:33:57
application_1490151019328_0010/	4096 bytes	22/03/2017 22:37:17
application_1490151019328_0011/	4096 bytes	22/03/2017 22:50:18
application_1490151019328_0012/	4096 bytes	22/03/2017 22:52:17
application_1490151019328_0013/	4096 bytes	22/03/2017 22:56:56
application_1490151019328_0014/	4096 bytes	22/03/2017 22:59:53
application_1490307791670_0002/	4096 bytes	23/03/2017 22:28:33
application_1490307791670_0003/	4096 bytes	23/03/2017 22:32:13
application_1490307791670_0004/	4096 bytes	23/03/2017 22:38:15
application_1490557613925_0001/	4096 bytes	26/03/2017 16:49:27
application_1490557613925_0002/	4096 bytes	26/03/2017 17:42:57
application_1490557613925_0003/	4096 bytes	26/03/2017 17:49:37
application_1490557613925_0004/	4096 bytes	26/03/2017 17:56:11
application_1490557613925_0005/	4096 bytes	26/03/2017 18:03:56



Directory: /logs/userlogs/application_1490557613925_0005/

[Parent Directory](#)

[container_1490557613925_0005_01_000001/](#) 4096 bytes 26/03/2017 18:03:36
[container_1490557613925_0005_01_000002/](#) 4096 bytes 26/03/2017 18:03:45
[container_1490557613925_0005_01_000003/](#) 4096 bytes 26/03/2017 18:03:45
[container_1490557613925_0005_01_000004/](#) 4096 bytes 26/03/2017 18:04:02



Depurando um job e os logs do Hadoop



Directory: /logs/userlogs/application_1490557613925_0005/container_1490557613925_0005_0001

[Parent Directory](#)

stderr	0 bytes	26/03/2017 18:03:56
stdout	163 bytes	26/03/2017 18:04:11
syslog	3125 bytes	26/03/2017 18:04:11
syslog.shuffle	3805 bytes	26/03/2017 18:04:10



Depurando um job e os logs do Hadoop



```
Chave 001 possui mais de 15000 itens ( 18664.428571428572 )
Chave 007 possui mais de 15000 itens ( 37482.75 )
Chave 009 possui mais de 15000 itens ( 21051.75 )
```

Usando System.err.println(...);



localhost:50070/logs/userlogs/application_1490557613925_0006/container_1490557613925_0006_01_000004/

Directory: /logs/userlogs/application_1490557613925_0006/container_1490557613925_0006_01_000004

[Parent Directory](#)

stderr	406 bytes	26/03/2017 18:29:22
stdout	0 bytes	26/03/2017 18:29:01
syslog	2746 bytes	26/03/2017 18:29:21
syslog.shuffle	4433 bytes	26/03/2017 18:29:21

localhost:50070/logs/userlogs/application_1490557613925_0006/container_1490557613925_0006_01_000004/stderr

```
Chave 001 possui mais de 15000 itens ( 18664.428571428572 )
Chave 007 possui mais de 15000 itens ( 37482.75 )
Chave 009 possui mais de 15000 itens ( 21051.75 )
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.impl.MetricsSystemImpl).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

Depurando um job e os logs do Hadoop

- Counters:

```
static enum Classificacao { MAIOR_15000, MENOR_15000, TODOS }  
  
public static class ReduceIGTI extends MapReduceBase implements Reducer<Text, Text, Text, Text> {  
  
    public void reduce (Text key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {  
        double media = 0.0;  
        int acumuladorItens = 0, contaVendas = 0;  
        Text value = new Text();  
  
        while (values.hasNext()) {  
            value = values.next();  
            contaVendas++;  
            acumuladorItens += Integer.parseInt(value.toString());  
        }  
        media = acumuladorItens / new Double(contaVendas);  
  
        if (media > 15000)  
            reporter.getCounter(Classificacao.MAIOR_15000).increment(1);  
        else  
            reporter.getCounter(Classificacao.MENOR_15000).increment(1);  
  
        reporter.getCounter(Classificacao.TODOS).increment(1);  
  
        value.set(String.valueOf(media));  
        output.collect(key, value);  
    }  
}
```

Depurando um job e os logs do Hadoop

- Counters:

```
Map-Reduce Framework
  Map input records=20
  Map output records=20
  Map output bytes=260
  Map output materialized bytes=312
  Input split bytes=226
  Combine input records=0
  Combine output records=0
  Reduce input groups=5
  Reduce shuffle bytes=312
  Reduce input records=20
  Reduce output records=5
  Spilled Records=40
  Shuffled Maps =2
  Failed Shuffles=0
  Merged Map outputs=2
  GC time elapsed (ms)=295
  CPU time spent (ms)=2940
  Physical memory (bytes) snapshot=651509760
  Virtual memory (bytes) snapshot=5749903360
  Total committed heap usage (bytes)=467140608
IGTI.ExemploIGTI$Classificacao ←
  MAIOR_15000=3
  MENOR_15000=2
  TODOS=5
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=3084
File Output Format Counters
  Bytes Written=71
```

Depurando um job e os logs do Hadoop

IGTI

- Plugins:
 - Eclipse.
 - Netbeans.

Conclusão

- ✓ Apresentadas maneiras de depurar código no Hadoop.

Próxima aula

- Algoritmos iterativos com o Hadoop.



Aula 4.9. Algoritmos iterativos com Hadoop



Nesta aula

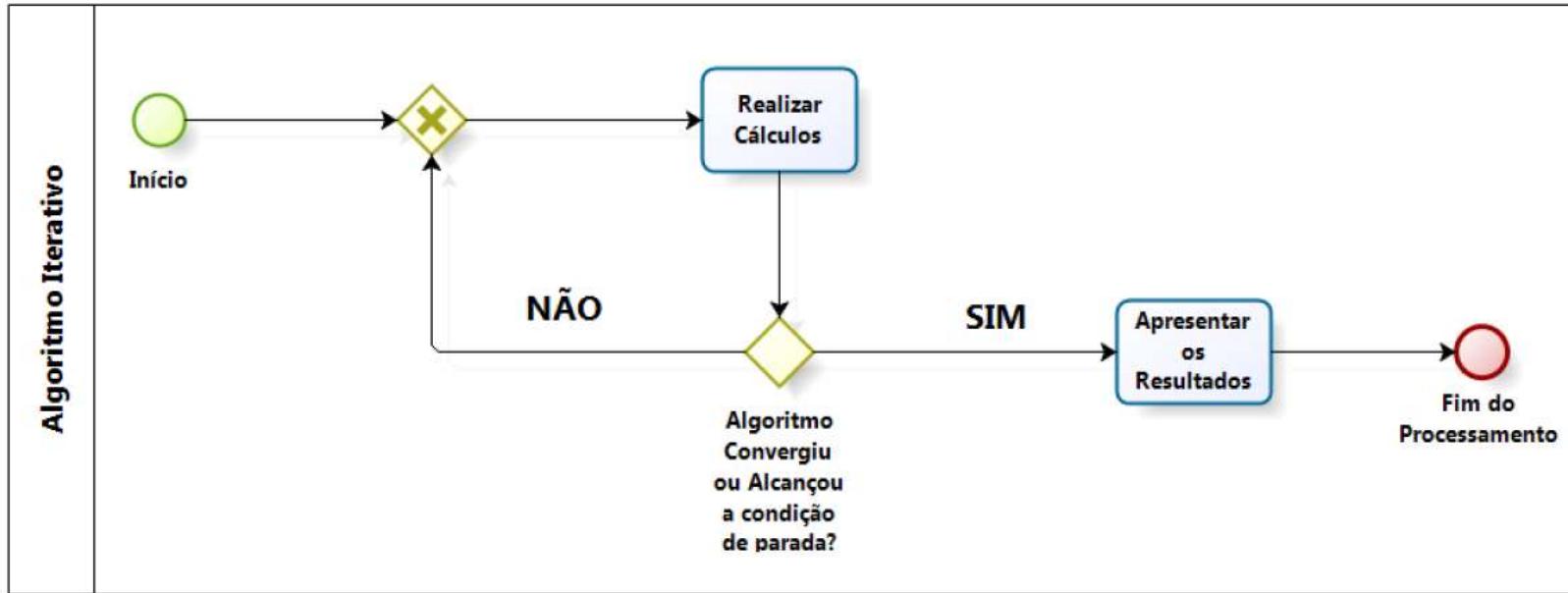
- ❑ Algoritmos iterativos e o Hadoop.
- ❑ Exemplo de um algoritmo iterativo.

Algoritmos iterativos com Hadoop



- Hadoop não é bom em lidar com algoritmos iterativos.
- A cada iteração os dados precisam ser gravados e recuperados do disco.
- Dados não são mantidos em memória.
- Isso causa overhead.

Fluxo de um algoritmo iterativo



- Exemplo:
 - 1 job para calcular as médias.
 - 1 job para encontrar qual a média com maior valor.

Algoritmos iterativos com Hadoop

```
public int run (final String[] args) throws Exception {
    try{
        JobConf conf = new JobConf(getConf(), ExemploIGTI.class);
        conf.setJobName("Media");
        conf.setStrings("codigoExecucao", Integer.toString(337));

        final FileSystem fs = FileSystem.get(conf);
        Path diretorioEntrada = new Path("Entrada"), diretorioSaida = new Path("Saida");

        /* Criar um diretório de entrada no HDFS */
        if (!fs.exists(diretorioEntrada))
            fs.mkdirs(diretorioEntrada);

        /* Adicionar um arquivo para ser processado */
        fs.copyFromLocalFile(new Path("/usr/local/hadoop/Dados/arquivoBigData.txt"), diretorioEntrada);

        /* Atribuindo os diretórios de Entrada e Saida para o Job */

        FileInputFormat.setInputPaths(conf, diretorioEntrada);
        FileOutputFormat.setOutputPath(conf, diretorioSaida);

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(Text.class);
        conf.setMapperClass(MapIGTI.class);
        conf.setReducerClass(ReduceIGTI.class);
        JobClient.runJob(conf);

        JobConf conf_total = new JobConf(getConf(), ExemploIGTI.class);
        conf_total.setJobName("Maior Valor");
        FileInputFormat.setInputPaths(conf_total, diretorioSaida);
        FileOutputFormat.setOutputPath(conf_total, new Path("Total"));
        conf_total.setOutputKeyClass(Text.class);
        conf_total.setOutputValueClass(Text.class);
        conf_total.setMapperClass(MapIGTIMaiorQuantidade.class);
        conf_total.setReducerClass(ReduceIGTIMaiorQuantidade.class);
        JobClient.runJob(conf_total);
    }
    catch ( Exception e ) {
        throw e;
    }
    return 0;
}
```

Algoritmos iterativos com Hadoop

IGTi

localhost:8088/cluster 90% C Pesquisa

All Applications

hadoop

Cluster Metrics												
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Vcores Used	Vcores Total	Vcores Reserved	Active Nodes	
10	0	0	10	0	0 B	8 GB	0 B	0	8	0	1	

Scheduler Metrics

Scheduler Type			Scheduling Resource Type			Minimum Allocation		
Capacity Scheduler			[MEMORY]			<memory:1024, vCores:1>		
Show 20	entries							
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus
application_1490557613925_0010	huser	Maior Valor	MAPREDUCE	default	Sun Mar 26 20:56:19 -0300 2017	Sun Mar 26 20:56:51 -0300 2017	FINISHED	SUCCEEDED
application_1490557613925_0009	huser	Media	MAPREDUCE	default	Sun Mar 26 20:55:26 -0300 2017	Sun Mar 26 20:56:17 -0300 2017	FINISHED	SUCCEEDED

Tools



Browse Directory

/user/hduser

Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	hduser	supergroup	0 B	26/03/2017 20:55:22	0	0 B	Entrada
drwxr-xr-x	hduser	supergroup	0 B	26/03/2017 20:56:16	0	0 B	Saida
drwxr-xr-x	hduser	supergroup	0 B	26/03/2017 20:56:50	0	0 B	Total

Hadoop, 2016.

Algoritmos iterativos com Hadoop

Diretório entrada – Arquivo original

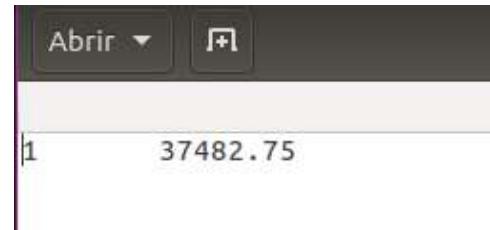
MapIGTI
ReduceIGTI

Diretório saída – Arquivo com as médias

001	18664.428
004	6644.5
007	37482.75
008	2235.0
009	21051.75

Diretório total – Maior média

MapIGTIMaiorQuantidade.class ReduceIGTIMaiorQuantidade.class



	Abrir	+
001	18664.428	
004	6644.5	
007	37482.75	
008	2235.0	
009	21051.75	

Arquivo de entrada

```
public static class MapIGTIMaiorQuantidade extends MapReduceBase implements Mapper<LongWritable, Text, Text, Text> {  
    public void map(LongWritable key, Text value, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {  
        Text txtChave = new Text();  
        Text txtValor = new Text();  
  
        String[] dados = value.toString().split("\t");  
  
        txtChave.set("1");  
        txtValor.set(dados[1]);  
  
        output.collect(txtChave, txtValor);  
    }  
}
```

Algoritmos iterativos com Hadoop



```
public static class ReduceIGTIMaiorQuantidade extends MapReduceBase implements Reducer<Text, Text, Text, Text> {

    public void reduce (Text key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {
        double maiorValor = 0.0;

        Text value = new Text();

        while (values.hasNext()) {
            value = values.next();
            if (Double.parseDouble(value.toString()) > maiorValor)
                maiorValor = Double.parseDouble(value.toString());

        }
        value.set(String.valueOf(maiorValor));
        output.collect(key, value);
    }
}
```

Algoritmos iterativos com Hadoop

```
    File System Counters
      FILE: Number of bytes read=77
      FILE: Number of bytes written=355485
      FILE: Number of read operations=0
      FILE: Number of large read operations=0
      FILE: Number of write operations=0
      HDFS: Number of bytes read=313
      HDFS: Number of bytes written=11
      HDFS: Number of read operations=9
      HDFS: Number of large read operations=0
      HDFS: Number of write operations=2
    Job Counters
      Launched map tasks=2
      Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
    File Input Format Counters
      Bytes Read=3084
    File Output Format Counters
      Bytes Written=71
17/03/26 20:56:18 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
17/03/26 20:56:18 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
17/03/26 20:56:19 INFO mapred.FileInputFormat: Total input paths to process : 1
17/03/26 20:56:19 INFO mapreduce.JobSubmitter: number of splits:2
17/03/26 20:56:19 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1490557613925_0010
17/03/26 20:56:19 INFO impl.YarnClientImpl: Submitted application application_1490557613925_0010
17/03/26 20:56:19 INFO mapreduce.Job: The url to track the job: http://JOAOLINUX:8088/proxy/application_1490557613925_0010/
17/03/26 20:56:19 INFO mapreduce.Job: Running job: job_1490557613925_0010
17/03/26 20:56:33 INFO mapreduce.Job: Job job_1490557613925_0010 running in uber mode : false
17/03/26 20:56:33 INFO mapreduce.Job: map 0% reduce 0%
17/03/26 20:56:43 INFO mapreduce.Job: map 100% reduce 0%
17/03/26 20:56:51 INFO mapreduce.Job: map 100% reduce 100%
17/03/26 20:56:52 INFO mapreduce.Job: Job job_1490557613925_0010 completed successfully
17/03/26 20:56:52 INFO mapreduce.Job: Counters: 49
```

Conclusão

- Iteratividade do Hadoop.
- Criação de um algoritmo iterativo.
- Fechamento do capítulo 4.



Módulo 3 - Solução de Dados utilizando Ecossistema Hadoop

Capítulo 5. Introdução ao Apache Spark

Prof. João Paulo Nascimento



Aula 5.1. Visão geral do Apache Spark



Nesta aula

- Primeiros conceitos.
- Introdução.

Visão geral

- Modelo MapReduce não é uma solução para todos os problemas.
- Existem lacunas que precisam ser preenchidas.
- Outras ferramentas especializadas surgiram.
- Apache Spark é uma delas.

- Framework para processamento distribuído.
- Oferece alto desempenho para processamento em lote e interativo.
- Possui APIs para Java, Python, R e Scala.
- Em seu núcleo existem projetos relacionados.

- Podemos executar aplicações Spark localmente ou de maneira distribuída (cluster).
- Um Shell Interativo ou submetendo uma aplicação.
- Spark requer um gerenciador de clusters.

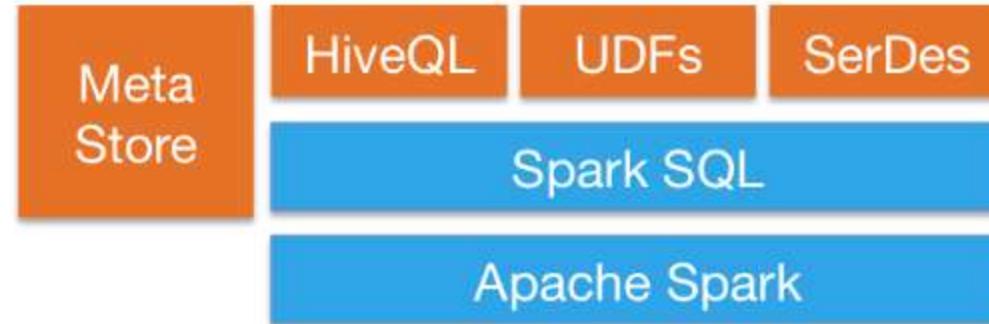
Visão geral

- Spark SQL:
 - Módulo para trabalhar com dados estruturados.
 - Permite utilizar consultas SQL ou DataFrames em conjunto com programas Spark.
 - Permite integrações com Hive.

```
results = spark.sql(  
    "SELECT * FROM people")  
names = results.map(lambda p: p.name)
```

Apply functions to results of SQL queries.

- Integração com Hive.



Fonte: <https://spark.apache.org/sql/>

- Conexão por meio de JDBC e ODBC.



Fonte: <https://spark.apache.org/sql/>

Visão geral

- Spark MLlib:

- Módulo que implementa algoritmos para Aprendizado de Máquinas.
- Inclui algoritmos de classificação.
- Regressão.
- Recomendação.
- Clusterização.
- Facilidade no uso.

```
data = spark.read.format("libsvm")\n    .load("hdfs://...")\n\nmodel = KMeans(k=10).fit(data)
```

Fonte: <https://spark.apache.org/mllib/>

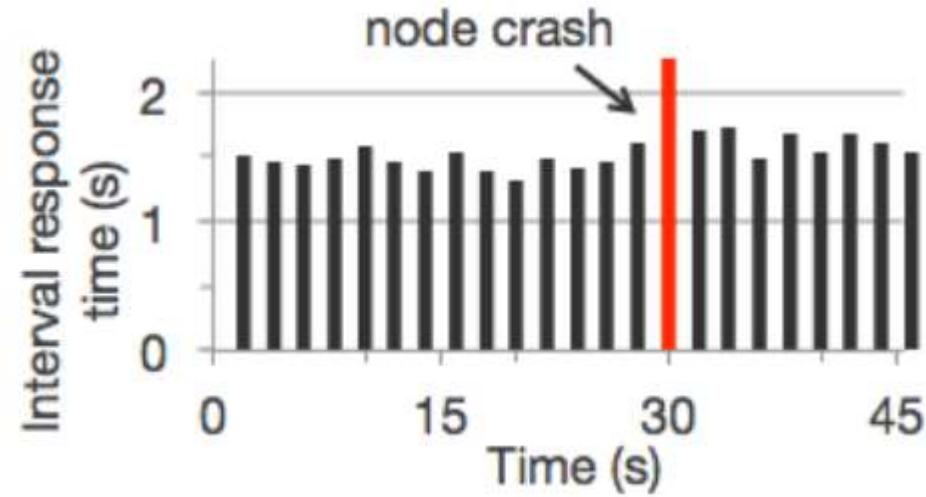
- Spark Streaming:
 - API que permite construir aplicações streaming.
 - Escaláveis e tolerantes a falhas.
 - Dados preferencialmente em memória.
 - Open-source.

- Spark Streaming:
 - Integrado ao Spark Core.
 - Pode ler dados do HDFS, Flume, Kafka, TCP sockets, dentre outros.
 - Possibilidade de combinar streaming com consultas SQL.
 - Sensores, tráfego de servidores, monitoramento em tempo real e eventos em geral.
 - Detecção de fraudes em tempo real.

```
TwitterUtils.createStream(...)  
    .filter(_.getText.contains("Spark"))  
    .countByWindow(Seconds(5))
```

Fonte: <https://spark.apache.org/streaming/>

Visão geral



Fonte: <https://spark.apache.org/streaming/>

- Spark GraphX:
 - API que permite realizar o processamento de grafos.
 - Compete em desempenho com os melhores processadores de grafos.
 - Oferece flexibilidade, tolerância a falhas e facilidade de uso.
 - Estruturas de dados para grafos.
 - Ampla variedade de algoritmos.
 - PageRank.
 - Componentes conectados.
 - Contagem de Triângulos.
 - Menor caminho.

- Spark SQL, MLlib, Streaming e GraphX podem ser combinados.
- Em um fluxo.



Conclusão

- Visão geral.
- Spark e seus componentes.

Próxima aula

- Características do Apache Spark.



Aula 5.2. Características do Apache Spark

Nesta aula

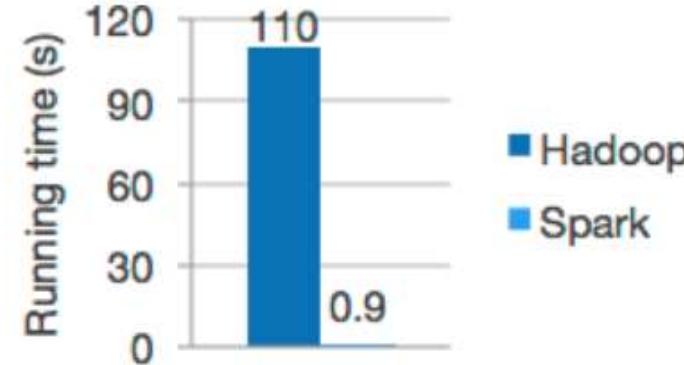
- Principais características.
- Diferenciais da ferramenta.



Características do Apache Spark

- Velocidade (processamento em memória).
- Utilização de RDDs.
- Resolveu grande deficiência do Hadoop MapReduce.
- Processamento distribuído em memória, mas também pode utilizar o disco.

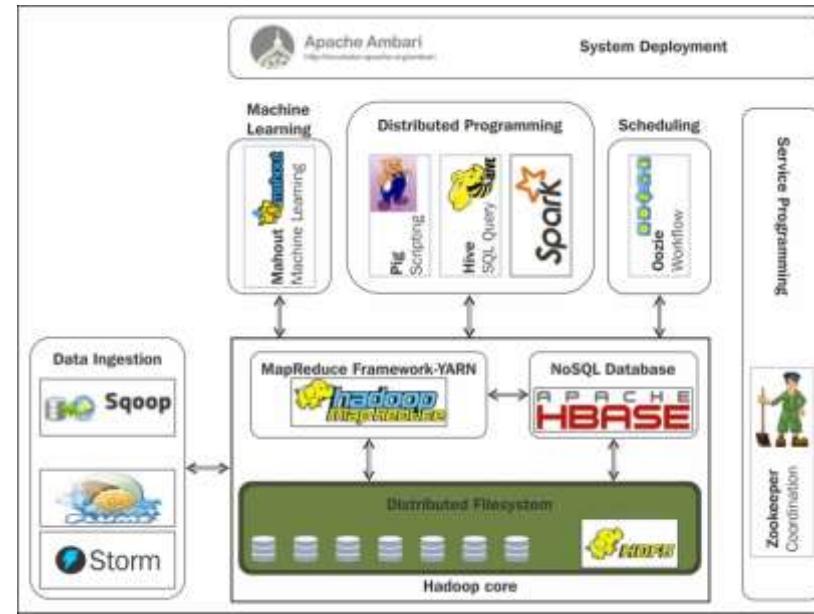
Características do Apache Spark



Fonte: <https://spark.apache.org/mllib/>

Características do Apache Spark

- Integrado e compatível com o ecossistema Hadoop.
- Possibilidade de uso de dados já existentes.





Características do Apache Spark

- Tolerante a falhas.
- Grande vantagem em relação ao Hadoop MapReduce.
- Resolve uma das deficiências do Hadoop MapReduce.
- Suporte a múltiplas linguagens (Poliglota).
- R, Java, Python e Scala.
- Facilita o desenvolvimento.



Características do Apache Spark

- Possibilidade de integrar com o SQL.
- Shell para Scala e Python.
- Uma grande comunidade.
- Muitos usuários em volta do mundo.
- Atualizações constantes.
- Várias integrações.



Conclusão

- Principais características.
- Diferenciais da ferramenta.

Próxima aula

- Componentes que formam a ferramenta.
- Suas integrações.



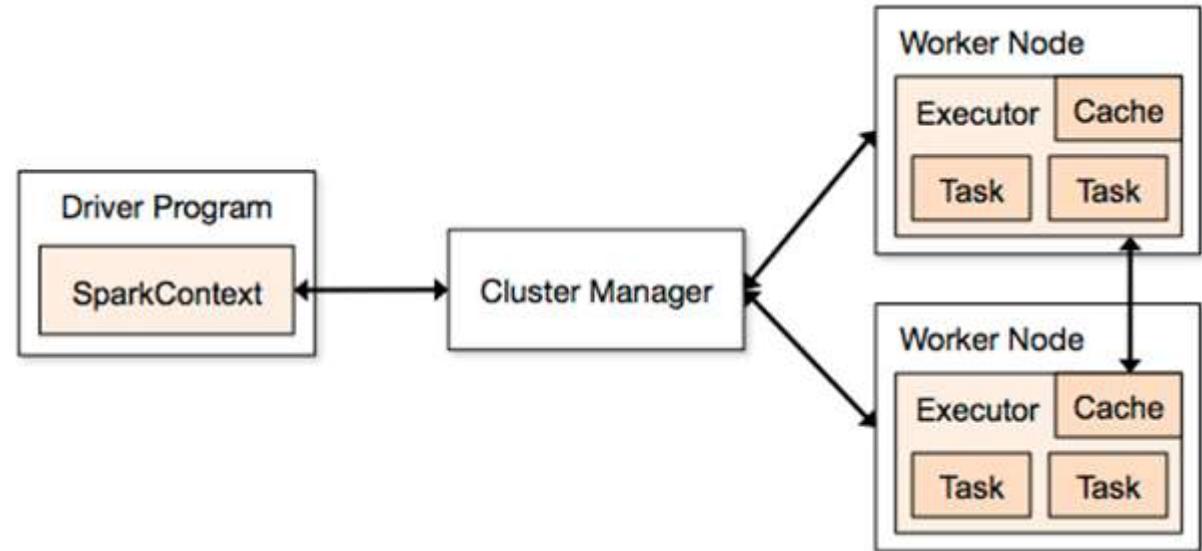
Aula 5.3. Arquitetura Spark

Nesta aula

- Componentes que formam a ferramenta.
- Suas integrações.

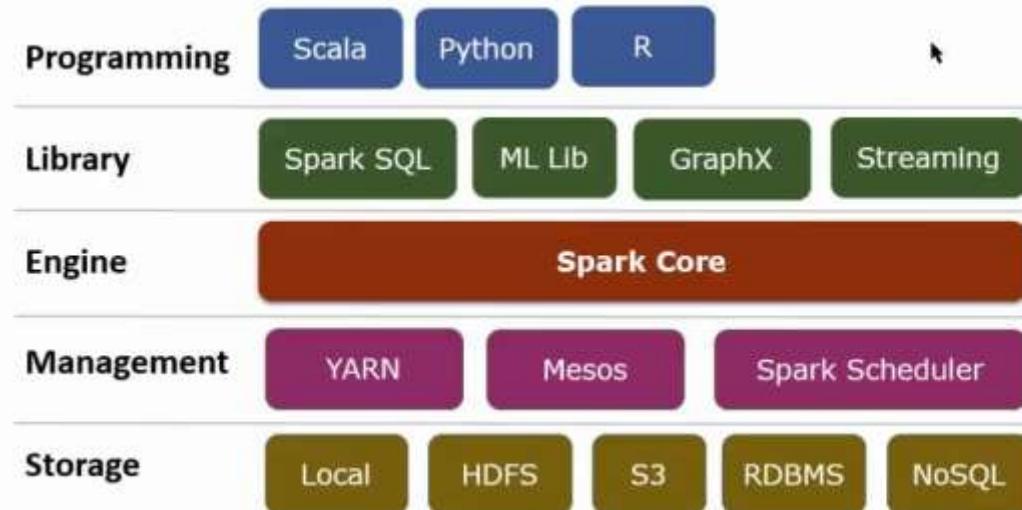
Arquitetura Spark

IGTI



Fonte: <https://spark.apache.org/>

Arquitetura Spark



Fonte: <https://maestros.com/>



Conclusão

- Componentes que formam a ferramenta.
- Suas integrações.



Próxima aula

- RDDs.
- Armazenamento de dados.



Aula 5.4. RDDs

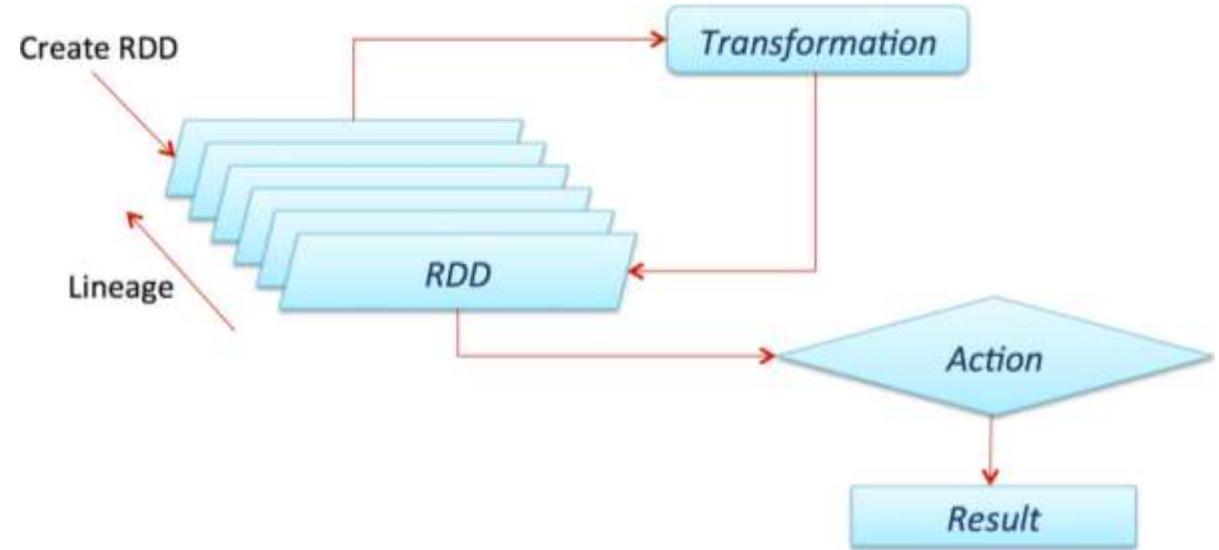


Nesta aula

- RDDs.
- Armazenamento de dados.

- Resilient Distributed Dataset.
- Coleção de objetos imutáveis.
- Núcleo do conceito do Spark.
- A maioria dos programas Spark é formado por manipulação de RDDs.

RDDs

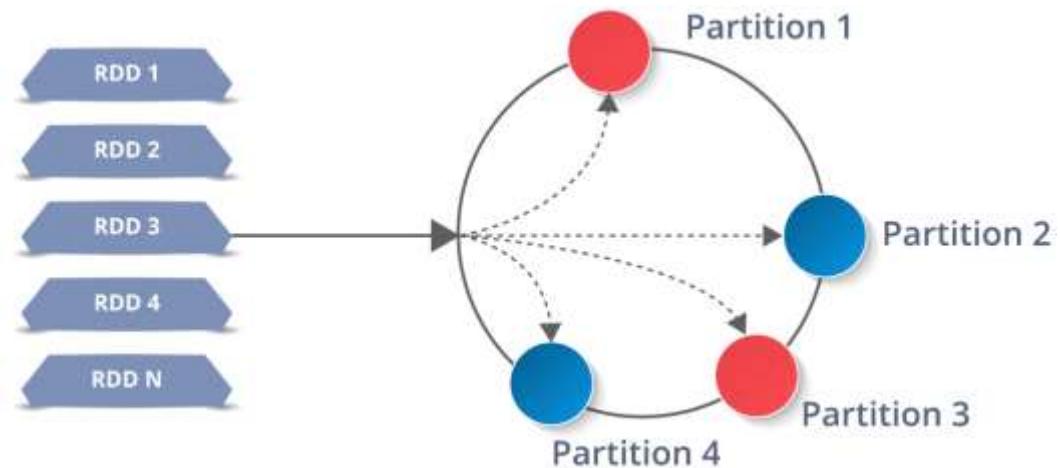


Fonte: <https://spark.apache.org/>

RDDs

- Cada RDD é dividido em múltiplas partições.
- Cada partição pode ser computada em um nó diferente do cluster.
- Pode conter um objeto Scala, Python, Java ou uma classe definida pelo usuário.
- Resilientes.
- Distribuídos.
- Imutáveis.

RDDs



- Formas de criar um RDD:
 - Paralelizando uma coleção de dados existente no próprio programa.
 - Carregando um conjunto de dados de um sistema de arquivos.

- Dois tipos de operações:
 - Transformações.
 - Ações.

- RDDs são lazy.
- Após uma transformação, nada do RDD é processado até que haja uma ação.



Conclusão

- RDDs.
- Estrutura de dados do Spark.



Próxima aula

- ❑ Vantagens do Apache Spark.



Aula 5.5. Vantagens de uso

Nesta aula

- ❑ Vantagens de uso do Apache Spark.
- ❑ Indicações de uso.

Vantagens de uso

- Velocidade.
- Poliglota.
- Processamento em memória.
- Integração com as outras ferramentas do ecossistema Hadoop.

Vantagens de uso

- Necessidade de uso.
- Grandes massas de dados.
- Quantidade que não cabe em uma só máquina.

Vantagens de uso

- Diversos experimentos realizados.
- Velocidade.
- Rapidez no desenvolvimento.
- Integração.
- Distribuição automática.
- Open-source.



Vantagens de uso

- O Spark é conhecido como o canivete suíço das ferramentas de Big Data Analytics.
- Muito popular devido a sua velocidade e computação iterativa.



Conclusão

- Vantagens de uso do Apache Spark.
- Indicações de uso.



Próxima aula

- ❑ Spark x MapReduce.



Aula 1.6. Spark x MapReduce

Nesta aula

Spark x Hadoop:

- Aplicação de cada tecnologia.
- Diferenças.
- Formas de trabalho.

- Hadoop e Spark foram projetados para:
 - Grandes massas de dados.
 - Baseadas no modelo MapReduce.
 - Tolerante a falhas.
- Mas nem sempre servem para as mesmas coisas.

- As duas ferramentas não são mutuamente exclusivas.
- Estão habilitadas para trabalharem juntas.
- Spark pode ser até 100 vezes mais rápido do que o Hadoop.
- Para alguns casos específicos.
- Mas não possui seu próprio sistema de arquivos distribuído.

- Armazenamento distribuído é fundamental para projetos de Big Data.
- Grande armazenamento em grande quantidade de máquinas.
- Ambos os sistemas são escaláveis.
- É necessário instalar o Spark sobre o Hadoop.
- Assim as aplicações podem usar o HDFS.

- O que realmente dá vantagem ao Spark é a velocidade.
- Spark manipula dados em memória.
- Isso reduz o tempo gasto em leitura e escrita (I/O).
- O Hadoop escreve os dados em disco após cada operação.
- O Spark mantém os dados nos RDDs.
- Isso garante a recuperação de falhas.



Visão geral do Apache Spark

- Spark possui mais funcionalidades implementadas:
 - Aprendizado de máquina.
 - Tempo real.
 - Grafos.
- Spark é melhor para aplicações interativas.

- Hadoop possui apenas dois operadores para interativas: map e reduce.
- Com o tempo observou-se que é necessário muito mais que isso.
- Por possuir vários operadores o Spark está mais preparado para lidar com aplicações interativas.

- Cada carga de trabalho (workload) tem a sua particularidade.
- Devemos estudar caso a caso antes de decidir pela aplicação de uma tecnologia.
- Outro fator a considerar é o tamanho do cluster disponível.

Conclusão

Spark x Hadoop:

- Spark: processamento em memória e aplicações interativas. Não possui sistema de arquivos próprio.

Devemos considerar:

- Tamanho da base de dados.
- Tamanho do cluster.
- Particularidades da aplicação.



Próxima aula

- Criando programas Apache Spark.



Módulo 3 - Solução de Dados utilizando Ecossistema Hadoop

Capítulo 6. Criando Programas Apache Spark

Prof. João Paulo Nascimento



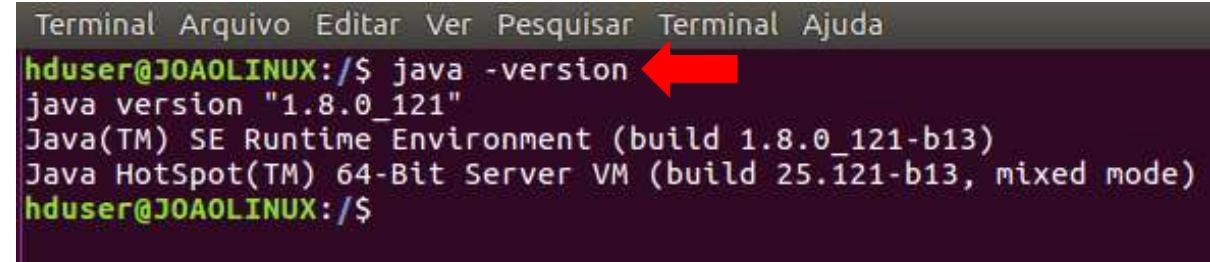
Aula 6.1. Fazendo o download e instalando o Apache Spark

Nesta aula

- ❑ Apache Spark.
- ❑ Instalação.
- ❑ Spark-submit.

- **Passo 1:** Java.

- Verificar a existência do Java no ambiente Linux.



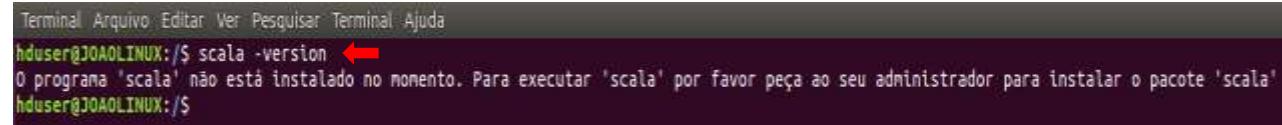
```
Terminal Arquivo Editar Ver Pesquisar Terminal Ajuda
hduser@JOAOLINUX:/$ java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
hduser@JOAOLINUX:/$
```

A screenshot of a terminal window on a Linux system. The window title bar includes 'Terminal', 'Arquivo', 'Editar', 'Ver', 'Pesquisar', 'Terminal', and 'Ajuda'. The main area of the terminal shows the command 'java -version' being run by the user 'hduser' at the prompt 'JOAOLINUX:/\$'. The output of the command is displayed below, indicating Java version 1.8.0_121, Java(TM) SE Runtime Environment, and Java HotSpot(TM) 64-Bit Server VM. A red arrow points to the command line where 'java -version' was typed.

- Caso não exista o Java, instale-o.

- **Passo 2:** verificando a existência do Scala.

- É a linguagem que vamos utilizar para implementar nossos programas no Spark.



```
Terminal Arquivo Editar Ver Pesquisar Terminal Ajuda
hduser@JOAOLINUX:~$ scala -version
O programa 'scala' não está instalado no momento. Para executar 'scala' por favor peça ao seu administrador para instalar o pacote 'scala'
hduser@JOAOLINUX:~$
```

- **Passo 3:** instalando o Scala.
 - <https://www.scala-lang.org/download/>
 - Versão: Scala 2.12.1
 - Salvar o arquivo no disco.
 - Após o download, o arquivo possivelmente estará no diretório downloads.

```
hduser@JOAOLINUX:/home/joao/Downloads$ ls
scala-2.12.1.tgz
hduser@JOAOLINUX:/home/joao/Downloads$ █
```

- **Passo 3:** instalando o Scala.

- Mova o arquivo para o diretório /usr/local/, conforme comando abaixo:

```
Terminal Arquivo Editar Ver Pesquisar Terminal Ajuda
hduser@JOAOLINUX:/home/joao/Downloads$ sudo mv scala-2.12.1.tgz /usr/local/
hduser@JOAOLINUX:/home/joao/Downloads$
```

- Vá para o diretório /usr/local/:

```
Terminal Arquivo Editar Ver Pesquisar Terminal Ajuda
hduser@JOAOLINUX:/home/joao/Downloads$ cd /usr/local
hduser@JOAOLINUX:/usr/local$
```

- Descompacte o arquivo tgz, conforme comando abaixo:

```
Terminal Arquivo Editar Ver Pesquisar Terminal Ajuda
hduser@JOAOLINUX:/usr/local$ sudo tar xvf scala-2.12.1.tgz
```

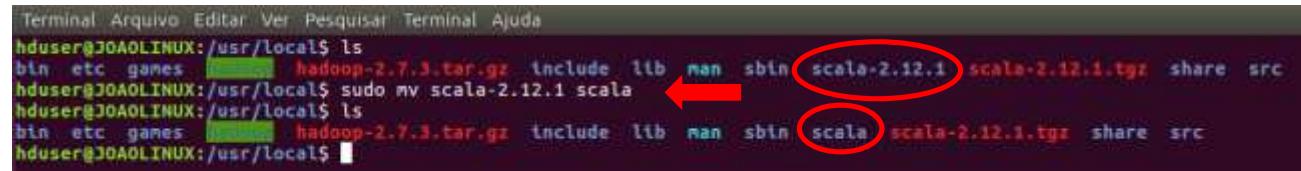
Instalando o Apache Spark

IGTI

■ Passo 3: instalando o Scala.

```
scala-2.12.1/
scala-2.12.1/lib/
scala-2.12.1/lib/scala-parser-combinators_2.12-1
scala-2.12.1/lib/scala-swing_2.12-2.0.0-M2.jar
scala-2.12.1/lib/scala-reflect.jar
scala-2.12.1/lib/jline-2.14.1.jar
scala-2.12.1/lib/scala-compiler.jar
scala-2.12.1/lib/scalap-2.12.1.jar
scala-2.12.1/lib/scala-library.jar
scala-2.12.1/lib/scala-xml_2.12-1.0.6.jar
scala-2.12.1/bin/
scala-2.12.1/bin/scala
scala-2.12.1/bin/scalac.bat
scala-2.12.1/bin/scala.bat
scala-2.12.1/bin/scalap
scala-2.12.1/bin/scalap.bat
scala-2.12.1/bin/scaladoc.bat
scala-2.12.1/bin/fsc
scala-2.12.1/bin/fsc.bat
scala-2.12.1/bin/scalac
scala-2.12.1/bin/scaladoc
scala-2.12.1/man/
scala-2.12.1/man/man1/
scala-2.12.1/man/man1/scalac.1
scala-2.12.1/man/man1/scaladoc.1
scala-2.12.1/man/man1/fsc.1
scala-2.12.1/man/man1/scala.1
scala-2.12.1/man/man1/scalap.1
scala-2.12.1/doc/
scala-2.12.1/doc/tools/
scala-2.12.1/doc/tools/scala.html
scala-2.12.1/doc/tools/index.html
scala-2.12.1/doc/tools/images/
scala-2.12.1/doc/tools/images/scala_logo.png
scala-2.12.1/doc/tools/images/external.gif
scala-2.12.1/doc/tools/fsc.html
scala-2.12.1/doc/tools/scalac.html
scala-2.12.1/doc/tools/css/
scala-2.12.1/doc/tools/css/style.css
scala-2.12.1/doc/tools/scalap.html
scala-2.12.1/doc/tools/scaladoc.html
scala-2.12.1/doc/license.rtf
scala-2.12.1/doc/licenses/
scala-2.12.1/doc/licensesbsd_asm.txt
scala-2.12.1/doc/licenses/mit_jquery.txt
scala-2.12.1/doc/licensesbsd_jline.txt
scala-2.12.1/doc/licenses/mit_sizzle.txt
scala-2.12.1/doc/licenses/mit_toolstooltip.txt
scala-2.12.1/doc/licenses/apache_janst.txt
scala-2.12.1/doc/LICENSE.md
scala-2.12.1/doc/README
```

- **Passo 3:** instalando o Scala.
 - Será criada o diretório scala-2.12.1.
 - Para simplificar o trabalho, renomeie esse diretório para Scala.



```
Terminal Arquivo Editar Ver Pesquisa Terminal Ajuda
hduser@JOAOLINUX:/usr/local$ ls
bin etc games hadoop-2.7.3.tar.gz include lib man sbin scala-2.12.1 scala-2.12.1.tgz share src
hduser@JOAOLINUX:/usr/local$ sudo mv scala-2.12.1 scala
hduser@JOAOLINUX:/usr/local$ ls
bin etc games hadoop-2.7.3.tar.gz include lib man sbin scala scala-2.12.1.tgz share src
hduser@JOAOLINUX:/usr/local$
```

- **Passo 3:** instalando o Scala.
 - Atualizando o Path.
 - Por meio do seguinte comando:

```
Terminal Arquivo Editar Ver Pesquisar Terminal Ajuda
hduser@JOAOLINUX:/usr/local$ export PATH=$PATH:/usr/local/scala/bin
hduser@JOAOLINUX:/usr/local$
```

- **Passo 3:** instalando o Scala.
 - Verifique a instalação.
 - Se o Scala estiver corretamente instalado, a seguinte mensagem será exibida:

```
hduser@JOAOLINUX:/usr/local$ scala -version
Scala code runner version 2.12.1 -- Copyright 2002-2016, LAMP/EPFL and Lightbend, Inc.
hduser@JOAOLINUX:/usr/local$
```

Instalando o Apache Spark

IGTI

- Passo 4: download e instalação do Apache Spark.

The screenshot shows the Apache Spark website at spark.apache.org/downloads.html. The page features the Apache logo and the text "Lightning-fast cluster computing". A navigation bar includes links for Download, Libraries, Documentation, Examples, Community, Developers, and Apache Software Foundation. On the left, a sidebar titled "Latest News" lists recent releases and events. The main content area is titled "Download Apache Spark™" and contains a numbered list of steps for downloading the software. Step 1: Choose a Spark release: 2.1.0 (Dec 26 2016). Step 2: Choose a package type: Pre-built for Hadoop 2.7 and later. Step 3: Choose a download type: Direct Download. Step 4: Download Spark: spark-2.1.0-bin-hadoop2.7.tgz. Step 5: Verify this release using the 2.1.0 signatures and checksums and project release KEYS. A note at the bottom states: "Note: Starting version 2.0, Spark is built with Scala 2.11 by default. Scala 2.10 users should download the Spark source package and build with Scala 2.10 support." A green "Download Spark" button is located at the bottom right.

spark.apache.org/downloads.html

marks Doutorado Eventos IGTI UNIBH TIMG Apartamentos Computing Shortest c| BIBLIOTECA NACIONAL Moedas | Valor Econômico Sistema On-Line HP - Portal de Serviços

APACHE Spark Lightning-fast cluster computing

Download Libraries Documentation Examples Community Developers Apache Software Foundation

Latest News

- Spark Summit East (Feb 7-9th, 2017, Boston) agenda posted (Jan 04, 2017)
- Spark 2.1.0 released (Dec 26, 2016)
- Spark wins CloudSoft Benchmark as the most efficient engine (Nov 15, 2016)
- Spark 2.0.2 released (Nov 14, 2016)

Archive

Download Apache Spark™

1. Choose a Spark release: 2.1.0 (Dec 26 2016)
2. Choose a package type: Pre-built for Hadoop 2.7 and later
3. Choose a download type: Direct Download
4. Download Spark: spark-2.1.0-bin-hadoop2.7.tgz
5. Verify this release using the 2.1.0 signatures and checksums and project release KEYS.

Note: Starting version 2.0, Spark is built with Scala 2.11 by default. Scala 2.10 users should download the Spark source package and build with Scala 2.10 support.

Link with Spark

Download Spark

Instalando o Apache Spark

- **Passo 4:** download e instalação do Apache Spark.
 - Possivelmente o download do Spark foi feito em seu diretório “Downloads”.

```
Terminal Arquivo Editar Ver Pesquisar Terminal Ajuda
hduser@JOAOLINUX:/home/joao/Downloads$ ls
spark-2.1.0-bin-hadoop2.7.tgz
hduser@JOAOLINUX:/home/joao/Downloads$
```

- Mova o arquivo de instalação para o diretório /usr/local/.

```
Terminal Arquivo Editar Ver Pesquisar Terminal Ajuda
hduser@JOAOLINUX:/home/joao/Downloads$ sudo mv spark-2.1.0-bin-hadoop2.7.tgz /usr/local
[sudo] senha para hduser:
hduser@JOAOLINUX:/home/joao/Downloads$ cd /usr/local
hduser@JOAOLINUX:/usr/local$ ls
bin  etc  games  hadoop-2.7.3.tar.gz  include  lib  man  sbin  scala  scala-2.12.1.tgz  share  spark-2.1.0-bin-hadoop2.7.tgz  src
hduser@JOAOLINUX:/usr/local$
```

Instalando o Apache Spark

- **Passo 4:** download e instalação do Apache Spark.
 - Extraia os arquivos dentro do diretório /usr/local/.

```
hduser@JOAOLINUX:/usr/local$ sudo tar xvf spark-2.1.0-bin-hadoop2.7.tgz
spark-2.1.0-bin-hadoop2.7/
spark-2.1.0-bin-hadoop2.7/NOTICE
spark-2.1.0-bin-hadoop2.7/jars/
spark-2.1.0-bin-hadoop2.7/jars/bonecp-0.8.0.RELEASE.jar
spark-2.1.0-bin-hadoop2.7/jars/commons-net-2.2.jar
spark-2.1.0-bin-hadoop2.7/jars/javax.servlet-api-3.1.0.jar
spark-2.1.0-bin-hadoop2.7/jars/hadoop-annotations-2.7.3.jar
spark-2.1.0-bin-hadoop2.7/jars/hadoop-hdfs-2.7.3.jar
```

- Foi criado o diretório spark-2.1.0-bin-hadoop2.7.
 - Renomeie o diretório para spark.

```
Terminal Arquivo Editar Ver Pesquisar Terminal Ajuda
hduser@JOAOLINUX:/usr/local$ sudo mv spark-2.1.0-bin-hadoop2.7 spark
hduser@JOAOLINUX:/usr/local$ ls
bin  etc  games  lib  libexec  mnt  opt  root  run  share  spark  spark-2.1.0-bin-hadoop2.7.tgz  src
hduser@JOAOLINUX:/usr/local$
```

- **Passo 4:** download e instalação do Apache Spark.
 - Atualizando a variável de ambiente.

```
Terminal Arquivo Editar Ver Pesquisar Terminal Ajuda
hduser@JOAOLINUX:/usr/local$ export PATH=$PATH:/usr/local/spark/bin
hduser@JOAOLINUX:/usr/local$
```

- Use o seguinte comando para atualizar o arquivo `~/.bashrc`.

```
hduser@JOAOLINUX:/usr/local$ source ~/.bashrc
hduser@JOAOLINUX:/usr/local$
```

Instalando o Apache Spark

■ Passo 5: verificando a instalação.

```
hduser@JOAOLINUX:/usr/local$ spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/03/27 01:56:09 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
17/03/27 01:56:09 WARN Utils: Your hostname, JOAOLINUX resolves to a loopback address: 127.0.1.1; using
17/03/27 01:56:09 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
17/03/27 01:56:21 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.ve
17/03/27 01:56:21 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
17/03/27 01:56:24 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1490590571153).
Spark session available as 'spark'.
Welcome to

    /____/\   \  /____/\   \ /____/
   /  \  / \  /  \ /  \  / \  / \  / \
  /    \ / \ /    \ / \ / \ / \ / \ / \
 /____/\ / \ /____/\ / \ / \ / \ / \ / \
                                     version 2.1.0

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_121)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```



Instalando o Apache Spark

- Problemas na instalação:
 - Permissões na pasta local.
 - sudo chown 777 /usr/local
 - sudo chown 777 /usr/local/spark

Conclusão

- Instalação do Scala e Spark.
- Teste do Shell.



Próxima aula

- Conceitos básicos do Apache Spark.



Aula 6.2. Conceitos básicos do Apache Spark

Nesta aula

- ❑ Introduzindo os conceitos do Spark:
 - Contexto.
 - Primeiro exemplo.
 - RDDs.

- No Spark realizamos o processamento através de operações.
- Operações em coleções distribuídas.
- Automaticamente paralelizadas no cluster.
- RDDs: Resilient Distributed Datasets.
- Vamos usar o Shell para um exemplo:

Conceitos básicos do Apache Spark

```
hduser@JOAOLINUX:/usr/local$ spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/03/29 20:56:15 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform..
17/03/29 20:56:16 WARN Utils: Your hostname, JOAOLINUX resolves to a loopback address: 127.0.1.1; set SPARK_LOCAL_IP if you need to bind to another address
17/03/29 20:56:16 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
17/03/29 20:56:26 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1490831777437).
Spark session available as 'spark'.
Welcome to

    / \ \ / -- \ \ \ \ \ \ / \ / \
   / \ \ / - \ \ \ \ \ \ / \ / \
  / \ \ / . \ \ \ \ \ / \ / \
 / \ \ / \ \ \ \ \ / \ / \
version 2.1.0

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_121)
Type in expressions to have them evaluated.
Type :help for more information.

scala> █
```

Conceitos básicos do Apache Spark

- Executando um teste:

```
version 2.1.0

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_121)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val linhas = sc.textFile("/usr/local/spark/README.md") \\Cria um RDD chamado linhas
linhas: org.apache.spark.rdd.RDD[String] = /usr/local/spark/README.md MapPartitionsRDD[1] at textFile at <console>:24

scala> linhas.count() \\ Conta o número de itens no RDD linhas
res0: Long = 104

scala> linhas.first() \\ Exibe o primeiro item do RDD (primeira linha de README.md)
res1: String = # Apache Spark

scala> ■
```

- Para sair do Shell: CTRL+D ou :q + Enter.

Conceitos básicos do Apache Spark

Apache Spark

Spark is a fast and general cluster computing system for Big Data. It provides high-level APIs in Scala, Java, Python, and R, and an optimized engine that supports general computation graphs for data analysis. It also supports a rich set of higher-level tools including Spark SQL for SQL and DataFrames, MLLib for machine learning, GraphX for graph processing, and Spark Streaming for stream processing.

<<http://spark.apache.org/>>

Online Documentation

You can find the latest Spark documentation, including a programming guide, on the [[project web page](#)](<http://spark.apache.org/documentation.html>). This README file only contains basic setup instructions.

Building Spark

Spark is built using [[Apache Maven](#)](<http://maven.apache.org/>). To build Spark and its example programs, run:

```
build/mvn -DskipTests clean package
```

(You do not need to do this if you downloaded a pre-built package.)

You can build Spark using more than one thread by using the -T option with Maven, see [["Parallel Build"](#)]. More detailed documentation is available from the project site, at [["Building Spark"](#)](<http://spark.apache.org/docs/latest/building-spark.html>).

For general development tips, including info on developing Spark using an IDE, see [[http://spark.apache.org/developer-tools.html](#)](the Useful Developer Tools page).

Interactive Scala Shell

The easiest way to start using Spark is through the Scala shell:

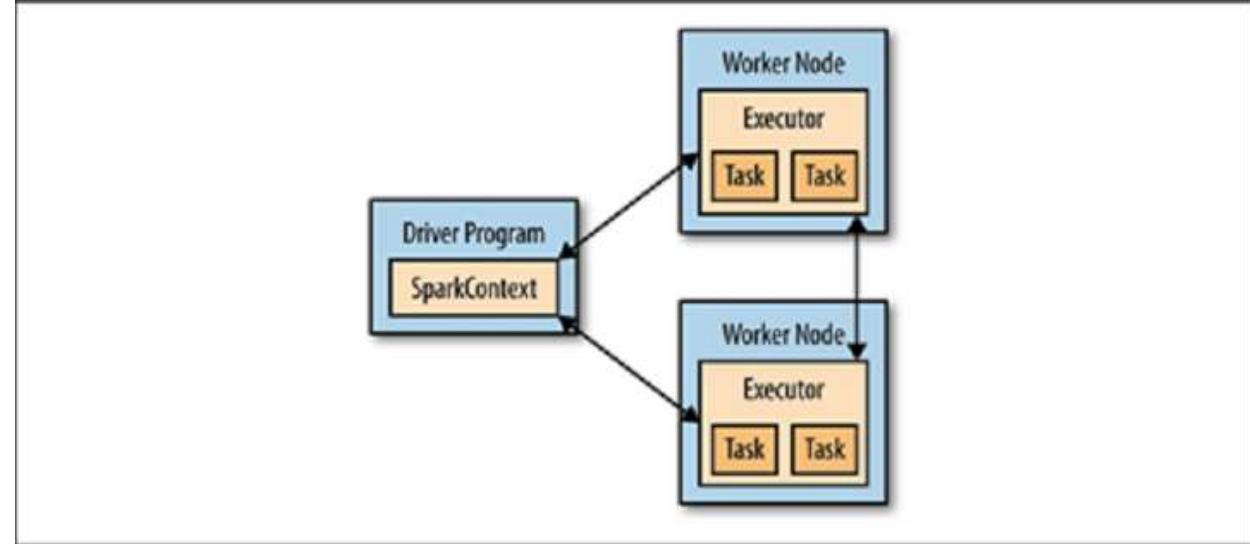
- A variável “linhas” é um RDD criado de um arquivo texto de sua máquina local.
- Várias operações podem ser executadas em um RDD.
- Posteriormente iremos analisar melhor os RDDs e suas operações.

- Cada aplicação Spark consiste de um programa driver.
- Ele executa várias operações em paralelo no cluster.
- O programa driver contém a função principal da aplicação.
- No exemplo anterior o PD foi o Spark Shell.
- O PD acessa o Spark através do objeto SparkContext.
- No Shell o SparkContext é automaticamente criado como uma variável chamada sc.

Conceitos básicos do Apache Spark

- Uma vez que você tem um SparkContext, você pode usá-lo para construir RDDs.
- sc.textFile(...) → Usamos para criar um RDD representando as linhas de um arquivo texto.
- Para executar as operações o programa driver gerencia um número de nós chamados de executors.
- Se você está executando o count() em um cluster, diferentes máquinas podem contar diferentes partes do arquivo.

Conceitos básicos do Apache Spark



Componentes para execução distribuída no Spark

Fonte: spark.apache.org

- Outras operações podem ser executadas.
- Podemos ampliar o programa anterior e filtrar as linhas do arquivo que possuem uma determinada palavra. Exemplo: palavra “cluster”.

```
scala> val linhas = sc.textFile("/usr/local/spark/README.md")
linhas: org.apache.spark.rdd.RDD[String] = /usr/local/spark/README.md MapPartitionsRDD[4] at textFile at <console>:24

scala> val linhasCluster = linhas.filter(line => line.contains("cluster"))
linhasCluster: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[5] at filter at <console>:26

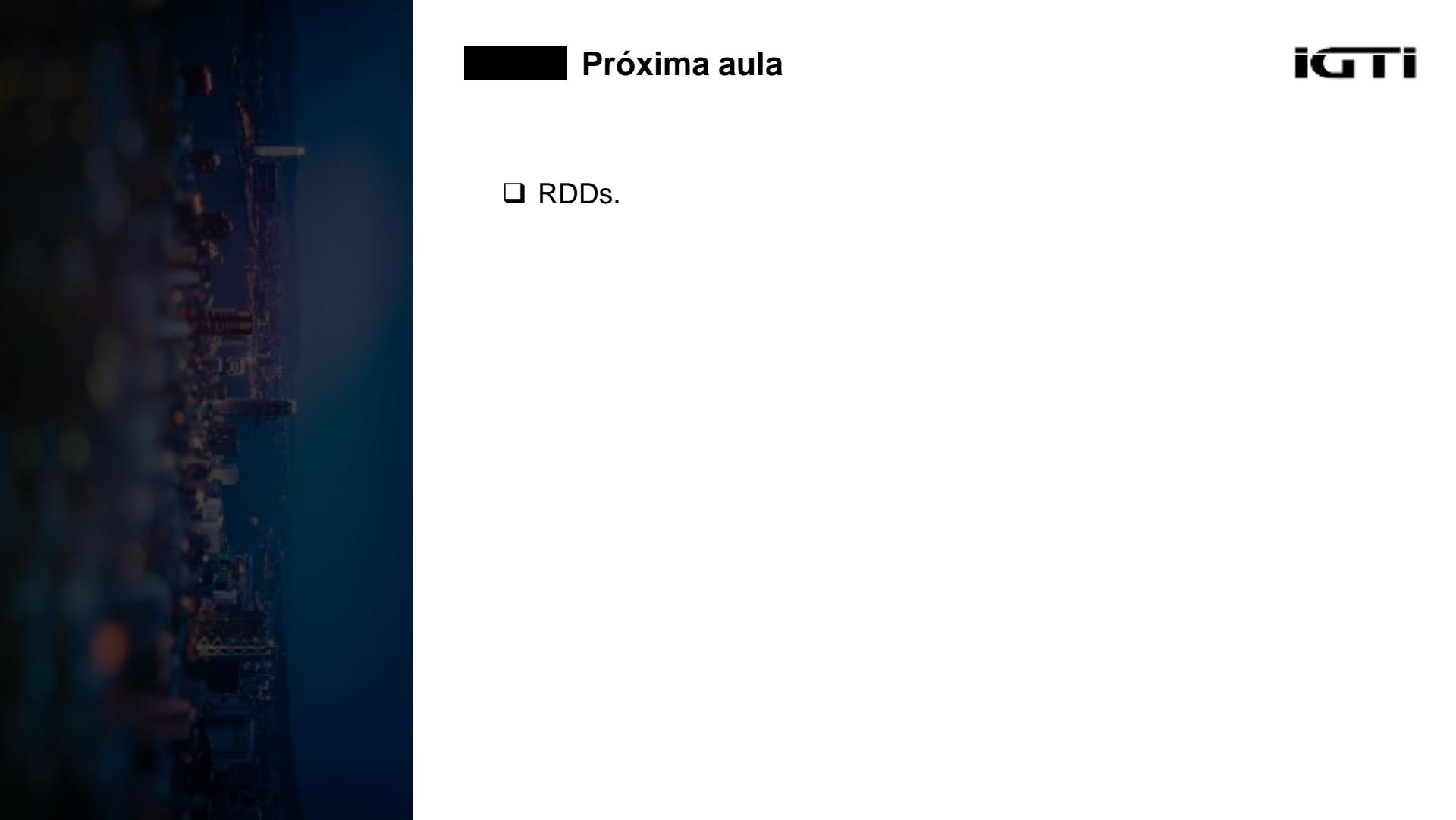
scala> linhasCluster.count()
res5: Long = 3

scala> linhasCluster.first()
res6: String = Spark is a fast and general cluster computing system for Big Data. It provides

scala> ■
```

Conclusão

- SparkContext.
- Primeiro exemplo.
- RDDs.



Próxima aula

- ❑ RDDs.



Aula 6.3. RDDs na prática



Nesta aula

- ❑ RDDs.

RDDs

- Uma coleção distribuída de elementos.
- Cada RDD é dividido em múltiplas partitions, os quais podem ser computados em diferentes nós do cluster.
- O usuário cria RDDs de duas formas:
 - Carregando um conjunto de dados externo (arquivo texto, por exemplo).
 - Distribuindo uma coleção de objetos em seu programa driver.

- Uma vez criado, RDDs têm dois tipos de operações:
 - Transformação.
 - Ação.

- **Transformação (Transformations).**
- Constrói um novo RDD a partir de um prévio.
- Exemplo: filter.

```
scala> val linhas = sc.textFile("/usr/local/spark/README.md")
linhas: org.apache.spark.rdd.RDD[String] = /usr/local/spark/README.md MapPartitionsRDD[4] at textFile at <console>:24

scala> val linhasCluster = linhas.filter(line => line.contains("cluster"))
linhasCluster: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[5] at filter at <console>:26

scala> linhasCluster.count()
res5: Long = 3

scala> linhasCluster.first()
res6: String = Spark is a fast and general cluster computing system for Big Data. It provides

scala>
```

- **Transformação.**
- Transformações são aplicadas a cada elemento do RDD.
- RDDs são imutáveis.
- Operação filter() não muda o RDD existente (“linhas”).
- Ao invés disso retorna um novo RDD (“linhasCluster”).
- RDD “linhas” pode ser reusado em outra parte do programa.

```
scala> val linhas = sc.textFile("/usr/local/spark/README.md")
linhas: org.apache.spark.rdd.RDD[String] = /usr/local/spark/README.md MapPartitionsRDD[4] at textFile at <console>:24
scala> val linhasCluster = linhas.filter(line => line.contains("cluster"))
linhasCluster: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[5] at filter at <console>:26
scala> linhasCluster.count()
res5: Long = 3
scala> linhasCluster.first()
res6: String = Spark is a fast and general cluster computing system for Big Data. It provides
scala>
```

- Transformação.

- Union.

```
Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_121)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val todasLinhas = sc.textFile("/usr/local/spark/README.md")
todasLinhas: org.apache.spark.rdd.RDD[String] = /usr/local/spark/README.md MapPartitionsRDD[1]

scala> todasLinhas.count()
res0: Long = 104

scala> val linhasCluster = todasLinhas.filter(linha => linha.contains("cluster"))
linhasCluster: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at <console>:26

scala> linhasCluster.count()
res1: Long = 3

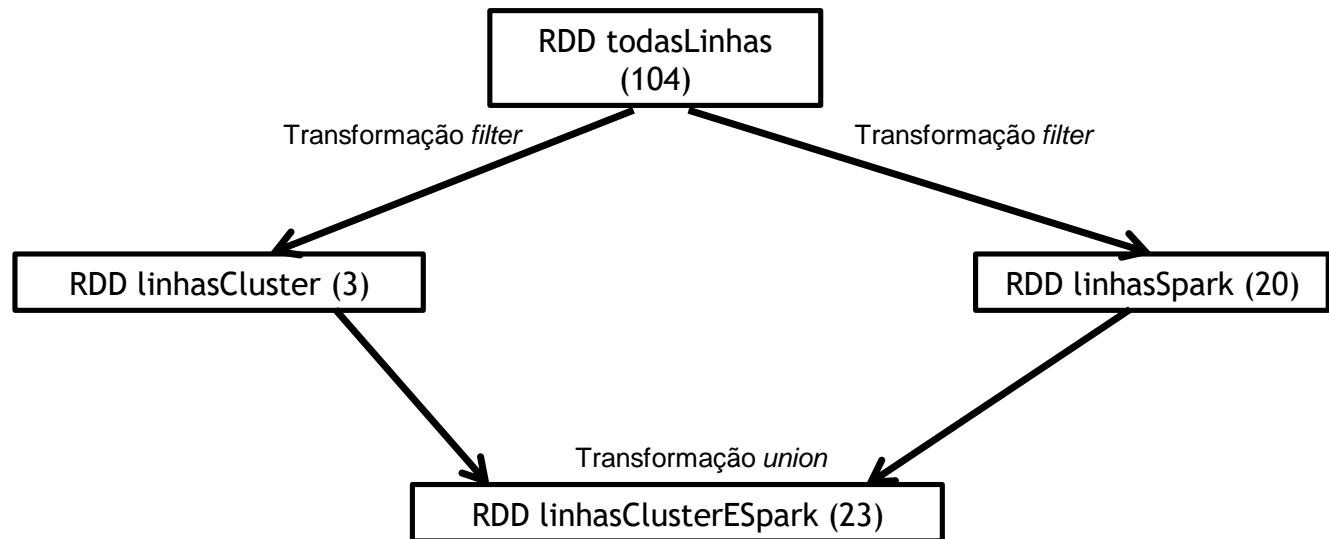
scala> val linhasSpark = todasLinhas.filter(linha => linha.contains("Spark"))
linhasSpark: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at filter at <console>:26

scala> linhasSpark.count()
res2: Long = 20

scala> val linhasClusterESpark = linhasCluster.union(linhasSpark)
linhasClusterESpark: org.apache.spark.rdd.RDD[String] = UnionRDD[4] at union at <console>:30

scala> linhasClusterESpark.count()
res3: Long = 23
```

RDDs



Transformação	Descrição
<i>map(func)</i>	Aplica a função <i>func</i> a cada elemento do RDD e retorna um novo RDD como resultado.
<i>filter</i>	Aplica uma função predicado a cada elemento do RDD e retorna um RDD com elementos que satisfizeram a condição do predicado.
<i>distinct</i>	Retorna um novo RDD sem os elementos duplicados.
<i>union</i>	Retorna um novo RDD formado pela união de dois outros.
<i>intersection</i>	Retorna um novo RDD apenas com os elementos que se repetem em dois determinados RDDs.
<i>cartesian</i>	Retorna um novo RDD com o produto cartesiano entre dois determinados RDDs.
<i>reduceByKey</i>	Executa uma operação de redução para cada valor que compartilha a mesma chave.

RDDs

- Transformações são lazy, ou seja, não são computadas imediatamente.
- O processamento somente será realizado quando o Spark encontrar uma action para ser executada.
- Spark armazena os dados da chamada à transformação para execução posterior.

RDDs

```
scala> val entrada = sc.parallelize(List(1, 2, 4, 5, 6, 7, 8, 3, 0))
entrada: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[5] at parallelize at <console>:24

scala> val resultado = entrada.map(x => x * x)
resultado: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[6] at map at <console>:26

scala> resultado.count()
res10: Long = 9

scala> resultado.take(9).foreach(println)
1
4
16
25
36
49
64
9
0

scala> █
```

- **Ações (Actions).**
- Actions são operações que retornam um valor final para o Program Driver ou que armazena os resultados no sistema de armazenamento (Ex.: HDFS).
- Actions forçam a avaliação dos transformations.
- Exemplo:

RDDs

```
scala> println("A palavra cluster aparece " + linhasCluster.count() + " vezes no arquivo README.md ")
A palavra cluster aparece 3 vezes no arquivo README.md

scala> println("A palavra Spark aparece " + linhasSpark.count() + " vezes no arquivo README.md ")
A palavra Spark aparece 20 vezes no arquivo README.md

scala> println("Aqui estão 10 exemplos")
Aqui estão 10 exemplos

scala> linhasSpark.take(10).foreach(println)
# Apache Spark
Spark is a fast and general cluster computing system for Big Data. It provides
rich set of higher-level tools including Spark SQL for SQL and DataFrames,
and Spark Streaming for stream processing.
You can find the latest Spark documentation, including a programming
## Building Spark
Spark is built using [Apache Maven](http://maven.apache.org/).
To build Spark and its example programs, run:
You can build Spark using more than one thread by using the -T option with Maven, see ["Parallel builds in Maven
["Building Spark"]](http://spark.apache.org/docs/latest/building-spark.html).

scala>
```

- Usamos a action take para recuperar alguns elementos de um RDD no programa driver.
- Realizamos iterações sobre o RDD para exibir as informações.
- RDDs têm a action collect() para recuperar o RDD inteiro.
- O conteúdo de um RDD pode ser salvo em disco com a action saveAsTextFile().

Transformação	Descrição
<code>reduce(func)</code>	Agrega os elementos do RDD usando uma função <i>func</i> como parâmetro.
<code>collect()</code>	Retorna todos os elementos do RDD em formato de <i>array</i> no <i>Driver Program</i> .
<code>count()</code>	Retorna o número de elementos do RDD.
<code>first()</code>	Retorna o primeiro elemento do RDD (Similar à <code>take(1)</code>).
<code>take(n)</code>	Retorna um <i>array</i> com os primeiros <i>n</i> elementos do RDD.
<code>foreach(func)</code>	Executa uma função <i>func</i> para cada elemento do RDD.

Conclusão

- RDDs.
- Transformations.
- Actions.
- Exemplos.



Próxima aula

- Transformações e ações comuns.



Aula 6.4. Transformações e ações comuns

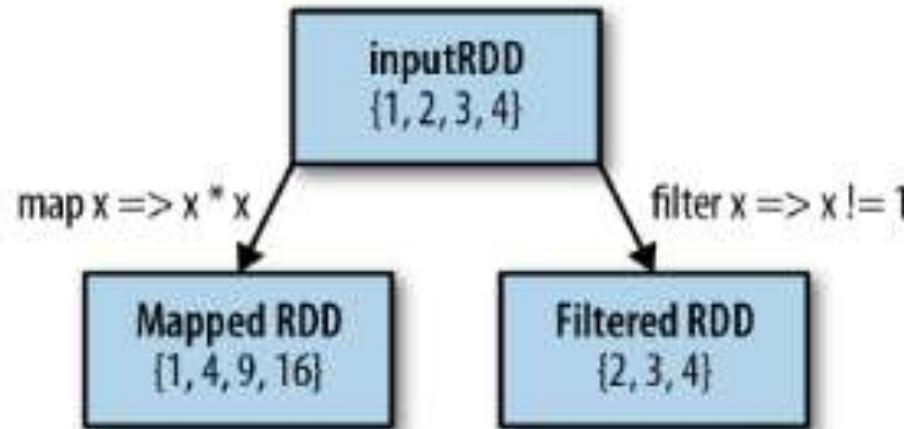


Nesta aula

- ❑ Transformações e ações comuns.

- As duas transformações mais comuns são:
 - Map.
 - Filter.
- `map()` aplica uma função a cada elemento do RDD. Com o resultado da função gera um novo RDD.
- `filter()` aplica uma função e retorna um novo RDD apenas com os elementos que passaram no filtro.

Transformações e ações comuns



RDDs mapeados e filtrados.

Transformações e ações comuns

```
scala> val numeros = sc.parallelize(List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12))
numeros: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[3] at parallelize at <console>:24

scala> val resultado = numeros.map(x => x*(-1))
resultado: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[4] at map at <console>:26

scala> println(resultado.collect().mkString(","))
-1,-2,-3,-4,-5,-6,-7,-8,-9,-10,-11,-12

scala> ■
```

```
scala> val numeros = sc.parallelize(List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12))
numeros: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[5] at parallelize at <console>:24

scala> val resultado = numeros.filter(x => x>10)
resultado: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[6] at filter at <console>:26

scala> println(resultado.collect().mkString(","))
11,12

scala> ■
```

- Map() pode ser usado para uma grande quantidade de funções.
- As vezes queremos produzir vários elementos para um elemento de entrada.
- A operação para fazer isso é o flatmap().
- Assim como em map() a função que fornecemos para flatMap() é executada individualmente para cada elemento do RDD de entrada.

- Ao invés de retornar um simples elemento, flatMap() retorna um iterator com os valores de retorno.
- Assim é montado um RDD com todos os elementos.

Transformações e ações comuns

```
scala> val nomes = sc.parallelize(List("João Silva", "Maria Costa", "Jorge Sa", "Martha Gouveia", "Jardel Souza"))
nomes: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[18] at parallelize at <console>:24

scala> val nomes_map = nomes.map(linha => linha.split(" "))
nomes_map: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[19] at map at <console>:26

scala> nomes_map.collect()
res25: Array[Array[String]] = Array(Array(João, Silva), Array(Maria, Costa), Array(Jorge, Sa), Array(Martha, Gouveia), Array(Jardel, Souza))

scala> nomes_map.first()
res26: Array[String] = Array(João, Silva)

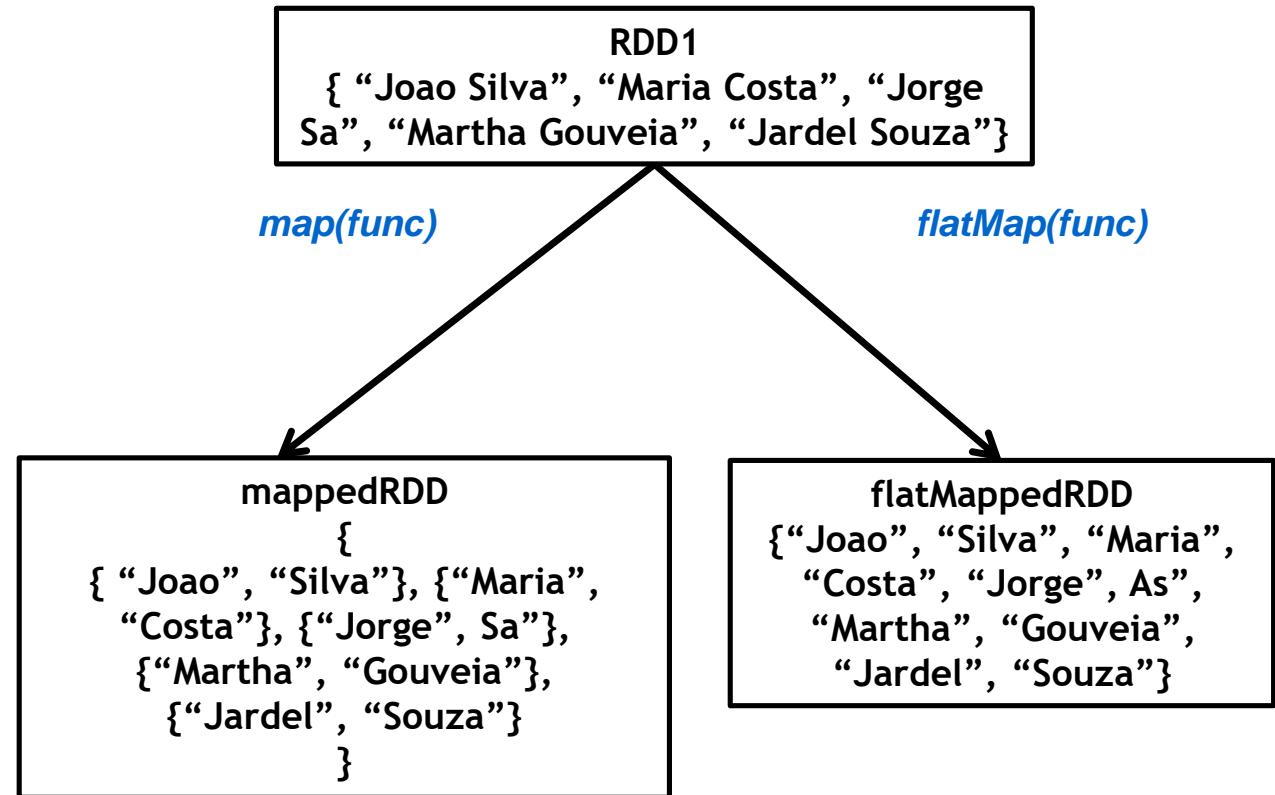
scala> val nomes_flatmap = nomes.flatMap(linha => linha.split(" "))
nomes_flatmap: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[20] at flatMap at <console>:26

scala> nomes_flatmap.collect()
res27: Array[String] = Array(João, Silva, Maria, Costa, Jorge, Sa, Martha, Gouveia, Jardel, Souza)

scala> nomes_flatmap.first()
res28: String = João

scala> nomes_flatmap.take(10).foreach(println)
João
Silva
Maria
Costa
Jorge
Sa
Martha
Gouveia
Jardel
Souza
```

Transformações e ações comuns



- Reduce é uma das ações mais tradicionais no Spark.
- Atua sobre dois elementos do RDD e retorna um novo elemento.
- Exemplos:

```
scala> val numeros = sc.parallelize(List(1, 2, 3, 4, 5))
numeros: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[35] at parallelize at <console>:24

scala> val aux = numeros.reduce((x, y) => x + y)
aux: Int = 15

scala> val aux = numeros.reduce((x, y) => x * y)
aux: Int = 120

scala>
```

- Outras ações:

- countByValue() – número de vezes que cada elemento aparece no RDD.
- top(num) – retorna os maiores elementos do RDD.

```
scala> val numeros = sc.parallelize(List(3, 2, 3, 4, 5, 7, 8, 6, 4, 3, 2, 1, 0, 2, 1, 2, 7, 8, 0, 0, 2, 1))
numeros: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[45] at parallelize at <console>:24

scala> numeros.countByValue()
res36: scala.collection.Map[Int,Long] = Map(0 -> 3, 5 -> 1, 1 -> 3, 6 -> 1, 2 -> 5, 7 -> 2, 3 -> 3, 8 -> 2, 4 -> 2)

scala> numeros.top(1)
res37: Array[Int] = Array(8)
```

Conclusão

- Outras transformations e actions.



Próxima aula

- Persistência (caching).



Aula 6.5. Persistência (caching)



Nesta aula

- Persistência.
- Caching.

- As vezes o Spark não executa imediatamente uma transformação.
- Um RDD pode ser usado diversas vezes.
- Se usamos um RDD repetidas vezes, o Spark irá recomputar o RDD e suas dependências cada vez que aplicamos uma ação no RDD.
- Isso pode ser muito custoso para algoritmos iterativos.
- Essa classe de algoritmo reúsa os dados muitas vezes.

- Para evitar computar o RDD diversas vezes, podemos pedir ao Spark para persistir os dados.
- Com isso cada nó irá persistir suas partições.
- Se um desses nós falha, o Spark recupera os dados que estava armazenando.
- Os dados podem ser replicados para múltiplos nós.

Persistência (caching)

- O Spark tem múltiplos níveis de persistência.
- Deve-se escolher um deles baseado no objetivo da aplicação.

Nível	Espaço usado	Tempo de CPU	Em memória	Em disco
MEMORY_ONLY	Alto	Baixo	Sim	Não
MEMORY_AND_DISK	Alto	Médio	Algum	Algum
DISK_ONLY	Baixo	Alto	Não	Sim

Persistência (caching)

- Perceba que persist() foi chamado antes da primeira ação (count()).

```
scala> import org.apache.spark.storage.StorageLevel
import org.apache.spark.storage.StorageLevel

scala> val nums = sc.parallelize(List(1, 2, 3, 4, 5, 6, 7, 8))
nums: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[6] at parallelize at <console>:28

scala> val resultado = nums.map(x => x*x)
resultado: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[7] at map at <console>:30

scala> resultado.persist(StorageLevel.DISK_ONLY)
res6: resultado.type = MapPartitionsRDD[7] at map at <console>:30

scala> println(resultado.count())
8

scala> println(resultado.collect().mkString(","))
1,4,9,16,25,36,49,64
```

- Cache com mais dados que a memória suporta → Política LRU - (Menos Recentemente Usado).
- MEMORY_ONLY, o Spark vai reprocessar o RDD quando precisar dos dados novamente.
- Se estiver usando MEMORY_AND_DISK os dados serão despejados no HD.

- Mesmo com dados grandes o programa não irá parar caso falte memória.
- Cache não deve ser uma preocupação do desenvolvedor.
- Cache cheia pode fazer com que o programa reprocesse ou despeje dados úteis em disco.
- RDD possui o método `unpersist()`, que permite remover manualmente dados do cache.

Conclusão

- Persistência.
- Cache.
- Níveis de persistência do Spark.

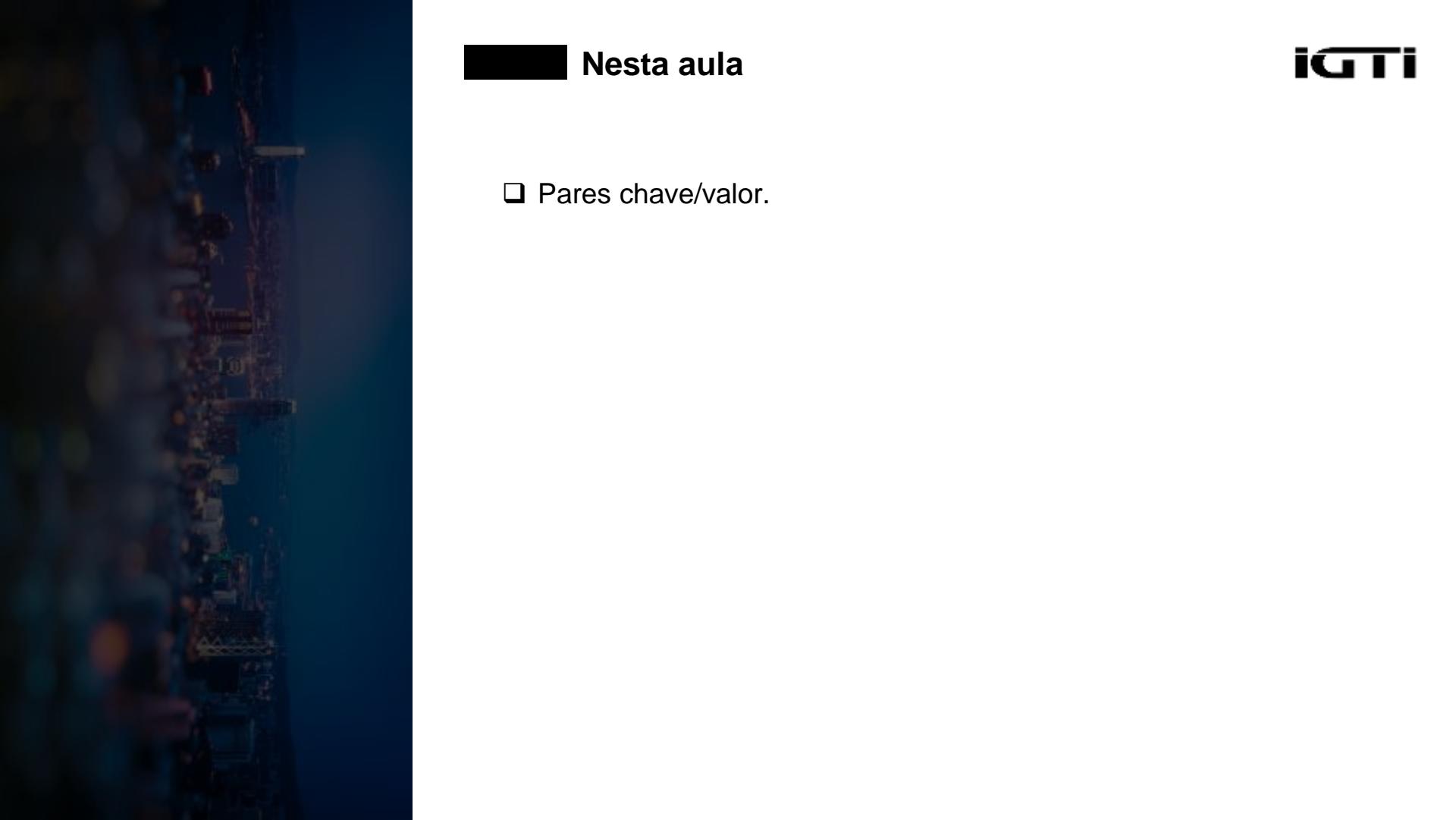


Próxima aula

- ❑ Pares chave/valor.



Aula 6.6. Pares chave/valor



Nesta aula

- Pares chave/valor.

- Pares chave/valor são comumente utilizados para realizar agregações.
- Frequentemente utilizados para ETL (Extract, transform and load).
- Agrupamento de dados (dados com a mesma chave).

- RDDs que possuem pares chave/valor são chamados de Pair RDDs.
- Existem muitas formas de criar Pair RDDS.
- Usando os próprios dados de entrada, podemos definir a chave e o valor.

Pares chave/valor

```
scala> val cliente_Vendas = sc.parallelize(List("1 500", "2 700", "1 300", "3 1000", "2 200", "1 800", "4 650"))
cliente_Vendas: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[63] at parallelize at <console>:28

scala> cliente_Vendas.count()
res56: Long = 7

scala> cliente_Vendas.collect()
res57: Array[String] = Array(1 500, 2 700, 1 300, 3 1000, 2 200, 1 800, 4 650)

scala> val pares = cliente_Vendas.map(x => (x.split(" ")(0), x))
pares: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[64] at map at <console>:30

scala> pares.count()
res58: Long = 7

scala> pares.collect()
res59: Array[(String, String)] = Array((1,1 500), (2,2 700), (1,1 300), (3,3 1000), (2,2 200), (1,1 800), (4,4 650))

scala> val pares = cliente_Vendas.map(x => (x.split(" ")(0), x.split(" ")(1)))
pares: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[65] at map at <console>:30

scala> pares.collect()
res60: Array[(String, String)] = Array((1,500), (2,700), (1,300), (3,1000), (2,200), (1,800), (4,650))
```

Pares chave/valor

- **Transformações em Pair RDDs.**
- `reduceByKey(func)` – Combina valores com a mesma chave.

```
scala> val cliente_Vendas = sc.parallelize(Array(("Abel", 500), ("Bel", 700), ("Abel", 300), ("Cris", 1000), ("Bel", 200), ("Abel", 800), ("Dan", 650 )))
cliente_Vendas: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[74] at parallelize at <console>:28

scala> val aux = cliente_Vendas.reduceByKey((x, y) => (x+y))
aux: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[75] at reduceByKey at <console>:30

scala> aux.collect()
res67: Array[(String, Int)] = Array((Abel,1600), (Cris,1000), (Dan,650), (Bel,900))
```

Pares chave/valor

- **Transformações em Pair RDDs.**
- `groupByKey()` – Agrupa valores com a mesma chave.

```
scala> val cliente_Vendas = sc.parallelize(Array( ("Abel", 500), ("Bel", 700), ("Abel", 300), ("Cris", 1000), ("Bel", 200), ("Abel", 800), ("Dan", 650) ))
cliente_Vendas: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[76] at parallelize at <console>:28

scala> val aux = cliente_Vendas.groupByKey()
aux: org.apache.spark.rdd.RDD[(String, Iterable[Int])] = ShuffledRDD[77] at groupByKey at <console>:36

scala> aux.collect()
res68: Array[(String, Iterable[Int])] = Array((Abel,CompactBuffer(500, 300, 800)), (Cris,CompactBuffer(1000)), (Dan,CompactBuffer(650)), (Bel,CompactBuffer(700, 200)))
```

- **Transformações em Pair RDDs.**
- `mapValues(func)` – Aplica uma função para cada valor de um Pair RDD, sem mudar o valor da chave.

```
scala> val cliente_Vendas = sc.parallelize(Array( ("Abel", 500), ("Bel", 700), ("Abel", 300), ("Cris", 1000)))
cliente_Vendas: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[79] at parallelize at <console>:28

scala> val aux = cliente_Vendas.mapValues(x => x + 10)
aux: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[80] at mapValues at <console>:30

scala> aux.collect()
res69: Array[(String, Int)] = Array((Abel,510), (Bel,710), (Abel,310), (Cris,1010))

scala> val aux = cliente_Vendas.mapValues(x => x * -1)
aux: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[81] at mapValues at <console>:30

scala> aux.collect()
res70: Array[(String, Int)] = Array((Abel,-500), (Bel,-700), (Abel,-300), (Cris,-1000))
```

Pares chave/valor

- **Transformações em Pair RDDs.**
- Keys – Retorna um RDD com apenas as chaves.

```
scala> val cliente_Vendas = sc.parallelize(Array( ("Abel", 500), ("Bel", 700), ("Abel", 300), ("Cris", 1000)))
cliente_Vendas: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[84] at parallelize at <console>:28

scala> val chaves = cliente_Vendas.keys
chaves: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[85] at keys at <console>:30

scala> chaves.collect()
res72: Array[String] = Array(Abel, Bel, Abel, Cris)
```

- Values – Retorna um RDD com apenas os valores.

```
scala> val cliente_Vendas = sc.parallelize(Array( ("Abel", 500), ("Bel", 700), ("Abel", 300), ("Cris", 1000)))
cliente_Vendas: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[86] at parallelize at <console>:28

scala> val valores = cliente_Vendas.values
valores: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[87] at values at <console>:30

scala> valores.collect()
res73: Array[Int] = Array(500, 700, 300, 1000)
```

Pares chave/valor

- **Transformações em em dois Pair RDDs.**
- **subtractByKey** – Remove elementos com a chave presente em outro RDD.

```
scala> val cliente_Vendas = sc.parallelize(Array( ("Abel", 500), ("Bel", 700), ("Abel", 300), ("Cris", 1000)))
cliente_Vendas: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[92] at parallelize at <console>:28

scala> val cliente_Debitos = sc.parallelize(Array( ("Bel", 70000), ("Jota", 10000)))
cliente_Debitos: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[93] at parallelize at <console>:28

scala> val autorizados = cliente_Vendas.subtractByKey(cliente_Debitos)
autorizados: org.apache.spark.rdd.RDD[(String, Int)] = SubtractedRDD[94] at subtractByKey at <console>:32

scala> autorizados.collect()
res75: Array[(String, Int)] = Array((Abel,500), (Abel,300), (Cris,1000))
```

- Transformações em em dois Pair RDDs.
- join – Executa um inner join entre dois RDDs.

```
scala> val cliente_Vendas = sc.parallelize(Array( ("Abel", 500), ("Bel", 700), ("Abel", 300), ("Cris", 1000)))
cliente_Vendas: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[106] at parallelize at <console>:28

scala> val cliente_Orcamentos = sc.parallelize(Array( ("Cris", 4500), ("Gil", 1300)))
cliente_Orcamentos: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[107] at parallelize at <console>:28

scala> val cliente_Vendas_Orcamentos = cliente_Vendas.join(cliente_Orcamentos)
cliente_Vendas_Orcamentos: org.apache.spark.rdd.RDD[(String, (Int, Int))] = MapPartitionsRDD[110] at join at <console>:32

scala> cliente_Vendas_Orcamentos.collect()
res81: Array[(String, (Int, Int))] = Array((Cris,(1000,4500)))
```

Pares chave/valor

- **Transformações em em dois Pair RDDs.**
- rightOuterJoin – Executa uma junção entre dois RDDs, onde a chave precisa estar presente no RDD “da direita”.

```
scala> val cliente_Vendas = sc.parallelize(Array( ("Abel", 500), ("Bel", 700), ("Abel", 300), ("Cris", 1000)))
cliente_Vendas: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[114] at parallelize at <console>:28

scala> val cliente_Orcamentos = sc.parallelize(Array( ("Cris", 4500), ("Gil", 1300)))
cliente_Orcamentos: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[115] at parallelize at <console>:28

scala> val aux = cliente_Vendas.rightOuterJoin(cliente_Orcamentos)
aux: org.apache.spark.rdd.RDD[(String, (Option[Int], Int))] = MapPartitionsRDD[118] at rightOuterJoin at <console>:32

scala> aux.collect()
res83: Array[(String, (Option[Int], Int))] = Array((Cris,(Some(1000),4500)), (Gil,(None,1300)))
```

Pares chave/valor

- **Transformações em dois Pair RDDs.**
- **leftOuterJoin** – Executa um join entre dois RDDs, onde a chave precisa estar presente no RDD “da esquerda”.

```
scala> val cliente_Vendas = sc.parallelize(Array( ("Abel", 500), ("Bel", 700), ("Abel", 300), ("Cris", 1000)))
cliente_Vendas: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[119] at parallelize at <console>:28
scala> val cliente_Orcamentos = sc.parallelize(Array( ("Cris", 4500), ("Gll", 1300)))
cliente_Orcamentos: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[120] at parallelize at <console>:28
scala> val aux  = cliente_Vendas.leftOuterJoin(cliente_Orcamentos)
aux: org.apache.spark.rdd.RDD[(String, (Int, Option[Int]))]] = MapPartitionsRDD[123] at leftOuterJoin at <console>:32
scala> aux.collect()
res84: Array[(String, (Int, Option[Int]))]] = Array((Abel,(500,None)), (Abel,(300,None)), (Cris,(1000,Some(4500))), (Bel,(700,None)))
```

Conclusão

- Pares chave/valor.
- Transformações em pares chave/valor.

Próxima aula

- ❑ Agregações / Agrupamento / Ordenação.



Aula 6.7. Agregações / Agrupamento / Ordenação

Nesta aula

- Agregações.
- Agrupamento.
- Ordenação.

- Em datasets com formato chave/valor é comum realizar agregações em elementos que possuem as mesmas chaves.
- `reduceByKey()` é similar a `reduce()`.
- Ambos têm uma função e combinam valores.
- `reduceByKey()` executa diversas operações `reduce()` em paralelo, uma para cada chave no RDD.

- Wordcount (contador de palavras).

```
scala> val arquivo = sc.textFile("/usr/local/spark/README.md")
arquivo: org.apache.spark.rdd.RDD[String] = /usr/local/spark/README.md MapPartitionsRDD[35] at textFile at <console>:28

scala> val palavras = arquivo.flatMap(x => x.split(" "))
palavras: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[36] at flatMap at <console>:30

scala> arquivo.count()
res35: Long = 104

scala> palavras.count()
res36: Long = 567

scala> val resultado = palavras.map(x => (x, 1)).reduceByKey((x, y) => x + y)
resultado: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[38] at reduceByKey at <console>:32

scala> resultado.take(12).foreach(println)
(package,1)
(this,1)
(Because,1)
(Python,2)
(page](http://spark.apache.org/docs/latest/building-spark.html#specifying-the-hadoop-version),1)
(cluster.,1)
(its,1)
([run,1)
(general,3)
(have,1)
(pre-built,1)
```

- Actions disponíveis em Pair RDDs.
- Assim como as transformações, todas as actions tradicionais estão disponíveis para os Pairs RDDs.

Action	Descrição
countByKey()	Conta o número de elementos de cada chave.
lookup(key)	Retorna todos os valores associados com uma chave fornecida.

- Exemplos:

```
scala> val cliente_Vendas = sc.parallelize(Array( ("Abel", 500), ("Bel", 700), ("Abel", 300), ("Cris", 1000), ("Bel", 200), ("Abel", 800), ("Dan", 650), ("Dan", 987) ))
cliente_Vendas: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[4] at parallelize at <console>:28

scala> cliente_Vendas.countByKey()
res40: scala.collection.Map[String,Long] = Map(abel -> 3, Cris -> 1, Dan -> 2, Bel -> 2)

scala> cliente_Vendas.lookup("Dan")
res47: Seq[Int] = WrappedArray(650, 987)
```

- Ordenando um RDD.
- Podemos usar sortByKey().

```
scala> val clientes = sc.parallelize(Array(("Abel", 50), ("Zul", 7), ("Yul", 3), ("Cris", 10), ("Joca", 2), ("Sil", 80), ("Paul", 6)))
clientes: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[68] at parallelize at <console>:28

scala> clientes.collect()
res56: Array[(String, Int)] = Array((Abel,50), (Zul,7), (Yul,3), (Cris,10), (Joca,2), (Sil,80), (Paul,6))

scala> val clientes_ord = clientes.sortByKey()
clientes_ord: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[71] at sortByKey at <console>:30

scala> clientes_ord.collect()
res57: Array[(String, Int)] = Array((Abel,50), (Cris,10), (Joca,2), (Paul,6), (Sil,80), (Yul,3), (Zul,7))

scala> val clientes_ord = clientes.sortByKey(false)
clientes_ord: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[74] at sortByKey at <console>:30

scala> clientes_ord.collect()
res58: Array[(String, Int)] = Array((Zul,7), (Yul,3), (Sil,80), (Paul,6), (Joca,2), (Cris,10), (Abel,50))
```

- Agrupando dois RDDs.
- Podemos usar cogroup().

```
scala> val rdd1 = sc.parallelize(Array((1, 2), (3, 4), (3, 6), (4, 2)))
rdd1: org.apache.spark.rdd.RDD[(Int, Int)] = ParallelCollectionRDD[79] at parallelize at <console>:28

scala> val rdd2 = sc.parallelize(Array((3,9), (4,7)))
rdd2: org.apache.spark.rdd.RDD[(Int, Int)] = ParallelCollectionRDD[80] at parallelize at <console>:28

scala> val rdd3 = rdd1.cogroup(rdd2)
rdd3: org.apache.spark.rdd.RDD[(Int, (Iterable[Int], Iterable[Int]))] = MapPartitionsRDD[82] at cogroup at <console>:32

scala> rdd3.take(3).foreach(println)
(4,(CompactBuffer(2),CompactBuffer(7)))
(1,(CompactBuffer(2),CompactBuffer()))
(3,(CompactBuffer(4, 6),CompactBuffer(9)))
```

- Agrupamento.
- GroupBy – Agrupando pela primeira letra do nome.

```
scala> val nomes = sc.parallelize(Array("Jose", "Jonas", "Tina", "Thomas", "Joca", "Cora", "Chris", "Jack", "David"))
nomes: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[86] at parallelize at <console>:28

scala> val aux = nomes.groupBy(palavra => palavra.charAt(0))
aux: org.apache.spark.rdd.RDD[(Char, Iterable[String])] = ShuffledRDD[88] at groupBy at <console>:38

scala> aux.count()
res68: Long = 4

scala> aux.take(4).foreach(println)
(T,CompactBuffer(Tina, Thomas))
(D,CompactBuffer(David))
(J,CompactBuffer(Jose, Jonas, Joca, Jack))
(C,CompactBuffer(Cora, Chris))
```

Conclusão

- Agregação.
- Ordenação.
- Agrupamento.
- Actions em RDDs.



Próxima aula

- Carregando e salvando dados.



Aula 6.8. Carregando e salvando dados



Nesta aula

- ❑ Tipos de arquivos.
- ❑ HDFS:
 - Carregando arquivos.
 - Salvando arquivos.

- Spark suporta uma ampla variedade de fontes de entrada e saída.
- Principalmente porque ele foi construído no ecossistema Hadoop.
- Spark pode acessar dados por meio das interfaces InputFormat e OutputFormat usadas pelo Hadoop.
- Vários formatos de arquivos e vários sistemas de armazenamento.
- Arquivos texto, JSON, CSV etc.
- S3, HDFS, Cassandra, HBase etc.

- Arquivos texto é o formato mais simples.
- Cada linha é um elemento no RDD.
- Podemos carregar vários arquivos em Pair RDDs. Chave será o nome e o valor o conteúdo do arquivo.
- Carregar um arquivo texto: `sc.textFile("...")`.
- Salvar um arquivo texto: `rdd.saveAsTextFile("...")`.

Carregando e salvando dados

- Arquivos JSON:
- clientes.json.

```
Terminal Arquivo Editar Ver Pesquisar Terminal Ajuda
[{"prim_nome": "Jose", "ult_nome": "Braga", "media": 5000, "endereco": {"rua": "Braga Silve 47", "cidade": "Belo Horizonte", "uf": "MG"}}, {"prim_nome": "Martha", "ult_nome": "Souza", "media": 8000, "endereco": {"rua": "Amazonas 368", "cidade": "Sao Paulo", "uf": "SP"}}, {"prim_nome": "Anisia", "ult_nome": "Flores", "media": 1200, "endereco": {"rua": "Antonio Carlos 317", "cidade": "Rio de Janeiro", "uf": "RJ"}}]
```

- Arquivos JSON:

```
scala> val clientes = spark.read.json("clientes.json")
clientes: org.apache.spark.sql.DataFrame = [endereco: struct<cidade: string, rua: string ... 1 more field>,
scala> clientes.count()
res19: Long = 3

scala> clientes.take(3).foreach(println)
[[Belo Horizonte,Braga Silva 47,MG],5000,Jose,Braga]
[[Sao Paulo,Amazonas 368,SP],8000,Martha,Souza]
[[Rio de Janeiro,Antonio Carlos 317,RJ],1300,Anisio,Flores]

scala> clientes.createOrReplaceTempView("cli")

scala> spark.sql("select * from cli").show
+-----+-----+-----+
|     endereco|media|prim_nome|ult_nome|
+-----+-----+-----+
|[Belo Horizonte,B...| 5000|    Jose|   Braga|
|[Sao Paulo,Amazon...| 8000|   Martha|   Souza|
|[Rio de Janeiro,A...| 1300|  Anisio|   Flores|
+-----+-----+-----+
```

Carregando e salvando dados

- Arquivos JSON:

```
scala> spark.sql("select prim_nome, ult_nome, media from cli where prim_nome = 'Jose'").show
+-----+-----+-----+
|prim_nome|ult_nome|media|
+-----+-----+-----+
|      Jose|    Braga|  5000|
+-----+-----+-----+


scala> spark.sql("select prim_nome, ult_nome, media from cli").show
+-----+-----+-----+
|prim_nome|ult_nome|media|
+-----+-----+-----+
|      Jose|    Braga|  5000|
|   Martha|   Souza|  8000|
| Anisio|   Flores| 1300|
+-----+-----+-----+
```

- Arquivos JSON:

```
scala> spark.sql("select prim_nome, ult_nome, media from cli where prim_nome = 'Jose'").show
+-----+-----+-----+
|prim_nome|ult_nome|media|
+-----+-----+-----+
|      Jose|    Braga|  5000|
+-----+-----+-----+


scala> spark.sql("select sum(media) from cli").show
+-----+
|sum(CAST(media AS DOUBLE))|
+-----+
|          14300.0|
+-----+


scala> spark.sql("select sum(media)/3 from cli").show
+-----+
|(sum(CAST(media AS DOUBLE)) / CAST(3 AS DOUBLE))|
+-----+
|           4766.666666666667|
+-----+
```

■ Carregando e salvando dados

- Carregando um RDD a partir de um arquivo do HDFS.

The screenshot shows a web browser window with the URL `localhost:50070/explorer.html#entrada`. The page title is "Browse Directory". At the top, there is a green navigation bar with tabs: Hadoop, Overview, Datanodes, Graphviz, Status Progress, and Utilities. Below the navigation bar, the main content area displays a table titled "Browse Directory" with one row of data. The table has columns: Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name. The single entry is: "rw-r--r--" (Owner), "superuser" (Group), "2.01 KB" (Size), "13/04/2017 00:08:45" (Last Modified), "1" (Replication), "128 MB" (Block Size), and "arquivodigita.txt" (Name). At the bottom of the table, it says "Hadoop, 2016."

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	rootuser	supergroup	2.01 KB	13/04/2017 00:08:45	1	128 MB	arquivodigita.txt

Carregando e salvando dados

igti

- Carregando um RDD a partir de um arquivo do HDFS.

Carregando e salvando dados

- Salvando os dados de um RDD no HDFS.
- `saveAsTextFile(...)`.

```
scala> val vendas = sc.textFile("hdfs://localhost:54310/entrada/arquivoBigData.txt")
vendas: org.apache.spark.rdd.RDD[String] = hdfs://localhost:54310/entrada/arquivoBigData.txt MapPartitionsRDD[19]

scala> val codigo_cliente = vendas.map(linha => linha.substring(58, 61))
codigo_cliente: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[20] at map at <console>:26

scala> codigo_cliente.count()
res14: Long = 20

scala> codigo_cliente.take(4).foreach(println)
081
007
001
009

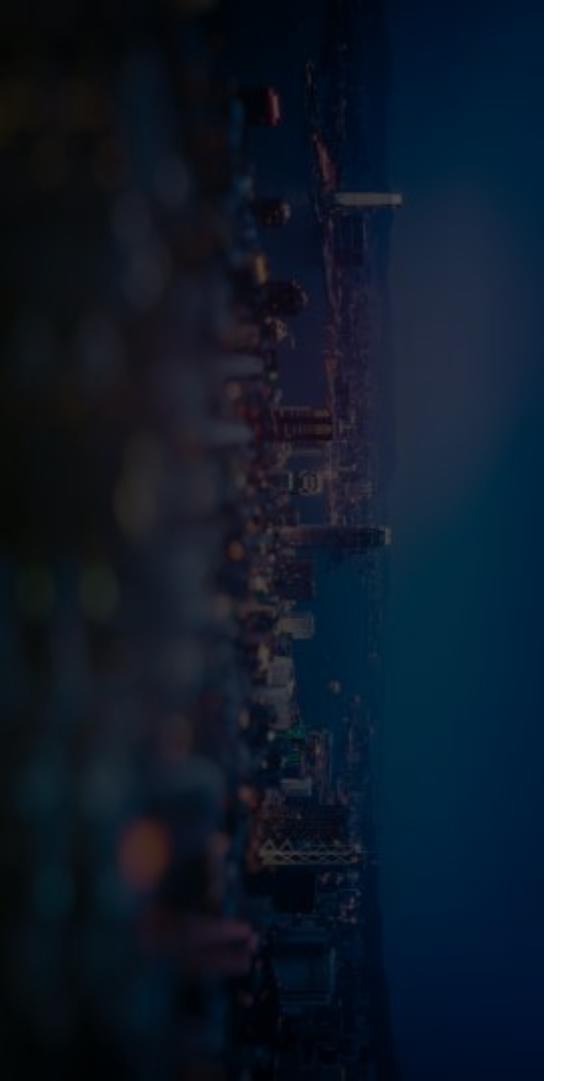
scala> codigo_cliente.saveAsTextFile("hdfs://localhost:54310/entrada/clientes.txt")
```

■ Carregando e salvando dados

- Salvando os dados de um RDD no HDFS.
- `saveAsTextFile(...)`.

The screenshot shows a web-based Hadoop file browser interface. At the top, there is a header bar with the URL "localhost:50070/explorer.html#/entrada" and a search bar labeled "Pesquisar". Below the header is a green navigation bar with links: Hadoop, Overview, Datanodes, Snapshot, Startup Progress, and Utilities. The main content area is titled "Browse Directory" and displays the contents of the "/entrada" directory. A table lists two files: "arquivandoDados.txt" and "clientes.txt". The table columns are: Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name. The "arquivandoDados.txt" file has a size of 2.01 KB and was last modified on 13/04/2017 at 00:08:45. The "clientes.txt" file has a size of 0 B and was last modified on 13/04/2017 at 00:54:03. At the bottom of the page, there is a footer note: "Hadoop, 2016."

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	supergroup	2.01 KB	13/04/2017 00:08:45	1	128 MB	arquivandoDados.txt
drwxr-xr-x	hduser	supergroup	0 B	13/04/2017 00:54:03	0	0 B	clientes.txt



Conclusão

- Tipos de arquivos manipulados pelo Spark.
- Armazenamento no HDFS.



Próxima aula

- ❑ Executando em um cluster.



Aula 6.9. Executando em um cluster



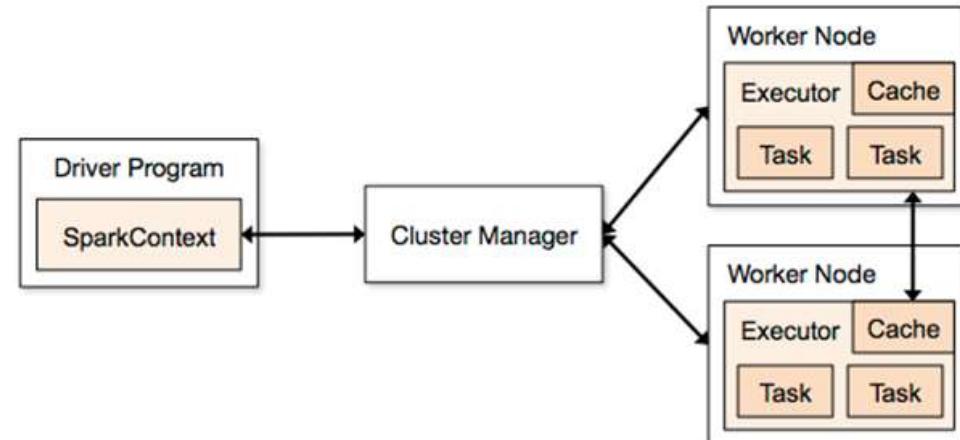
Nesta aula

- Execução do Spark em cluster.
- Arquitetura.

- Até agora usamos o Spark em modo local.
- Uma possibilidade de construir aplicações no Spark é a escalabilidade.
- Uma aplicação funcionando em “local mode” funciona também em um cluster.
- Desenvolvedor pode prototipar uma aplicação com um dataset pequeno.
- Testar toda a aplicação e depois enviar ao cluster.

Executando em um cluster

- Arquitetura Spark em modo distribuído.



Executando em um cluster

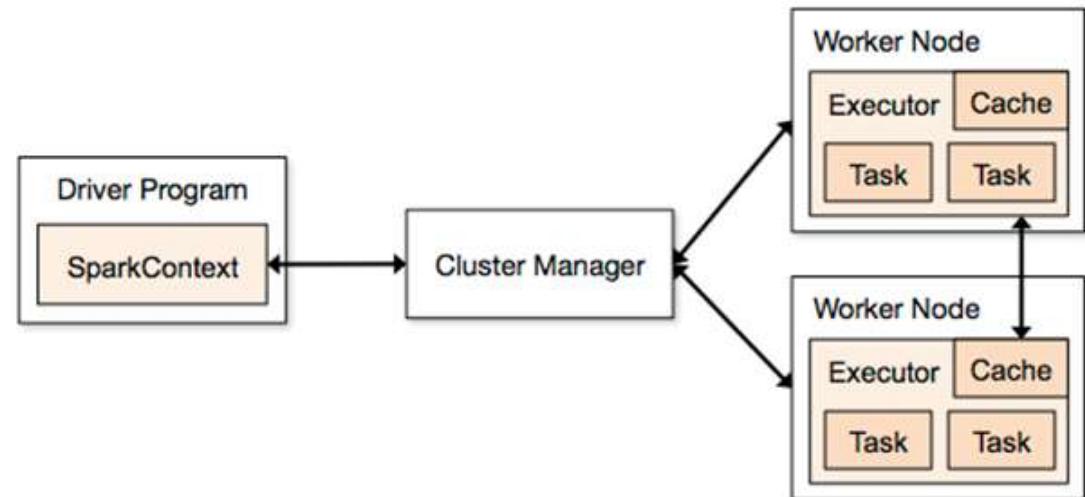
- Cluster Manager.
- Spark depende de um gerenciador de cluster.
- Cluster Manager é um componente plugável ao Spark.
- Isso permite que o Spark utilize diferentes gerenciadores de cluster:
 - Yarn.
 - Mesos.



- Iniciando um programa:
 - Não importa qual cluster manager que estamos utilizando.
 - O Spark fornece um script para submeter os programas, chamado spark-submit.
 - spark-submit pode se conectar a diferentes gerenciadores de cluster.

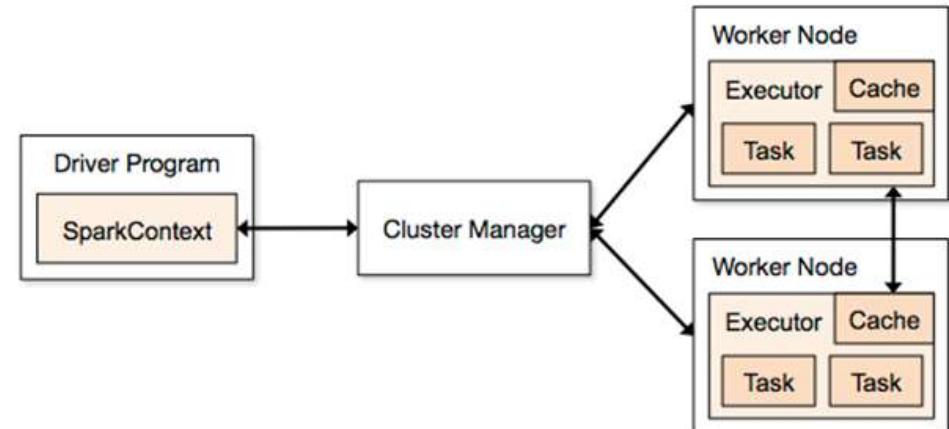
Executando em um cluster

1. Usuário submete uma aplicação usando spark-submit.
2. spark-submit inicia o driver program invocando o método main().
3. O driver program entra em contato com o cluster manager e solicita recursos para iniciar o programa.



Executando em um cluster

4. O cluster manager inicia os executors para o driver program.
5. Baseado nas transformations e actions, o driver envia trabalhos para os executors em forma de tarefas
6. Tarefas rodam nos executors para computar e salvar dados.
7. Executors retornam os dados com o resultado para o driver program.





Executando em um cluster

- Caminho do spark-submit: {caminho_de_instalação_do_Spark}/bin/spark-submit

Conclusão

- Arquitetura do Spark.
- Lançando um programa Spark.
- spark-submit.



Próxima aula

- Utilizando o spark-submit.



Aula 6.10. Utilizando o spark-submit

Nesta aula

- ❑ Formato do comando de chamada.
- ❑ Parâmetros do spark-submit.

Utilizando o spark-submit

- Quando o spark-submit é chamado sem nenhum parâmetro além do nome do script, ele simplesmente executa um programa localmente.
- Para executar uma aplicação Spark em modo cluster:

```
$ ./bin/spark-submit --class path.to.your.Class --master yarn --deploy-mode cluster [options] <app jar> [app options]
```

Utilizando o spark-submit

- - - class: a classe principal da sua aplicação, se você estiver executando uma aplicação Java ou Scala.
- - - master: indica o cluster manager que iremos conectar nossa aplicação.
 - spark://host:port – conecta ao modo standalone do Spark na porta especificada. Por padrão, o Spark usa a porta 7077.
 - mesos://host:port – conecta ao cluster manager Mesos.
 - yarn: conecta ao Yarn cluster manager.
 - local: executa em modo local com um único núcleo.
 - local[N]: executa e modo local com N núcleos.
 - local[*]: executa e modo local com todos os núcleos.

```
$ ./bin/spark-submit --class path.to.your.Class --master yarn --deploy-mode cluster [options] <app jar> [app options]
```

Utilizando o spark-submit

- - - deploy mode: definir se o modo é client ou cluster.
- <app jar>: seu jar com o programa Java ou Scala.

```
$ ./bin/spark-submit --class path.to.your.Class --master yarn --deploy-mode cluster [options] <app jar> [app options]
```

Utilizando o spark-submit

```
$ ./bin/spark-submit --class org.apache.spark.examples.SparkPi \
--master yarn \
--deploy-mode cluster \
--driver-memory 4g \
--executor-memory 2g \
--executor-cores 1 \
--queue thequeue \
lib/spark-examples*.jar \
10
```

Conclusão

- Utilização do spark-submit.
- Parâmetros.
- Formato do comando.



Próxima aula

- Construindo e executando aplicações Spark.



Aula 6.11. Construindo e executando uma aplicação Spark

Nesta aula

- Construindo uma aplicação.
- Compilar.
- Executar.
- Verificar os resultados.

- SBT é uma ferramenta popular para construir aplicações Scala.
- Construir aplicações Spark com SBT é simples, pois o Spark foi construído com o SBT.
- Primeiro passo: instalar o SBT.

- Instalando o SBT no Ubuntu:

```
echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2EE0EA64E40A89B84B2DF73499E82A75642AC823
sudo apt-get update
sudo apt-get install sbt
```

- Construindo uma aplicação simples.
- Iremos utilizar dois arquivos em um novo diretório.
- build.sbt: dados necessários para a compilação.
- myscript.scala: instruções do programa.

```
hduser@JOAOLINUX:/usr/local/spark/MPRApp$ ls  
build.sbt  myscript.scala  
hduser@JOAOLINUX:/usr/local/spark/MPRApp$ █
```

- **sbt clean.**

- Exclui todos os arquivos gerados anteriormente (diretório target).

```
hduser@JOAOLINUX:/usr/local/spark/MPRApp$ ls
build.sbt myscript.scala
hduser@JOAOLINUX:/usr/local/spark/MPRApp$ sbt clean
Updated file /usr/local/spark/MPRApp/project/build.properties setting sbt.version to: 0.13.15
[warn] Executing in batch mode.
[warn]   For better performance, hit [ENTER] to switch to interactive mode, or
[warn]   consider launching sbt without any commands, or explicitly passing 'shell'
[info] Loading project definition from /usr/local/spark/MPRApp/project
[info] Updating {file:/usr/local/spark/MPRApp/project/}mprapp-build...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Set current project to IGTI MPRAPP (in build file:/usr/local/spark/MPRApp/)
[success] Total time: 0 s, completed 14/04/2017 12:03:24
hduser@JOAOLINUX:/usr/local/spark/MPRApp$ ls
build.sbt myscript.scala project target
hduser@JOAOLINUX:/usr/local/spark/MPRApp$ █
```

- **sbt package.**

- Compila o projeto e cria o arquivo jar.
- Arquivo jar que será enviado ao spark-submit.

```
hduser@JOAOLINUX:/usr/local/spark/MPRApp$ sbt package
[warn] Executing in batch mode.
[warn] For better performance, hit [ENTER] to switch to interactive mode, or
[warn] consider launching sbt without any commands, or explicitly passing 'shell'
[info] Loading project definition from /usr/local/spark/MPRApp/project
[info] Set current project to IGTI MPRAPP (in build file:/usr/local/spark/MPRApp/)
[info] Updating {file:/usr/local/spark/MPRApp/}mprapp...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] downloading https://repo1.maven.org/maven2/org/apache/avro/avro/1.7.7/avro-1.7.7.jar ...
[info]  [SUCCESSFUL ] org.apache.avro#avro;1.7.7!avro.jar (2276ms)
[info] Done updating.
[info] Compiling 1 Scala source to /usr/local/spark/MPRApp/target/scala-2.10/classes...
[info] Packaging /usr/local/spark/MPRApp/target/scala-2.10/igti-mprapp_2.10-0.13.5.jar ...
[info] Done packaging.
[success] Total time: 20 s, completed 14/04/2017 12:09:05
hduser@JOAOLINUX:/usr/local/spark/MPRApp$ cd target
hduser@JOAOLINUX:/usr/local/spark/MPRApp/target$ cd scala-2.10/
hduser@JOAOLINUX:/usr/local/spark/MPRApp/target/scala-2.10$ ls
classes  igti-mprapp_2.10-0.13.5.jar
hduser@JOAOLINUX:/usr/local/spark/MPRApp/target/scala-2.10$ █
```

- **myscript.scala.**

```
package IGTI

import org.apache.spark.{SparkContext, SparkConf}

object MPRApp {
    def main(args: Array[String]) {
        var conf = new SparkConf().setAppName("IGTI MPRApp")
        var sc = new SparkContext(conf)
        val vendas = sc.textFile("hdfs://localhost:54310/entrada/arquivoBigData.txt")
        vendas.saveAsTextFile("hdfs://localhost:54310/entrada/resultado.txt");
    }
}
```

- **build.sbt.**

```
name := "IGTI MPRAPP"  
  
version := "0.13.5"  
  
scalaVersion := "2.10.4"  
  
libraryDependencies ++= Seq("org.apache.spark" %% "spark-core" % "2.1.0" % "provided")
```

Construindo e executando uma aplicação Spark



■ Executando:

```
hadoop@290L3HDX:~/usr/local/march/07spark$ /usr/local/spark/bin/spark-submit --master yarn-cluster --class IGTI.MPRAApp /usr/local/spark/MPRAApp/target/scala-2.10/igti-mpraapp_2.10-0.13.5.jar
Warning: Master yarn-cluster is deprecated since 2.0. Please use master "yarn" with specified deploy mode instead.
17/04/14 13:06:25 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/04/14 13:06:25 WARN util.Utils: Your hostname, 290L3HDX resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
17/04/14 13:06:25 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another address
17/04/14 13:06:26 INFO Client: RMProxy: Connecting to ResourceManager at /0.0.0.0:8882
17/04/14 13:06:29 INFO yarn.Client: Requesting a new application from cluster with 1 NodeManagers
17/04/14 13:06:29 INFO yarn.Client: Verifying our application has not requested more than the maximum memory capability of the cluster (8192 MB per container)
17/04/14 13:06:29 INFO yarn.Client: Will allow 1 AM container, with 8192 MB memory including 384 MB overhead
17/04/14 13:06:29 INFO yarn.Client: Setting up the launch environment for our AM
17/04/14 13:06:29 INFO yarn.Client: Setting up the launch environment for our AM container
17/04/14 13:06:29 INFO yarn.Client: Preparing resources for our AM container
17/04/14 13:06:29 WARN yarn.Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
```

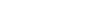
Construindo e executando uma aplicação Spark

IGTI

- Executando:

The screenshot shows the Hadoop ResourceManager UI at the URL `http://jasonmax:8088/cluster`. The title bar says "All Applications". The page features a sidebar with cluster metrics and a main table listing four completed applications.

ID	User	Name	Application Type	Queue	StartTime	FinishTime	Status	FinalStatus	Progress	Tracking URL
application_1492123981765_0009	hduser	IGTI.MPRApp	SPARK	default	Fri Apr 14 16:04:35 -0300 2017	Fri Apr 14 16:05:07 -0300 2017	FINISHED	SUCCEEDED	<div style="width: 100%;"></div>	History
application_1492123981765_0008	hduser	IGTI.MPRApp	SPARK	default	Fri Apr 14 13:54:11 -0300 2017	Fri Apr 14 13:54:32 -0300 2017	FINISHED	SUCCEEDED	<div style="width: 100%;"></div>	History
application_1492123981765_0007	hduser	IGTI.MPRApp	SPARK	default	Fri Apr 14 13:47:27 -0300 2017	Fri Apr 14 13:47:55 -0300 2017	FINISHED	SUCCEEDED	<div style="width: 100%;"></div>	History
application_1492123981765_0006	hduser	IGTI.MPRApp	SPARK	default	Fri Apr 14 13:16:34 -0300 2017	Fri Apr 14 13:17:04 -0300 2017	FINISHED	SUCCEEDED	<div style="width: 100%;"></div>	History

Construindo e executando uma aplicação Spark 

- Outro exemplo (arquivoBigData.txt):

- Outro exemplo:

```
JOAOLINUX:/usr/local/spark/MPRApp
package IGTI

import org.apache.spark.{SparkContext, SparkConf}

object MPRApp {
    def main(args: Array[String]) {
        var conf = new SparkConf().setAppName("IGTI MPRApp")
        var sc = new SparkContext(conf)

        val arquivo = sc.textFile("hdfs://localhost:54310/entrada/arquivoBigData.txt")

        val clientes_vendas = arquivo.map(linha => (linha.substring(58, 61).toInt, linha.substring(76, 84).toDouble))
        val grupo = clientes_vendas.reduceByKey((x, y) => (x+y))
        val grupo_ord = grupo.sortByKey()

        grupo_ord.saveAsTextFile("hdfs://localhost:54310/entrada/resultado");
    }
}
```

Construindo e executando uma aplicação Spark



■ Outro exemplo:

```
hadoop@JOAOLINHA:/usr/local/spark/MRapp$ ./spark/bin/spark-submit --master yarn-cluster --class IGTI.MRApp /usr/local/spark/MRApp/target/scala-2.10/igti-mrapp_2.10-0.13.5.jar
Warning: Master yarn-cluster is deprecated since 2.0. Please use master 'yarn' with specified deploy mode instead.
17/04/14 23:18:43 WARN util.NativeCodeLoader: Unable to load native hadoop library for your platform... using builtin-Java classes where applicable
17/04/14 23:18:43 WARN util.Utils: Your hostname, JOAOLINHA resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s)
17/04/14 23:18:45 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another address
17/04/14 23:18:44 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
17/04/14 23:18:44 INFO yarn.Client: Requesting a new application from cluster with 1 nodeManagers
17/04/14 23:18:44 INFO yarn.Client: Verifying our application has not requested more than the maximum memory capability of the cluster (8192 MB per container)
17/04/14 23:18:44 INFO yarn.Client: Will allocate AM container, with 3488 MB memory including 384 MB overhead
17/04/14 23:18:44 INFO yarn.Client: Setting up container launch context for our AM
17/04/14 23:18:44 INFO yarn.Client: Setting up the launch environment for our AM container
17/04/14 23:18:44 INFO yarn.Client: Preparing resources for our AM container
17/04/14 23:18:46 WARN yarn.Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
17/04/14 23:18:46 INFO yarn.Client: Uploading resource file:/tmp/spark-d74e32b2-8102-478f-8d77-3af58a39643c/_spark_libs_4642941974044264293.zip -> hdfs://localhost:54310/user/hduser/.sparkStaging/application_148222130316_0009/_spark_libs_4642941974044264293.zip
17/04/14 23:18:52 INFO yarn.Client: Uploading resource file:/usr/local/spark/MRApp/target/scala-2.10/igti-mrapp_2.10-0.13.5.jar -> hdfs://localhost:54310/user/hduser/.sparkStaging/application_148222130316_0009/igti-mrapp_2.10-0.13.5.jar
17/04/14 23:18:52 INFO yarn.Client: Uploading resource file:/tmp/spark-d74e32b2-8102-478f-8d77-3af58a39643c/_spark_conf_718003247265523861.zip -> hdfs://localhost:54310/user/hduser/.sparkStaging/application_148222130316_0009/_spark_conf_.zip
17/04/14 23:18:53 INFO spark.SecurityManager: Changing view acls to: hduser
```

- Outro exemplo:

```
17/04/14 23:17:03 INFO yarn.Client:  
  client token: N/A  
  diagnostics: N/A  
  ApplicationMaster host: 10.0.2.15  
  ApplicationMaster RPC port: 0  
  queue: default  
  start time: 1492222613043  
  final status: UNDEFINED  
  tracking URL: http://JOAOLINUX:8088/proxy/application_1492221381316_0009/  
  user: hduser  
17/04/14 23:17:04 INFO yarn.Client: Application report for application_1492221381316_0009 (state: RUNNING)  
17/04/14 23:17:05 INFO yarn.Client: Application report for application_1492221381316_0009 (state: RUNNING)  
17/04/14 23:17:06 INFO yarn.Client: Application report for application_1492221381316_0009 (state: RUNNING)  
17/04/14 23:17:07 INFO yarn.Client: Application report for application_1492221381316_0009 (state: RUNNING)  
17/04/14 23:17:08 INFO yarn.Client: Application report for application_1492221381316_0009 (state: RUNNING)  
17/04/14 23:17:09 INFO yarn.Client: Application report for application_1492221381316_0009 (state: RUNNING)  
17/04/14 23:17:10 INFO yarn.Client: Application report for application_1492221381316_0009 (state: RUNNING)  
17/04/14 23:17:11 INFO yarn.Client: Application report for application_1492221381316_0009 (state: RUNNING)  
17/04/14 23:17:12 INFO yarn.Client: Application report for application_1492221381316_0009 (state: RUNNING)  
17/04/14 23:17:13 INFO yarn.Client: Application report for application_1492221381316_0009 (state: RUNNING)  
17/04/14 23:17:14 INFO yarn.Client: Application report for application_1492221381316_0009 (state: RUNNING)  
17/04/14 23:17:16 INFO yarn.Client: Application report for application_1492221381316_0009 (state: RUNNING)  
17/04/14 23:17:17 INFO yarn.Client: Application report for application_1492221381316_0009 (state: RUNNING)  
17/04/14 23:17:18 INFO yarn.Client: Application report for application_1492221381316_0009 (state: FINISHED)  
17/04/14 23:17:18 INFO yarn.client:  
  client token: N/A  
  diagnostics: N/A  
  ApplicationMaster host: 10.0.2.15  
  ApplicationMaster RPC port: 0  
  queue: default  
  start time: 1492222613043  
  final status: SUCCEEDED  
  tracking URL: http://JOAOLINUX:8088/proxy/application_1492221381316_0009/  
  user: hduser  
17/04/14 23:17:18 INFO util.ShutdownHookManager: Shutdown hook called  
17/04/14 23:17:18 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-d74e32b2-8162-476f-8df7-36f58a39643c  
hduser@JOAOLINUX:/usr/local/spark/MRApp$
```

- Outro exemplo:

The screenshot shows a web-based Hadoop File Explorer interface. The URL in the address bar is `localhost:50070/explorer.html#/entrada`. The page title is "Browse Directory". The top navigation bar includes links for "Hadoop", "Overview", "Datanodes", "Snapshot", "Startup Progress", and "Utilities". A search bar at the top right contains the placeholder "Pesquisar". The main content area displays a table of files in the "/entrada" directory. The table has columns: Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name. Two files are listed:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hduser	supergroup	2.01 KB	14/04/2017 23:01:12	1	128 MB	archiveBigData.txt
drwxr-xp-x	hduser	supergroup	0 B	14/04/2017 23:17:17	9	0 B	resultado

At the bottom of the page, the text "Hadoop. 2016." is visible.

Construindo e executando uma aplicação Spark

IGTI

- Outro exemplo:



A screenshot of a terminal window titled "part-00000(1)". The window displays a list of five numerical pairs, each enclosed in parentheses and separated by commas. The numbers represent floating-point values.

```
(1,130651.0)
(4,26578.0)
(7,149931.0)
(8,2235.0)
(9,84267.0)
```

Conclusão

- Construir.
- Compilar.
- Executar.
- Manipular dados no HDFS.



Próxima aula

- ❑ Componentes do Apache Spark.