FIAP GRADUAÇÃO

FIAP

DATA SCIENCE

BIG DATA ARCHITECTURING & DATA INTEGRATION

Prof. Dr. Renê de Ávila Mendes

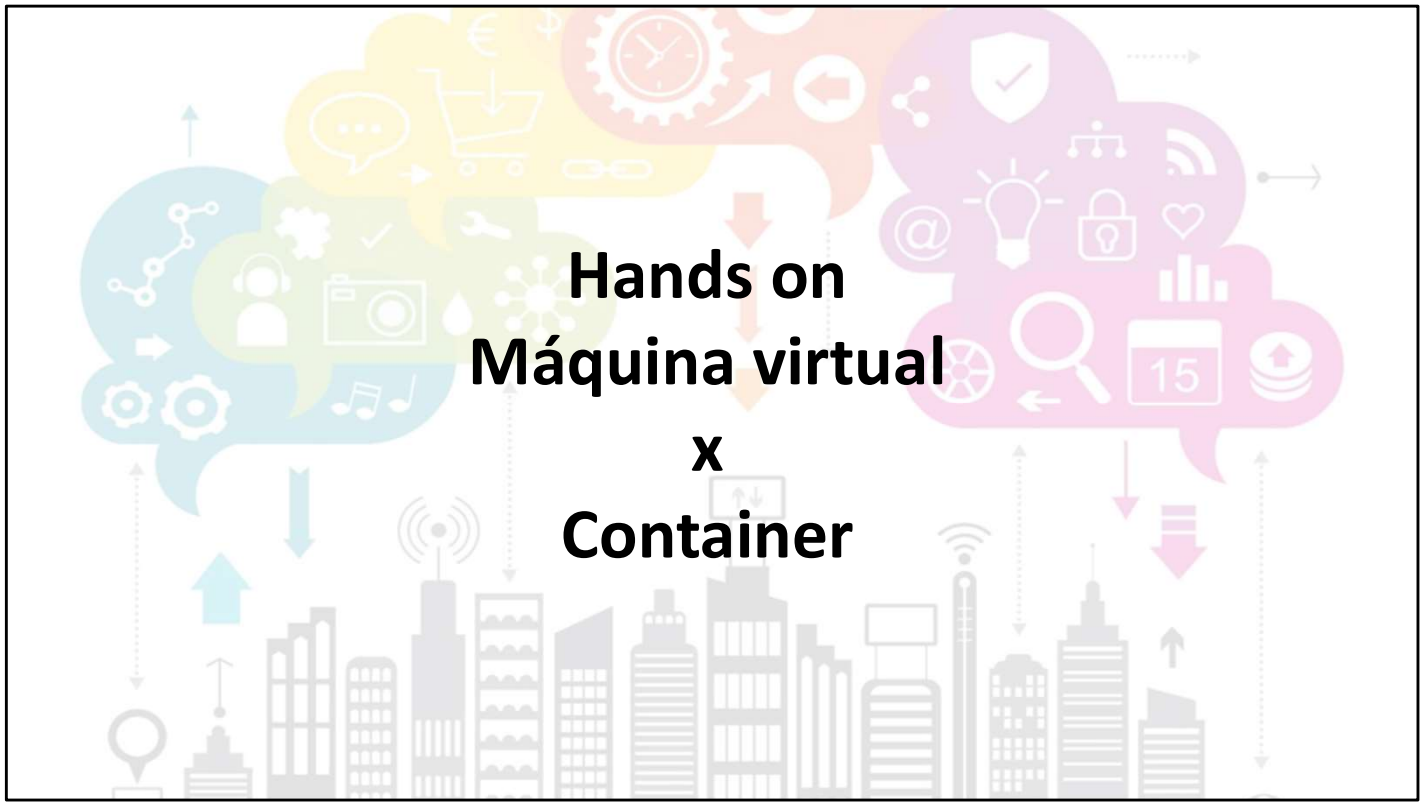
Objetivos da disciplina

DISCIPLINA: Big Data Architecturing & Data Integration

OBJETIVOS: Entenda as principais **arquiteturas** para ingestão, processamento e análise de grandes volumes de dados. Conheça as principais **ferramentas** open-source de Big Data como Hadoop, MapReduce, Spark, Sqoop, NiFi, Flume, Kafka, Zookeeper, HBase, Hive e as **integre** com as ferramentas de **extração, transformação e carga** de dados em modelos dimensionais. Entenda conceitos sobre **computação paralela e distribuída**, aplicação do Hadoop e bases Apache e arquiteturas *serverless* e desacopladas. Veja como **visualizar os dados** estruturados ou não estruturados com ferramentas de Self-Service Business Intelligence como PowerBI, utilizando as melhores práticas de **visualização de dados**.

Assuntos – 1º Semestre

- Introdução a Big Data
- Conceitos Computação Paralela e Distribuída, Lei de Moore, Sistema HDFS
- **Aplicação Hadoop. Usos e Administração de Ambientes Hadoop**
- **Introdução a MapReduce**
- Introdução à Integração de Dados
- Integração entre SQL e Hadoop - SQOOP
- Bases Apache
- Bases Apache - PIG
- Introdução da Data Streaming - FLUME
- Introdução a análise de dados com SPARK

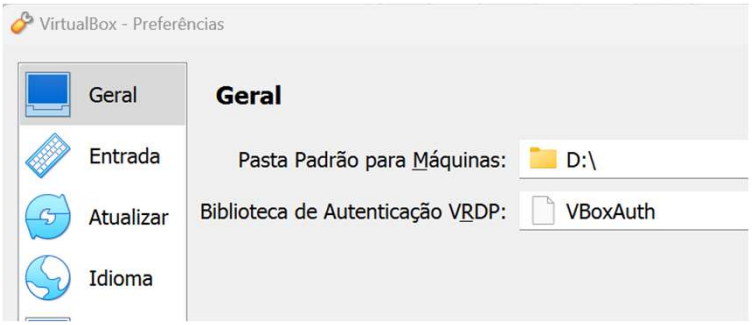

Hands on
Máquina virtual
x
Container

VM Hadoop

- Alterar a configuração do local:

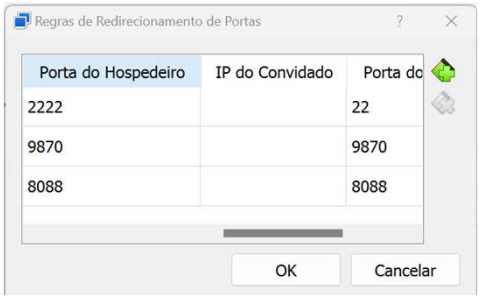
Arquivo > Preferências > Pasta padrão para máquinas: “D:\”

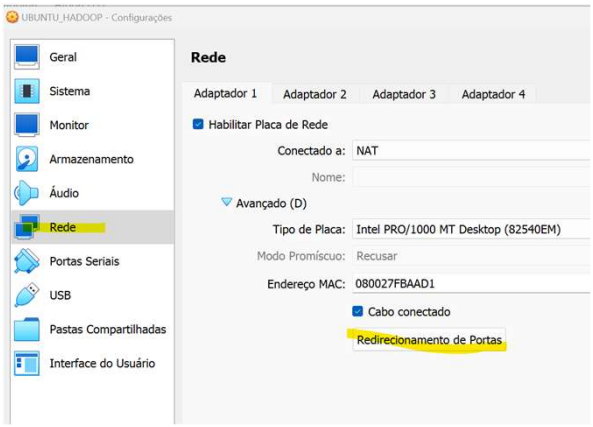
- Importar a imagem
- Usuário e senha
  - hadoop / hadoop



VM Hadoop

- Mapear portas
  - SSH





FIAP

Linux - Comandos

- ls – lista o diretório
  - ls
  - ls > teste.txt
- cat – lista o conteúdo de um arquivo
  - cat teste.txt
- mkdir – cria diretório
  - mkdir teste

FIAP

Linux - Comandos

- cp – copia arquivo
  - cp teste.txt teste1.txt
- mv – move um arquivo
  - mv teste1.txt /etc/teste2.txt
- rm – apaga arquivo
  - rm /etc/teste2.txt

FIAP

Comandos frequentemente usados

- Comandos *shell* são usados para executar várias operações Hadoop HDFS e para gerenciar os arquivos presentes em clusters HDFS.
- Todos os comandos são invocados pelo script bin/hdfs ou bin/hadoop.
- version.
  - Imprime a versão do hadoop
    - `hadoop version`

10

FIAP

Comandos frequentemente usados

- classpath.
  - Exibe o caminho das classes e bibliotecas do Hadoop. Definido no arquivo hadoop-config.sh.
    - `hadoop classpath`

11

http://namenode-name:50070/

FIAP

Comandos frequentemente usados

- getconf.
  - Exibe informações de configuração do diretório de configurações.
    - `hdfs getconf -namenodes`
    - `hdfs getconf -secondaryNameNodes`
    - `hdfs getconf -backupNodes`
    - `hdfs getconf -includeFile`
    - `hdfs getconf -excludeFile`
    - `hdfs getconf -nnRpcAddresses`

12

-namenodes gets list of namenodes in the cluster.
-secondaryNameNodes gets list of secondary namenodes in the cluster.
-backupNodes gets list of backup nodes in the cluster.
-includeFile gets the include file path that defines the datanodes that can join the cluster.
-excludeFile gets the exclude file path that defines the datanodes that need to decommissioned.
-nnRpcAddresses gets the namenode rpc addresses
-confKey [key] gets a specific key from the configuration

FIAP

Comandos frequentemente usados

- mkdir.
  - Cria um diretório no *path* informado.
    - `hadoop dfs -mkdir teste1-seunome`
    - `hadoop dfs -mkdir teste2-seunome`
    - `hadoop dfs -mkdir teste3-seunome`

FIAP

Comandos frequentemente usados

- ls.
  - Exibe uma lista do conteúdo de um diretório especificado no *path.* Pode exibir permissões, proprietário, tamanho e data de criação/alteração de cada arquivo.
    - `hadoop dfs –ls`
    - `hadoop dfs –ls -R`

FIAP

Comandos frequentemente usados

- copyFromLocal.
  - Copia um arquivo ou diretório do *file system* local para o destino especificado no HDFS.
  - `hadoop dfs -copyFromLocal /usr/local/hadoop/etc/hadoop/*.xml teste1-seunome`

FIAP

Comandos frequentemente usados

- cat.
  - Exibe o conteúdo de um arquivo na console ou na **stdout**.
  - `hadoop dfs -cat teste1-seunome/core-site.xml`

FIAP

Comandos frequentemente usados

- copyToLocal.
  - Copia um arquivo ou diretório do HDFS para o *file system* local.
  - `hadoop dfs -copyToLocal teste1-seunome/core-site.xml .`
  - `hadoop dfs -ls teste1-seunome`

17

FIAP

Comandos frequentemente usados

- cp
  - Copia um arquivo ou diretório de uma origem identificada no comando para um destino identificado no comando
  - `hadoop dfs -cp teste1-seunome/*.xml teste2-seunome`

FIAP

Comandos frequentemente usados

- mv
  - Move um arquivo ou diretório de uma origem identificada no comando para um destino identificado no comando
  - `hadoop dfs -mv teste1-seunome/core-site.xml teste3-seunome`

FIAP

Comandos frequentemente usados

- chmod
  - Altera as permissões de leitura, gravação e execução de um arquivo
  - `hadoop dfs -chmod 777 teste2-seunome/*`

	Owner	Group	Other
Read (r)	4	4	4
Write (w)	2	2	2
Execute (x)	1	1	1
Total	7	7	7

20

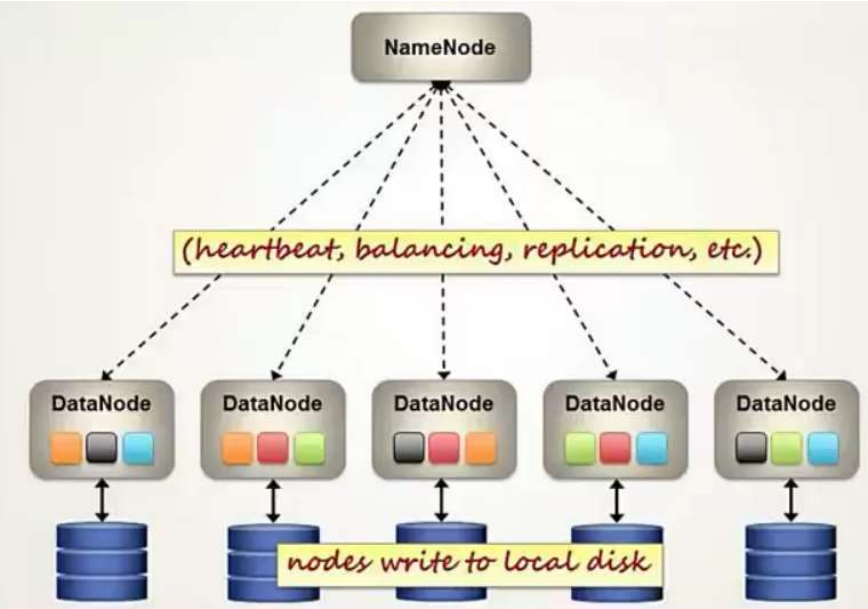
FIAP

Comandos frequentemente usados

- rm
  - Apaga um arquivo ou diretório
  - `hadoop dfs -rm teste2-seunome/core-site.xml`
  - `hadoop dfs -rm -r teste1-seunome`
  - `hadoop dfs -rm -r teste2-seunome`
  - `hadoop dfs -rm -r teste3-seunome`

HDFS

FIAP


NameNode
(heartbeat, balancing, replication, etc.)
DataNode
DataNode
DataNode
DataNode
DataNode
nodes write to local disk

Namespace

FIAP

- “É um delimitador abstrato (container) que fornece um contexto para os itens que ele armazena (nomes, termos técnicos, conceitos), o que permite uma desambiguação para itens que possuem o mesmo nome mas que residem em espaços de nomes diferentes. Como um contexto distinto é fornecido para cada container, o significado de um nome pode variar de acordo com o espaço de nomes o qual ele pertence” (O’Reilly Strata, 2015)

NameNode

FIAP

- Conhecido do como nó mestre (*master node)*
- É o servidor que armazena os metadados da árvore de diretório, réplicas, número de bloco de dados e outros detalhes
- Os metadados estão disponíveis na memória para recuperação mais rápida dos dados.

NameNode

FIAP

- Mantém e gerencia os nós escravos e atribui tarefas a eles.
- Normalmente implementado em hardware confiável pois é peça central do HDFS.

NameNode

FIAP

```
cd $HADOOP_HOME/etc/hadoop/
ls -ltr
vi core-site.xml
```

- A **property fs.defaultFS** especifica a localização do **namenode**.

```
<property>
 <name>fs.defaultFS</name>
 <value>hdfs://ip-172-31-45-216.ec2.internal:8020</value>
</property>
```

- Este arquivo fica disponível em todos o **nodes** do **cluster** permitindo que todos **nodes** saibam a localização do **namenode**.

Tarefas do NameNode

FIAP

- Gerencia o *namespace* do sistema de arquivos
- Regula o acesso do cliente aos arquivos
  - Não existe quota de usuário
- Executa operações do *file system*
  - nomeação, abertura e fechamento de arquivos ou diretórios.

Tarefas do NameNode

FIAP

- Todos os DataNodes enviam *Heartbeats* e *block reports* para o NameNode.
  - *Heartbeats* garantem que o nó está ativo
  - *Block Report* contém a lista de blocos do datanode

- Responsável por cuidar do *Replicator Factor* de todos os blocos.

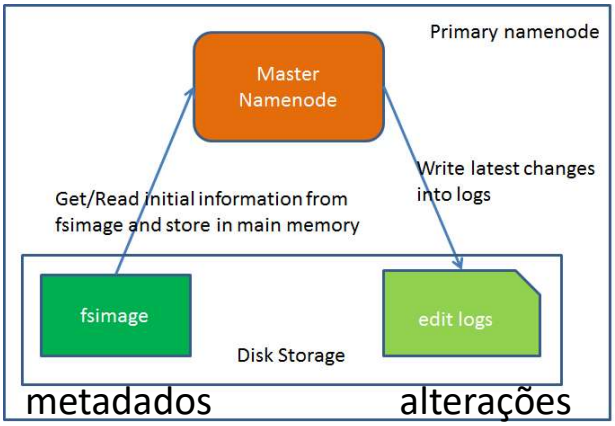
Arquivos do NameNode

FIAP

- Arquivos presentes nos metadados do NameNode:
- **FsImage**
  - É um "arquivo de imagem".
  - Contém o *namespace* inteiro do sistema de arquivos
  - Armazenado como um arquivo no file system no namenode
  - Contém uma forma serializada de todos os diretórios e arquivos ***inode*** do sistema
    - **Inode** é uma representação interna dos metadados dos arquivos e diretórios.
- **EditLogs**
  - Contém todas as alterações recentes feitas no **FsImage** mais recente.
  - Ao receber uma solicitação de criação, atualização ou exclusão do cliente, o Namenode primeiro registra essa solicitação no arquivo de edição.

Arquivos do NameNode

FIAP



DataNode

FIAP

- Conhecido como *Slave*.
- Armazena dados reais no HDFS.
- Executa a operação de leitura e gravação de acordo com a solicitação do cliente.
- DataNodes podem ser implantados em hardware de commodities.

NameNode

FIAP

```
cd etc/hadoop
ls -ltr
vi hdfs-site.xml
```

- **dfs.namenode.name.dir** especifica onde o **NameNode** pode armazenar seus arquivos, localmente.
- **dfs.datanode.name.dir** especifica onde o **DataNode** pode armazenar arquivos e blocos, localmente
- **dfs.namenode.http.address** fornece o endereço no **NameNode**. O endereço pode ser acessado através de um *browser* e fornecer informações sobre o HDFS.

```
<property>
  <name>dfs.namenode.http-address</name>
  <value>ec2-54-92-244-237.compute-1.amazonaws.com:50070</value>
```

Tarefas do DataNode

FIAP

- Bloquear a criação, exclusão e replicação conforme as instruções enviadas pelo **Namenode**.
- Gerenciar o armazenamento de dados do sistema.
- Enviar **heartbeats** para o **NameNode**
  - Por padrão, a freqüência é de 3 segundos.

Replicação

FIAP

- Replicação de blocos fornece tolerância a falhas.
- Se uma cópia não for acessível ou estiver corrompida, pode ser lida de outra cópia.
- O número de cópias ou réplicas de cada bloco de um arquivo é o fator de replicação (3, por padrão). Cada bloco e replicado três vezes e é armazenado em diferentes **DataNodes**.
- Em uma configuração padrão, um arquivo de 128 MB em HDFS ocupa 384 MB (3 * 128 MB) de espaço.
- **NameNode** recebe relatório de bloco do **DataNode** periodicamente para manter o fator de replicação.
- Se um bloco estiver super-replicado / sub-replicado, o **NameNode** adiciona ou exclui as réplicas conforme necessário.

Replicação

FIAP

- `hadoop dfs -ls teste3-seunome`

- `hadoop dfs -Ddfs.replication=2 -cp teste2-seunome/yarn-site.xml teste2-seunome/teste.xml`
- `hadoop dfs -ls teste2-seunome`
- `hadoop dfs -ls teste2-seunome/teste.xml`

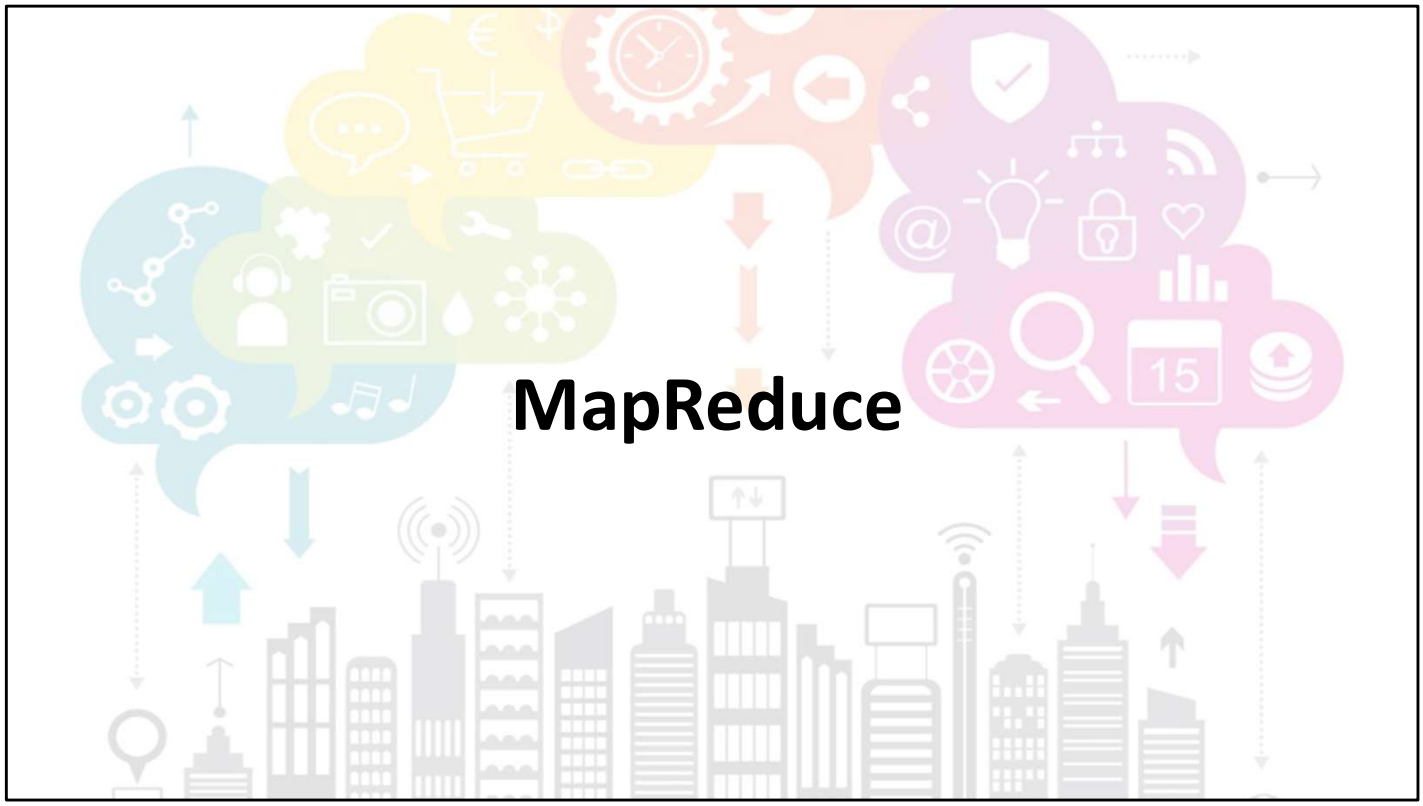

CHECKPOINT 1

Checkpoint 1

Data: 27/03

Data de entrega: 02/04

Entrega: Portal do aluno


MapReduce

Map Reduce

- Em 2004 o Google publicou o artigo chamado MapReduce: Simplified Data Processing on Large Clusters.
- MapReduce é um modelo de programação para processamento de grandes volumes de dados.
- Esta abstração foi inspirada nas primitivas **map** e **reduce** do Lisp e outras linguagens funcionais.

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

1 Introduction

Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, web request logs, etc., to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance.

The major contributions of this work are a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

Section 2 describes the basic programming model and gives several examples. Section 3 describes an implementation of the MapReduce interface tailored towards our cluster-based computing environment. Section 4 describes several refinements of the programming model that we have found useful. Section 5 has performance measurements of our implementation for a variety of tasks. Section 6 explores the use of MapReduce within Google including our experiences in using it as the basis

To appear in OSDI 2004

https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

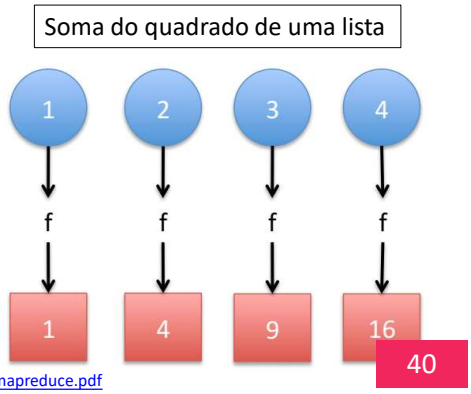
FIAP

Map e Reduce no Lisp

- Lisp é uma linguagem funcional, projetada por John McCarthy que apareceu pela primeira vez em 1958.
- A primitiva **map** recebe como parâmetros um operador unário e uma lista, onde este operador será chamada para cada elemento da lista

```
(map square '(1 2 3 4))
```

Resultado: (1 4 9 16)



https://cse.buffalo.edu/~stevko/courses/cse704/fall10/papers/cse704-mapreduce.pdf

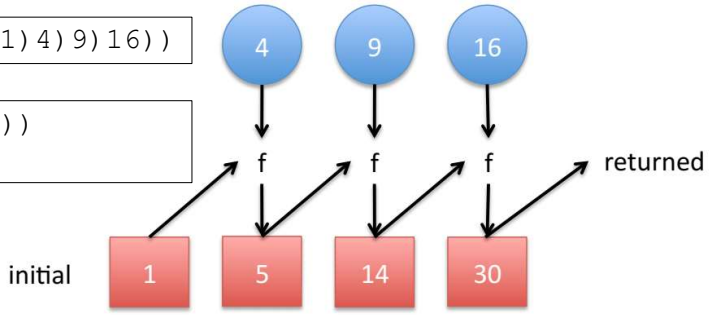
FIAP

Map e Reduce no Lisp

- A primitiva **reduce** recebe como parâmetros um operador binário e uma lista, onde os elementos da lista serão combinados utilizando este operador

```
(reduce)+)'(1)4)9)16))
```

```
(+16(+9(+4 1)))
Resultado: 30
```



https://cse.buffalo.edu/~stevko/courses/cse704/fall10/papers/cse704-mapreduce.pdf

FIAP

Map Reduce em Lisp

- Map
  - processa cada registro individualmente
- Reduce
  - Processa (combina) o conjunto de todos os registros em lote

https://cse.buffalo.edu/~stevko/courses/cse704/fall10/papers/cse704-mapreduce.pdf

O que a Google percebeu

FIAP

- Pesquisa por palavra-chave

Map – Encontre uma palavra-chave em cada página da web individualmente e, se ela for encontrada, retorne sua URL

Reduce – Combine todos os resultados (URLs) e devolva-os

- Contagem do número de ocorrências de cada palavra

– Conte o número de ocorrências em cada página da web

Map individualmente e retorne a lista de <palavra, número>

Reduce – Para cada palavra, some (combine) a contagem

https://renil.wordpress.com/2007/05/03/mapreduce-functional-programming-lisp/

FIAP

Visão Geral do Map Reduce

Map

Os dados na fase map são divididos entre os nós task tracker onde os dados estão localizados. Cada nó na fase map emite pares chave-valor com base no registro de entrada, um registro por vez.

Shuffle

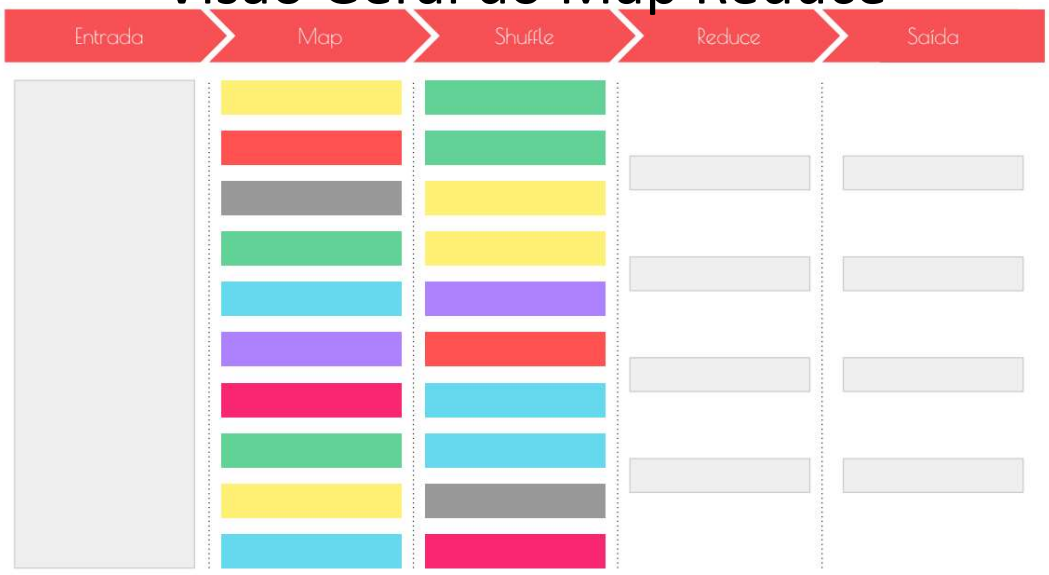
A fase shuffle é tratada pela estrutura Hadoop. Ela transfere os resultados dos mappers para os reducers juntando os resultados por meio chave.

Reduce

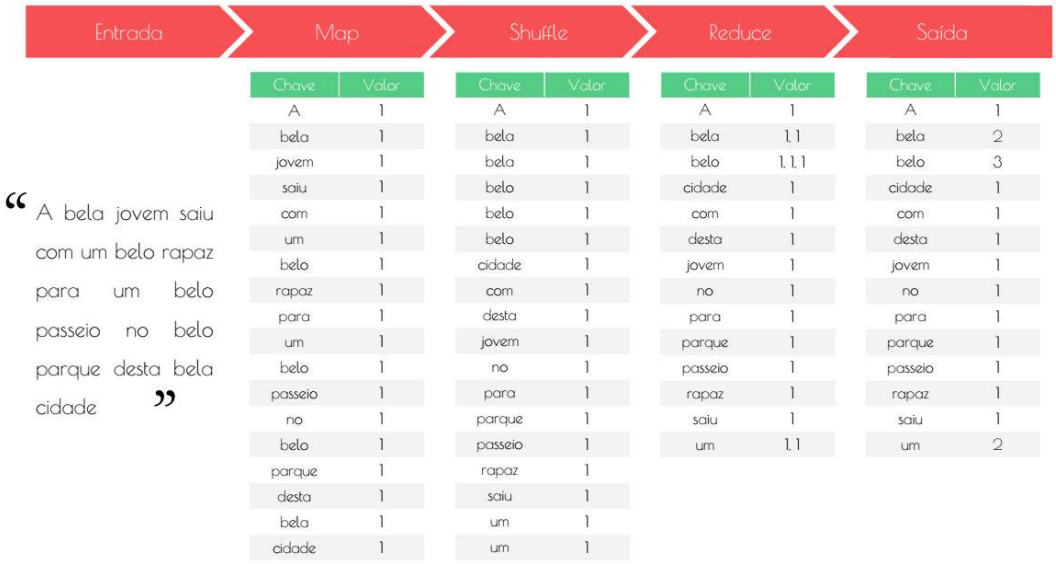
A saída dos mappers é enviado para os reducers. Os dados na fase reduce são divididos em partições onde cada reducer lê uma chave e uma lista de valores associados a essa chave. Os reducers emitem zero ou mais pares chave-valor com base na lógica utilizada

Visão Geral do Map Reduce

FIAP



Visão Geral do Map Reduce FIAP



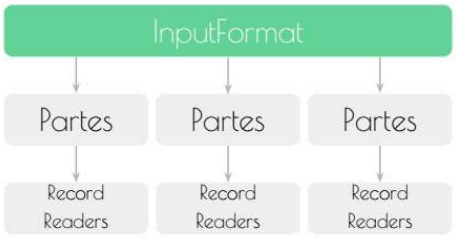
FIAP

InputFormat

- O primeiro passo para executar um programa MapReduce é localizar e ler o arquivo de entrada contendo os dados brutos.
- O formato do arquivo é completamente arbitrário, mas os dados devem ser convertidos para algo que o programa pode processar.
- Isso é feito pelo InputFormat e RecordReader.
- InputFormat decide como o arquivo será dividido em peças menores para processamento usando uma função chamada InputSplit.

FIAP

InputFormat

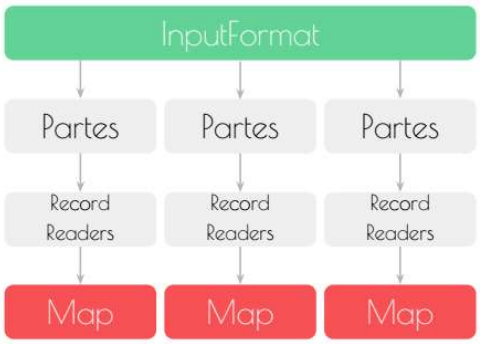


- O objeto **InputFormat** é responsável por validar a entrada, dividindo os arquivos entre os **mappers** e instanciando os objetos **RecordReaders**.
- Por padrão, o tamanho de uma parte é igual ao tamanho de um bloco que no Hadoop o padrão é 64 Mb.
- As partes possuem um conjunto de registros onde cada um deles será quebrado em pares chave-valor para o **map**. A separação dos registros é feita antes mesmo da instanciação do processo map.
- O **job** que está executando a tarefa **MapReduce** tentará colocar a tarefa **map** o mais próximo dos dados possível, ou seja, executar o processo **map** no mesmo nó do cluster onde o dados está armazenado.

FIAP

Método MAP

- O método map recebe como argumento 3 parâmetros:
  - Writable chave,
  - Writable valor e
  - context.
- Por padrão, o **RecordReader** define a chave para o método **map** como sendo o byte offset do registro no arquivo de entrada e o valor é basicamente a linha inteira.



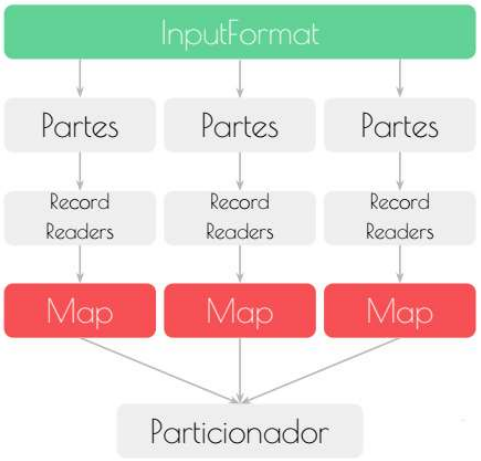
Computer-science World
Quantum Computing

<0,Computer Science World>
<23,Quantum Computing>

FIAP

Particionador

- O particionador recebe a saída gerada pelo método **map** e faz um **hash** da chave e cria uma partição com base no **hash** da chave.
- Cada partição se destina a um **reducer**, assim, todos os registros com a mesma chave serão enviados para a mesmo partição (e, portanto, enviados para o mesmo **reducer**).



FIAP

Reduce

- O método **reduce** é chamado para cada chave e a lista de valores associados a essa chave.
- O método **reduce** processa cada valor e grava o envia o resultado para o contexto.
- O **OutputCommitter** cria um arquivo de saída para cada reduce executado.

FIAP

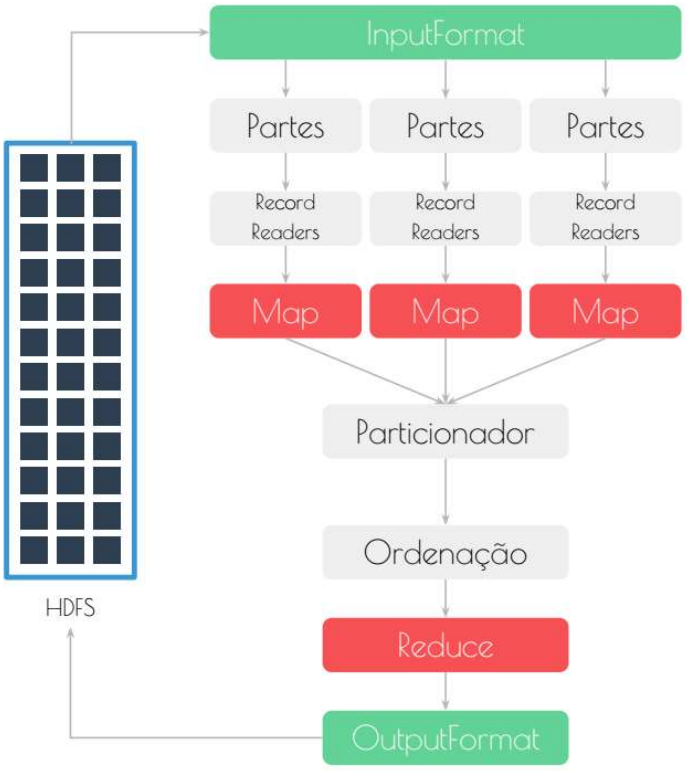
OutputFormat

- Os resultados de um job MapReduce são armazenados no diretório especificado pelo usuário.
- Um arquivo vazio chamado _SUCCESS é criado para indicar que o job foi concluído com sucesso (mas não necessariamente sem erros).

FIAP

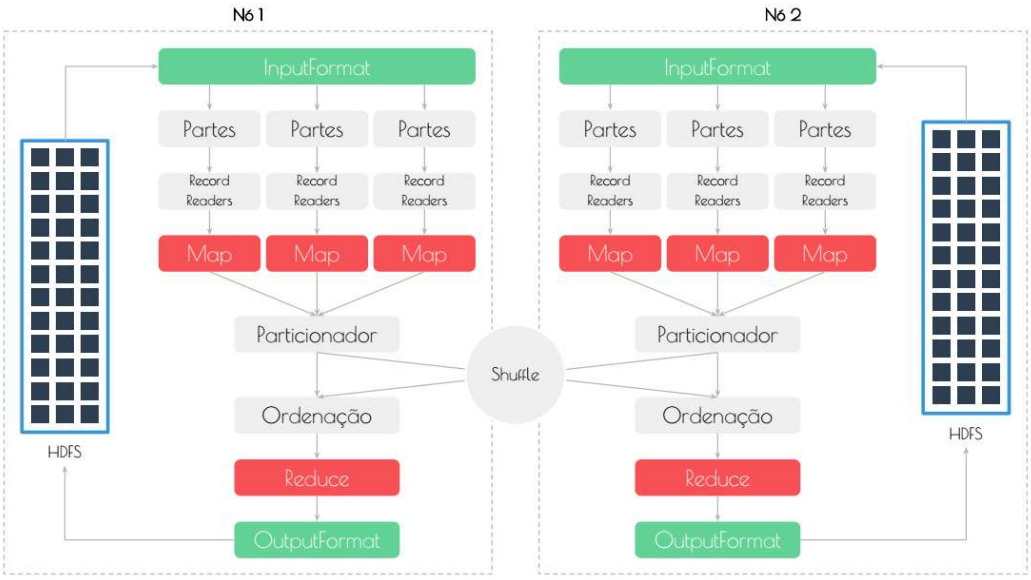
OutputFormat

- O histórico do job é armazenado em _logs/history*.
- A saída do método reduce é salvo em arquivos denominados part-r-00000, part-r-00001, ... (um para cada reduce).
- Caso seja executada uma tarefa apenas de mapeamento, implementação apenas do map, os nomes dos arquivos de saída serão part-m-00000, part-m-00001, ... etc.


InputFormat
Partes
Partes
Partes
Record Readers
Record Readers
Record Readers
Map
Map
Map
Particionador
Ordenação
Reduce
OutputFormat
HDFS

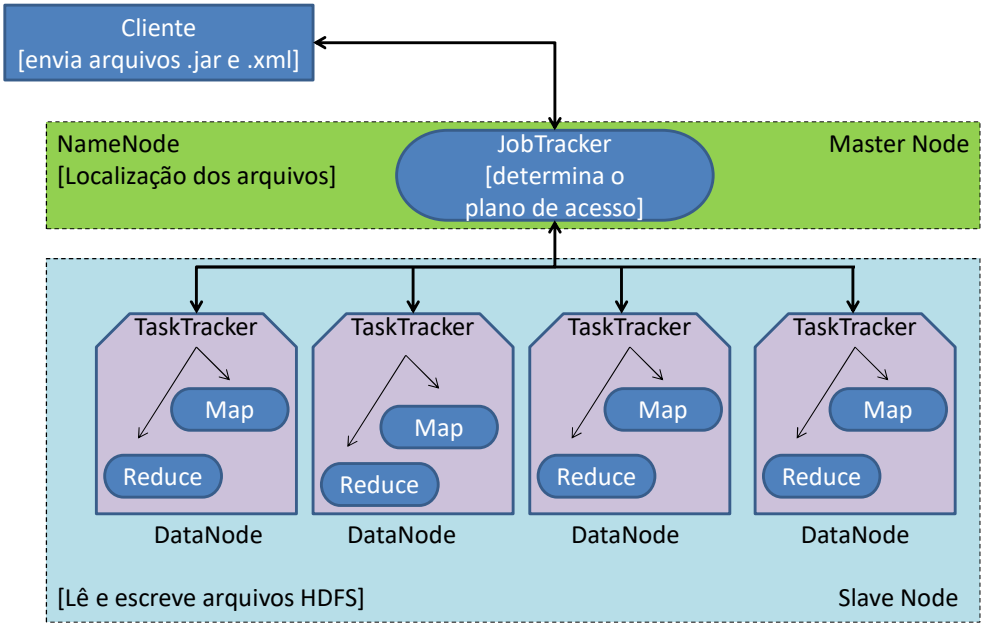
FIAP

FIAP


N6 1
N6 2
InputFormat
Partes
Partes
Partes
Record Readers
Record Readers
Record Readers
Map
Map
Map
Particionador
Shuffle
Ordenação
Reduce
OutputFormat
HDFS
HDFS

MapReduce Engine

FIAP



Exemplo: Contador de Palavras

- Incluindo as palavras dog, cat, mouse e hippo em um banco orientado a documentos.

```
db.items.insert({tags: ['dog', 'cat']})
db.items.insert({tags: ['dog']})
db.items.insert({tags: ['dog', 'mouse']})
db.items.insert({tags: ['dog', 'mouse', 'hippo']})
db.items.insert({tags: ['dog', 'mouse', 'hippo']})
db.items.insert({tags: ['dog', 'hippo']})
```

Fonte: https://docs.mongodb.org/manual/tutorial/map-reduce-Exemplos/

Exemplo: Contador de Palavras

- Criando o mapeamento

```
var map = function() {
  this.tags.forEach(function(t) {
    emit(t, {count: 1});
  });
}
```

Fonte: https://docs.mongodb.org/manual/tutorial/map-reduce-Exemplos/

Exemplo: Contador de Palavras

- Criando o reduce

```
var reduce = function(key, values) {
  var count = 0;
  for(var i=0, len=values.length; i<len; i++) {
    count += values[i].count;
  }
  return {count: count};
}
```

Fonte: https://docs.mongodb.org/manual/tutorial/map-reduce-Exemplos/

Exemplo: Contador de Palavras

- Executando o MapReduce

```
var result = db.items.mapReduce(map, reduce);
result
{
  "ok"         : 1,
  "timeMillis" : 86,
  "result"     : "tmp.mr.mapreduce_1273861517_683",
  "counts"     : {
    "input"    : 6,
    "emit"     : 13,
    "output"   : 4
  }
}
```

Fonte: https://docs.mongodb.org/manual/tutorial/map-reduce-Exemplos/

Exemplo: Contador de Palavras

- Executando o MapReduce

```
> db[result.result].find()
{ "_id" : "cat",   "value" : { "count" : 1 } }
{ "_id" : "dog",   "value" : { "count" : 6 } }
{ "_id" : "hippo", "value" : { "count" : 3 } }
{ "_id" : "mouse", "value" : { "count" : 3 } }
```

Fonte: https://docs.mongodb.org/manual/tutorial/map-reduce-Exemplos/

FIAP

Submetendo um job MapReduce

- WordCount.java
  - Classe de drive do MapReduce para executar o job
- WordMapper.java
  - Uma classe mapper para emitir para emitir palavras
- SumReducer.java
  - Uma classe reducer para contar palavras

https://www.dropbox.com/sh/bsopakcxg7wctfu/AAA-r0_LQkT0ze54KnzYzhDMa?dl=0