

HD01 - Agenda	2
HD02 - Conceitos	5
HD03.1 - Hadoop - Introdução	32
HD03.1- Hadoop - Prática HUE e FSSHELL	51
HD03.3 - Hadoop - MapReduce	54
HD03.3.1 - Hadoop - Prática Map Reduce textos	65
HD03.3.2 - Hadoop - Prática Map Reduce SQL	67
HD03.3.3 - Hadoop - Prática Map Reduce Java	70
HD03.4 - Hadoop - YARN	75
HD03.4.1 - Hadoop - Prática Ambari	86
HD03.5 - Hadoop - Trabalho Prático	89
HD04.1 - Spark - Introdução	90
HD04.1.1 - Spark - Prática PySpark	109
HD04.2 - Spark - RDD	114
HD04.2.1 - Spark - Prática RDD	122
HD04.3 - Spark - Algumas operações	130
HD04.3.1 - Spark - Prática Transformações e ações	141
HD04.4 - Spark - SQL	144
HD04.4.1 - Spark - Prática SQL	156
HD04.5 - Spark - DataFrame	163
HD04.5.1 - Spark - Prática DataFrame	167
HD06 - Spark - Trabalho Prático	170

Soluções para processamento paralelo e distribuído de dados

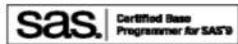


Ilustração de Oliver Munday

Soluções Processamento paralelo e distribuído de dados

Apresentação

Cláudio Lúcio



Doutorando em Modelagem Matemática e Computacional, Mestre em informática PUC MG, Especialista em estatística UFMG, pós-graduado em gerência de projetos, Bacharel em ciência da computação;

18 anos de experiência com Bancos de dados atuando em projetos para clientes do cenário nacional: Arcelor Mittal, Banco Mercantil do Brasil, FIDIS, BDMG, BMG, CEMIG, IVECO, FIAT - FCA, JEEP, GM do Brasil, Mendes Júnior, Localiza, SEBRAE-SC, SUDECAP, Telefônica, Oi, Vale, VIVO, VMM – Votorantim Mineração e Metais.

Treinamentos ministrados para várias empresas: Assurant, Athos Pharma, Banco do Brasil, Best Forecast, BM&F, Caixa Econômica, CEMIG, E-Lucid, GM do Brasil, HDI Seguros, Mapfre, Marítima Seguros, Telemar, FIAT, Telemig Celular, SEF-MG e Unibanco;

Soluções Processamento paralelo e distribuído de dados

Agenda:

Módulo 1: Conceitos

- Motivação
- Arquitetura de soluções paralelas
- Paradigmas de programação paralela
- Exemplo práticos de soluções paralelas 'atuais'

Módulo 2: Hadoop

- Conceitos
- Hadoop Core(DFS, MapReduce e Yarn)

Módulo 3: Spark

- Conceitos
- SparkSQL, RDD, DataFrames

Soluções Processamento paralelo e distribuído de dados

Objetivos:

- Entender os desafios tecnológicos criados pelo Big Data;
- Entender o paradigma proposto por dados massivos ;
- Introduzir as premissas de funcionamento de plataformas para processamento paralelo e distribuído;
- Introdução das ferramentas básicas do Hadoop;
- Introdução as ferramentas básicas do Spark;

Soluções Processamento paralelo e distribuído de dados

Dinâmica:

Frequência;

Demonstrações e exercícios em laboratório

HortonWorks

Python

Trabalhos práticos

Soluções Processamento paralelo e distribuído de dados

Referências bibliográficas:

Ciprian Dobre and Fatos Xhafa. 2014. Parallel Programming Paradigms and Frameworks in Big Data Era. Int. J. Parallel Program. 42, 5 (October 2014), 710-738. DOI=<http://dx.doi.org/10.1007/s10766-013-0272-7>

EMC Summer School 2013. Disponível em: <http://emcbigdataschool.nce.ufrj.br/index.php/speakers-and-schedule/slides.html>. Acesso em: 25/02/2013;

Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters"; pub. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December 2004

Krishna Sankar;Holden Karau. Fast Data Processing with Spark (Second Edition): Perform real-time analytics using Spark in a fast,distributed, and scalable way.Published by Packt Publishing Ltd.Birmingham B3 2PB, UK. ISBN 978-1-78439-257-4

M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," USENIX NSDI, 2012.

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google File System"; pub. 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October 2003.

WILKINSON, B.; ALLEN, M. Parallel Programming: Using Networked Workstations and Parallel Computers. New Jersey: Prentice Hall, 1998.

Ullman ,J.D.; Rajaraman ,A. Mining Massive Datasets. Cambridge University Press, UK 2012. Disponível em: <http://i.stanford.edu/~ullman/mmds.html> Acesso em: 22/09/2012. Capítulo 2

Soluções para processamento paralelo e distribuído de dados



Ilustração de Oliver Munday

Soluções Processamento paralelo e distribuído de dados

Conceitos

- Motivação
- Arquitetura de soluções paralelas
- Paradigmas de programação paralela
- Exemplo práticos de soluções paralelas 'atuais'

Soluções para processamento paralelo e distribuído de dados

Conceitos – Motivação



Ilustração de Oliver Munday

Conceitos - Motivação

Porque computação paralela e distribuída?

Sistemas de computadores sequenciais cada vez mais velozes

- Velocidade de processador

- Memória

- Comunicação com o mundo externo

Conceitos - Motivação

Porque computação paralela e distribuída?

Mas, e os dados massivos ???



Conceitos - Motivação

Porque computação paralela e distribuída?

Mas, e os problemas derivados de dados massivos ???

Cosmologia e Astrofísica;

Clima global e modelagem do ambiente;

E os carros autônomos, e as montadoras e locadoras de carros?

Como será a digitalização deste negócio com dados granulares em tempo real?

Detecção de padrões em vídeos em tempo real ?

E o atacadista ali perto? Cruzar o histórico de 5 anos de compras de cada um de seus clientes por produto (novas oportunidades de vendas): 5.000 produtos * 100.000 clientes * 1825 dias = 912.500.000.000

Conceitos - Motivação

Porque computação paralela e distribuída?

Em resumo:

As aplicações que exigem computadores cada vez mais rápidos estão por toda parte;

Estas aplicações ou requerem um grande poder de computação ou requerem o processamento de grandes quantidades de informação;

Conceitos - Motivação

Porque computação paralela e distribuída?

Fatos:

Estrutura de arquivos convencional não é capaz de lidar com dados massivos;

É necessário um paradigma que comporte escalabilidade elástica;

Bancos de dados relacionais utilizam conceitos, que às vezes, não são um requisito para dados massivos;

Um infra estrutura tolerante a falhas e que permita computação paralela é necessária;

Soluções para processamento paralelo e distribuído de dados

Conceitos – Arquitetura de soluções paralelas



Ilustração de Oliver Munday

Conceitos - Arquitetura de soluções paralelas

Características interessantes de soluções paralelas

- Conectividade ⇒ rede de interconexão
- Heterogeneidade ⇒ hardware e software distintos
- Compartilhamento ⇒ utilização de recursos (memória, disco, CPU)
- Imagem do sistema ⇒ como usuário o percebe
- Escalabilidade ⇒ mais estações de trabalho levam a melhor desempenho/eficiência

Conceitos - Arquitetura de soluções paralelas

Taxonomia de Flynn

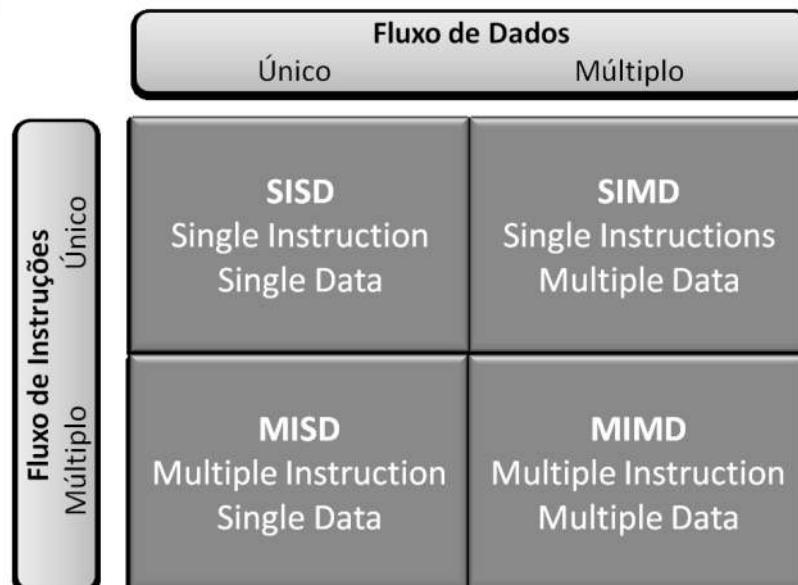
Uma das metodologias mais conhecidas e utilizadas para classificar uma arquitetura de computadores ou conjunto de computadores é a taxonomia de Flynn (1966);

Duas dimensões: instruções e dados;

Cada dimensão assume dois valores distintos: *single* ou *multiple*;

Conceitos - Arquitetura de soluções paralelas

Taxonomia de Flynn



Conceitos - Arquitetura de soluções paralelas

Taxonomia de Flynn

SISD – Single Instruction Single Data

Arquitetura dos computadores com um único processador;

Apenas uma instrução é processada a cada momento.

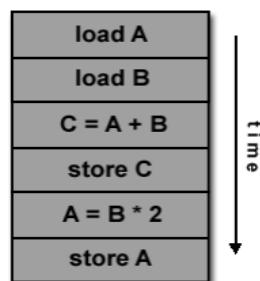
Apenas um fluxo de dados é processado a cada momento.

Exemplos: PCs, workstations e servidores com um único processador.

Conceitos - Arquitetura de soluções paralelas

Taxonomia de Flynn

SISD – Single Instruction Single Data



Conceitos - Arquitetura de soluções paralelas

Taxonomia de Flynn

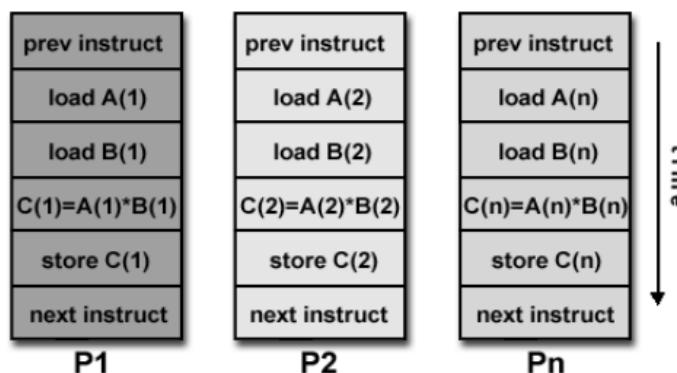
SIMD – Single Instruction Multiple Data

- Tipo de arquitetura paralela desenhada para problemas específicos (alto padrão de regularidade nos dados, ex.:processamento de imagem);
- Todas as unidades de processamento executam a mesma instrução a cada momento;
- Cada unidade de processamento pode operar sobre um fluxo de dados diferente;
- Exemplos: Computadores com unidades de processadores gráficos (GPUs);

Conceitos - Arquitetura de soluções paralelas

Taxonomia de Flynn

SIMD – Single Instruction Multiple Data



Conceitos - Arquitetura de soluções paralelas

Taxonomia de Flynn

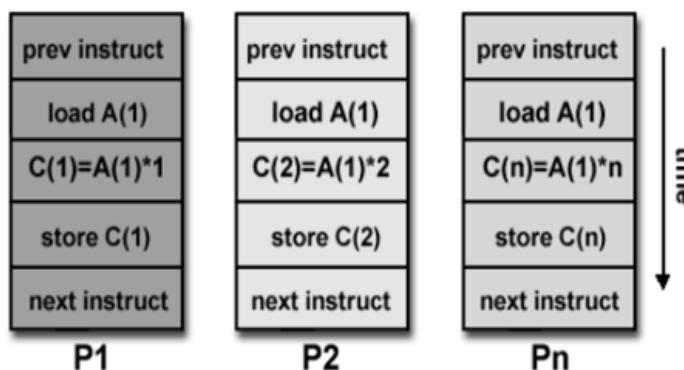
MISD – Multiple Instruction Single Data

- Arquitetura paralela desenhada para problemas caracterizados por um alto padrão de regularidade funcional (ex.: processamento de sinal);
- Constituída por uma pipeline de unidades de processamento independentes que operam sobre um mesmo fluxo de dados enviando os resultados de uma unidade para a próxima;
- Cada unidade de processamento executa instruções diferentes a cada momento;
- Exemplos: Não existem exemplos práticos

Conceitos - Arquitetura de soluções paralelas

Taxonomia de Flynn

MISD – Multiple Instruction Single Data



Conceitos - Arquitetura de soluções paralelas

Taxonomia de Flynn

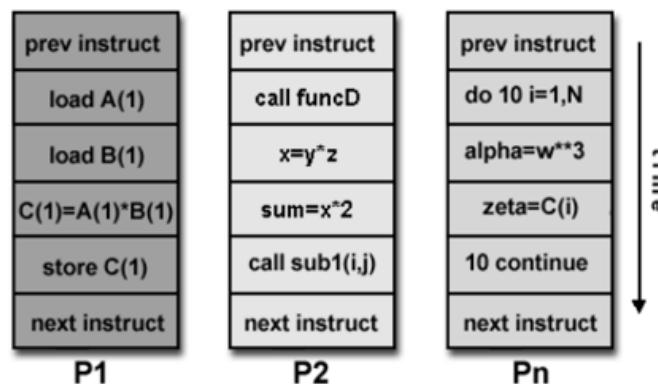
MIMD – Multiple Instruction Multiple Data

- Arquitetura paralela predominante;
- Cada unidade de processamento executa instruções diferentes a cada momento;
- Cada unidade de processamento pode operar sobre um fluxo de dados diferente;
- Exemplos: alguns supercomputadores, clusters de computadores paralelos em rede, PCs multi-core;

Conceitos - Arquitetura de soluções paralelas

Taxonomia de Flynn

MIMD – Multiple Instruction Multiple Data



Conceitos - Arquitetura de soluções paralelas

Mas, no fundo, o que se espera com computadores paralelos ?

- Queremos melhorar o “Speedup”....
- Lei de Amdahl: Speedup esperado: $\frac{T(1)}{T(P)}$
- Speedup esperado com número de processadores:

$$\frac{1}{\frac{P}{N} + S}$$

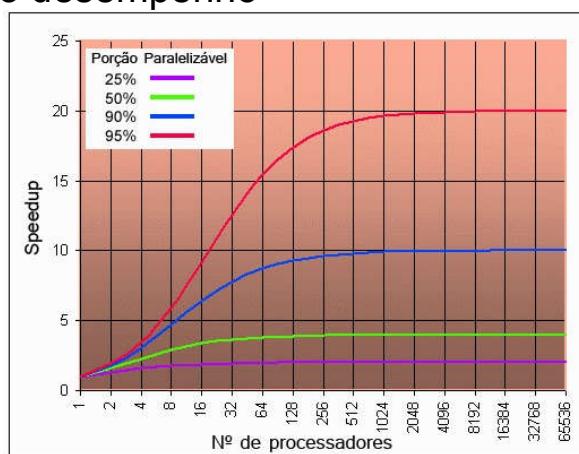
P = fração paralelizável
N = número dos processadores
S = fração sequencial
Exemplo: P = 0.7 S = 0.3 N = 1 Speedup = 1
P = 0.7 S = 0.3 N = 2 Speedup = 1.53
P = 0.7 S = 0.3 N = 3 Speedup = 1.87

Conceitos - Arquitetura de soluções paralelas

Mas, no fundo, o que se espera com computadores paralelos ?

- Speedup : melhorar o desempenho

speedup			
N	P = .50	P = .90	P = .99
10	1.82	5.26	9.17
100	1.98	9.17	50.25
1,000	1.99	9.91	90.99
10,000	1.99	9.91	99.02
100,000	1.99	9.99	99.90



Conceitos - Arquitetura de soluções paralelas

Plataformas MIMD - Multiple Instruction Multiple Data

- Espaço de endereçamento
- Mecanismo de comunicação

Podem ser agrupadas em quatro grupos

- SMPs (Symmetric MultiProcessors)
- MPPs (Massively Parallel Processors)
- Cluster ou NOWs (Network Of Workstations)
- Grades Computacionais

Conceitos - Arquitetura de soluções paralelas

SMPs ou Multiprocessadores

- Espaço de endereçamento
 - Único espaço de endereçamento lógico
 - Mecanismo de hardware (memória centralizada)
- Mecanismo de comunicação
 - Comunicação entre processadores se dá através de um espaço de endereçamento compartilhado
 - Operações de loads(leitura) e stores(escrita) feitos com acesso a memória ;

Conceitos - Arquitetura de soluções paralelas

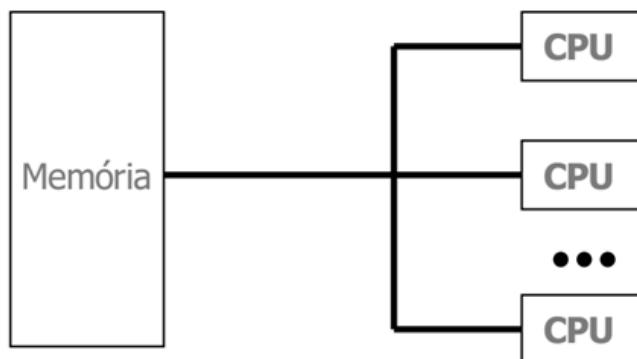
SMPs ou Multiprocessadores

- Características

- Sistema homogêneo
- Comunicação através da mesma memória
- Uma única cópia do Sistema Operacional
- Imagem única do sistema
- Fortemente acoplados
- Não escalável

Conceitos - Arquitetura de soluções paralelas

SMPs ou Multiprocessadores



Conceitos - Arquitetura de soluções paralelas

MPPs (Multicomputadores)

- Espaço de endereçamento
 - Não compartilhado
 - Memória distribuída
- Mecanismo de comunicação
 - Troca de mensagens;
- Unidades de processamento
 - Múltiplos processadores com memória privativa
 - Computadores completos

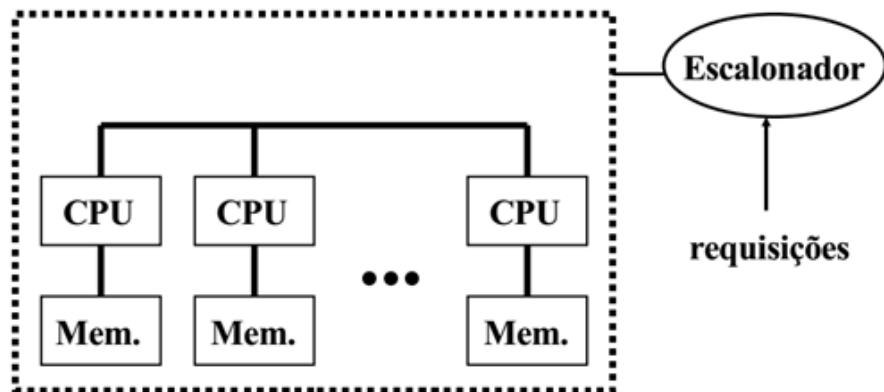
Conceitos - Arquitetura de soluções paralelas

MPPs (Multicomputadores)

- Características
 - Sistema homogêneo ou heterogêneo
 - Cada nó executa sua própria cópia do Sistema Operacional
 - Imagem única do sistema
 - Rede de interconexão: diferentes topologias
 - Fracamente acoplados
 - Escaláveis
 - Aplicações não compartilham recursos: Pode ocorrer que uma aplicação permaneça em estado de espera

Conceitos - Arquitetura de soluções paralelas

MPPs (Multicomputadores)



Conceitos - Arquitetura de soluções paralelas

Clusters

- Espaço de endereçamento
 - Não compartilhado
- Mecanismo de comunicação
 - Troca de mensagens;
- Unidades de processamento
 - Conjunto de estações de trabalho ou Pcs (Nós:
elementos de processamento = processador + memória)

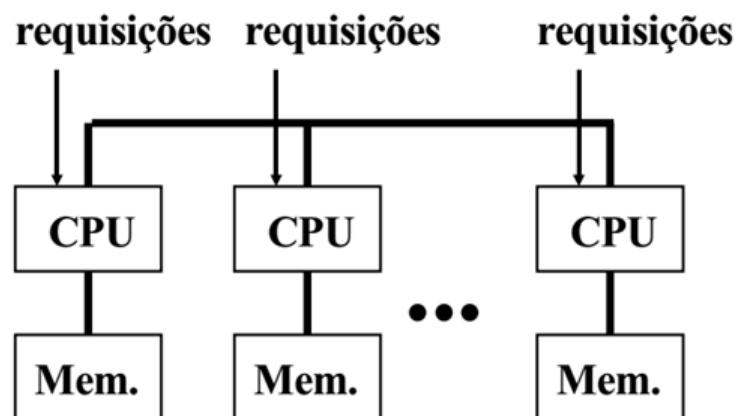
Conceitos - Arquitetura de soluções paralelas

Clusters

- Características
 - Sistema homogêneo ou heterogêneo
 - Interconexão: redes locais e tendem a ser mais lentas que no MPP
 - Não existe um escalonador centralizado
 - Cada nó tem seu próprio escalonador local
 - Possibilidade de compor um sistema de alto desempenho e um baixo custo (principalmente quando comparados com MPP).

Conceitos - Arquitetura de soluções paralelas

Clusters



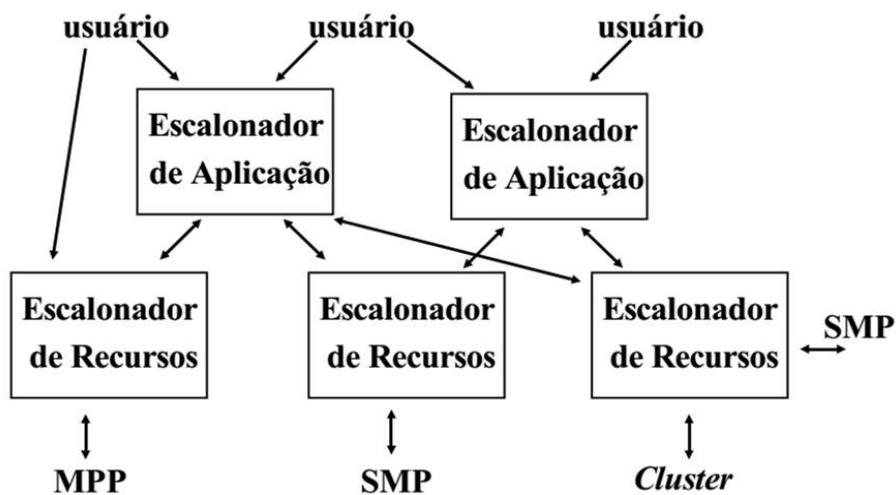
Conceitos - Arquitetura de soluções paralelas

Grids

- Utilização de computadores
 - independentes
 - geograficamente distantes
- Não têm uma imagem única do sistema a princípio (em desenvolvimento)
- Pode ser formado por PCs, SMPs, MPPs e clusters

Conceitos - Arquitetura de soluções paralelas

Grids



Soluções para processamento paralelo e distribuído de dados

Conceitos – Paradigmas de programação paralela



Ilustração de Oliver Munday

Conceitos – Paradigmas de programação Paralela

Fatores que limitam o desempenho

Código Sequencial: existem partes do código que são inherentemente sequenciais

Concorrência: o número de tarefas pode ser escasso e/ou de difícil definição.

Granularidade: o número e o tamanho das tarefas é importante porque o tempo que demoram a ser executadas tem de compensar os custos da execução em paralelo (custos de criação, comunicação e sincronização);

Balanceamento de Carga: ter os processadores o máximo ocupados (durante toda a execução) é decisivo para o desempenho global do sistema;

Conceitos – Paradigmas de programação Paralela

Modelos de programação paralela

Programação em Memória Partilhada

- Programação usando processos ou threads
- Comunicação através de memória partilhada.

Programação em Memória Distribuída

- Programação usando troca de mensagens.
- Comunicação e sincronização por troca de mensagens.

Conceitos – Paradigmas de programação Paralela

Principais Paradigmas de Programação Paralela

Diversos problemas aplicáveis

O desenvolvimento de algoritmos paralelos pode ser classificado em diferentes paradigmas;

Cada paradigma representa uma classe de algoritmos (similares):

- Master/Slave
- Single Program Multiple Data (SPMD)
- Data Pipelining
- Divide and Conquer
- Speculative Parallelism

Conceitos – Paradigmas de programação Paralela

Principais Paradigmas de Programação Paralela

A escolha do paradigma para aplicar a um dado problema é determinado pelo:

- Tipo de paralelismo inerente ao problema
- Tipo de recursos computacionais disponíveis

Conceitos – Paradigmas de programação Paralela

Master/Slave

Cada paradigma representa uma classe de algoritmos (similares):

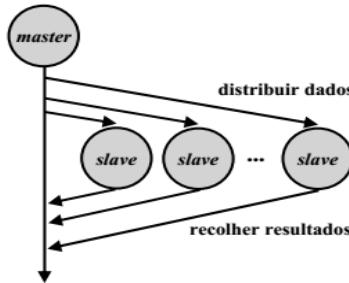
- Master/Slave
- Single Program Multiple Data (SPMD)
- Data Pipelining
- Divide and Conquer
- Speculative Parallelism

Conceitos – Paradigmas de programação Paralela

Master/Slave

Divide a computação em duas entidades distintas:

- **Master**: é o responsável por decompor o problema em tarefas, distribuir as tarefas pelos slaves e recolher os resultados parciais dos slaves de modo a calcular o resultado final
- **Slaves**: responsabilidade trivais e simples: obter uma tarefa do master, processar a tarefa e enviar o resultado de volta para o master.



Conceitos – Paradigmas de programação Paralela

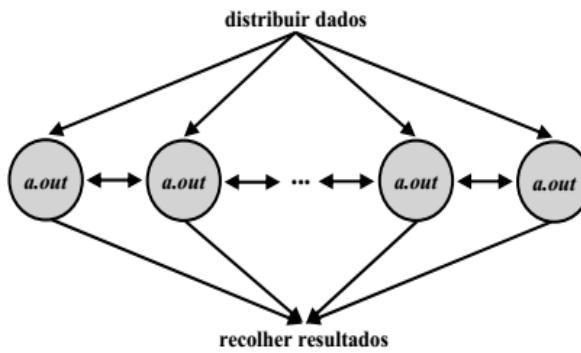
Master/Slave

- O balanceamento de carga pode ser estático ou dinâmico:
 - Estático: a divisão de tarefas é feita no início da computação;
 - Dinâmico: quando o tempo de execução das tarefas é desconhecido no início da computação;
- Existe apenas comunicação entre o master e os slaves: bons desempenhos e um elevado grau de escalabilidade
- Controle centralizado no master pode ser um problema quando o número de slaves é elevado;
- Possível aumentar a escalabilidade do paradigma considerando vários masters em que cada um controla um grupo diferente de slaves;

Conceitos – Paradigmas de programação Paralela

Single Program Multiple Data (SPMD)

- Consiste em processos que executam o mesmo programa (executável) mas sobre diferentes partes dos dados:



Conceitos – Paradigmas de programação Paralela

Single Program Multiple Data (SPMD)

- Dados devem ser bem distribuídos (mesma quantidade e regularidade);
- São lidos individualmente por cada processo ou um dos processos é o responsável por ler todos os dados e depois distribui-los para os demais;
- Os processos comunicam quase sempre com processos vizinhos e apenas esporadicamente existem pontos de sincronização global;
- Consegue bons desempenhos e um elevado grau de escalabilidade;

Conceitos – Paradigmas de programação Paralela

Single Program Multiple Data (SPMD)

- Muito sensível a falhas: no caso de falha de um nó há problemas na resolução da computação global;

Conceitos – Paradigmas de programação Paralela

Data Pipelining

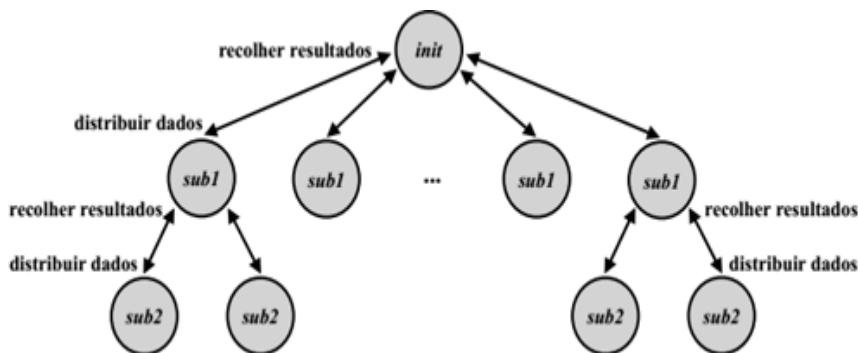
- Utiliza uma decomposição funcional do problema em que cada processo executa apenas uma parte do algoritmo completo e total;
- O padrão de comunicação é definido e simples: processos são organizados em sequência (pipeline) e cada processo só troca informação com o processo seguinte;
- Dependente muito da capacidade de balancear a carga entre as diferentes etapas da pipeline;



Conceitos – Paradigmas de programação Paralela

Divide and Conquer

- Utiliza uma divisão recursiva do problema inicial em subproblemas independentes instâncias mais pequenas do problema inicial) cujos resultados são depois combinados para obter o resultado final



Conceitos – Paradigmas de programação Paralela

Divide and Conquer

Visão dos processos como uma espécie de árvore:

- Os processos nos nós folha processam as subtarefas;
- Os restantes processos são responsáveis por criar as subtarefas e por agregar os seus resultados parciais;

Subtarefas são totalmente independentes, não há qualquer tipo de comunicação durante o processamento das mesmas;

Apenas existe comunicação entre o processo que cria as subtarefas e os processos que as executam;

Conceitos – Paradigmas de programação Paralela

Speculative Parallelism

- Utilizado quando as dependências entre os dados são tão complexas que tornam difícil explorar paralelismo usando os paradigmas anteriores;
- Uma aplicação deste paradigma é quando se utiliza simultaneamente diversos algoritmos para resolver um determinado problema e se escolhe aquele que primeiro obtiver uma solução;

Soluções para processamento paralelo e distribuído de dados

Conceitos – Exemplo práticos de soluções paralelas 'atuais'



Ilustração de Oliver Munday

Conceitos - Exemplo práticos de soluções paralelas 'atuais'

Hadoop



- Talvez seja a solução 'clássica' e mais conhecida atualmente;
- É um framework para processamento paralelo e distribuído;
- Utiliza paradigma de Master\Slave;
- É uma arquitetura do tipo cluster ou NOW;
- Possui vários componentes: HDFS, MapReduce, Yarn, Pig, Hive, Oozie, Ambari....
- Grande alteração da versão 1.0 para a versão 2.0 em diante

Conceitos - Exemplo práticos de soluções paralelas 'atuais'

Spark



- Spark também é uma plataforma para processamento distribuído;
- Pode trabalhar em conjunto com o Hadoop ou não...
- Spark não possui uma componente de armazenamento
- Processamento pode ser feito em memória e flexibiliza o pipeline de processamento de dados

Conceitos - Exemplo práticos de soluções paralelas 'atuais'

Storm



- É um sistema de computação distribuída em tempo real, por natureza;
- Utilizado em real-time analytics (Processamento de streaming de dados);
- Pode ser usado com muitas linguagens de programação;
- Capaz de processar 1 milhão de linhas por nó, por segundo;
- Integra-se com o Hadoop;
- Não possui suporte para processamento Batch;

Conceitos - Exemplo práticos de soluções paralelas 'atuais'

Flink



- É um engine de fluxo de dados em real time;
- Trata tanto fluxos em batch quanto em real time (apesar de ser orientado para streamings de dados);
- Possui APIs para Java, Scala, Python e uma API SQL;
- Possui módulos para grafos e aprendizado de máquina;
- Consegue trabalhar com granularidade de registros (linhas de uma tabela - Hadoop e Spark vão ter dificuldades neste caso);

Soluções para processamento paralelo e distribuído de dados



Ilustração de Oliver Munday

Soluções Processamento paralelo e distribuído de dados

Hadoop

- Introdução ao Hadoop;
- Componentes principais: HDFS e MapReduce;
- Entendendo o MapReduce – prática com python;
- Arquitetura do Hadoop 2.0: YARN;

Soluções para processamento paralelo e distribuído de dados

Hadoop – Introdução



Ilustração de Oliver Munday

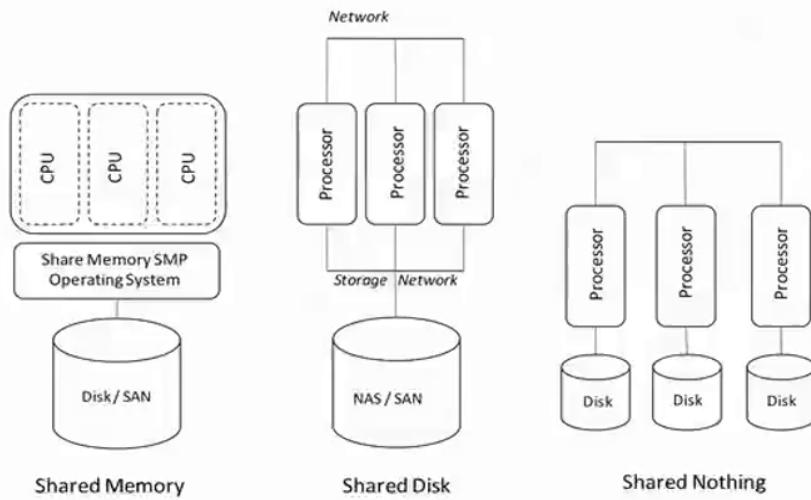
Hadoop - Introdução

Motivação:

- Estrutura de arquivos convencional não é capaz de lidar com dados massivos;
- É necessário um paradigma que comporte escalabilidade elástica;
- Bancos de dados relacionais utilizam conceitos, que às vezes, não são um requisito para dados massivos;
- Um infra estrutura tolerante a falhas e que permita computação paralela é necessária;

Hadoop - Introdução

Mudança arquitetural:



Hadoop - Introdução

Mudança arquitetural:

Requisitos:

- Demandar estrutura de computação paralela;
- Esquemas de particionamentos (*shared nothing*);
- Tolerância a falhas;
- Escalabilidade horizontal e elástica;

Hadoop - Introdução

Mudança arquitetural:

Alguns paradigmas de computação são fundamentais:

- Várias soluções vão utilizar sistemas de arquivos distribuídos;
- Esquemas de controle de réplicas e consistência dos dados;
- Algoritmos para execução paralela em conjunto com estruturas de arquivos;

Hadoop - Introdução

Visão Geral

In Memory Storage

NoSQL

BigTable

MPP

Hive

Cloud

Hadoop

HBase

Pig

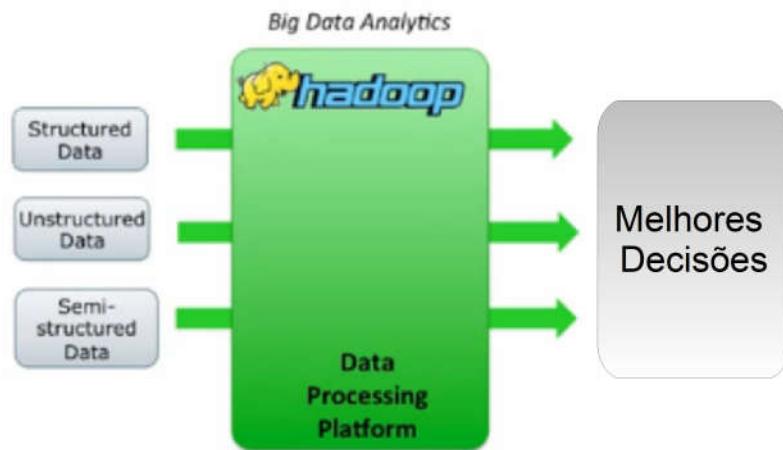
Apache

MapReduce

YARN

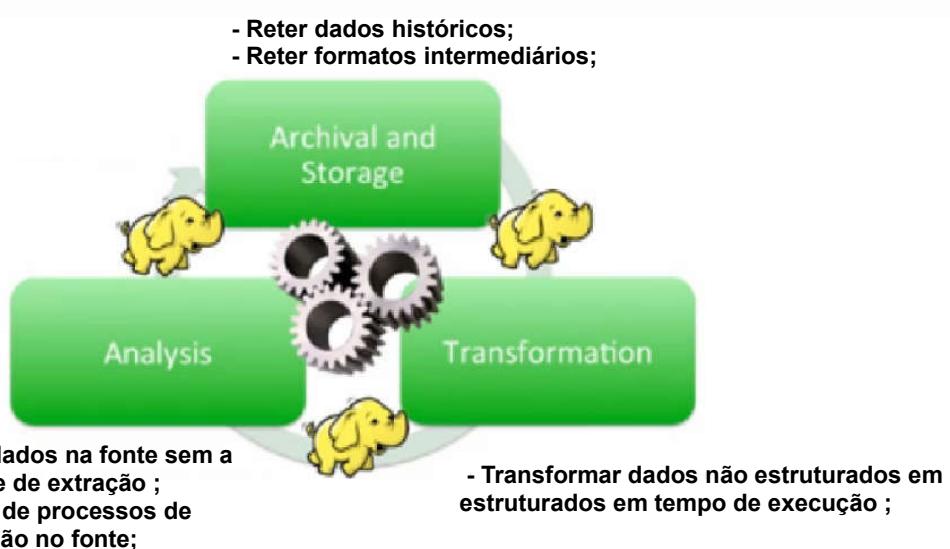
Hadoop - Introdução

Visão Geral



Hadoop - Introdução

Visão Geral



Hadoop - Introdução

Hadoop

- Plataforma para acesso a dados estruturados, semi estruturados(logs, tweets, sensor data) e não estruturados;
- Executar o ciclo de: obter, reter e analisar grandes volumes de dados;
- *Open Source* e mantida pela fundação Apache;
- Características:
 - Escalabilidade;
 - Replicação de dados distribuída;
 - Utilização de '*commodity hardware*' (NOW);

Hadoop - Introdução

Hadoop versus SGBDR

Características	SGBDR	HADOOP
Esquema	Esquema na escrita	Esquema na leitura
Desempenho	Leituras são rápidas	Escrita é rápida
Governança	Dados estruturados e padrões	Não estruturado
Processamento	Limitado	Ilimitado
Tipos de dados	Estruturado	Não estruturado
Escalabilidade	Vertical, em casos especiais é horizontal	Elástica
Exemplo de uso	- Transações ACID - <i>Operational Data Store</i> - Acesso a dados interativo	- <i>Data Discovery</i> - Armazenamento e processamento de dados massivos

Hadoop - Introdução

Hadoop Componentes – Core

Hadoop: Plataforma para processamento e armazenamento de dados massivos (larga escala)

HDFS



- Sistema de arquivos distribuídos para dados estruturados, semi-estruturados e não estruturados. Arquivos são divididos em bloco e armazenados com redundância no cluster;

Map Reduce

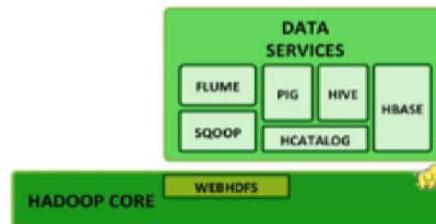
- Framework para execução de processamento paralelos em múltiplos nós de trabalho para posteriormente combinar os resultados;

YARN(Hadoop 2.0)

- Gerenciamento da execução das aplicações no cluster;

Hadoop - Introdução

Hadoop Componentes – Data Services



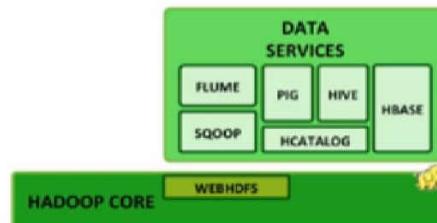
WEBHDFS

- API REST que permite acesso via HTTP ao HDFS: movimentação de arquivos(entrada e saída) exclusão de arquivos;
- Executar funções de arquivos e diretórios;
- `webhdfs://<host>:<http port>/path`

Hadoop - Introdução

Hadoop Componentes – Data Services

Flume

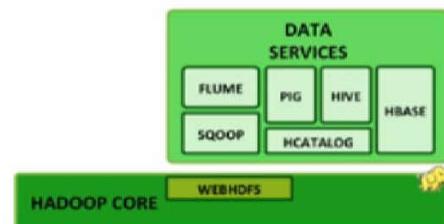


- Serviço distribuído para coleta, agregação e movimentação de *streams* de logs de dados para o HDFS;
- Executar funções de *streaming* com capacidade de recuperação e tolerante a falhas;
- Uso principal é para movimentação de arquivos de log diretamente para o HDFS/Hadoop;

Hadoop - Introdução

Hadoop Componentes – Data Services

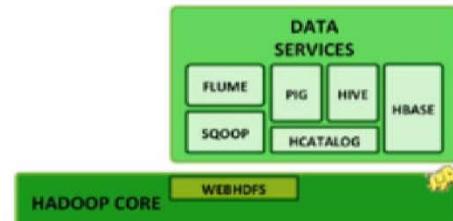
Sqoop



- Movimentação de dados para dentro do Hadoop a partir de bancos de dados relacionais e vice versa;
- Ferramentas e conectores que permitem dados de bancos de dados relacionais(Oracle, DB2, MySQL) serem armazenados e obtidos do Hadoop;

Hadoop - Introdução

Hadoop Componentes – Data Services

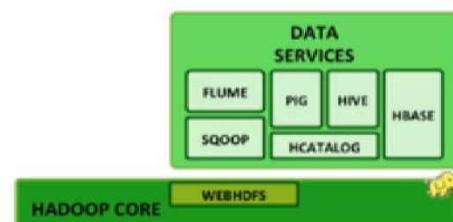


PIG

- Possibilitar a escrita de programas de transformação e movimentação de dados utilizando uma linguagem simples de script;
- Pig Latin é a linguagem que define um conjunto de transformações nos dados permitindo: junção, agregação, ordenação entre outros;
- A linguagem também pode ser estendida utilizando UDF's que podem ser escritas em JAVA e executadas a partir de comando Pig latin;

Hadoop - Introdução

Hadoop Componentes – Data Services

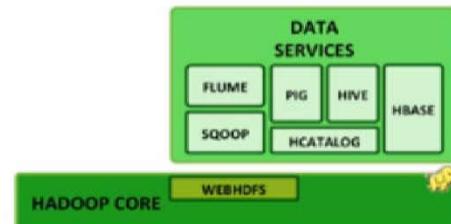


Hive

- Interface SQL para o Hadoop que possibilita summarização de dados, consultas ad-hoc e análise de grandes volumes de dados;
- Existem conectores que utilizam Hive e permitem conexões de algumas ferramentas de BI: Microstrategy, Excel, PowerPivot, Tableau e outros;
- HiveQL (HQL) é a linguagem utilizada no Hive: 'compatibilidade' com o SQL;

Hadoop - Introdução

Hadoop Componentes – Data Services

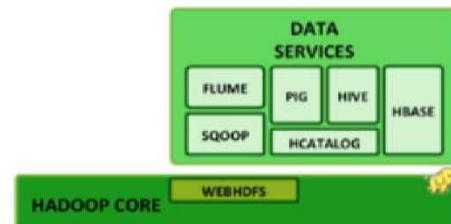


HCatalog

- Serviço de metadados que permite usuários acessarem dados no hadoop como um conjunto de tabelas sem a necessidade de fornecer detalhes sobre onde e como os arquivos estão armazenados;
- Possibilita compartilhamento e interoperabilidade entre outras ferramentas de data service:Pig, Map Reduce e Hive;
- Permite também interoperabilidade e acesso de dados para outros bancos de dados como SQL Server e Teradata(conexão hadoop via HCatalog);

Hadoop - Introdução

Hadoop Componentes – Data Services

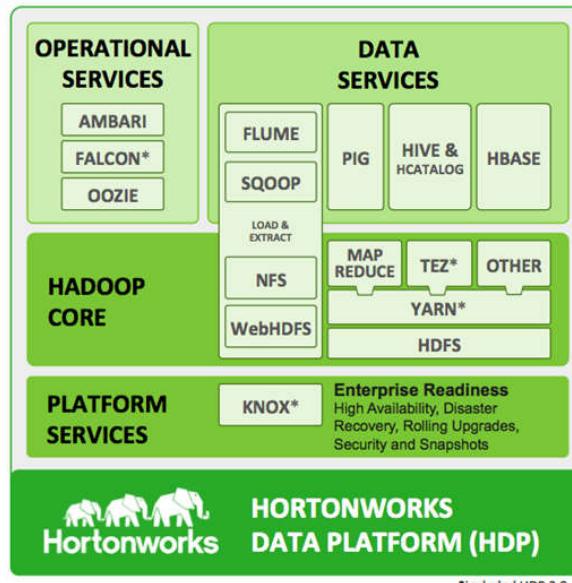


HBase

- Banco de dados NoSQL(colunar -'*big table clone*') para utilização de dados de forma interativa (não batch) ;
- Comumente utilizando para servir aplicações inteligentes: recomendações de produtos, predição de comportamento de usuários;

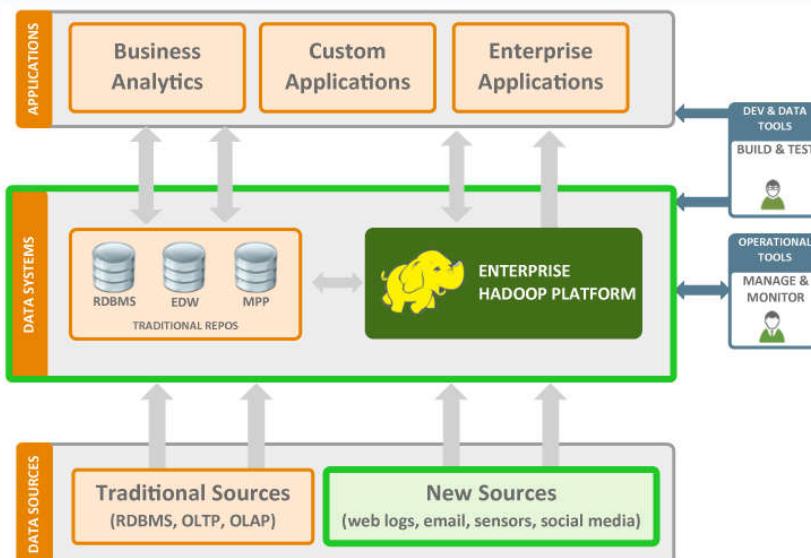
Hadoop - Introdução

Hadoop - Hortonworks



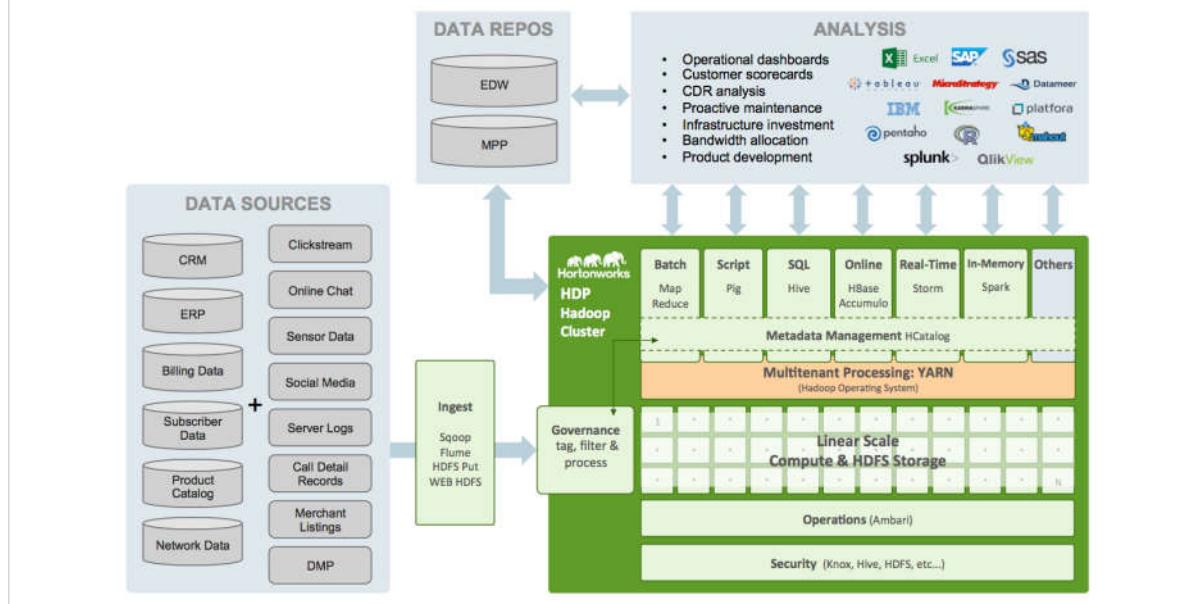
Hadoop - Introdução

Hadoop - Hortonworks – BI Approach



Hadoop - Introdução

Hadoop - Hortonworks – Another Approach



Soluções para processamento paralelo e distribuído de dados

Hadoop – Componentes Principais: HDFS



Ilustração de Oliver Munday

Hadoop – Componentes principais: HDFS

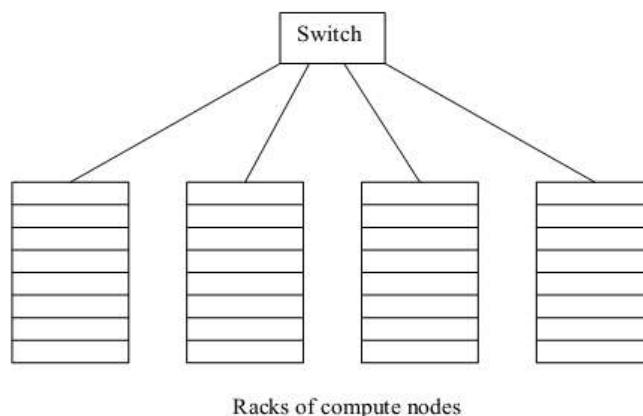
Histórico:

- Processamento intensivo era feito em hardware especializado (processadores, cache, discos e memória);
- A Web e o *Big Data* exigem processamento intensivo, mas em outra estrutura de hardware:
 - Centenas ou milhares de computadores em rede (nós);
 - Operação destes computadores de forma mais ou menos independente;
 - Cada um dos nós é um '*commodity hardware*' – custo reduzido;
 - A estrutura em geral é tolerante a falhas;
 - Utilizam sistemas de arquivos especializados;

Hadoop – Componentes principais: HDFS

Organização física da estrutura:

- A organização física destas máquinas pode seguir este exemplo:



Hadoop – Componentes principais: HDFS

Características DFS:

- Arquivos devem ser 'grandes', gigabytes, pelo menos; Arquivos menores não fazem sentido no *DFS*;
- Arquivos no *DFS* são raramente atualizados (*write-once-read-many*). Adicionalmente dados são adicionados para os arquivos (periodicidade, processamento batch);
- Arquivos são divididos em partes ('*chunks*' ou blocos), normalmente 64 megabytes e replicados para, pelo menos, 3 nós (em *racks* diferentes);
- Os Itens acima são customizáveis;

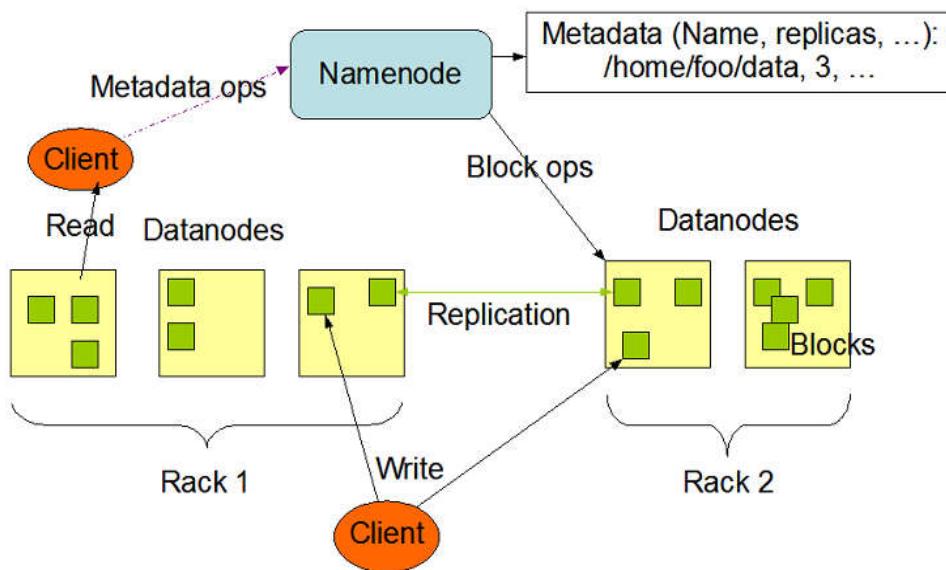
Hadoop – Componentes principais: HDFS

Características *DFS*:

- As informações dos blocos e replicas é controlado utilizando metadados e com um figura central no cluster: '*name node*' ou '*master node*';
- *Name node*:
 - Gerencia o sistema de arquivos(rélicas, blocos, nós e *racks*): abrir, fechar, renomear arquivos;
 - Gerencia o acesso dos clientes ao arquivos;
- Os outros nós do *cluster* são chamados de '*data node*' ou '*slave node*';
 - Executam as operações enviados pelo '*Name node*': criação, exclusão e replicação de blocos;

Hadoop – Componentes principais: HDFS

Características *DFS*:



Hadoop – Componentes principais: HDFS

Características DFS:

- Possuem regras de sistemas de arquivos: *rack*, '*data node*', *namespaces*, diretórios e arquivos;
- Além disto o DFS gerencia os blocos e sua distribuição/replicação nos '*data nodes*';
- Padrão de réplicas 1/3(forá do rack) e 2/3(no rack);
- O '*name node*' periodicamente recebe um relatório de blocos do '*data node*';

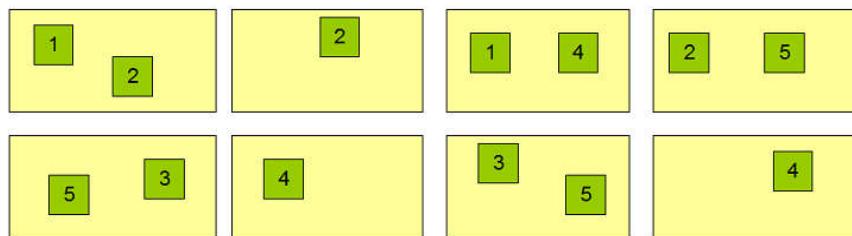
Hadoop – Componentes principais: HDFS

Características DFS:

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



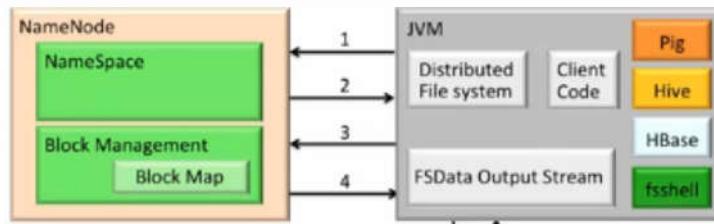
Hadoop – Componentes principais: HDFS

Características DFS:

- O '*name node*' é replicado para outra estrutura no *DFS* e também possui *log* de alterações;
- A estrutura de arquivos é, tipicamente, mantida em memória;
- '*Name node*' também faz rebalanceamento de carga: elevada demanda para um bloco, por exemplo;

Hadoop – Componentes principais: HDFS

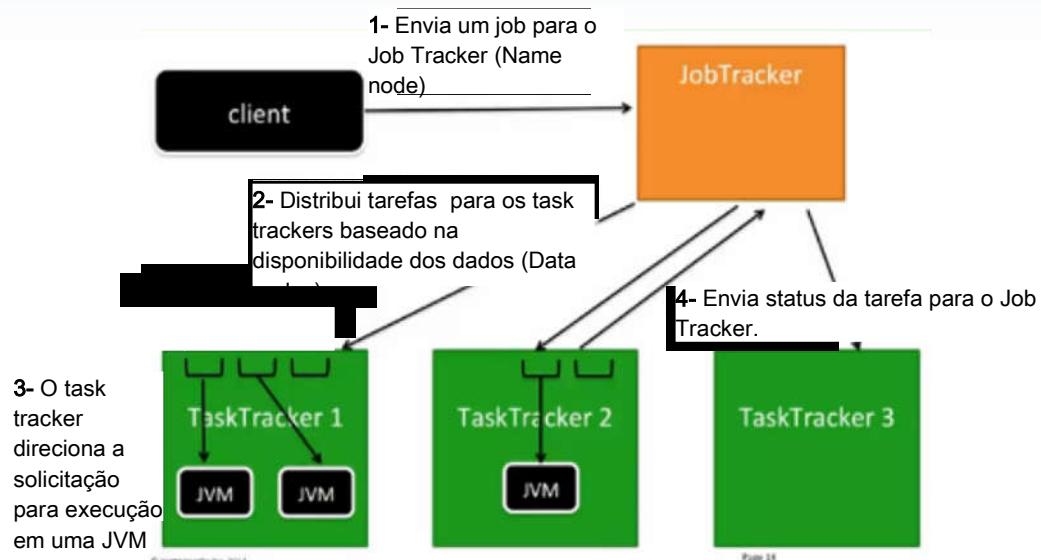
Hadoop - Exemplo RPC Hadoop 1.0



O name node não é utilizado para o acesso direto aos dados;

Hadoop – Componentes principais: HDFS

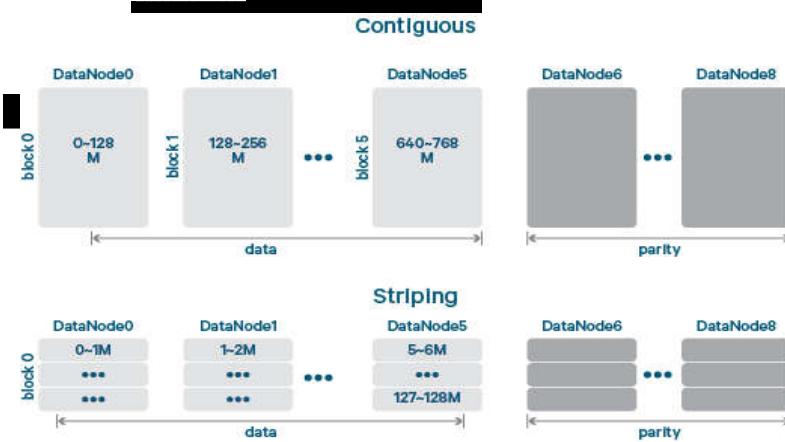
Hadoop - Organização de backend Hadoop 1.0



Hadoop – Componentes principais: HDFS

Novidades HDFS 3.0

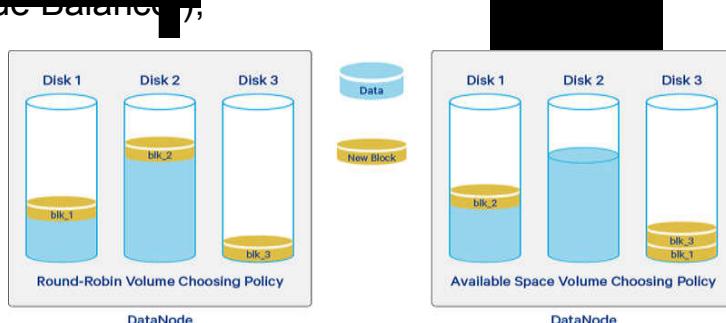
- Tolerância a falha: o ~~modo de reparo~~ do bloco foi substituída por um algoritmo chamado *Erasure Coding*;
- Utiliza o algoritmo Reed-Solomon – similar ao RAID6;



Hadoop – Componentes principais: HDFS

Novidades HDFS 3.0

- Storage: antes era necessário um acréscimo de +200% em função da replicação. Agora é necessário apenas 50% em função do esquema da paridade;
- Número de Name Nodes: agora pode ter mais de 2 name nodes;
- Melhoria na distribuição de volumes internas do Data Node(Intra-DataNode Balancer);



Hadoop – Componentes principais: HDFS

Implementações DFS:

'*Google File System*' (*GFS*), implementação original e primeira a ficar 'famosa';

Hadoop Distributed File System (HDFS), *open-source*. É disponibilizada pela fundação apache e implementada em java;

CloudStore, é outra implementação *DFS open-source*, originalmente desenvolvida pela Kosmix.

Hadoop – Componentes principais: HDFS

Atividade

Demonstração utilizando os comando FSShell e HUE da Hortonworks

Curso: Ciência de dados e Big Data

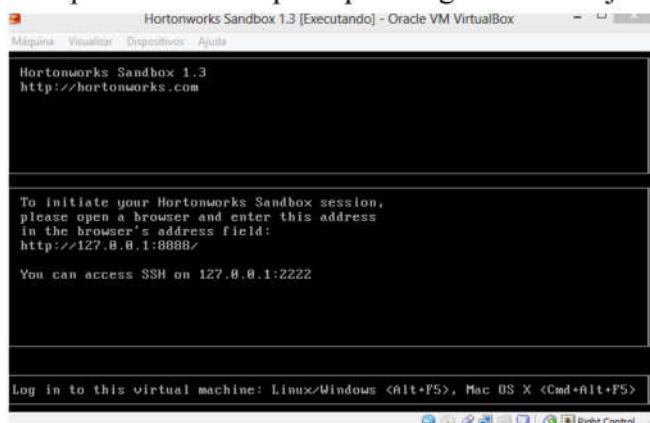
Professor: Cláudio Lúcio

Atividade Prática sobre HUE e FSSHELL

- 1) Para utilização das práticas com o Hadoop vamos usar uma sandbox (máquina virtual) fornecida pela HortonWorks. Esta máquina virtual possui os seguintes produtos instalados:

Esta é a versão 2.1 do produto e pode ser obtida em:
<http://hortonworks.com/products/hortonworks-sandbox/>

- 2) Inicialize a máquina virtual. Espere que a seguinte tela seja exibida:



- 3) Pressione Alt+F5 e faça o login com o usuário *root* e senha indicada:*hadoop*;
- 4) Para listar o conteúdo dos arquivos:

```
hadoop fs -ls /
```

- 5) Inicialmente vamos fazer a criação de uma estrutura de diretório no cluster:

```
hadoop fs -mkdir /dados
hadoop fs -ls /
hadoop fs -ls /dados
hadoop fs -mkdir /dados/bigdata
hadoop fs -ls /dados
```

- 6) Testando a exclusão de um diretório:

```
hadoop fs -rmr /dados/bigdata
hadoop fs -ls /dados
hadoop fs -mkdir /dados/bigdata
hadoop fs -ls /dados
```

- 7) Adicionando um arquivo externo para o cluster:

```
cd root
ls
hadoop fs -ls /dados/bigdata
hadoop fs -put /root/start_ambari.sh /dados/bigdata
hadoop fs -ls /dados/bigdata
```

- 8) Copiando arquivos no cluster:

```
hadoop fs -ls /apps/hive/warehouse/sample_07
hadoop fs -ls /dados/bigdata
hadoop fs -cp /apps/hive/warehouse/sample_07/sample_07.csv /dados/bigdata
hadoop fs -ls /dados/bigdata
```

9) Obtendo arquivos do cluster:

```
cd root
ls
hadoop fs -ls /dados/bigdata
hadoop fs -get /dados/bigdata/sample_07.csv /root
cd root
ls
rm sample_07.csv
ls
```

10) Listando o conteúdo de um arquivo:

```
hadoop fs -cat /dados/bigdata/sample_07.csv
```

11) HUE: Outra forma de acesso:

Abra um navegador e execute a url <http://localhost:8888> ;

Clicando no ultimo ícone a seguinte tela será apresentada:

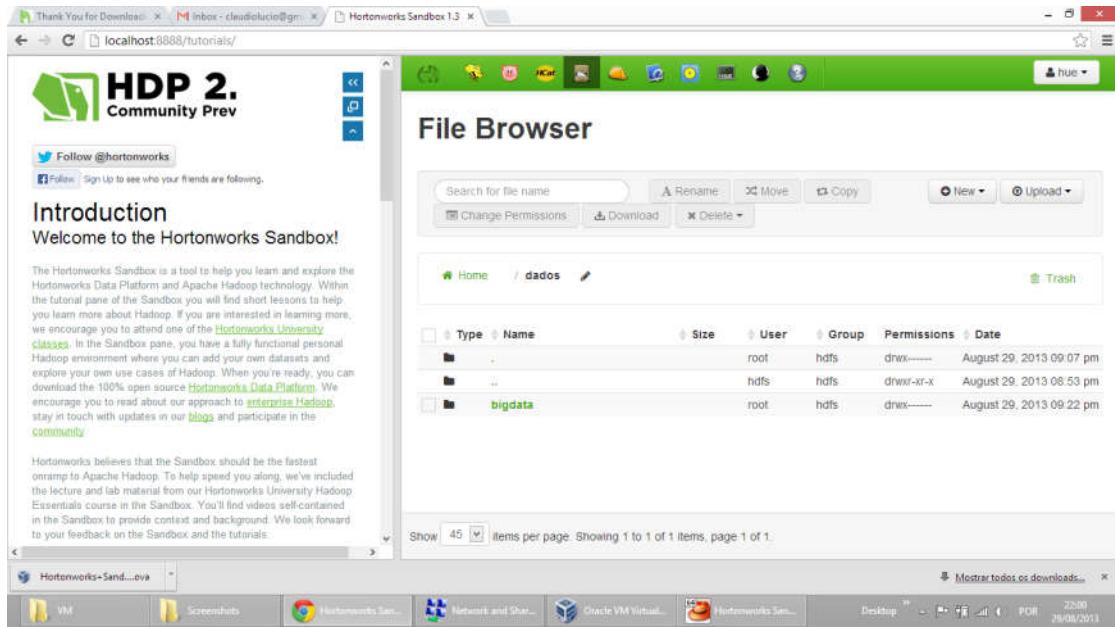


Esta ferramenta se chama HUE e permite interações com o cluster hadoop. Inclue várias ferramentas de fácil utilização, dentre elas: Pig, Hive, Hbase, Jobs Map Reduce, dentre outros.

As aplicações HUE são totalmente Web, permitindo que se faça execute as aplicações de acordo com os botões abaixo:



- 12) No caso de selecionarmos o quinto botão ele exibe uma aplicação que o “File Browser” . As atividades realizadas anteriormente via linha de comando podem ser feitas utilizando esta interface.



- 13) Navegue na estrutura e veja o arquivo anteriormente carregado;
 14) Exclua o arquivo start_ambari.sh utilizando o botão “Delete”;
 15) Faça também a exclusão do arquivo sample_07.csv;
 16) Acesse o seguinte caminho: /apps/hive/warehouse/ ;
 17) Copie o arquivo sample_07.csv utilizando o botão “Copy”(não se esqueça de marcar o arquivo antes de clicar no botão);
 18) Para o novo destino indique o caminho: /dados/bigdata

Soluções para processamento paralelo e distribuído de dados

Hadoop – Componentes Principais: Map Reduce



Ilustração de Oliver Munday

Hadoop – Componentes principais: Map Reduce

Origens:

- Patente original é do Google, mas é utilizado em várias outros sistemas de computação paralela;

- A ideia é derivada da programação funcional:

- *Map* e *reduce* são dois tipos de funções comuns;

- *Map*:

- Aplica um função ou operação para cada elemento em uma lista; Ex.: multiplicação por 2;

[1,2,3,4] Map function → [2,4,6,8,]

- Não altera o dado original. Evita o princípio '*Shared Data*';
- Pode ser executado de forma paralela;

Hadoop – Componentes principais: Map Reduce

Origens:

- A ideia é derivada da programação funcional:

- *Reduce*:

- É uma função de agrupamento ou compressão;
- Aplica uma função em conjunto de dados reduzindo para um simples valor;
- Pode ser executado de forma paralela;
- Ex.: [2,4,6,8,] → Reduce function → [20]

De forma geral:

- O algoritmo pode ser usado sempre que houver uma lista;
- Para cada elemento da lista uma função que a transforme;
- Outra função que possa ser aplicada ao conjunto de dados transformados de forma a agregá-los;

Hadoop – Componentes principais: Map Reduce

Detalhes de funcionamento:

- A implementação do algoritmo é utilizada para realizar computação no *DFS* para arquivos 'grandes' e com execução tolerante a falha;
- É necessário escrever as duas funções: *Map* e *reduce*;
- O sistema lida com os demais detalhes:
 - Execução paralela;
 - Coordenação de tarefas (*Map* e *reduce*);
 - Lidar com a tolerância a falhas;

Hadoop – Componentes principais: Map Reduce

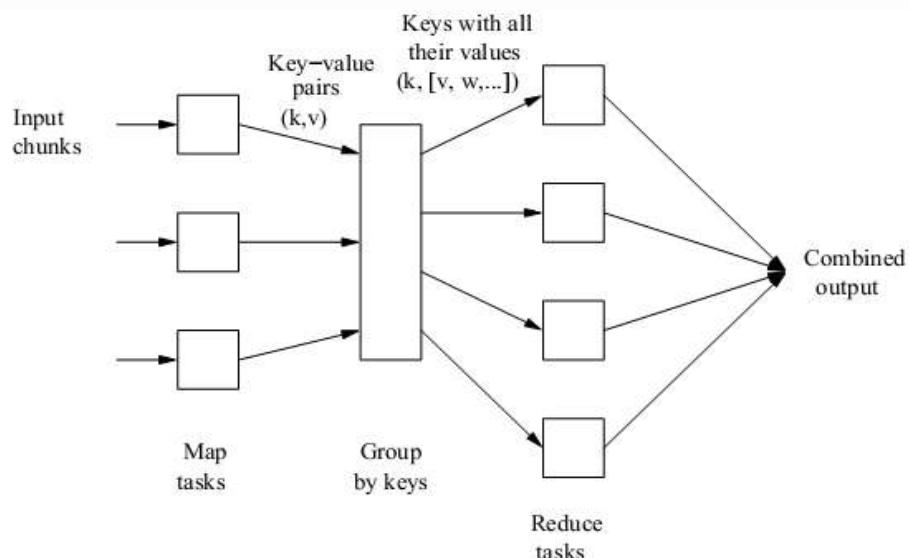
Detalhes de funcionamento:

Seguintes passos de execução:

- Um arquivo é armazenado no *DFS* com vários blocos em vários nós e *racks*;
- Um conjunto de tarefas do tipo *Map* é criado, para cada *Map* existe um ou mais blocos que serão processados; As tarefas *Map* vão transformar o dado em um estrutura chave valor ou tuplas;
- As estruturas chave valor são coletadas pelo controlador *master* e ordenadas pelas suas chaves;
- As chaves serão agrupadas e divididas para as tarefas do tipo *Reduce* (uma chave, com vários valores será processado por uma e só uma tarefa *Reduce*);
- As tarefas do tipo *Reduce* vão então agrupar os dados pelas chave, uma por vez.

Hadoop – Componentes principais: Map Reduce

Detalhes de funcionamento:



Hadoop – Componentes principais: Map Reduce

Detalhes de funcionamento:

Tarefas *Map*:

- Um bloco no DFS possui vários 'membros' que serão processados. Cada membro só pertence a apenas 1 bloco;
- A estrutura chave valor é importante pois permite a execução de várias tarefas *Map* em paralelo;
- A função *Map* é responsável por converter os dados para a estrutura chave valor;
- Chaves não são 'chave', no sentido estrito, não precisam ser únicas;

Hadoop – Componentes principais: Map Reduce

Detalhes de funcionamento:

Tarefas *Map*:

Exemplo clássico - Contar o número de palavras em uma coleção de documentos:

- Cada tarefa *Map* lê um documento ou vários documentos;
- Cada palavra será considerada uma chave:
 w_1, w_2, \dots, w_n ;
- A seguinte estrutura chave valor será criada:

Sugestões para melhorar a função *Map*
???

Hadoop – Componentes principais: Map Reduce

Detalhes de funcionamento:

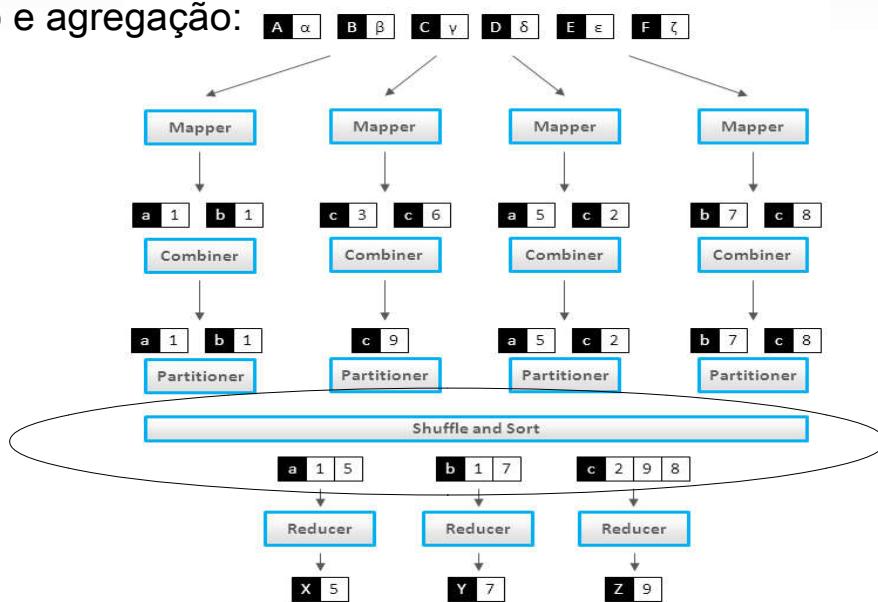
Agrupamento e agregação:

- São realizados, sempre da mesma forma, independente do que as funções *Map* e *Reduce* façam:
- O “*Name node*” controla este processo:
 - O número de tarefas *reduce* já são conhecidos: r (pode ser previamente determinada);
 - Cria um função *hash* (0 até $r-1$) que é aplicada nas chaves;
 - Cada chave gerada pela tarefa *Map* é então gravada em um dos r arquivos locais;
 - Após todas as tarefas *map* serem finalizadas o “*master node*” faz um *merge* dos arquivos e que são então destinadas para as tarefas de *reduce*.

Hadoop – Componentes principais: Map Reduce

Detalhes de funcionamento:

Agrupamento e agregação:



Hadoop – Componentes principais: Map Reduce

Detalhes de funcionamento:

Tarefas *Reduce*:

- Para cada chave k , a tarefa *reduce* recebe um conjunto de pares na forma $(k,[v_1,v_2,\dots,v_n])$ oriundos de várias tarefas *map* na forma $(k,v_1)(k,v_2)\dots(k,v_n)$;
- A tarefa *reduce* deve combinar os valores de alguma forma;

Hadoop – Componentes principais: Map Reduce

Detalhes de funcionamento:

Tarefas *Reduce*:

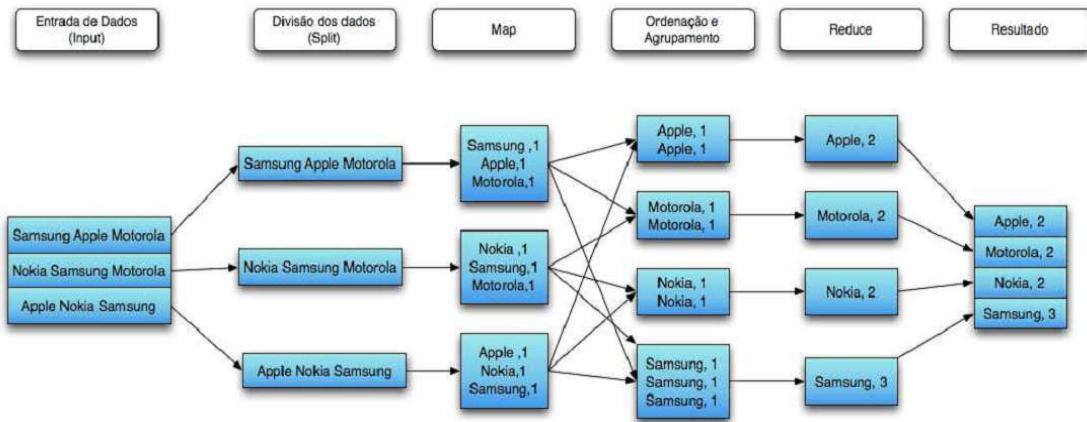
Exemplo clássico - Contar o número de palavras em um coleção de documentos:

- A função *reduce* apenas faz a soma dos valores para cada uma das chaves w_n
- Desta forma a saída da função *reduce* será um conjunto de chaves-valor na forma (w,m) ;
- Em que m é o total de vezes que a palavra w aparece em todos os documentos;

Hadoop – Componentes principais: Map Reduce

Detalhes de funcionamento:

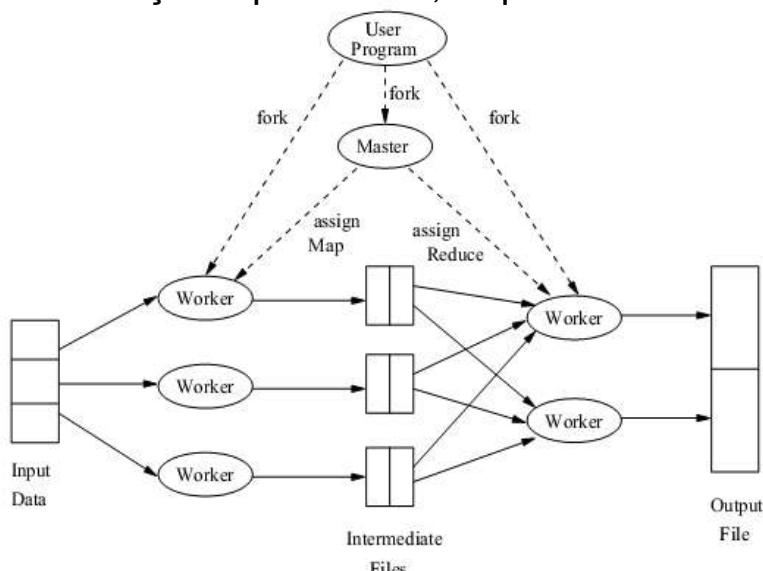
Exemplo clássico:



Hadoop – Componentes principais: Map Reduce

Detalhes de funcionamento:

Outras considerações: processos, arquivos e tarefas



Hadoop – Componentes principais: Map Reduce

Detalhes de funcionamento:

Outras considerações:

Revisão: processos, arquivos e tarefas

- O '*Name node*' é responsável pela criação das tarefas de Map e Reduce nos '*data nodes*' ;
- É ideal que se crie um tarefa *map* para cada bloco;
- O número de tarefas *reduce* deve ser mais reduzido, a intenção é evitar a explosão do número de arquivos gerados pela tarefa *map* (um arquivo para cada *reduce*);
- O '*name node*' acompanha a execução das tarefas (em espera, executando, completo); Quando um processo termina ele comunica para o '*name node*';

Hadoop – Componentes principais: Map Reduce

Detalhes de funcionamento:

Outras considerações:

Revisão: processos, arquivos e tarefas

- Cada tarefa *map* processa vários blocos de vários arquivos;
- Os arquivos gerados pelo *map* são gravados localmente e o '*name node*' armazena todos estes dados: nomes dos arquivos e tamanho;
- Uma tarefa *reduce* recebe todos os arquivos intermediários para o processamento da função *reduce*;

Hadoop – Componentes principais: Map Reduce

Resumo Python:

Variáveis e Substituição:

```

lista = [ 1, 2, "texto", 3.5 ]
print lista[ 0 ] # imprime 1
print lista[ 1 : 2 ] # imprime [ 2, "texto" ]
print lista[ -1 ] # imprime [ 1, 2, "texto" ]
lista += [ "novo" ]
print lista # imprime [ 1, 2, "texto", 3.5, "novo" ]

tupla = ( 1, 2, "texto", 3.5 ) # Elementos não podem ser alterados!
print tupla[ 0 ] # imprime 1
print tupla[ 1 : 2 ] # imprime ( 2, "texto" )
print tupla[ : -1 ] # imprime ( 1, 2, "texto" )
tupla += ( "novo", )
print tupla # imprime ( 1, 2, "texto", 3.5, "novo" )

dicionario = { "chave": "valor", "c2": "v2" }
print dicionario[ "chave" ] # imprime valor

newstring1 = "string '%s' int=%d float=%03.2f" % ( "txt", 12, 4.56 )
newstring2 = "chave=%(chave)s c2=%(c2)s" % dicionario
newstring3 = "chave=%s c2=%s" % ( dicionario[ "chave" ],
                                 dicionario[ "c2" ] )

```

Controle de Fluxo e Laços:

```

if a > b and a < c:
    print "a entre b e c"
elif a > c:
    print "a maior que c"
else:
    print "a menor ou igual a b ou igual a c"

for elemento in lista:
    print "elemento: %s" % elemento

coordenadas = [ ( 0, 0, 0 ), ( 1, 0, 0 ), ( 0, 1, 0 ), ( 0, 0, 1 ) ]
for x, y, z in coordenadas:
    print "Ponto: x=%d, y=%d, z=%d" % ( x, y, z )

loop = 1
while loop:
    resultado = faca_acao()
    if resultado < 0:
        break # Para o laço
    else resultado > 0:
        continua # Volta para o começo do laço
    print "teste"

```

Funções:

```

def funcao( p1, p2="Valor Padrao" ):
    print "p1: %s" % p1, "p2: %s" % ( p1, p2 )

def f_param_variaveis( p1, *args ):
    print "p1: %s" % p1
    for arg in args:
        print "arg: %s" % arg

def f_param_nome_variaveis( p1, **args ):
    print "p1: %s" % p1
    for p_name, p_value in args:
        print "arg: %s=%s" % ( p_name, p_value )

```

Classes:

```

class A:
    atributo = 1
    __privado = 123
    def __init__( self ):
        self.atributo = valor
        self.__metodo_privado()
    def __metodo_privado( self ):
        print "chamando metodo privado"

class B:
    atributo = 2
    __privado = 123
    def __init__( self ):
        self.novo_atributo = 2
class C( A, B ):
    def __init__( self ):
        B.__init__( self )
class D( B, A ):
    def __init__( self ):
        B.__init__( self )
a = A( 1 )
b = B( 2 )
c = C( 3 )
d = D( 4 )

print a.atributo # imprime 1
print b.atributo # imprime 2
print c.atributo # imprime 1; heranca mult. (A,B) A sobrepoese a B
print d.atributo # imprime 2; heranca mult. (B,A) B sobrepoese a A

```

Módulos e Espaço de Nomes:

```

import urllib
url = "http://www.unicamp.br/* + urllib.quote( "index.html" )
conteudo = urllib.urlopen( url ).read()

# importa simbolos para espaço de nomes atual:
from urllib import *
url = "http://www.unicamp.br/* + quote( "index.html" )
conteudo = urlopen( url ).read()

```

Hadoop – Componentes principais: Map Reduce

Exemplo - Mincemeat:

```

#!/usr/bin/env python
import mincemeat

data = [ "Humpty Dumpty sat on a wall",
         "Humpty Dumpty had a great fall",
         "All the King's horses and all the King's men",
         "Couldn't put Humpty together again",
         ]

def mapfn(k, v):
    for w in v.split():
        yield w, 1

def reducefn(k, vs):
    result = 0
    for v in vs:
        result += v
    return result

s = mincemeat.Server()

# The data source can be any dictionary-like object
s.datasource = dict(enumerate(data))
s.mapfn = mapfn
s.reducefn = reducefn

results = s.run_server(password="changeme")
print results

```

Hadoop – Componentes principais: Map Reduce

Exemplo – Mincemeat:

Servidor – '*Name node*:

```
python example.py
```

Clientes – '*data node*:

```
python mincemeat.py -p changeme [server address]
```

Resultado:

```
{'a': 2, 'on': 1, 'great': 1, 'Humpty': 3, 'again': 1, 'wall': 1, 'Dumpty': 2, 'men': 1, 'had': 1, 'all': 1, 'together': 1}
```

Hadoop – Componentes principais: Map Reduce

Tipo de processamentos:

Map reduce não é uma solução para qualquer tipo de problema. Ex. de exceção: site de comércio eletrônico;

Pode ser utilizado:

- Encontrar palavras chaves;
- Operações que envolvam multiplicação de matrizes;
- Operações de álgebra relacional:
 - Seleção - σ
 - Projeção - π
 - União, interseção e diferença
 - Natural *join* -
 - Agrupamentos e agregações

Hadoop – Componentes principais: Map Reduce

Natural Join – SQL:

Suponha a duas relações $R(A,B)$ e $S(B,C)$

Função *Map*:

- Para cada tupla a, b de R produza um membro chave valor $(b, (R, a))$;
- Para cada tupla de b, c de S produza um membro chave valor $(b, (S, c))$;

Função *Reduce*:

- Cada chave b deve ser associada com ambos os itens (R, a) e (S, c) ;
- A saída para a chave b deve ser $(b, [(a_1, b, c_1), (a_2, b, c_2), \dots])$

Hadoop – Componentes principais: Map Reduce

Atividade

- Exemplo de contagem de palavras para textos
- Exemplo de implementação de natural join e agrupamentos;
- Exemplo de execução de wordcount no Hadoop

Curso: Ciência de dados e Big Data

Professor: Cláudio Lúcio

Atividade Prática sobre Map Reduce

Um conjunto de comentários sobre os produtos da sua empresa foi disponibilizado (textos em inglês). Todos eles são comentários positivos sobre os produtos. O seu gerente solicitou que você prepare uma solução que possa atender os seguinte requisitos:

- I. Armazenar os textos e extrair 15 palavras chaves que são mais citadas pelos clientes;
- II. Sua companhia possui 4 milhões de clientes e há uma interação com estes clientes pelo menos uma vez por mês;
- III. Para cada mês deve haver uma atualização nas palavras ;
- IV. Em média um cliente interage com a companhia 1 vez por mês; Cada interação gera um arquivo texto a partir do sistema fonte de informação que armazena este dado;
- V. Algumas palavras deve ser descartadas: 'by', 'above', 'all', 'none', 'nobody'.....

Sua missão é fazer uma prova de conceito para o seu gerente. Ele ouviu falar de processamento paralelo e leu um artigo na '*Computer magazine*' falando sobre o movimento NoSQL: quer entender no cenário da empresa como isto poderia funcionar. Ele já solicitou um extração de 1000 arquivos sobre os comentário dos clientes. Em uma reunião foi definido que você deve entregar um exemplo funcionando amanhã

Você deve então:

- a- Usar uma implementação simples e rápida;
- b- Permitir que isto seja executada em 50 máquinas na empresa;
- c- No final gerar um arquivo com a contagem das palavras;

Vamos começar com a implementação:

1. Faça a instalação do python 2.7;
2. Crie o diretório exerc\;
3. Crie o diretório exerc\textos;
4. Copie o arquivo zipado(2.1 - textos.zip) para o diretório criado no passo anterior;
5. Copie o arquivo python do mincemeat para o diretório exerc\
<https://github.com/michaelfairley/mincemeatpy>
6. Crie um arquivo texto e digite a primeira parte do código – Importação dos arquivos:

```
import mincemeat
import glob
import csv

text_files = glob.glob('Exerc\\textos\\*')

def file_contents(file_name):
    f = open(file_name)
    try:
        return f.read()
    finally:
        f.close()

source = dict((file_name, file_contents(file_name))for file_name in text_files)
```

Estas parte faz a importação dos dados e gera um objeto 'source' que é do tipo dicionário: nome do arquivo e conteúdo do arquivo;

7. Faça a implementação do método *map*:

```

def mapfn(k, v):
    print 'map ' + k
    from stopwords import allstopwords
    for line in v.splitlines():
        for word in line.split():
            if (word not in allstopwords):
                yield word, 1

```

Neste caso esta sendo utilizada um implementação rápida e simples;

8. Faça a implementação do método *reduce*:

```

def reducefn(k, v):
    print 'reduce ' + k
    return sum(v)

```

9. Utize agora o *mincemeat* ele vai simular o papel da DFS e do '*name mode*':

```

s = mincemeat.Server()
# A fonte de dados pode ser qualquer objeto do tipo dicionário
s.datasource = source
s.mapfn = mapfn
s.reducefn = reducefn
results = s.run_server(password="changeme")
w = csv.writer(open("Exerc\\RESULT.csv", "w"))
for k, v in results.items():
    w.writerow([k, v])

```

10. Salve o nome do arquivo como exerc21.py;

11. Vamos fazer a execução paralela deste código:

a- server: python exerc21.py

b- Crie dois ou três clientes com o comando:

python mincemeat.py -p changeme localhost

Para executar remoto,faça a instalação do python e mincemeat conforme passos 1 e 4, respectivamente;

12. Veja o arquivo gerado

Curso: Ciência de dados e Big Data

Professor: Cláudio Lúcio

Atividade Prática sobre Map Reduce

Neste caso vamos implementar um junção e agrupamento de duas tabelas que estão representadas por dois arquivos:

I. Vendas:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Store 01	Customer 144	80759 Books	29/11/2006 06:48	9	80,5	1												
2	Store 04	Customer 1716	56810 Clothing	14/01/2007 21:15	5	42,3	2												
3	Store 04	Customer 129	60636 Movies	17/12/2005 13:26	5	92,9	3												
4	Store 10	Customer 802	82105 Home/Garden	06/03/2007 19:57	4	70,6	4												
5	Store 14	Customer 1	52349 Clothing	09/12/2005 21:52	8	76,6	5												
6	Store 07	Customer 1190	43024 Movies	31/05/2005 11:18	7	90,2	6												
7	Store 14	Customer 1268	82054 Movies	08/02/2006 20:58	1	26,1	7												
8	Store 10	Customer 1433	57181 Electronics	19/06/2007 18:06	9	66,4	8												
9	Store 05	Customer 103	15211 Health	11/02/2005 18:14	9	83,5	9												
10	Store 01	Customer 1871	72478 Electronics	24/06/2006 10:10	6	77,4	10												
11	Store 07	Customer 1193	50985 Movies	14/09/2006 02:40	2	69,8	11												
12	Store 07	Customer 291	97284 Movies	01/07/2008 16:59	7	42,7	12												
13	Store 14	Customer 751	30213 Home/Garden	04/09/2005 15:43	6	39,3	13												
14	Store 13	Customer 1736	96441 Sports	27/11/2006 21:44	9	89,8	14												
15	Store 13	Customer 1740	51940 Sports	16/02/2005 10:18	4	41,2	15												
16	Store 14	Customer 307	96842 Clothing	18/01/2008 16:03	3	14,9	16												
17	Store 09	Customer 30	89799 Clothing	21/03/2007 13:17	1	81,2	17												
18	Store 15	Customer 1837	84658 Toys	04/11/2005 06:00	8	76,3	18												
19	Store 02	Customer 1641	62589 Toys	11/09/2006 22:59	4	12,7	19												
20	Store 03	Customer 191	18438 Electronics	16/09/2008 22:40	9	14,2	20												
21	Store 13	Customer 1289	53573 Sports	07/11/2008 03:51	8	17,1	21												
22	Store 10	Customer 1575	50955 Home/Garden	13/01/2008 05:38	9	27	22												
23	Store 09	Customer 483	92160 Home/Garden	14/08/2007 19:50	5	63,3	23												

II. Filiais:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Store 01	CO_BHS																	
2	Store 04	CO_POA																	
3	Store 10	CO_NAS																	
4	Store 14	CO_CPT																	
5	Store 07	CO_NHS																	
6	Store 05	CO_BET																	
7	Store 13	NA_GFT																	
8	Store 09	GH_DSE																	
9	Store 15	DF_FGE																	
10	Store 02	QW_ERT																	
11	Store 03	MN_DFG																	
12	Store 12	YU_GHJ																	
13	Store 11	EV_MKL																	
14	Store 08	TR_RTE																	
15	Store 06	PO_FGO																	

Neste caso o que se deseja é fazer um junção dos dois arquivos para que seja apresentado um resultado que seria: Código da filial (arquivo Filial - campo1), descrição da filial (arquivo Filial - campo2) e total de itens pedidos (arquivo Filial – campo6);
Na verdade seria um SQL com join e um group by:

```
Select cod_filial, des_filial, sum(qtd_item) as total
from vendas inner join filial on(filial.cod_filial = vendas.cod_filial)
group by cod_filial, des_filial
```

Implementação:

1. Faça a instalação do python 2.7;

2. Crie o diretório exerc\;
3. Crie o diretório exerc\join;
4. Copie o arquivo zipado(2.2 Join.zip) para o diretório criado no passo anterior;
5. Copie o arquivo python do mincemeat para o diretório exerc\
- <https://github.com/michaelfairley/mincemeatpy>
6. Crie um arquivo texto e digite a primeira parte do código – Importação dos arquivos:

```
import mincemeat
import glob
import csv

text_files = glob.glob('G:\\NOSQL\\Exerc\\Join\\*')

def file_contents(file_name):
    f = open(file_name)
    try:
        return f.read()
    finally:
        f.close()

source = dict((file_name, file_contents(file_name))for file_name in text_files)
```

Estas parte faz a importação dos dados e gera um objeto 'source' que é do tipo dicionário: nome do arquivo e conteúdo do arquivo;

7. Faça a implementação do método *map*:

```
def mapfn(k, v):
    print 'map ' + k
    for line in v.splitlines():
        if k == 'G:\\NOSQL\\Exercicios\\1.3 Join\\1.3-vendas.csv':
            yield line.split(';')[0], 'Vendas' + ':' + line.split(';')[5]
        if k == 'G:\\NOSQL\\Exercicios\\1.3 Join\\1.3-filiais.csv':
            yield line.split(';')[0], 'Filial' + ':' + line.split(';')[1]
```

Altera os 'if' acima para o caminho e nomes dos arquivos da sua estação de trabalho;

8. Faça a implementação do método *reduce*. Implementação simples para entender o que acontece:

```
def reducefn(k, v):
    print 'reduce ' + k
    return v
```

9. Utize agora o *mincemeat* ele vai simular o papel da DFS e do '*name mode*':

```
s = mincemeat.Server()

# A fonte de dados pode ser qualquer objeto do tipo dicionário
s.datasource = source
s.mapfn = mapfn
s.reducefn = reducefn

results = s.run_server(password="changeme")

w = csv.writer(open("Exerc\\RESULT.csv", "w"))
for k, v in results.items():
    w.writerow([k, v])
```

10. Salve o nome do arquivo como exerc22.py;

11. Faça a execução paralela deste código:

a- server: python exerc22.py

b- Crie dois ou três clientes com o comando:

python mincemeat.py -p changeme localhost

Para executar remoto,faça a instalação do python e mincemeat conforme passos 1 e 4, respectivamente;

12. Veja o arquivo gerado

13. Faça a implementação final do método *reduce*:

```

def reducefn(k, v):
    print 'reduce' + k
    total = 0
    for index, item in enumerate(v):
        if item.split(":")[0] == 'vendas':
            total = int(item.split(":")[1]) +total
        if item.split(":")[0] == 'Filial':
            NomeFilial = item.split(":")[1]
    L = list()
    L.append(NomeFilial + " , " + str(total))
    return L

```

14. Utize agora o *mincemeat* novamente:

```

s = mincemeat.Server()
s.datasource = source
s.mapfn = mapfn
s.reducefn = reducefn
results = s.run_server(password="changeme")
w = csv.writer(open("G:\\\\NOSQL\\\\Exercicios\\\\RESULT_1.3.csv", "w"))
for k, v in results.items():
    w.writerow([k, str(v).replace("[", "").replace("]", "").replace("'", "").replace(' ', '')])

```

15. Faça a execução paralela deste código:

a- server: python exerc22.py

b- Crie dois ou três clientes com o comando:

python mincemeat.py -p changeme localhost

Para executar remoto, faça a instalação do python e mincemeat conforme passos 1 e 4, respectivamente;

16. Veja o arquivo gerado

Curso: Ciência de dados e Big Data

Professor: Cláudio Lúcio

Atividade Prática sobre Map Reduce

Neste caso vamos analisar uma implementação de map reduce no Hadoop.

Vamos utilizar os um exemplo de contagem de palavras já existente na sandbox da HortonWorks. Veja abaixo o programa que será executado (obviamente, implemetanto em Java):

1. Porção relativa ao Map

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context context) throws IOException,
        InterruptedException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
    }
}
```

2. Porção relativa ao Reduce

```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

3. Porção relativa ao programa principal

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
}
```

4. Para executar este programa vamos antes criar dois diretórios no usuário Hue, veja a tela a seguir:

1. MapReduce

Type	Name	Size	User	Group	Permissions	Date
..	.		hue	hue	drwxr-xr-x	December 19, 2016 03:32 pm
..	..		hdfs	hdfs	drwxr-xr-x	July 20, 2015 06:01 pm
Folder	.Trash		hue	hue	drwx-----	July 13, 2016 09:00 pm
Folder	.staging		hue	hue	drwx-----	July 13, 2016 09:57 pm
Folder	MapReduce		hue	hue	drwxr-xr-x	December 19, 2016 03:12 pm
Folder	jobs		hue	hue	drwxrwxrwx	March 28, 2014 10:48 am
Folder	oozie		hue	hue	drwxrwxrwx	March 28, 2014 10:48 am

5. Agora utilize o arquivo 2.1 Textos.zip e faça seu upload para o HDFS.

6. Verifique se os arquivos foram todos carregados

File Browser

Search for file name:

A Rename Move Copy Change Permissions Download Delete

New Upload

Home / user / hue / MapReduce / 2.1 - textos / 2.1 - textos

Trash

Type	Name	Size	User	Group	Permissions	Date
..	.		hue	hue	drwxr-xr-x	December 19, 2016 03:15 pm
..	..		hue	hue	drwxr-xr-x	December 19, 2016 03:12 pm
cv000_29590.txt	4.1 KB	hue	hue	-rwxr-xr-x	December 19, 2016 03:12 pm	
cv001_18431.txt	4.0 KB	hue	hue	-rwxr-xr-x	December 19, 2016 03:14 pm	
cv002_15918.txt	2.4 KB	hue	hue	-rwxr-xr-x	December 19, 2016 03:14 pm	
cv003_11664.txt	5.9 KB	hue	hue	-rwxr-xr-x	December 19, 2016 03:14 pm	
cv004_11636.txt	3.8 KB	hue	hue	-rwxr-xr-x	December 19, 2016 03:14 pm	
cv005_29443.txt	5.2 KB	hue	hue	-rwxr-xr-x	December 19, 2016 03:14 pm	
cv006_15448.txt	4.4 KB	hue	hue	-rwxr-xr-x	December 19, 2016 03:13 pm	

Show 45 items per page. Showing 1 to 45 of 1000 items, page 1 of 23.

Next Page End of List

7. Com os arquivo vamos abrir uma linha de comando:

```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount
'/user/hue/MapReduce/2.1 - textos/2.1 - textos/' '/user/hue/MapReduceSaida/'
```

```
root@sandbox:/# hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount '/user/hue/MapReduce/2.1 - textos/2.1 - textos/' '/user/hue/MapReduceSaida/
16/12/19 09:33:25 INFO client.RMProxy: Connecting to ResourceManager at sandbox.hortonworks.com/10.0.2.15:8050
16/12/19 09:33:28 INFO input.FileInputFormat: Total input paths to process : 100
0
16/12/19 09:33:29 INFO mapreduce.JobSubmitter: number of splits:1000
16/12/19 09:33:29 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1482166846963_0001
16/12/19 09:33:30 INFO impl.YarnClientImpl: Submitted application application_1482166846963_0001
16/12/19 09:33:30 INFO mapreduce.Job: The url to track the job: http://sandbox.hortonworks.com:8088/proxy/application_1482166846963_0001/
16/12/19 09:33:30 INFO mapreduce.Job: Running job: job_1482166846963_0001
```

Ainda em execução:

```
hortonworks.com:8088/proxy/application_1482166846963_0001/
16/12/19 09:33:30 INFO mapreduce.Job: Running job: job_1482166846963_0001
16/12/19 09:33:41 INFO mapreduce.Job: Job job_1482166846963_0001 running in uber mode : false
16/12/19 09:33:41 INFO mapreduce.Job: map 0% reduce 0%
16/12/19 09:34:15 INFO mapreduce.Job: map 1% reduce 0%
16/12/19 09:34:47 INFO mapreduce.Job: map 2% reduce 0%
16/12/19 09:35:57 INFO mapreduce.Job: map 3% reduce 0%
16/12/19 09:36:29 INFO mapreduce.Job: map 4% reduce 0%
16/12/19 09:37:00 INFO mapreduce.Job: map 5% reduce 0%
16/12/19 09:37:31 INFO mapreduce.Job: map 6% reduce 0%
```

Finalização da execução:

```
16/12/19 13:05:17 INFO mapreduce.Job: map 95% reduce 31%
16/12/19 13:05:03 INFO mapreduce.Job: map 94% reduce 31%
16/12/19 13:05:45 INFO mapreduce.Job: map 95% reduce 31%
16/12/19 13:05:46 INFO mapreduce.Job: map 95% reduce 32%
16/12/19 13:06:27 INFO mapreduce.Job: map 96% reduce 32%
16/12/19 13:07:05 INFO mapreduce.Job: map 97% reduce 32%
16/12/19 13:07:33 INFO mapreduce.Job: map 98% reduce 32%
16/12/19 13:07:34 INFO mapreduce.Job: map 98% reduce 33%
16/12/19 13:08:22 INFO mapreduce.Job: map 99% reduce 33%
16/12/19 13:08:50 INFO mapreduce.Job: map 100% reduce 33%
16/12/19 13:09:01 INFO mapreduce.Job: map 100% reduce 100%
16/12/19 13:09:04 INFO mapreduce.Job: Job job_1482166846963_0001 completed successfully
16/12/19 13:09:05 INFO mapreduce.Job: Counters: 51
  File System Counters
    FILE: Number of bytes read=4578315
    FILE: Number of bytes written=108161357
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=4285076
    HDFS: Number of bytes written=410680
    HDFS: Number of read operations=3003
    HDFS: Number of large read operations=0
```

```
CDS Counters
  Failed map tasks=14
  Launched map tasks=1014
  Launched reduce tasks=1
  Other local map tasks=14
  Data-local map tasks=1000
  Total time spent by all maps in occupied slots (ms)=89570481
  Total time spent by all reduces in occupied slots (ms)=12373936
  Total time spent by all map tasks (ms)=89570481
  Total time spent by all reduce tasks (ms)=12373936
  Total vcore-seconds taken by all map tasks=89570481
  Total vcore-seconds taken by all reduce tasks=12373936
  Total megabyte-seconds taken by all map tasks=22392620250
  Total megabyte-seconds taken by all reduce tasks=3093484000
Map-Reduce Framework
  Map input records=32937
  Map output records=787051
  Map output bytes=7238653
  Map output materialized bytes=4584309
  Input split bytes=160793
  Combine input records=787051
  Combine output records=358530
  Reduce input groups=36805
  Reduce shuffle bytes=4584309
```

```
Reduce output records=36805
Spilled Records=717060
Shuffled Maps =1000
Failed Shuffles=0
Merged Map outputs=1000
GC time elapsed (ms)=102170
CPU time spent (ms)=594960
Physical memory (bytes) snapshot=211428573184
Virtual memory (bytes) snapshot=886697873408
Total committed heap usage (bytes)=166730924032
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=4124283
File Output Format Counters
  Bytes Written=410680
[root@sandbox /]#
```

8. Veja agora os arquivos gerados pela execução do programa map reduce:

Name	Type	Size	User	Group	Permissions	Date
_SUCCESS	File	0 bytes	root	hue	-rwxr-xr-x	December 19, 2016 07:09 pm
part-r-00000	File	401.1 KB	root	hue	-rw-r--r--	December 19, 2016 07:09 pm

9. Veja o conteúdo dos dois arquivos gerados (principalmente part-r-0000):

Word	Count
ty	1
really	1
rushmore	6
scream	1
_seven_nights_	2
_shaft's_big_score	1
_shaft_7	1
_shaft_s	1
_shaft_in_africa_	1
_shime_3	1
_six_days	2
_so	1
_star	1
that	2
the	24
_the_broadway_musical_	1
_the_fugitive_	1
_the_lion_king	1
their	1
titanic	1
titus	1
_titus_andronicus	1

Soluções para processamento paralelo e distribuído de dados

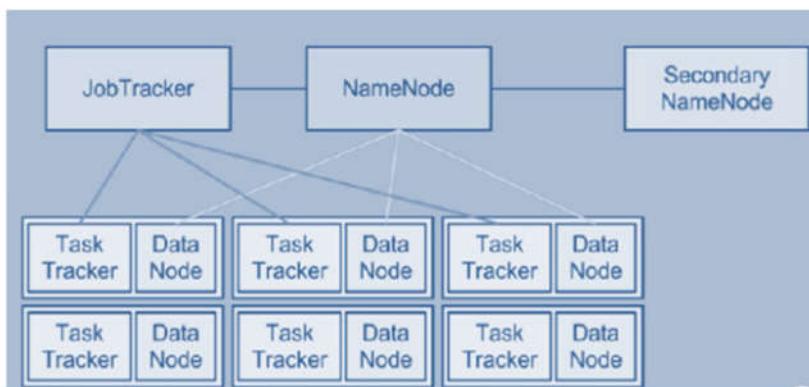
Hadoop – Arquitetura do Hadoop 2.0: YARN



Ilustração de Oliver Munday

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0



Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

- Características

Name Node

- Todos os metadados são mantidos em RAM;
- Executar todas as operações relativas aos metadados;
- Ponto de falha da arquitetura (há condições de contorno);

Limitações da arquitetura

- Escalabilidade 'limitada';
- Utilização de recurso pode ser melhorada;
- Não há suporte para outras paradigmas de processamento;

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

- Características

- Escalabilidade 'limitada'
 - ~ 4000 nós de dados
 - ~ 40000 tarefas concorrentes



- Utilização de recursos pode ser melhorada

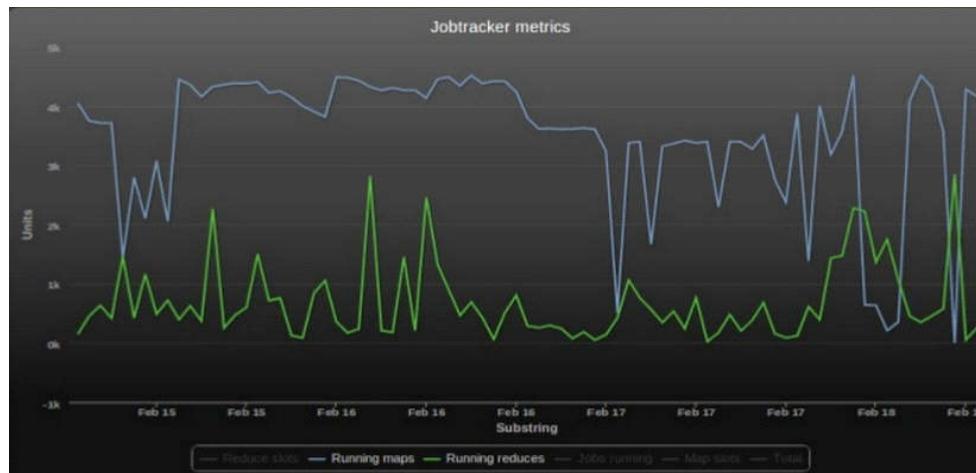
Cluster Summary (Heap Size is 12.58 GB/23.97 GB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node
4512	285	2223	188	4512	285	0	0	4512	3008	40.00

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

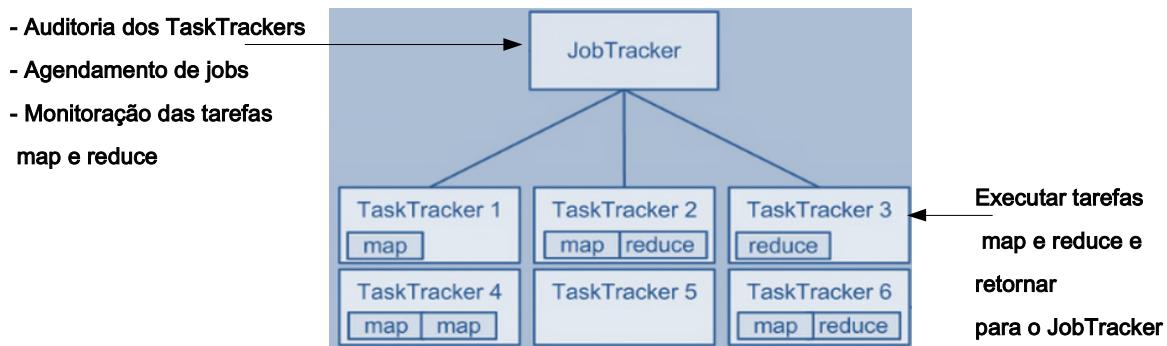
- Características
 - Utilização de recursos pode ser melhorada



Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

- Características
 - Utilização de recursos pode ser melhorada



Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

Características

- Tarefas do JobTracker

Gerencia dos recursos computacionais(tarefas map e reduce);

Agendamento de todas as tarefas de usuários:

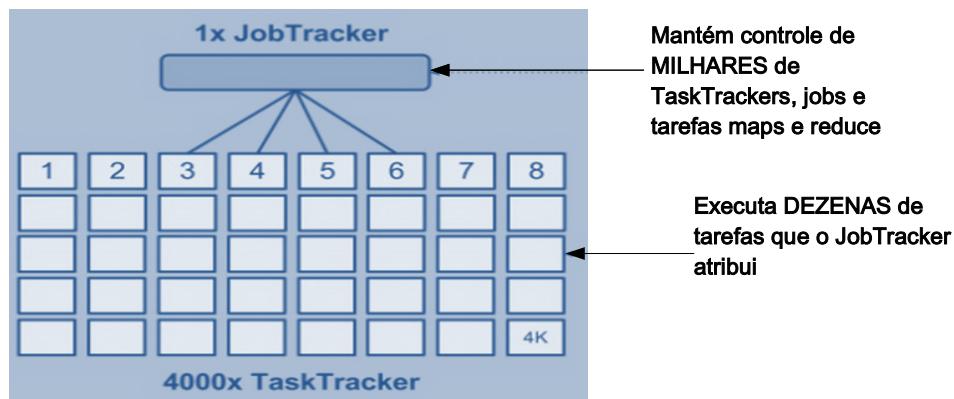
- Agendar tarefas de um job;
- Monitorar execução das tarefas;
- Reiniciar tarefas que falharam e verificar recursos disponíveis;
- Calcular e manter atualizados os registros/ contadores de jobs;

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

Características

- Algumas observações



Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

Características

- Propostas 'naturais'

Reducir as responsabilidades do JobTracker;

- Separar o gerenciamento de recursos do cluster da coordenação de Jobs;
- Usar outros nós para gerenciar o ciclo de vida dos jobs;

Aumentar a escalabilidade

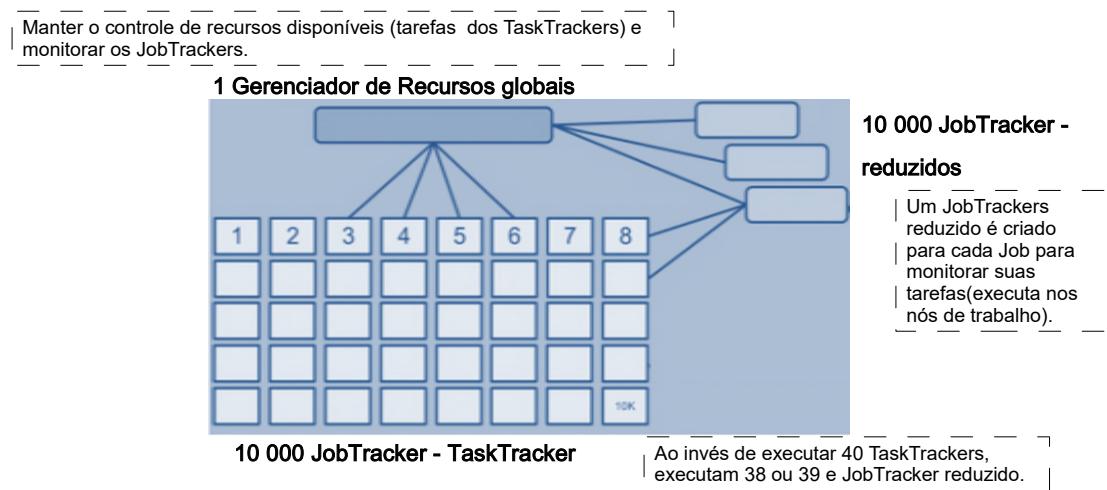
- Mais de 10.000 nós;
- Mais de 10.000 jobs;
- Mais de 100.000 tarefas;

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Características

- Desaparecimento do JobTracker



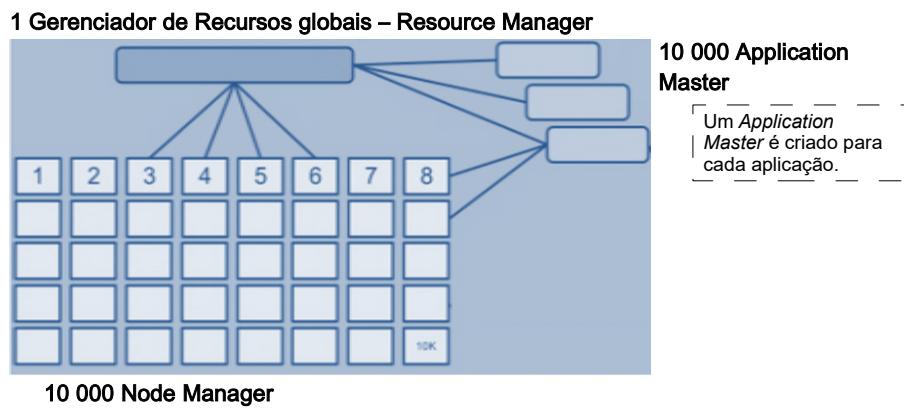
Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Características

- Novos nomes e funções no Hadoop 2.0

YARN -YET ANOTHER RESOURCE NEGOTIATOR

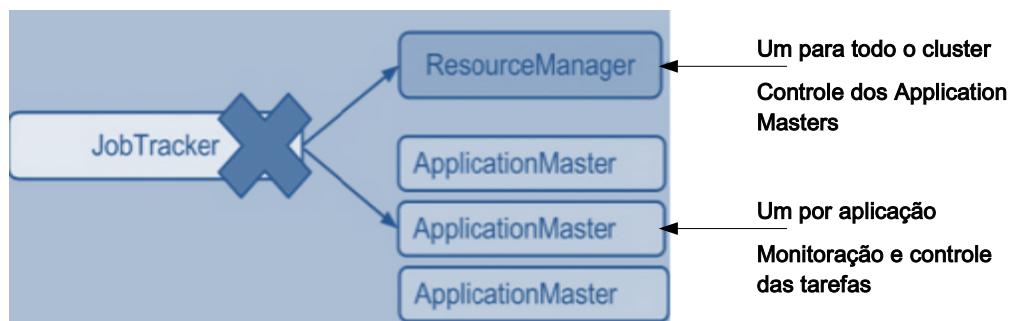


Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Características

- Novos nomes e funções no Hadoop 2.0- YARN



Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Características

- Node Manager

- Mais flexível e eficiente que o *TaskTracker*;
- Executa qualquer tipo de computação que faz sentido no contexto do *Application Manager*; não somente *map e reduce*;
- Possui o conceito de *containers*:
 - Podem lidar com recursos variáveis (RAM, CPU, IO);
 - Não exige número de funções map's e reduce;

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Características

- Node Manager

Possui o conceito de *containers*:

- Cria um *container* para cada tarefa
- Possui propriedades físicas:
 - 1 CPU, 2GB RAM e 100GB disco;
- Número de *containers* é limitado;
- Não pode exceder os recursos do *Node Manager*;

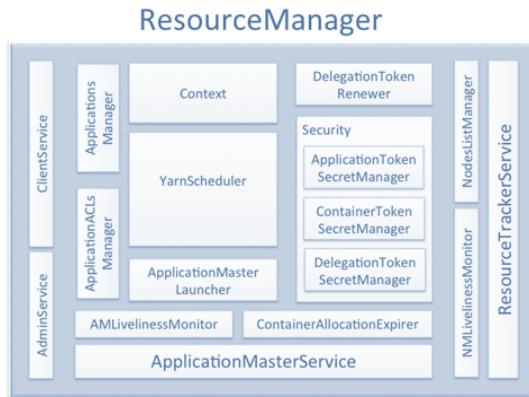
Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Características

- Resource Manager

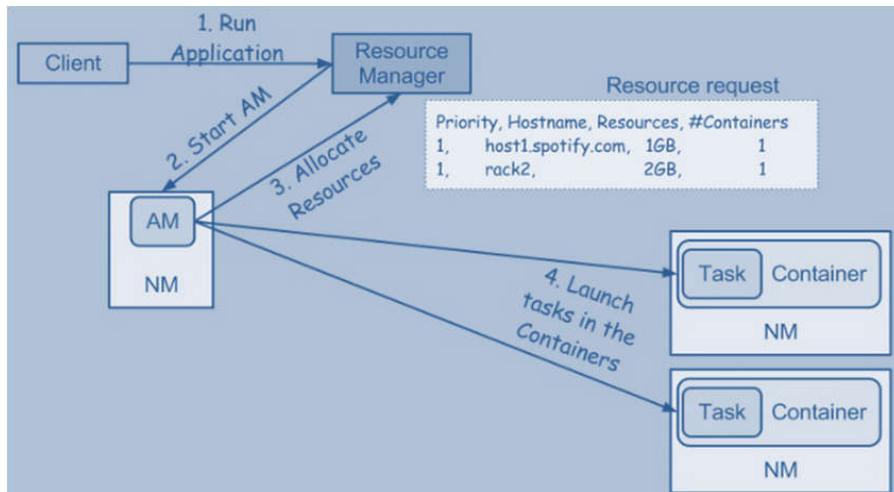
Na verdade é um pouco mais complexo:



Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Requisição usando YARN



Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Muitas novas aplicações usam YARN

Hadoop Wiki [Login](#)

PoweredByYarn

[FrontPage](#) [RecentChanges](#) [FindPage](#) [HelpContents](#) [PoweredByYarn](#)

Immutable Page [Info](#) [Attachments](#) More Actions: ▾

This wiki tracks the applications written (or being ported to run) on top of YARN i.e. Next Generation Hadoop processing framework aka [NextGenMapReduce](#).

- Apache Hadoop MapReduce, of course! - [https://issues.apache.org/jira/browse/MAPREDUCE-279](#)
- Spark - [https://github.com/mesos/spark-yarn](#)
- Apache HAMA - [https://issues.apache.org/jira/browse/HAMA-431](#)
- Apache Giraph - [https://issues.apache.org/jira/browse/GIRAPH-13](#)
- Open MPI - [https://issues.apache.org/jira/browse/MAPREDUCE-2911](#)
- Generic Co-Processors for Apache HBase - [https://issues.apache.org/jira/browse/HBASE-4047](#)

Other ideas:

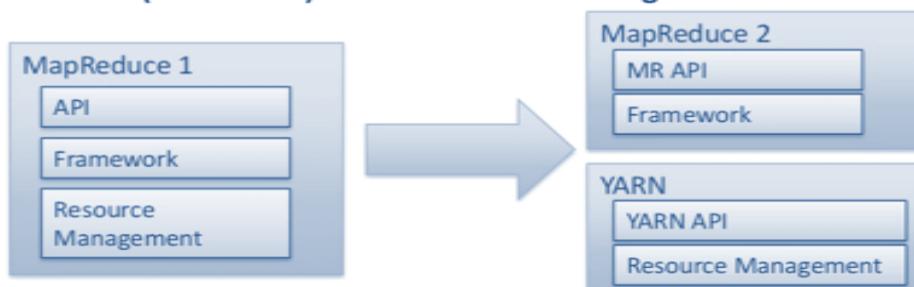
- Apache HBase deployment using YARN - [https://issues.apache.org/jira/browse/HBASE-4329](#)

We encourage you to add your application if you are using [NextGenMapReduce](#) to build a new processing framework or tool.

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

E o MapReduce???



Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

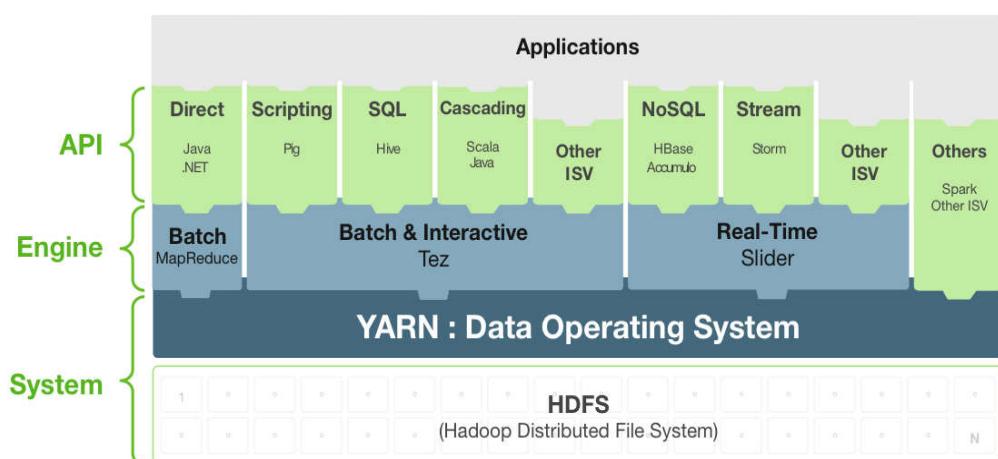
Na Prática....

- Hadoop se torna uma plataforma de processamento paralelo mais flexível;
- Não executar apenas MapReduce;
- Uma nova família de aplicações vem sendo desenvolvidas;
- Torna-se orientada a aplicações empresariais: flexibilidade;

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Na Prática....



Hadoop - Arquitetura do Hadoop 2.0: YARN

Atividade

- Exploração da interface do Ambari;
- Verificação de execução do YARN no Ambari;

Curso: Ciência de dados e Big Data

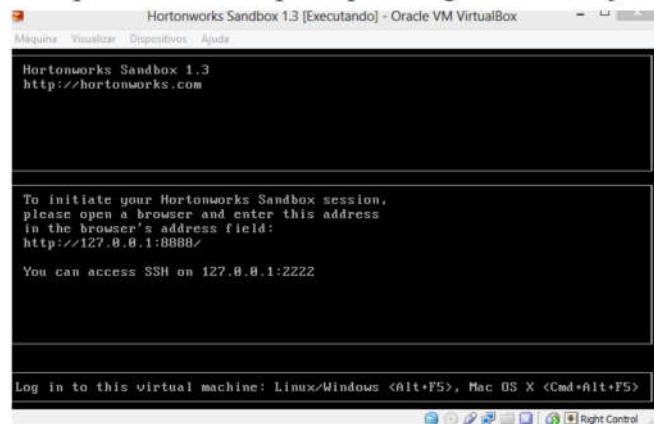
Professor: Cláudio Lúcio

Atividade Prática sobre Ambari

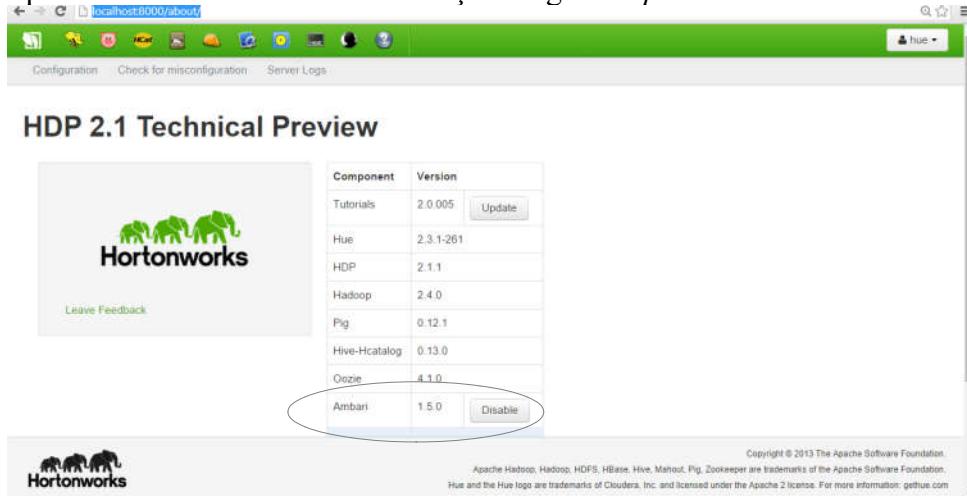
- 1) Para utilização das práticas com o Hadoop vamos usar uma sandbox (máquina virtual) fornecida pela HortonWorks. Esta máquina virtual possui os seguintes produtos instalados:

Esta é a versão 2.1 do produto e pode ser obtida em:
<http://hortonworks.com/products/hortonworks-sandbox/>

- 2) Inicialize a máquina virtual. Espere que a seguinte tela seja exibida:



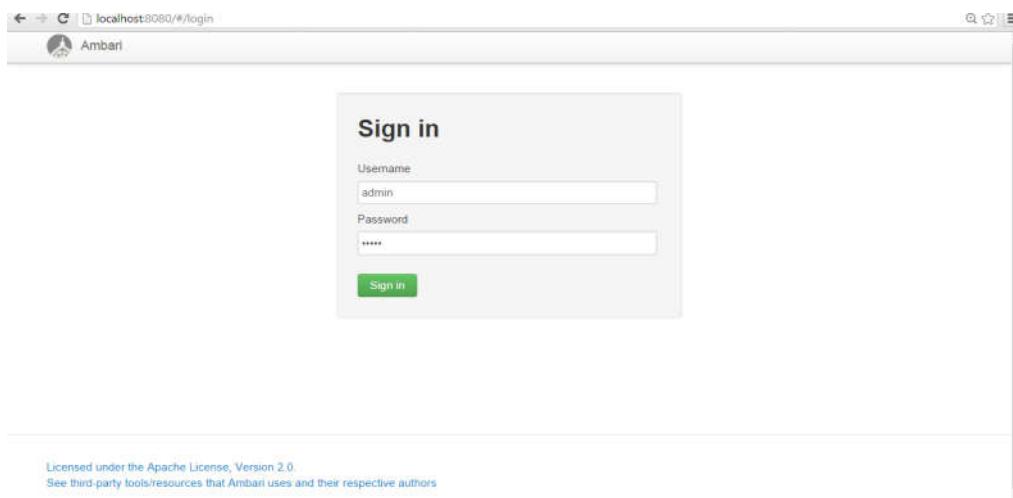
- 3) Ambari é uma ferramenta que é incluída no HDP para simplificar a instalação e configuração, e manutenção de clusters Hadoop;
- 4) Verifique se o Ambari está no em execução. Digite: <http://localhost:8000>



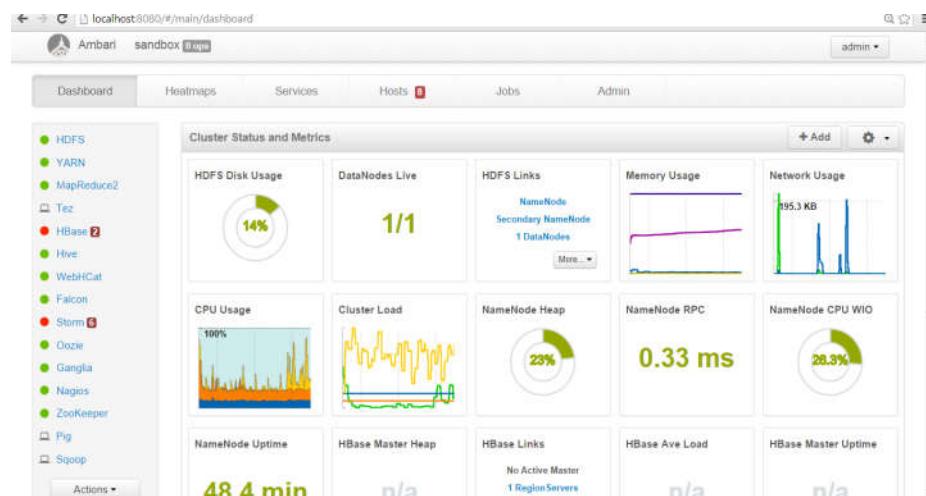
Component	Version	Action
Tutorials	2.0.005	Update
Hue	2.3.1-261	
HDP	2.1.1	
Hadoop	2.4.0	
Pig	0.12.1	
Hive-Hcatalog	0.13.0	
Oozie	4.1.0	
Ambari	1.5.0	Disable

5) Faça então acesso à interface do Ambari: <http://localhost:8080>

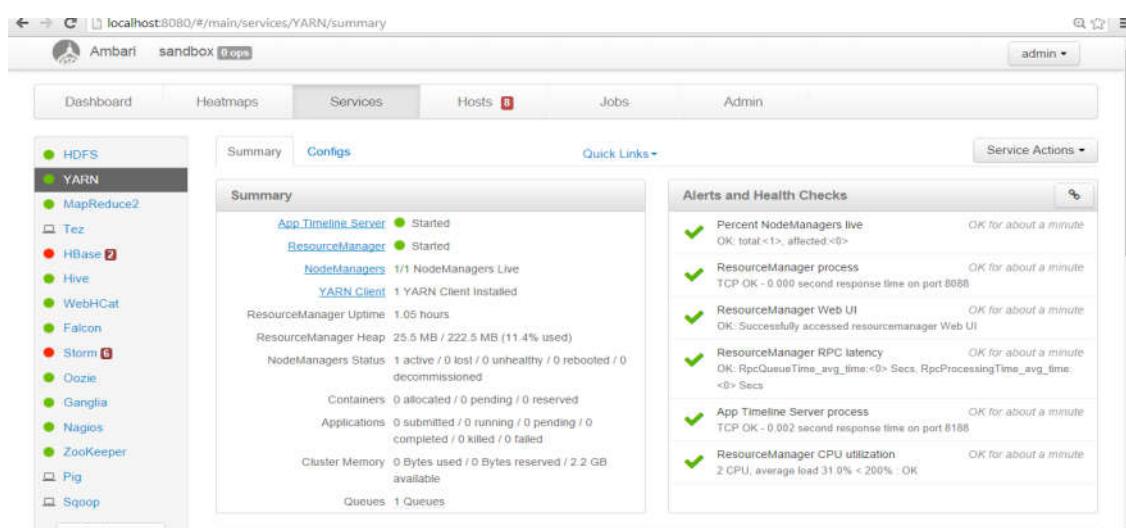
Acesse com usuário: *admin* e senha: *admin*



6) A tela padrão do Ambari é um dashboard mostrando alguns indicadores do cluster Hadoop. Veja na parte esquerda que há links para todos os produtos que estão instalados no cluster.



7) Clique na aba “Services”, na parte esquerda também selecione o *YARN*:



Observa os elementos: *ResourceManager*, *NodeManagers*, *Containers* e etc...

8) Clique na sub-aba Configs

Veja que você pode alterar a configuração dos componentes YARN: *ResourceManager*, *NodeManagers*, *Scheduler* e outros;

9) Finalmente, clique na opção “Quick Links--> Resource Manager” para verificar o que está em execução no cluster:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
No data available in table										

[About Apache Hadoop](#)

Curso: Ciência de dados e Big Data

Professor: Cláudio Lúcio

Trabalho prático de Map Reduce

Faça o download dos arquivos Trab1.0.zip

Cada arquivo neste diretório possui a seguinte estrutura:

journals/cl/SantoNR90:::Michele Di Santo::Libero Nigro::Wilma Russo:::Programmer-Defined Control Abstractions in Modula-2.

Cada uma das linhas pode ser entendida como:

Informação bibliográfica;

autores separados por ':::';

Título da obra

paper-id:::author1::author2::.... ::authorN:::title

- 1) A tarefa é fazer o cálculo de quantas vezes cada termo (no título da obra) acontece por autor;
- 2) Por exemplo para o autor 'Alberto Pettorossi' os seguintes termos acontecem (considerando todos os documentos): program:3, transformation:2, transforming:2, using:2, programs:2, and logic:2.
- 3) Observe os seguintes itens:

O separador de campos é “:::” e separador de autores é “::”;

Cada autor pode ter escrito múltiplas obras, que por sua vez podem estar em vários arquivos;

Existe uma lista de palavras que não serão consideradas. Utilize o arquivo `stopword.py` do exercício prático: exclua todas estas palavras para os autores;

Se possível faça a exclusão de pontuações também, de forma que a palavra `logic` e `logic.` sejam tratadas como se fossem uma única palavra;

- 4) Entregue a implementação do seu grupo;
- 5) Responda quais são as duas palavras que mais acontecem para os seguintes autores:
 - a- “Grzerorz Rozenberg”
 - b- “Philip S. Yu”

Soluções para processamento paralelo e distribuído de dados



Ilustração de Oliver Munday

Soluções Processamento paralelo e distribuído de dados

Spark - Básico

- Introdução ao Spark
- Histórico e motivações
- Arquitetura e Conceitos do Spark
- RDD's
- Algumas Ações e transformações
- Spark SQL
- DataFrame API

Soluções para processamento paralelo e distribuído de dados

Spark – Introdução



Ilustração de Oliver Munday

Spark - Introdução

Definição:

- É um plataforma(em cluster) para processamento rápido, paralelo e de propósito geral;
- Extende o algoritmo padrão: MapReduce;
- Muitos cálculos são feitos em memória (podem ser executados no disco também);
- API's: Java, Scala, Python, SQL e R

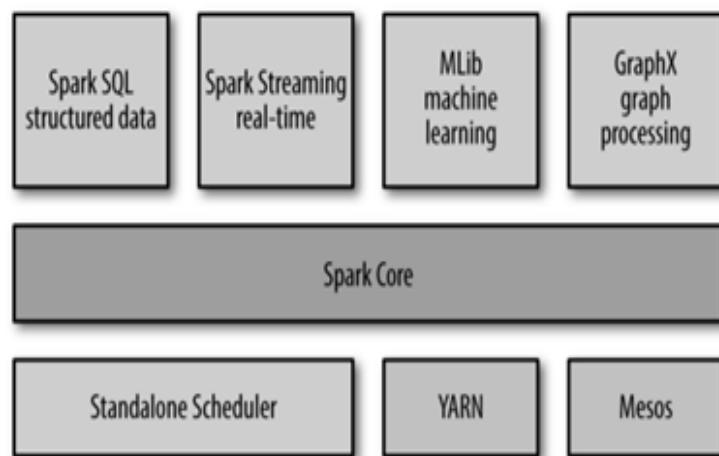
Spark - Introdução

Motivação:

- Ambiente único para execução de vários tipos de processamentos:
 - Batch
 - Processamentos interativas
 - Processamentos de streamings
- Multi-propósito: ML, Grafos, Processamento e Consultas;
- Stack de tecnologias

Spark - Introdução

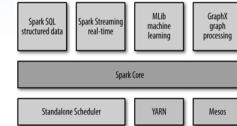
Macro arquitetura:



Spark - Introdução

Spark Core:

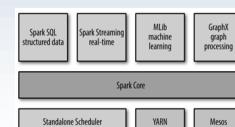
- Funcionalidade básicas:
 - Agendamento de tarefas
 - Gerenciamento de memória
 - Tolerância a falhas
 - Interação com sistemas de armazenamento e outros...
- Possui a API os RDD's (resilient distributed datasets) - principal abstração da programação no Spark;
- RDD's = coleção de itens distribuídos no cluster e que podem ser manipulados de forma paralela;



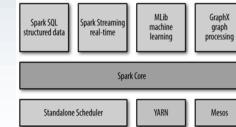
Spark - Introdução

Spark SQL:

- Componente para lidar com dados estruturados;
- Acesso a dados via SQL(de forma similar ao Hive do Hadoop);
- Suporta vários tipos de dados: Hive Tables, Parquet e arquivos Json;
- É possível num único programa mesclar SQL com outras operações suportadas pelos RDD's;



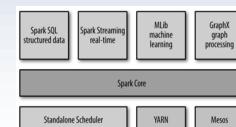
Spark - Introdução



Spark Streaming:

- Componente para lidar com *Streams* de dados;
- Desenvolvido, também, para lidar com logs de servidores ou filas de mensagens
- API do Spark Streaming, assim como as outras, são integradas ao API Spark Core;
- Fornece também: escalabilidade, tolerância a falhas, interações com dados em memória e disco;

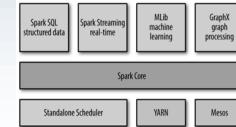
Spark - Introdução



Spark MLlib:

- Possui algoritmos típicos de aprendizado de máquina;
- Alguns tipos: classificação, regressão, agrupamento, filtros colaborativos(recomendação);
- Parte do time que trabalhou no Mahout ajudou no desenvolvimento deste componente;
- Outras funcionalidades: importação de dados e avaliação de modelos;
- Oferece também suporte para algumas etapas comuns em alguns algoritmos(ex.: otimização via gradiente descendente)

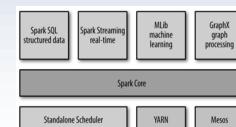
Spark - Introdução



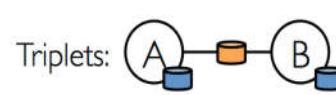
Spark GraphX:

- Componente para manipulação de grafos;
- Computação paralela é feita nos métodos de manipulação dos grafos;
- Exemplo de métodos: PageRank, detecção de triângulos, componentes conectados e outros

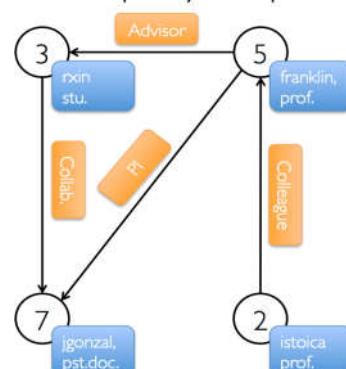
Spark - Introdução



Spark GraphX:



Property Graph



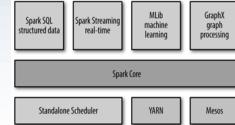
Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Spark - Introdução



Gerenciadores:

- Spark é escalável para milhares de nós em execução;
- Pode executar sobre vários gerenciadores de clusters;
- Exemplos: Hadoop YARN(integração), Apache Mesos ou mesmo uma versão simplificado do próprio Spark;

Soluções Processamento paralelo e distribuído de dados

Spark

- Introdução ao Spark;
- Histórico

Soluções para processamento paralelo e distribuído de dados

Spark – Histórico e Motivações



Ilustração de Oliver Munday

Spark – Histórico e Motivações

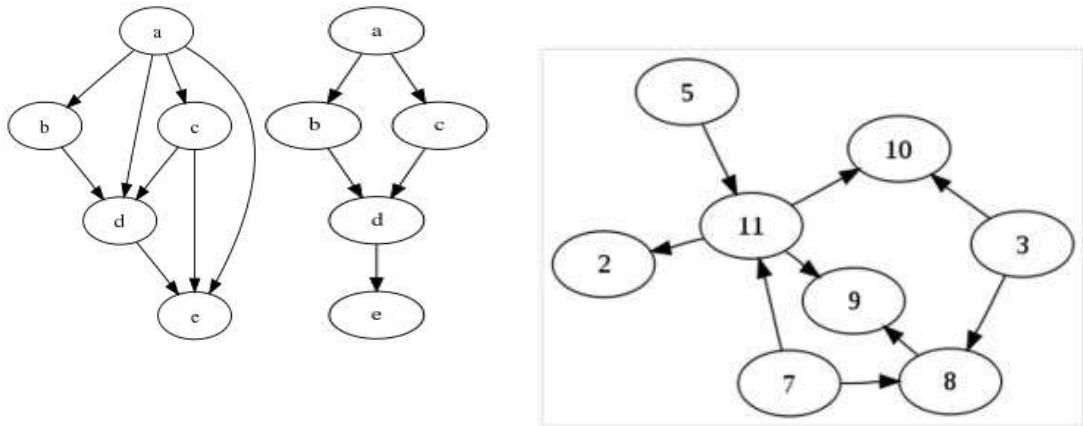
Histórico:

- É um projeto aberto e mantido por uma 'comunidade' de desenvolvedores;
- Projeto foi iniciado em 2009 na UC Berkeley RAD Lab;
 - Posteriormente é criada a AMPLab;
 - Boa parte do membros da laboratório trabalharam implementação MapReduce do Hadoop;
 - No Hadoop 1.0 perceberam, claramente, as limitações de MapReduce para jobs interativos e iterativos;
 - Em 2009 as primeiras versões do Spark são de 10 a 20 vezes mais rápidas que o MapReduce;

Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

- DAG ou Grafos direcionados acíclicos

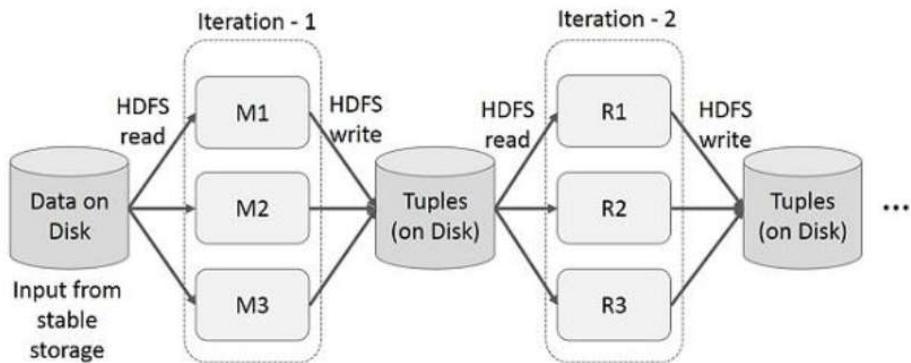


Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

DAG ou Grafos direcionados acíclicos

- MapReduce traduz as etapas da computação para apenas duas etapas;



Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

DAG ou Grafos direcionados acíclicos

- Processo complexos tipicamente requerem vários etapas (por exemplo vários MapReduces 'aninhados');
- Na verdade estas etapas formam um DAG, que é uma generalização do MapReduce;
- Desta forma, DAG não conflita com o paradigma MapReduce;

Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

DAG ou Grafos direcionados acíclicos

- Originalmente, MapReduce, tem dificuldades de parallelizar vários jobs MapReduce;
- A abordagem DAG resolve este problema;
- Imagine 3 Jobs: A, B e C:
 - C depende de A e B;
 - Então A e B podem executar de forma paralela (dois *maps*, por exemplo);
 - Esta precedência é explicitada em um DAG facilmente;

Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

DAG ou Grafos direcionados acíclicos

- Conceitualmente a solução não podeira ser adaptada ao MapReduce?
- Caraterísticas MapReduce:
 - Ler dados do HDFS;
 - Processar resultados aplicando paradigma Map e Reduce;
 - Gravar dados novamente no HDFS;
 - Cada Job MapReduce é completamente independente um do outro;

Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

DAG ou Grafos direcionados acíclicos

- Caraterísticas MapReduce:
 - O 'Name node' não faz ideia do que cada um dos MapReduce's esta executando e não há possibilidade de 'aproveitamento' de dados;
 - Para lógicas iterativas estas características são totalmente indesejáveis;
 - Para processos interativos estas características também devem ser otimizadas;

Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

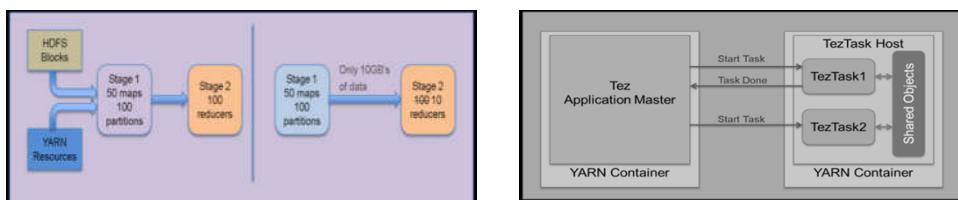
DAG ou Grafos direcionados acíclicos

- Características MapReduce:
 - Processo mais complexos exigem cadeias de processo MapReduce;
 - Com cadeias de processos MapReduce a paralelização é comprometida: o próximo Map só inicia após o seu Reduce predecessor;
 - Mesmo cadeias de processos com pequenos volumes de dados podem ter desempenho pobre:
 - Gravação em disco em todas as etapas;
 - Processo de inicialização para cada etapa;

Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

- Este é o problema resolvido pelo Spark, aliado com a utilização do dados reaproveitáveis e em memória;
- Outras implementações também fazem uso da mesma ideia, utilizando o YARN(Hadoop 2.X);
- Um exemplo é o TEZ que é utilizado, dentre outros, pela HortonWorks ;



Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

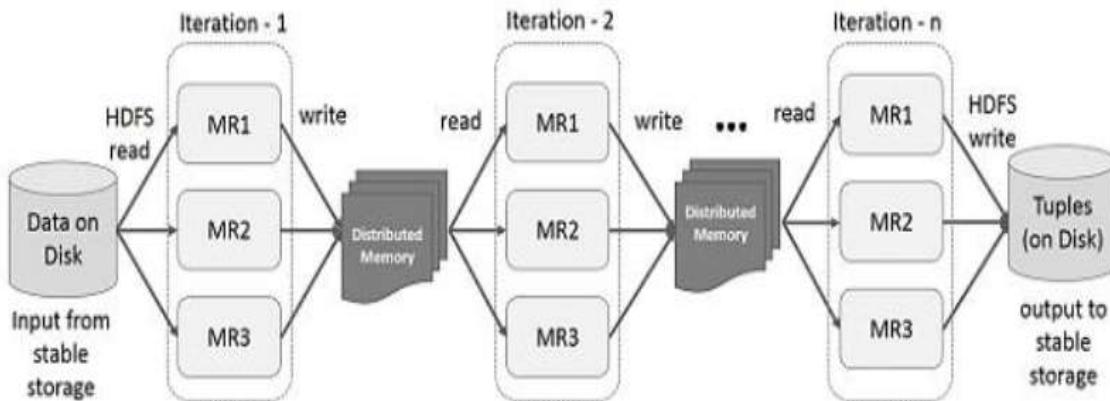
- Proposta Spark :

- Trata todo os fluxos de processamento como DAG;
- Carregar dados em memória e evitar leituras em disco em cada uma das etapas;
- Utiliza os RDD's para compartilhar dados e aproveitar para não fazer escrita no disco ao final de cada etapa (não escreve dados no HDFS ao fim de cada Reduce);

Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

- Proposta Spark :



Spark – Histórico e Motivações

Histórico: E depois ???

- Spark foi adotado por alguns grupos em Berkeley;
- Algumas empresas começaram a usar a solução também(Chaordic aqui no Brasil, por exemplo);
- Em 2010 o projeto teve seu código aberto;
- Em 2011 a AMPLab começou o desenvolvimento dos outros componentes Spark: Shark(Hive on Spark) e Spark Streaming;
- Em 2013 foi transferido para a fundação Apache

Soluções para processamento paralelo e distribuído de dados

Spark – Arquitetura e Conceitos do Spark



Ilustração de Oliver Munday

Spark - Arquitetura e Conceitos do Spark

Exemplo de programa Spark (*python*):

```
>>> lines = sc.textFile("/apps/hive/warehouse/sample_07/sample_07") #  
Criação do RDD  
>>> lines.count() # Operação no RDD  
>>> lines.first() # linha 1 do arquivo
```

Conceitos importantes em uma aplicação Spark:

- Driver Program
- Contexto Spark
- Executors

Spark - Arquitetura e Conceitos do Spark

Driver Program

- Qualquer aplicação Spark é um '*Driver program*';
- Esta aplicação é capaz de fazer operações paralelas no cluster Spark;
- Possui um função principal (*main*) e:
 - Define/cria dados distribuídos no cluster;
 - Aplicações operações paralelas aos dados distribuídos no cluster;
 - Operações não são apenas Map e Reduce;

Spark - Arquitetura e Conceitos do Spark

Contexto Spark

- Um 'Driver program' acessa o spark através de um objeto contexto;
- Na verdade, representa a conexão da aplicação com o Cluster;
- Usando o Spark Shell, automaticamente, um contexto é criado;



```
version 1.4.1

Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkContext available as sc, HiveContext available as sqlContext.
>>> sc
<pyspark.context.SparkContext object at 0x21f9310>
>>> 
```

Spark - Arquitetura e Conceitos do Spark

Contexto Spark

- Para inicializar um contexto Spark em uma aplicação são necessários dois parâmetros:

```
from pyspark import SparkConf, SparkContext

conf = SparkConf().setMaster("local").setAppName("My App")
sc = SparkContext(conf = conf)
```

- URL do Cluster : no exemplo acima usamos '*local*' (executa o Spark em uma única *thread* na máquina local sem usar o cluster);
- Nome da aplicação: irá identificar o nome de sua aplicação no gerenciador do *cluster*;
- Outros métodos;

Spark - Arquitetura e Conceitos do Spark

Contexto Spark - SpartConf

Instance Methods	
<code>__init__(self, loadDefaults=True, _jvm=None)</code>	source code
Create a new Spark configuration.	
<code>set(self, key, value)</code>	source code
Set a configuration property.	
<code>setMaster(self, value)</code>	source code
Set master URL to connect to.	
<code>setAppName(self, value)</code>	source code
Set application name.	
<code>setSparkHome(self, value)</code>	source code
Set path where Spark is installed on worker nodes.	
<code>setExecutorEnv(self, key=None, value=None, pairs=None)</code>	source code
Set an environment variable to be passed to executors.	
<code>setAll(self, pairs)</code>	source code
Set multiple parameters, passed as a list of key-value pairs.	
<code>get(self, key, defaultValue=None)</code>	source code
Get the configured value for some key, or return a default otherwise.	
<code>getAll(self)</code>	source code
Get all values as a list of key-value pairs.	
<code>contains(self, key)</code>	source code
Does this configuration contain a given key?	
<code>toDebugString(self)</code>	source code
Returns a printable version of the configuration, as a list of key=value pairs, one per line.	
Inherited from object: <code>__delattr__, __format__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __sizeof__, __str__, __subclasshook__</code>	

Spark - Arquitetura e Conceitos do Spark

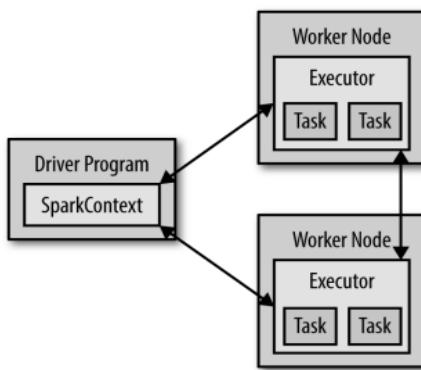
Contexto Spark - SparkContext

Instance Methods	
<code>__init__(self, master=None, appName=None, sparkHome=None, pyFiles=None, environment=None, batchSize=1024, serializer=PickleSerializer(), conf=None)</code>	source code
Create a new SparkContext.	
<code>defaultParallelism(self)</code>	source code
Default level of parallelism to use when not given by user (e.g.	
<code>__del__(self)</code>	source code
<code>stop(self)</code>	source code
Shut down the SparkContext.	
<code>parallelize(self, c, numSlices=None)</code>	source code
Distribute a local Python collection to form an RDD.	
<code>textFile(self, name, minSplits=None)</code>	source code
Read a text file from HDFS, a local file system (available on all nodes), or any Hadoop-supported file system URI, and return it as an RDD of Strings.	
<code>union(self, rdds)</code>	source code
Build the union of a list of RDDs.	
<code>broadcast(self, value)</code>	source code
Broadcast a read-only variable to the cluster, returning a <code>Broadcast</code> object for reading it in distributed functions.	
<code>accumulator(self, value, accum_param=None)</code>	source code
Create an <code>Accumulator</code> with the given initial value, using a given <code>AccumulatorParam</code> helper object to define how to add values of the data type if provided.	
<code>addFile(self, path)</code>	source code
Add a file to be downloaded with this Spark job on every node.	
<code>clearFiles(self)</code>	source code
Clear the job's list of files added by <code>addFile</code> or <code>addPyFile</code> so that they do not get downloaded to any new nodes.	
<code>addPyFile(self, path)</code>	source code
Add a .py or .zip dependency for all tasks to be executed on this SparkContext in the future.	
<code>setCheckpointDir(self, dirName)</code>	source code
Set the directory under which RDDs are going to be checkpointed.	
Inherited from object: <code>__delattr__, __format__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __sizeof__, __str__, __subclasshook__</code>	

Spark - Arquitetura e Conceitos do Spark

Executor

- Para realizar as operações do '*Driver Program*', o mesmo faz uso de vários nós que possuem '*Executor*':



Spark - Arquitetura e Conceitos do Spark

Exemplo de Wordcount - Scala

```
// Create a Scala Spark Context.  
val conf = new SparkConf().setAppName("wordCount")  
val sc = new SparkContext(conf)  
// Load our input data.  
val input = sc.textFile(inputFile)  
// Split it up into words.  
val words = input.flatMap(line => line.split(" "))  
// Transform into pairs and count.  
val counts = words.map(word => (word, 1)).reduceByKey(case (x, y) => x + y)  
// Save the word count back out to a text file, causing evaluation.  
counts.saveAsTextFile(outputFile)
```

Spark - Arquitetura e Conceitos do Spark

Prática – Interagindo com o Spark Shell Python

Prática – Interagindo com o Spark Shell Scala

Curso: Ciência de dados e Big Data

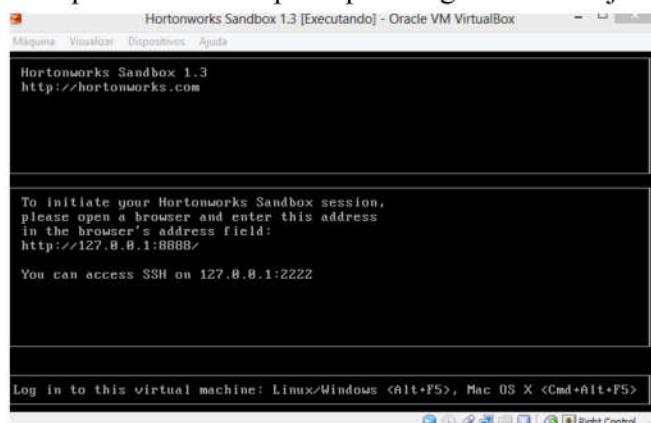
Professor: Cláudio Lúcio

Atividade para acesso ao Spark Shell (Python e Scala)

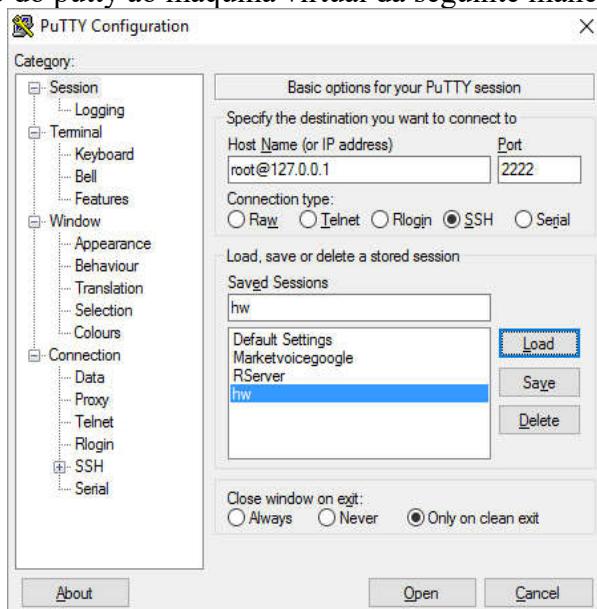
- 1) Para acesso ao Shell do Spark vamos usar a versão do Spark que vem na máquina virtual do HortonWorks:

Esta é a versão 2.3.2 do produto e pode ser obtida em:
<http://hortonworks.com/products/hortonworks-sandbox/>

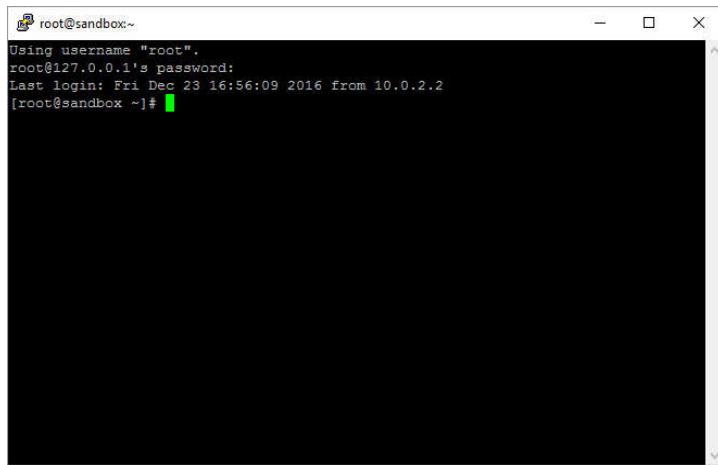
- 2) Inicialize a máquina virtual. Espere que a seguinte tela seja exibida:



- 3) A recomendação é usar um cliente para acesso SSH ao servidor do spark. Baixe a ferramenta putty.exe;
- 4) Configure o acesso do putty ao máquina virtual da seguinte maneira:



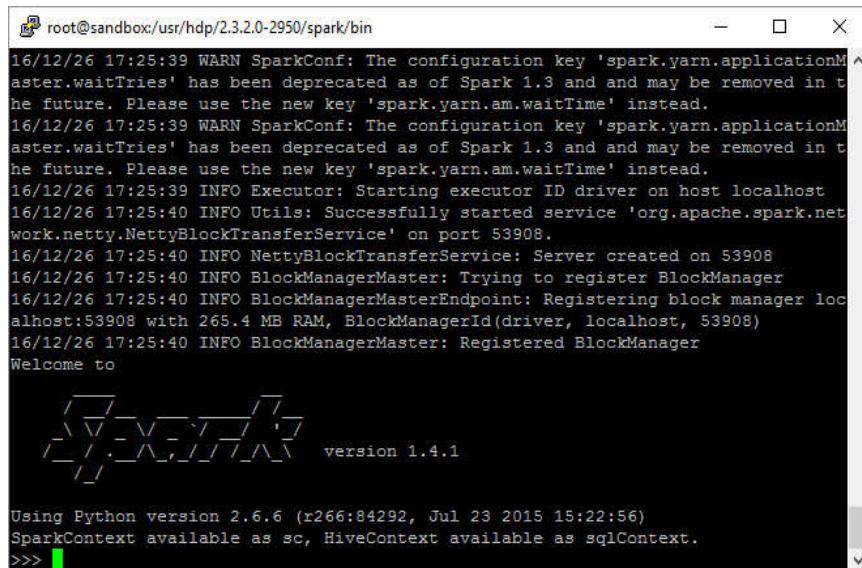
- 5) Clique em “Open” e então digite a senha do Root: hadoop



- 6) Primeiramente vamos interagir com a interface python para o Spark, para tal vamos acessar o bin/pyspark:

7) Digite os seguintes comandos:

```
cd /usr/hdp/2.3.2.0-2950/spark/bin  
pyspark
```



- 8) Veja que estamos com a versão 1.4.1 do Spark;

9) Por padrão as mensagens de log apresentadas no spark são muito detalhadas e podem dificultar as tarefas de manipulação de dados; Vamos ajustar os logs que serão apresentados. Vamos editar o arquivo ..//conf/log4j.properties :

```
cd ..  
cd conf/  
vi log4j.properties  
Pressione a tecla "insert"  
Altere:
```

```
root@sandbox:/usr/hdp/2.3.2.0-2950/spark/conf
# Set everything to be logged to the console
log4j.rootCategory=INFO, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c(1):
%m%n

# Settings to quiet third party logs that are too verbose
log4j.logger.org.eclipse.jetty=WARN
log4j.logger.org.eclipse.jetty.util.component.AbstractLifeCycle=ERROR
log4j.logger.org.apache.spark.repl.SparkIMain$exprTyper=INFO
log4j.logger.org.apache.spark.repl.SparkILoop$SparkILoopInterpreter=INFO

-- INSERT --
```

Para:

```
root@sandbox:/usr/hdp/2.3.2.0-2950/spark
# Set everything to be logged to the console
log4j.rootCategory=ERROR, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c(1):
%m%n

# Settings to quiet third party logs that are too verbose
log4j.logger.org.eclipse.jetty=WARN
log4j.logger.org.eclipse.jetty.util.component.AbstractLifeCycle=ERROR
log4j.logger.org.apache.spark.repl.SparkIMain$exprTyper=INFO
log4j.logger.org.apache.spark.repl.SparkILoop$SparkILoopInterpreter=INFO

-- INSERT --
```

Aperte a tecla "ESC"

Aperte agora as teclas ":wq" e depois "ENTER"

Digite agora cd ..

bin/pyspark

```
Warning! Please use the new key spark.yarn.am.waittime instead.
Welcome to
```



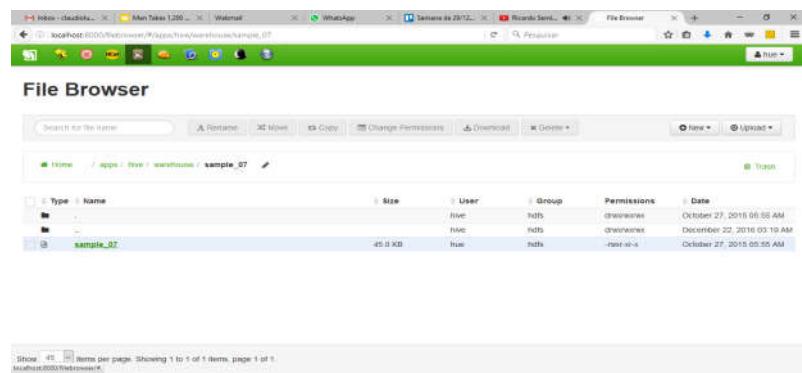
version 1.4.1

```
Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
```

```
SparkContext available as sc, HiveContext available as sqlContext.
```

```
>>>
```

- 10) Vamos agora listar um programa simples para entendimento do pyspark
Veja o seguinte arquivo de exemplo do HIVE que esta no HDFS:



11) O seguinte programa Spark acesso o arquivo listado anteriormente:

```
lines=sc.textFile("/apps/hive/warehouse/sample_07/sample_07")
lines.count()
lines.first()
```

```
[root@sandbox ~]# root@sandbox:/usr/hdp/2.3.0.2950/spark
>>>
KeyboardInterrupt
>>>
[2]+  Stopped                  bin/pyspark
[root@sandbox spark]# vi conf/log4j.properties
[root@sandbox spark]# bin/pyspark
Python 2.6.6 (r266:84292, Jul 23 2015, 15:22:56)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Welcome to


$$\sqrt{\frac{\sqrt{5}-1}{2}} \approx \frac{1}{\sqrt{5}}$$

version 1.4.1

Using Python version 2.6.6 (r266:84292, Jul 23 2015 15:22:56)
SparkContext available as sc, HiveContext available as sqlContext.
>>> lines=sc.textFile("/apps/hive/warehouse/sample_07/sample_07")
>>> lines.count()
823
>>> lines.first()
u'00-0000\tAll Occupations\t134354250\t40690'
>>>
```

Digitale CTRL + Z

12) Vamos agora acessar a interface usando o Scala e executar a mesma tarefa:

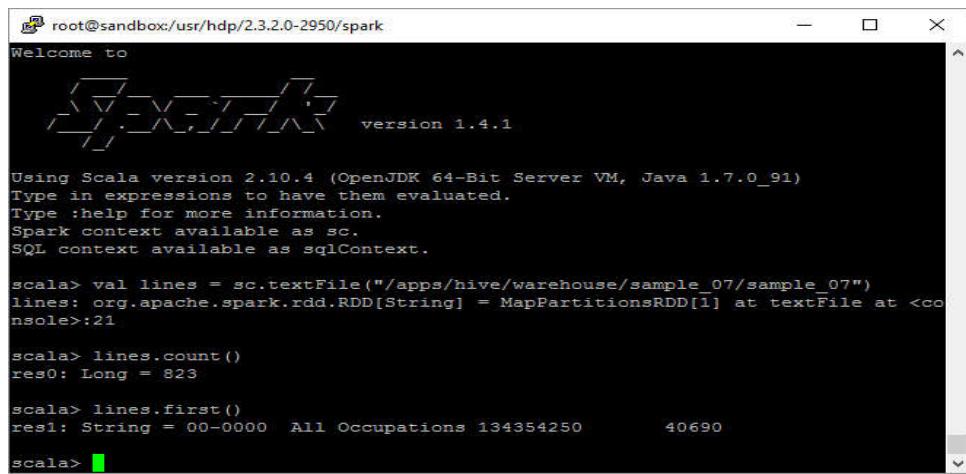
Digit /bin/spark-shell

```
Digitec / $ bin/Spark Shell
spark-class  sparkr          spark-shell   spark-sql    Spark-submit
[root@sandbox spark]# bin/spark-shell
Welcome to


$$\begin{array}{c} \backslash \swarrow \\ \diagup \end{array} \begin{array}{c} \diagdown \end{array} \begin{array}{c} \diagup \diagdown \\ \diagup \diagdown \end{array}$$
 version 1.4.1

Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.7.0_91)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.
SQL context available as sqlContext.

scala> |
Digitec:
```



The screenshot shows a terminal window with the following text:

```
root@sandbox:/usr/hdp/2.3.2.0-2950/spark
Welcome to

$$\sqrt{v} \times \sqrt{v} = \sqrt{v^2}$$

version 1.4.1

Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.7.0_91)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context available as sc.
SQL context available as sqlContext.

scala> val lines = sc.textFile("/apps/hive/warehouse/sample_07/sample_07")
lines: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[1] at textFile at <console>:21

scala> lines.count()
res0: Long = 823

scala> lines.first()
res1: String = 00-0000 All Occupations 134354250 40690

scala>
```

Soluções para processamento paralelo e distribuído de dados

Spark – RDD's



Ilustração de Oliver Munday

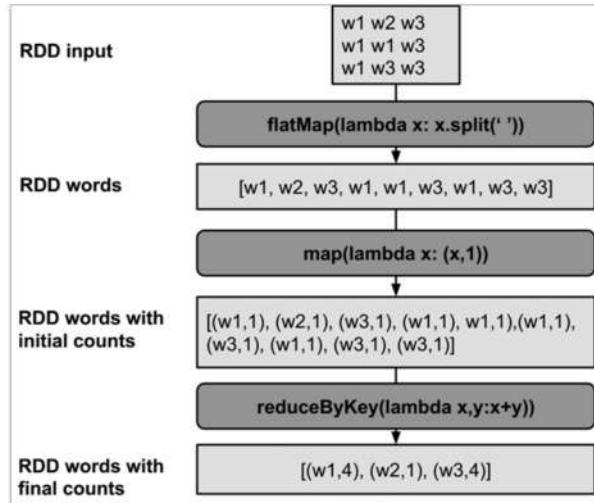
Spark - RDD's

Conceitos

- É uma coleção distribuída de elementos;
- Lógica de trabalho no Spark:
 - Criar RDD's
 - Transformar RDD's existentes
 - Executar operações nos RDD's para gerar resultados
- De forma transparente:
 - Spark distribui os dados dos RDD's no cluster;
 - Paraleliza as operações executadas nos RDD's

Spark - RDD's

Conceitos



Spark - RDD's

Conceitos

- RDD's representam um conjunto distribuídos de objetos 'imutáveis';
- Imutável = não volatilidade:
 - Facilita o processo de distribuição, replicação e compartilhamento dos dados
 - Dados imutáveis podem permanecer em disco ou memória de maneira indistinta
- Facilitam a tolerância a falhas

Spark - RDD's

Conceitos

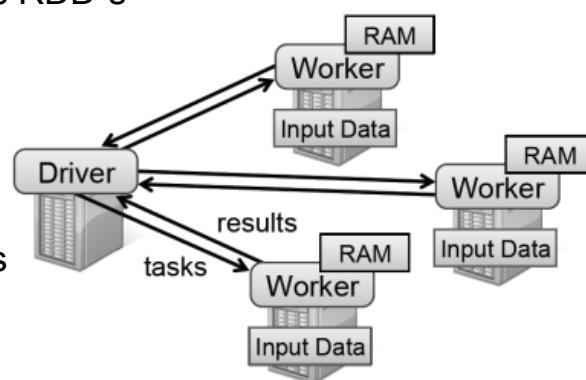
- RDD's são divididos em várias '*partitions*';
- As '*partitions*' podem residir em vários nós do *Cluster*;
- Criação de RDD's é feito de duas maneiras
 - Carregando dados de forma externa, por exemplo;
 - Distribuindo uma coleção de objetos no programa *Driver*
- O *Driver program* é quem coordena as atividades dos e os *Workers* armazena e manipulam as partições dos RDD's

Spark - RDD's

Conceitos

- Driver:
 - Define e invoca os ações(actions) nos RDD's
 - Rastreia as alterações nos RDD's

- Workers:
 - Armazenam as partições dos RDD's
 - Executam transformações (transformations) nos RDD's



Spark - RDD's

Conceitos

- Uma vez criado os RDD's duas operações são possíveis;
- Spark trata estas duas operações de maneira distinta;

-Transformations:

- Cria um RDD a partir de um anterior;
- Exemplo é o *transformation filter*

```
lines.filter(lambda line: "Python" in line)
```

- O resultado do comando anterior irá gerar outro RDD, resultado do filtro aplicado;
- Sempre retornam um RDD

Spark - RDD's

Conceitos

- Actions

- Calculam resultados a partir de um RDD;
- Este resultado é então enviado para o *Driver program* ou pode ser salvo, por exemplo, no HDFS;
- Retornam algum outro tipo de dados, com exceção de RDD;
- Os valores de retorno são enviados para o *Driver program* ou persistidos em disco;
- Por padrão todas as vezes que um *action* é acionada os RDD's envolvidos são recalculados;

Spark - RDD's

Conceitos

- **Actions:**

- Forçam a execução dos *transformations*;
- Exemplos de ações já vistas anteriormente:

```
>>> lines.count() # Operação no RDD  
>>> lines.first() # linha 1 do arquivo
```

Spark - RDD's

Conceitos

- *Lazy Evaluation*

- As transformações (*transformations*) são sempre postergadas até encontrar uma ação (*action*);
- A princípio isto pode parecer estranho, mas para Big Data faz todo o sentido;
- Deve-se considerar que todo programa Spark é um DAG, por conceito de implementação. Veja o programa abaixo:

```
lines = sc.textFile("/apps/hive/warehouse/sample_07/sample_07")  
lines.count()
```

Spark - RDD's

Conceitos

- *Lazy Evaluation*

- Considere este programa agora:

```
lines = sc.textFile("/apps/hive/warehouse/sample_07/sample_07")
lines.first()
```

- O conceito de avaliação tardia em conjunto com os DAG's fazem com as otimizações de código no Spark sejam muito boas, conforme o exemplo anterior;
- OBS.: por padrão os RDD's são recalculados a cada vez que uma '*action*' é executada;

Spark - RDD's

Conceitos

- *Lazy Evaluation*

- Este conceito altera o entendimento sobre RDD's:

- São formas de calcular *stages* de processamento;
- Formas como as lógicas são realizadas a partir do dado de origem;
- Um programa spark, em essência, segue a lógica de um processo ETL (clássico em ambientes de DW);
- Um recomendado importante é: quebre seu programa spark, via de regra, em várias estágios de processamento;

Spark - RDD's

Conceitos

- Noções de persistência
 - Para reusar o RDD em várias '*actions*' o comando '*persist*' deve ser explicitado
 - Há algumas opções para persistência dos dados;
 - Uma vez que o '*persist*' é utilizado, não haverá mais recálculos quando houver outras '*actions*';
 - Pragmaticamente: você vai persistir os dados em memória uma vez e fazer várias consultas ('*actions*') nestes dados em memória;

Spark - RDD's

Conceitos

- Noções de persistência

Exemplo:

```
>>> lines = sc.textFile("/apps/hive/warehouse/sample_07/sample_07") # Criação do RDD
>>> lines.persist() # persistência dos dados ou cache() para memoria

>>> lines.persist().is_cached # Esta na memoria

>>> lines.count() # Operação no RDD
>>> lines.first() # linha 1 do arquivo
```

Spark - RDD's

Conceitos

- Noções de persistência

StorageLevel	Espaço usado	CPU	Em memória	Em Disco
MEMORY_ONLY	Alto	Baixo	S	N
MEMORY_ONLY_SER	Baixo	Alto	S	N
MEMORY_AND_DISK	Alto	Médio	Parcial	Parcial
MEMORY_AND_DISK_SER	Baixo	Alto	Parcial	Parcial
DISK_ONLY	Baixo	Alto	N	S

- Há o método *unpersist()*;

Spark - RDD's

Prática – Verificando processos e RDD's

Curso: Ciência de dados e Big Data

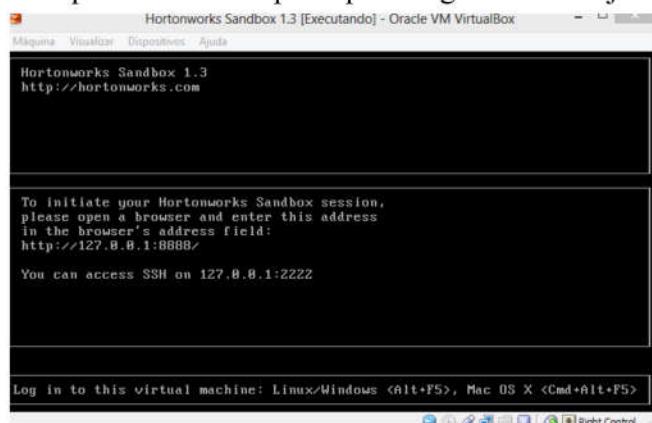
Professor: Cláudio Lúcio

Atividade Verificando processos e RDD's

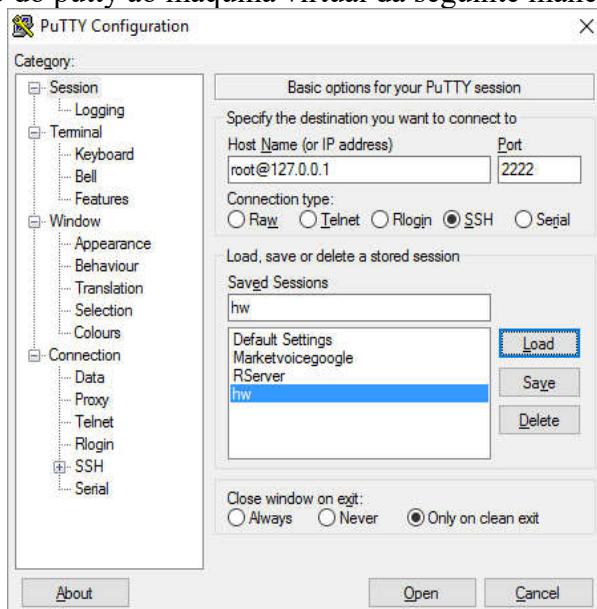
- 1) Para acesso ao Shell do Spark vamos usar a versão do Spark que vem na máquina virtual do HortonWorks:

Esta é a versão 2.3.2 do produto e pode ser obtida em:
<http://hortonworks.com/products/hortonworks-sandbox/>

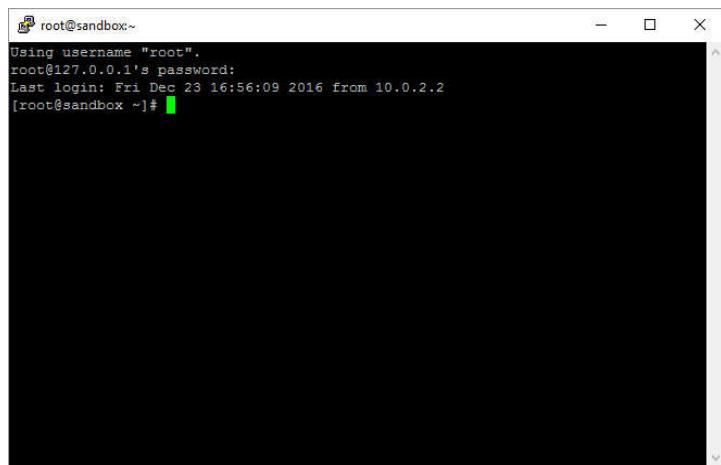
- 2) Inicialize a máquina virtual. Espere que a seguinte tela seja exibida:



- 3) A recomendação é usar um cliente para acesso SSH ao servidor do spark. Baixe a ferramenta putty.exe;
- 4) Configure o acesso do putty ao máquina virtual da seguinte maneira:

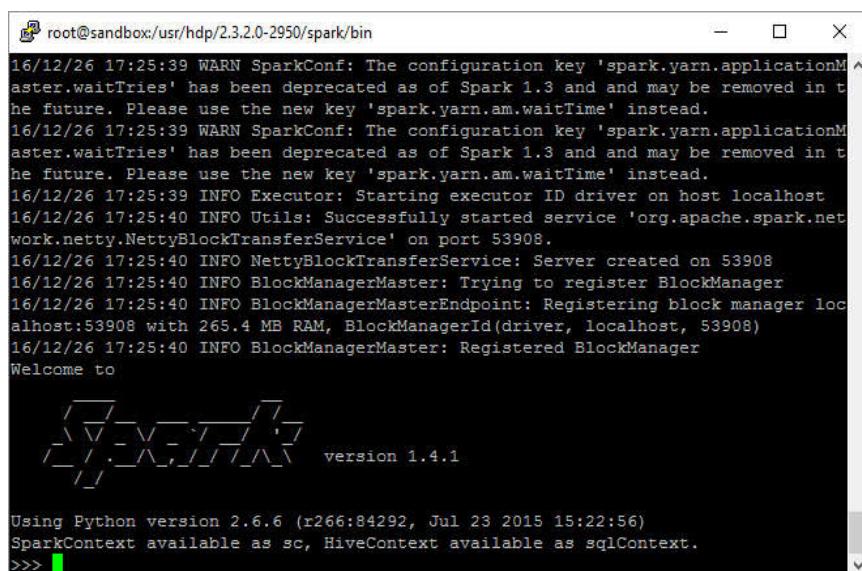


- 5) Clique em “Open” e então digite a senha do Root: hadoop



- 6) Primeiramente vamos interagir com a interface python para o Spark, para tal vamos acessar o bin/pyspark:
 - 7) Digite os seguintes comandos:

```
cd /usr/hdp/2.3.2.0-2950/spark/bin  
pyspark
```



- 8) Veja que estamos com a versão 1.4.1 do Spark;
 - 9) Vamos agora fazer o upload de arquivo para o HDFS e posteriormente vamos processá-lo utilizando um programa Spark. Crie uma pasta chamada **sparkfiles** no diretório do usuário Hue:

The screenshot shows the Hue File Browser interface at the URL `localhost:8000/filebrowser/view/User/hue`. The browser has a green header bar with various icons. The main area is titled "File Browser". At the top, there's a search bar labeled "Search for file name" and several action buttons: "Rename", "Move", "Copy", "Change Permissions", "Download", "Delete", "Now", and "Upload". Below the toolbar, the breadcrumb navigation shows "Home / user / hue". On the right, there's a "Trash" link. The main content area displays a table of HDFS items:

Type	Name	Size	User	Group	Permissions	Date
File	~		hue	hdfs	drwxrwxrwx	December 27, 2016 08:46 AM
File	hdfs		hue	hdfs	drwxr-xr-x	December 22, 2016 10:56 AM
File	hue		hue	hdfs	drwxrwxrwx	October 27, 2015 05:55 AM
File	jobsub		hue	hdfs	drwxrwxrwx	October 27, 2015 05:55 AM
File	oozie		hue	hdfs	drwxrwxrwx	October 27, 2015 05:55 AM
File	sparkfiles		hue	hdfs	drwxr-xr-x	December 27, 2016 08:46 AM

At the bottom, there are pagination controls: "Show 45" and "Items per page: Showing 1 to 3 of 3 items, page 1 of 1".

10) Vamos agora fazer o upload do arquivo **PrideandPrejudice.zip**

The screenshot shows the Hue File Browser interface. At the top, there's a toolbar with icons for search, rename, move, copy, change permissions, download, delete, new, and upload. Below the toolbar is a breadcrumb navigation bar showing the path: Home / user / hue / sparkfiles. The main area is a table listing files. One file, 'PrideandPrejudice.txt', is highlighted in green. The table columns are Type, Name, Size, User, Group, Permissions, and Date. The file details are: Type: File, Name: PrideandPrejudice.txt, Size: 700.8 KB, User: hue, Group: hdfs, Permissions: drwxr-xr-x, Date: December 27, 2016 08:51 AM. Below the table, there's a pagination message: 'Show 45 items per page. Showing 1 to 1 of 1 items, page 1 of 1.'

11) Adicione o direito de outros usuários escreverem neste diretório.

12) Marque o checkbox do diretório e clique no botão 'Change permissions'

13) Marque o checkbox da linha 'Write' e da coluna 'Other' e clique no botão 'submit'

The screenshot shows the Hue File Browser with a 'Change Permissions' dialog box open. The dialog has three tabs: User, Group, and Other. Under the User tab, the 'Write' checkbox is checked. Under the Other tab, the 'Other' checkbox is checked. There are also 'Sticky' and 'Recursive' checkboxes. At the bottom right of the dialog are 'Cancel' and 'Submit' buttons. In the background, the file browser lists several directories: home, user, hue, jesus, osbie, and sparkfiles. To the right of the dialog, there's a sidebar with a list of files and their last modified dates.

14) Vamos agora entrar no pySpark e trabalhar com este arquivo

15) Veja o programa abaixo:

```
tokenized = sc.textFile('/user/hue/sparkfiles/PrideandPrejudice.txt').flatMap(lambda line:  
    line.split(" ")).  
  
allStopWords={'about':1, 'above':1, 'after':1, 'again':1, 'against':1, 'all':1, 'am':1,  
    'an':1, 'and':1, 'any':1, 'are':1, 'arent':1, 'as':1, 'at':1, 'be':1, 'because':1, 'been':1,  
    'before':1, 'being':1, 'below':1, 'between':1, 'both':1, 'but':1, 'by':1, 'cant':1,  
    'cannot':1, 'could':1, 'couldnt':1, 'did':1, 'didnt':1, 'do':1, 'does':1, 'doesnt':1,  
    'doing':1, 'dont':1, 'down':1, 'during':1, 'each':1, 'few':1, 'for':1, 'from':1, 'further':1,  
    'had':1, 'hadnt':1, 'has':1, 'hasnt':1, 'have':1, 'havent':1, 'having':1, 'he':1, 'hed':1,  
    'hell':1, 'hes':1, 'her':1, 'here':1, 'heres':1, 'hers':1, 'herself':1, 'him':1, 'himself':1,  
    'his':1, 'how':1, 'hows':1, 'i':1, 'id':1, 'ill':1, 'im':1, 'ive':1, 'if':1, 'in':1,  
    'into':1, 'is':1, 'isnt':1, 'it':1, 'its':1, 'itself':1, 'lets':1, 'me':1, 'more':1,  
    'most':1, 'mustnt':1, 'my':1, 'myself':1, 'no':1, 'nor':1, 'not':1, 'of':1, 'off':1, 'on':1,  
    'once':1, 'only':1, 'or':1, 'other':1, 'ought':1, 'our':1, 'ours':1, 'ourselves':1, 'out':1,  
    'over':1, 'own':1, 'same':1, 'shant':1, 'she':1, 'shed':1, 'shell':1, 'shes':1, 'should':1,  
    'shouldnt':1, 'so':1, 'some':1, 'such':1, 'than':1, 'that':1, 'thats':1, 'the':1, 'their':1,  
    'theirs':1, 'them':1, 'themselves':1, 'then':1, 'there':1, 'theres':1, 'these':1, 'they':1,  
    'theyd':1, 'theyll':1, 'theyre':1, 'theyve':1, 'this':1, 'those':1, 'through':1, 'to':1,  
    'too':1, 'under':1, 'until':1, 'up':1, 'very':1, 'was':1, 'wasnt':1, 'we':1, 'wed':1,  
    'well':1, 'were':1, 'weve':1, 'were':1, 'werent':1, 'what':1, 'whats':1, 'when':1, 'whens':1,  
    'where':1, 'wheres':1, 'which':1, 'while':1, 'who':1, 'whos':1, 'whom':1, 'why':1, 'whys':1,  
    'with':1, 'wont':1, 'would':1, 'wouldnt':1, 'you':1, 'youd':1, 'youll':1, 'youre':1,  
    'youve':1, 'your':1, 'yours':1, 'yourself':1, 'yourselves':1}
```

```

# contar a ocorrência das palavras com alguns tratamentos
wordCounts = tokenized.map( lambda x: x.replace(',',' ').replace('.',' ').replace('-',' '))
.lower() ) \
.flatMap(lambda x: x.split()) \
.filter(lambda x: x not in allStopWords ) \
.map(lambda x: (x, 1)) \
.reduceByKey(lambda x,y:x+y) \
.map(lambda x:(x[1],x[0])) \
.sortByKey(False)

wordCounts.cache()
wordCounts.count()
wordCounts.take(10)

threshold = 10
filtered = wordCounts.filter(lambda pair:pair[0] >= threshold)
filtered.count()
filtered.first()

filtered.saveAsTextFile("/user/hue/sparkfiles/PrideandPrejudiceFILTERED_"+str(threshold)
+" .txt")

```

16) Execute o código acima e veja se o arquivo final será criado:

Type	Name	Size	User	Group	Permissions	Date
File	PrideandPrejudice.txt	700.8 KB	hue	hdfs	drwxr-xrwx	December 27, 2016 08:50 AM
File	PrideandPrejudiceFILTERED_10.txt	0 bytes	root	hdfs	drwxr-xr-x	December 27, 2016 11:15 AM

Type	Name	Size	User	Group	Permissions	Date
File	_SUCCESS	0 bytes	root	hdfs	drwxr-xr-x	December 27, 2016 11:15 AM
File	part-00000	18.9 KB	root	hdfs	-rw-r--r--	December 27, 2016 11:15 AM
File	part-00001	0 bytes	root	hdfs	-rw-r--r--	December 27, 2016 11:15 AM

17) Veja o conteúdo do arquivo **part-0000**

The screenshot shows a browser window with the URL localhost:8000/filebrowser/view//user/hue/sparkfiles/PrideandPrejudiceFILTERED_10.txt/part-00000. The page displays the first 4096 bytes of the file, which contains a list of tuples. The tuples represent words from the book "Pride and Prejudice" and their counts. The interface includes actions like View As Binary, Edit File, Download, and Refresh, along with file metadata such as Last Modified (Dec. 27, 2016, 11:15 a.m.), User (root), Group (hdfs), Size (18.9 KB), and Mode (100644).

Index	Word
1998	u'a'
1381	u''
769	u'm'
578	u'elizabeth'
409	u'will'
376	u'said'
344	u'darcy'
338	u'mrs'
328	u'much'
314	u'must'
297	u'i'
281	u'bennet'
272	u'miss'
262	u'one'
249	u'jane'
233	u'bingley'
228	u'know'
222	u'though'
217	u'never'
215	u'soon'
209	u'think'
209	u'can't'

18) Quando o pyspark é utilizado, de maneira local, como neste exemplo o Spark irá também apresentar uma interface para visualizar os jobs submetidos. Digite o endereço <http://localhost:4040> no browser:

The screenshot shows the PySparkShell application UI at localhost:4040/jobs/. It displays the completed jobs table and the event timeline.

Completed Jobs (6)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
5	runJob at PythonRDD.scala:366	2016/12/27 19:44:50	63 ms	1/1 (2 skipped)	1/1 (4 skipped)
4	count at <stdin>:1	2016/12/27 19:44:49	68 ms	1/1 (2 skipped)	2/2 (4 skipped)
3	runJob at PythonRDD.scala:366	2016/12/27 19:44:49	69 ms	1/1 (2 skipped)	1/1 (4 skipped)
2	count at <stdin>:1	2016/12/27 19:44:49	0.2 s	2/2 (1 skipped)	4/4 (2 skipped)
1	sortByKey at <stdin>:8	2016/12/27 19:44:48	71 ms	1/1 (1 skipped)	2/2 (2 skipped)
0	sortByKey at <stdin>:8	2016/12/27 19:44:47	1 s	2/2	4/4

Event Timeline

The event timeline shows the execution of the jobs over time. The x-axis represents time from 17:44:47 to 17:44:50. The y-axis shows executors and jobs. The timeline highlights the execution of sortByKey at <stdin>:8 (Job 0), count at <stdin>:1, runJob at PythonRDD.scala:366, and sortByKey at <stdin>:8 (Job 1).

19) Clique no link 'Event Time' e veja a ordem de execução do programa

The screenshot shows the PySparkShell application UI at localhost:4040/jobs/. It displays the completed jobs table and the event timeline.

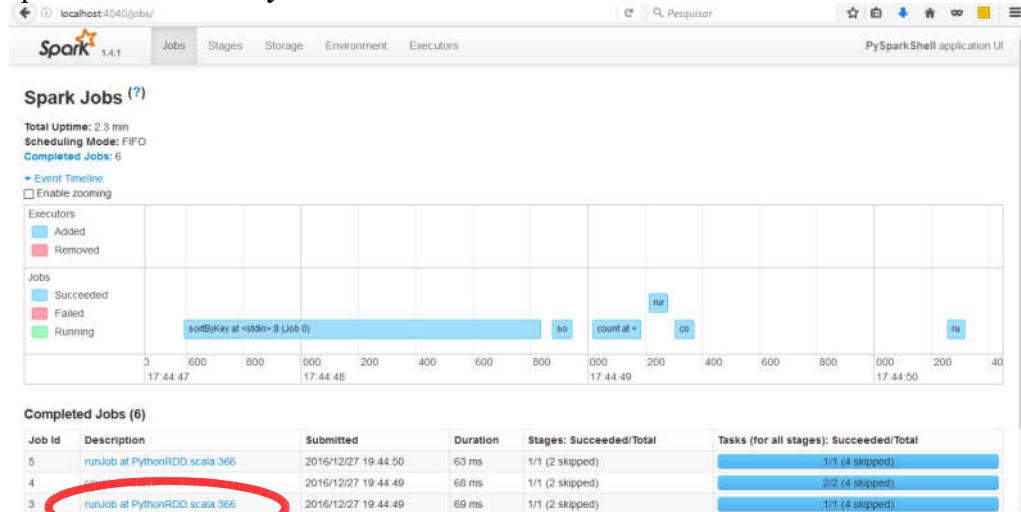
Completed Jobs (6)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
5	runJob at PythonRDD.scala:366	2016/12/27 19:44:50	63 ms	1/1 (2 skipped)	1/1 (4 skipped)
4	count at <stdin>:1	2016/12/27 19:44:49	68 ms	1/1 (2 skipped)	2/2 (4 skipped)
3	runJob at PythonRDD.scala:366	2016/12/27 19:44:49	69 ms	1/1 (2 skipped)	1/1 (4 skipped)

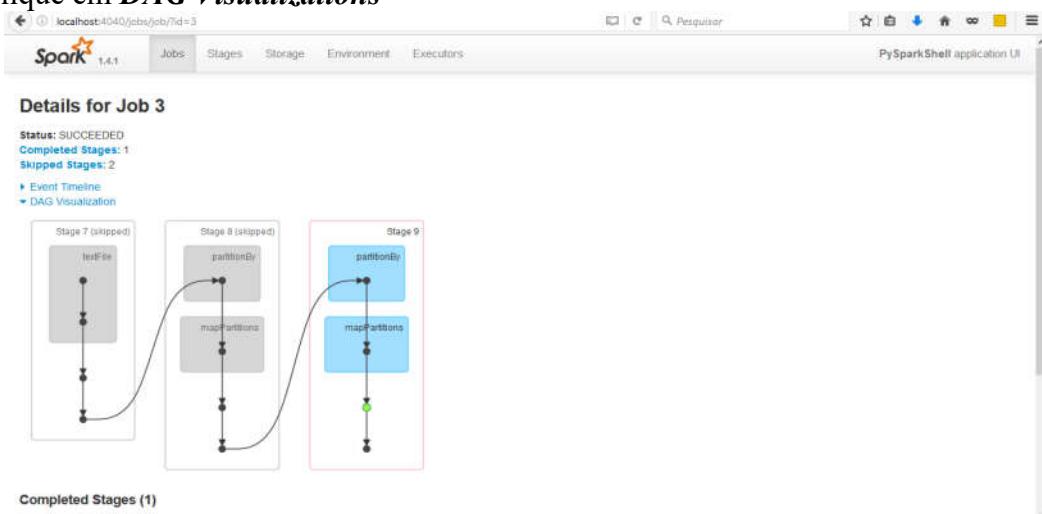
Event Timeline

The event timeline shows the execution of the jobs over time. The x-axis represents time from 17:44:47 to 17:44:50. The y-axis shows executors and jobs. The timeline highlights the execution of sortByKey at <stdin>:8 (Job 0), count at <stdin>:1, runJob at PythonRDD.scala:366, and sortByKey at <stdin>:8 (Job 1).

20) Clique em ***runJob at PythonRDD Scala...***



21) Clique em ***DAG Visualizations***



22) Clique na aba 'Stages' em que cada um dos passos é apresentado de forma mais detalhada:

localhost:4040/stages/

Spark 1.4.1 Jobs Stages Storage Environment Executors PySpark Shell application UI

Stages for All Jobs

Completed Stages: 8
Completed Stages (8)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
15	runJob at PythonRDD.scala:366	+details 2016/12/27 19:44:50	52 ms	1/1	38.9 KB			
12	count at <stdin>:1	2016/12/27 19:44:49	58 ms	2/2	105.4 KB			
9	runJob at PythonRDD.scala:366	+details 2016/12/27 19:44:49	57 ms	1/1	38.9 KB			
6	count at <stdin>:1	2016/12/27 19:44:49	65 ms	2/2			121.2 KB	
5	sortByKey at <stdin>:8	2016/12/27 19:44:49	78 ms	2/2			184.9 KB	121.2 KB
3	sortByKey at <stdin>:8	2016/12/27 19:44:48	61 ms	2/2			184.9 KB	
1	sortByKey at <stdin>:8	2016/12/27 19:44:48	73 ms	2/2			184.9 KB	
0	reduceByKey at <stdin>:6	2016/12/27 19:44:47	1 s	2/2	734.4 KB			184.9 KB

localhost:4040/stages/stage?id=9&attempt=0

23) Agora veja os detalhes da primeira execução **runJob at PythonRDD Scala...**. Clique neste link:

localhost:4040/stages/stage/9/attempt/0

Spark 1.4.1 Jobs Stages Storage Environment Executors PySpark Shell application UI

Details for Stage 9 (Attempt 0)

Total Time Across All Tasks: 44 ms
Input Size / Records: 38.9 KB / 12

- DAG Visualization
- Show Additional Metrics
- Event Timeline

Summary Metrics for 1 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	44 ms	44 ms	44 ms	44 ms	44 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Input Size / Records	38.9 KB / 12	38.9 KB / 12	38.9 KB / 12	38.9 KB / 12	38.9 KB / 12

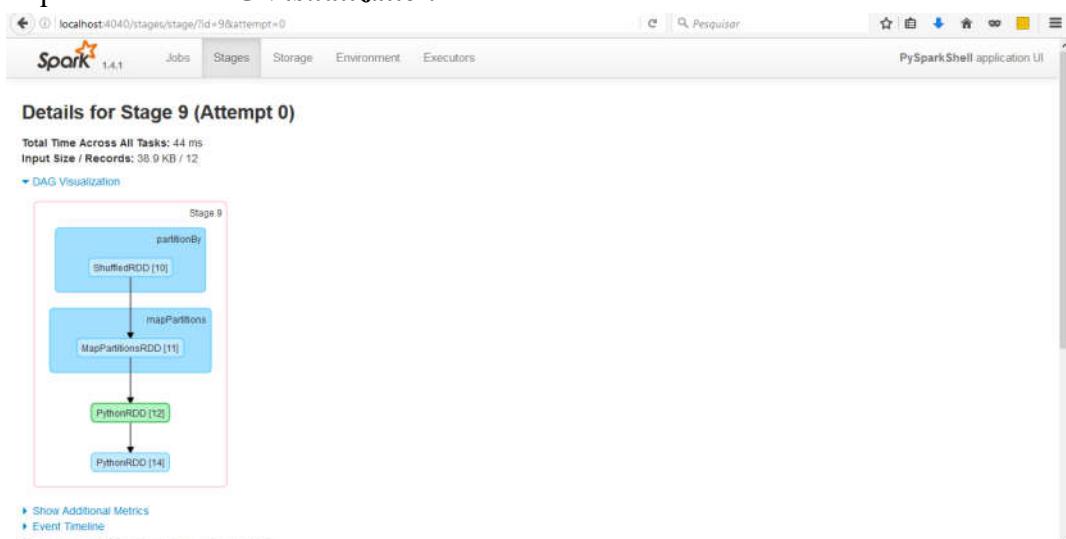
Aggregated Metrics by Executor

Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Succeeded Tasks	Input Size / Records
driver	localhost:44973	56 ms	1	0	1	38.9 KB / 12

Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	10	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/12/27 19:44:49	44 ms		38.9 KB (memory) / 12	

24) Clique no item **DAG Visualization**



25) Outra visualização é dos RDD's, clique em '**Storage**'

RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size in External Block Store	Size on Disk
PythonRDD	Memory Serialized 1x Replicated	2	100%	105.4 KB	0.0 B	0.0 B

26) Clique em Python RDD para ver as partições:

RDD Storage Info for PythonRDD

Storage Level: Memory Serialized 1x Replicated
Cached Partitions: 2
Total Partitions: 2
Memory Size: 105.4 KB
Disk Size: 0.0 B

Data Distribution on 1 Executors

Host	Memory Usage	Disk Usage
localhost:44973	105.4 KB (265.3 MB Remaining)	0.0 B

2 Partitions

Block Name	Storage Level	Size in Memory	Size on Disk	Executors
rdd_12_0	Memory Serialized 1x Replicated	38.9 KB	0.0 B	localhost:44973
rdd_12_1	Memory Serialized 1x Replicated	66.5 KB	0.0 B	localhost:44973

27) Por ultimo apague todo o diretório criado, utilizando seguinte linha de comando:

```
hadoop fs -rmr /user/hue/sparkfiles ou  
hadoop fs rm -r /user/hue/sparkfiles
```

Soluções para processamento paralelo e distribuído de dados

Spark – Algumas ações e transformações



Ilustração de Oliver Munday

Spark – Algumas Ações e Transformações

Passando funções:

- Muitas *transformations* e *actions* vão exigir como parâmetros funções que irão ser aplicadas aos dados ;
- Cada uma das linguagens utilizadas no Spark possui formas de passar tais funções para os *transformations* e *actions*;
- Vamos focar em Python e Scala;

Spark – Algumas Ações e Transformações

Passando funções:

Python

- Para funções curtas podemos utilizar o *lambda*

```
word = rdd.filter(lambda s: "error" in s)

def containsError(s):
    return "error" in s
word = rdd.filter(containsError)
```

Spark – Algumas Ações e Transformações

Passando funções:

Python

- Para funções mais longas use a função

```
def myFunc(s):
    palavras = s.split(" ")
    return len(palavras)

sc = SparkContext(...)
sc.textFile('arquivo.txt').map(MyFunc)
```

- Tome cuidado para não passar métodos de objetos, em alguns caso o objeto será serializado e enviado para o cluster;

Spark – Algumas Ações e Transformações

Passando funções:

Scala

- Para funções mais curtas use comando *inline*:

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

- Para funções longas:

```
object MyFunctions {
    def func1(s: String): String = { ... }
}

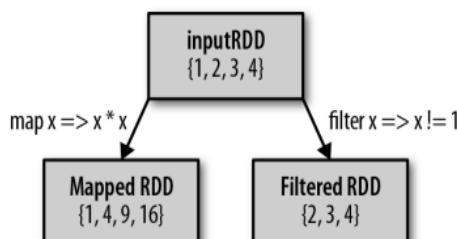
myRdd.map(MyFunctions.func1)
```

Spark – Algumas Ações e Transformações

Transformações - Básico:

- Duas transformações muito comuns são:

- *Map()*: aplica uma função a cada elemento do RDD, e os elementos gerados pela função formam o novo RDD
- *Filter()*: aplica uma função e retorna o conjunto de elementos filtrados com um novo RDD



Spark – Algumas Ações e Transformações

Transformações - Básico:

- Duas transformações muito comuns são:

- *Map()*

- Pode ser utilizada para um grande variedade de tarefas: buscar o conteúdo de uma URL ou retornar um número elevado ao cubo;
 - O RDD de entrada não precisa ser do mesmo tipo do RDD de saída:

```
txt_number = sc.parallelize([u'1', u'2', u'3', u'4', u'1', u'2'])
int_number = txt_number.map(lambda x:int(x))
sum(int_number.collect())
```

Spark – Algumas Ações e Transformações

Transformações - Básico:

- Outras transformações:

- *flatMap()*

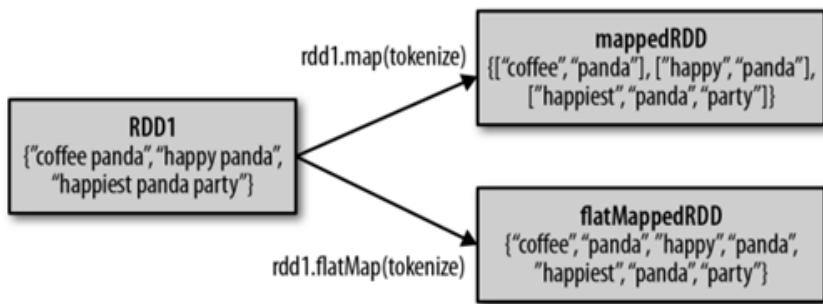
- Similar ao *Map* com a diferença que para cada item de entrada pode ser gerada um mais itens de saída;
 - Retorna um *iterator* para o resultado;
 - Exemplo clássico de *flatMap* é quebrar um linha em várias palavras

```
linhas = sc.parallelize(["hello world", "hi"])
palavra = linhas.flatMap(lambda linha: linha.split(" "))
palavra.take(3) # retorno ['hello', 'world', 'hi']
```

Spark – Algumas Ações e Transformações

Transformações - Básico:

- Outras transformações:
 - Diferença entre *Map()* e *flatMap()*



Spark – Algumas Ações e Transformações

Transformações - Básico:

- Transformações de conjuntos:
 - Os RDD's não são exatamente conjuntos(sets ou tabelas), mas há um bom conjunto de transformações para lidar com conjuntos;
 - *Distinct()* = para gerar elementos distintos de um RDD;
 - Tome cuidado com esta operação pois ela é, computacionalmente, custosa;
 - *Union()* = faz união dos dados dos conjuntos. As repetições são mantidas;

Spark – Algumas Ações e Transformações

Transformações - Básico:

- Transformações de conjuntos:
 - *intersection()* = retorna os elementos existente em ambos os RDD's (elimina os duplicados = atua como o *distinct()*, mantendo sua observação quanto ao desempenho);
 - *subtract()* = retorna os valores existentes apenas no primeiro RDD e que não existem no segundo RDD (tome cuidado com o desempenho);

Spark – Algumas Ações e Transformações

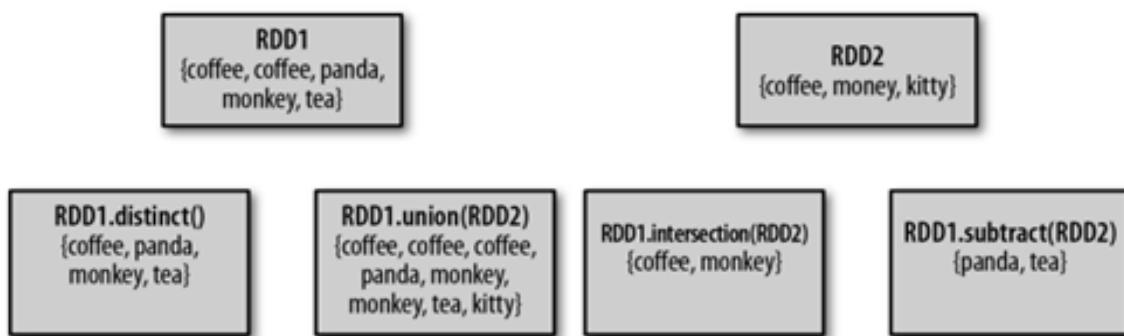
Transformações - Básico:

- Transformações de conjuntos:
 - *intersection()* = retorna os elementos existente em ambos os RDD's (elimina os duplicados = atua como o *distinct()*, mantendo sua observação quanto ao desempenho);
 - *subtract()* = retorna os valores existentes apenas no primeiro RDD e que não existem no segundo RDD (tome cuidado com o desempenho);

Spark – Algumas Ações e Transformações

Transformações - Básico:

- Transformações de conjuntos:

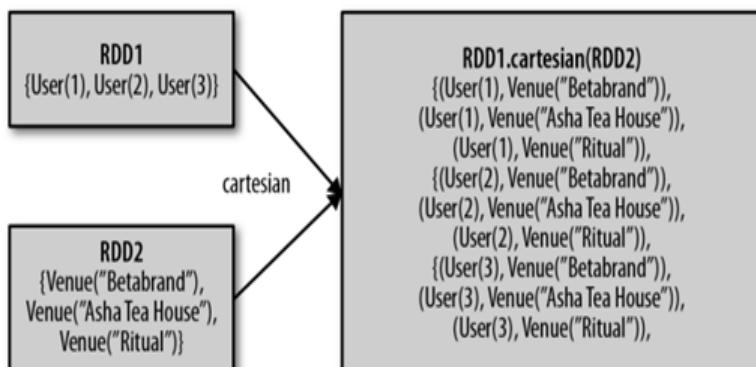


Spark – Algumas Ações e Transformações

Transformações - Básico:

- Transformações de conjuntos:

- *cartesian()* = realiza um produto cartesiano entre os dois RDD's envolvidos. Veja o exemplo:



Spark – Algumas Ações e Transformações

Transformações - Básico:

- Transformações de conjuntos:
 - *parallelize()*
 - É um trasnformation
 - Não é muito utilizado na prática, pois os seus dados tem caber na memória de uma única máquina

Spark – Algumas Ações e Transformações

Ações - Básico:

- *reduce()*
 - Talvez seja a ação mais comum;
 - Aplica uma função em dois elementos (tipo de dados) do RDD gerando um resultado do mesmo tipo;
 - O exemplo mais comum de função é “+”(somar o RDD);
 - Em geral o reduce é utilizado para agregar elementos;
- *fold()*
 - Faz a mesma coisa que o Reduce, mas atribui um valor inicial para cada elemento na soma;

```
sum = rdd.reduce(lambda x, y: x + y)
```

```
sc.parallelize([1, 2, 3, 4, 5]).fold(0, add)
```

Spark – Algumas Ações e Transformações

Ações - Básico:

- *Aggregate()*

- Tanto fold quanto reduce exigem que o retorno seja igual ao tipo do RDD trabalhado;
- Para evitar este problema o que poderia ser feito é criar uma função *map* e depois uma função *reduce*;
- É exatamente isto que a função *aggregate* faz, para tal devemos:
 - Fornecer valores iniciais (* atenção use sempre os valores = 0);
 - Função que será aplicada em cada elemento de cada partição para gerar o resultado intermediário de cada partição
 - Função para fazer *merge* dos resultados de parciais de cada partição

Spark – Algumas Ações e Transformações

Ações - Básico:

- *Aggregate()*

```
nums = sc.parallelize([1,2,3,4], 2)
seqOp = (lambda x, y: (x[0] + y, x[1] + 1))
combOp = (lambda x, y: (x[0] + y[0], x[1] + y[1]))
nums.aggregate((0, 0), seqOp, combOp)
```

(0, 0) <-- zeroValue

[1, 2] [3, 4]

0 + 1 = 1 0 + 3 = 3
0 + 1 = 1 0 + 1 = 1

1 + 2 = 3 3 + 4 = 7
1 + 1 = 2 1 + 1 = 2

| v | v |

(3, 2) (7, 2)

/ \ / \

-----|-----

| combOp |

| v |

(10, 4)

Spark – Algumas Ações e Transformações

Ações - Básico:

- *collect()*

- Forma mais simples de obter todo conteúdo de um RDD. Obs.: cuidado no uso para grandes bases de dados ;

- *take(n)*

- Retorna os *n* elementos do RDD;

- *takeSample (withReplacement, num, seed)*

- Faz uma amostragem com ou sem reposição (assume uma distribuição uniforme dos dados);

Spark – Algumas Ações e Transformações

Ações - Básico:

- *foreach()*

- Executa uma função em cada elemento do RDD, mas não retorna nada para o Driver ;

- Pense no caso, por exemplo, de ler um arquivo Json e armazená-lo em um outro banco de dados;

- *count()*

- Retorna a quantidade de elementos do RDD;

- *countByValue ()*

- Retorna um mapa com a quantidade de elementos, por valor, do RDD;

Spark – Algumas Ações e Transformações

Ações - Básico:

- Exemplos

Entrada	Função	Saída
{1, 2, 3, 3}	rdd.collect()	{1, 2, 3, 3}
{1, 2, 3, 3}	rdd.count()	4
{1, 2, 3, 3}	rdd.countByValue()	{(1, 1), (2, 1), (3, 2)}
{1, 2, 3, 3}	rdd.top(2)	{3, 3}
{1, 2, 3, 3}	rdd.takeSample(false, 1)	1 – estocástico
{1, 2, 3, 3}	rdd.foreach(func)	Não retorna nada

Spark – Algumas Ações e Transformações

Prática

- Trabalhando com algumas funções do Spark

Curso: Ciência de dados e Big Data

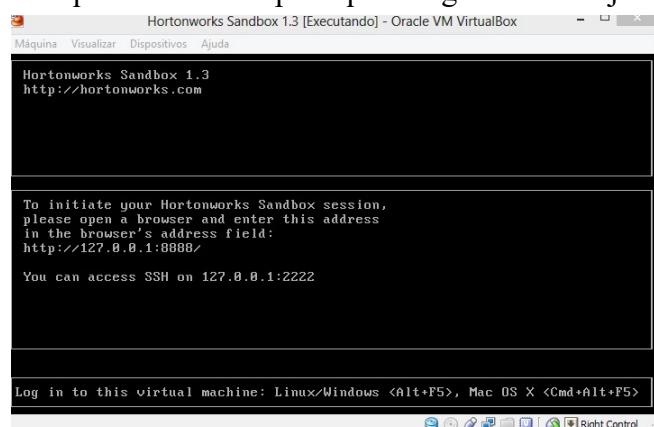
Professor: Cláudio Lúcio

Atividade Verificando processos e RDD's

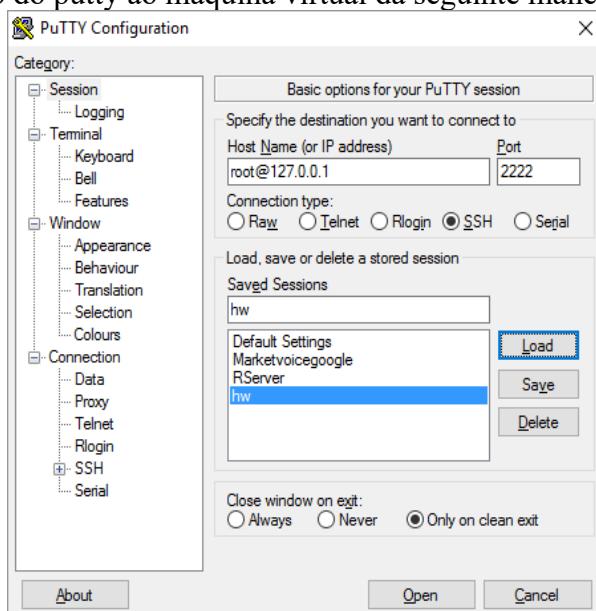
- 1) Para acesso ao Shell do Spark vamos usar a versão do Spark que vem na máquina virtual do HortonWorks:

Esta é a versão 2.3.2 do produto e pode ser obtida em:
<http://hortonworks.com/products/hortonworks-sandbox/>

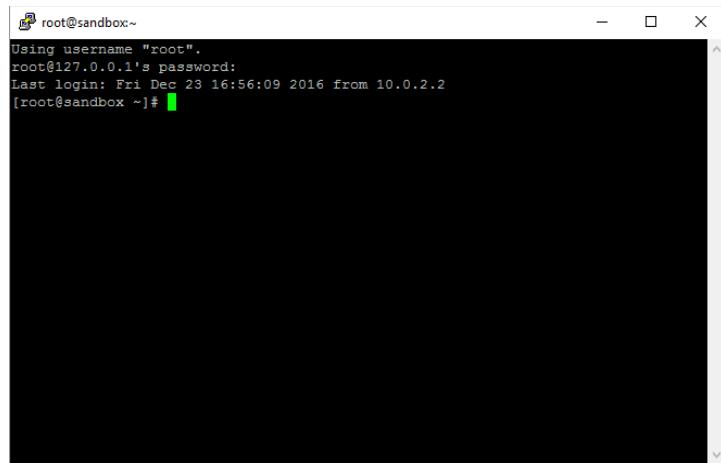
- 2) Inicialize a máquina virtual. Espere que a seguinte tela seja exibida:



- 3) A recomendação é usar um cliente para acesso SSH ao servidor do spark. Baixe a ferramenta putty.exe;
- 4) Configure o acesso do putty ao máquina virtual da seguinte maneira:



- 5) Clique em “Open” e então digite a senha do Root: hadoop



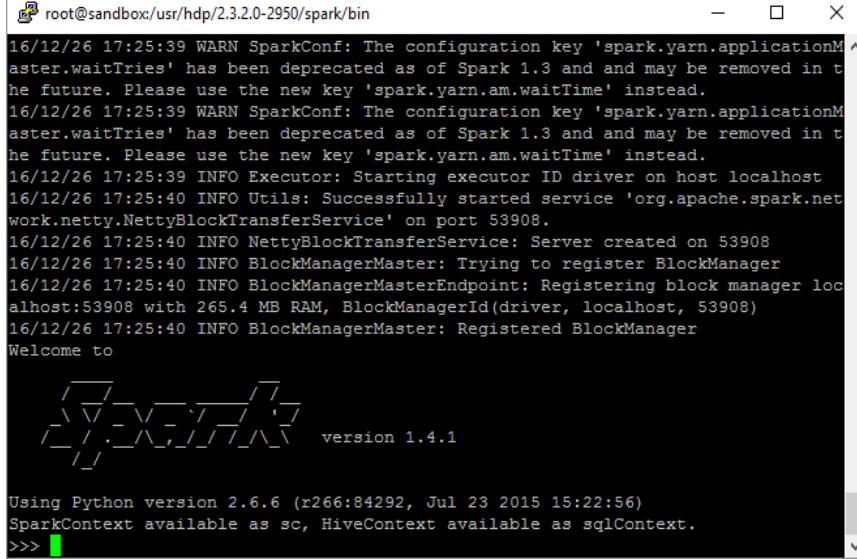
- 6) Vamos garantir que estamos com o usuário Admin do HDFS

```
sudo -u hdfs bash
```

- 7) Primeiramente vamos interagir com a interface python para o Spark, para tal vamos acessar o bin/pyspark:
 - 8) Digite os seguintes comandos:

```
cd /usr/hdp/2.3.2.0
```

pyspark



- 9) Veja que estamos com a versão 1.4.1 do Spark;

- 10) Vamos inicialmente utilizar uma base de dados para testar alguns métodos

```
pessoas = []
pessoas.append({'id':1,'nome':'Bob', 'idade':45,'gen':'M'})
pessoas.append({'id':2,'nome':'Gloria', 'idade':43,'gen':'F'})
pessoas.append({'id':4,'nome':'Albert', 'idade':28,'gen':'M'})
pessoas.append({'id':5,'nome':'Laura', 'idade':33,'gen':'F'})
pessoas.append({'id':8,'nome':'Simone', 'idade':18,'gen':'T'})
pessoas.append({'id':12,'nome':'Marta', 'idade':45,'gen':'F'})
pessoas.append({'id':45,'nome':'Jairo', 'idade':82,'gen':'M'})
pessoas.append({'id':13,'nome':'Teste', 'idade':38,'gen':'T'})
pessoasRdd=sc.parallelize(pessoas)
```

- 11) Utilizar o comando collect para vizualizar o conteúdo do RDD

```
pessoasRdd.collect()
```

12) Agora vamos fazer uma 'consulta' para verificar as pessoas com idade maior do que 30 anos

```
pessoas_M30 = pessoasRdd.filter(lambda x: x['idade']>30)
pessoas_M30.collect()
pessoas_M30.count()
```

13) Agora vamos fazer uma 'consulta' para verificar do sexo masculino

```
pessoas_GenM = pessoasRdd.filter(lambda x: x['gen'].upper() == 'M')
pessoas_GenM.collect()
pessoas_GenM.count()
```

14) Agora vamos verificar a quantidade de pessoas por sexo

```
pessoas_AggGen = pessoasRdd.map(lambda gen: gen['gen'])
pessoas_AggGen.collect()
pessoas_AggGen.countByValue()
```

15) Precisamos fazer a média da idade das pessoas. Para isto vamos usar a função *aggregate*:

```
seqOp = (lambda x,y: (x[0] + y['idade'],x[1] + 1))
combOp = (lambda x,y: (x[0] + y[0], x[1] + y[1]))
[soma,quantidade] = pessoasRdd.aggregate((0,0), seqOp, combOp)
media = soma/quantidade
media
```

16) Vamos fazer a união de pessoas acima de 30 mais pessoas do sexo masculino:

```
pessoas_GenM_30 = pessoas_M30.union(pessoas_GenM)
pessoas_GenM_30.collect()
```

17) Pessoas que estão nos dois critérios:

```
pessoas_M30_id = pessoas_M30.map(lambda x: x['id'])
pessoas_GenM_id = pessoas_GenM.map(lambda x: x['id'])
pessoas_GenM_And_30 = pessoas_M30_id.intersection(pessoas_GenM_id)
int_GenM_And_30 = pessoas_GenM_And_30.collect()
pessoas_GenM_And_30_full = pessoasRdd.filter(lambda x: x['id'] in int_GenM_And_30 )
pessoas_GenM_And_30_full.collect()
```

18) Seleção aleatório de duas pessoas do conjunto:

```
pessoasRdd.takeSample(True,2)
```

19) Importação de arquivo :

```
import json
pessoasValFile = sc.textFile("/user/hue/pessoasval.json")
pessoasValRdd = pessoasValFile.flatMap(lambda arq : (json.loads(arq)))
pessoasValRdd.collect()
```

20) Quantidade de ocorrências por id :

```
pessoasValRdd_id = pessoasValRdd.map(lambda x: x['id'])
pessoasValRdd_id.countByValue()
```

Soluções para processamento paralelo e distribuído de dados

Spark – SQL



Spark – SQL

Introdução

- É a proposta Spark para lidar com dados estruturados e semi-estruturados (dados com algum tipo de esquema);
- Características
 - Carga de vários tipos de dados(Json, Hive, Parquet);
 - Consultas são realizadas utilizando SQL (através de um programa spark ou através conexões externas: JDBC/ODBC);
 - Forte integração para ser utilizada com as linguagens spark: Java, Scala e Python. Permitir mesclar um RDD com dados do SQL Spark

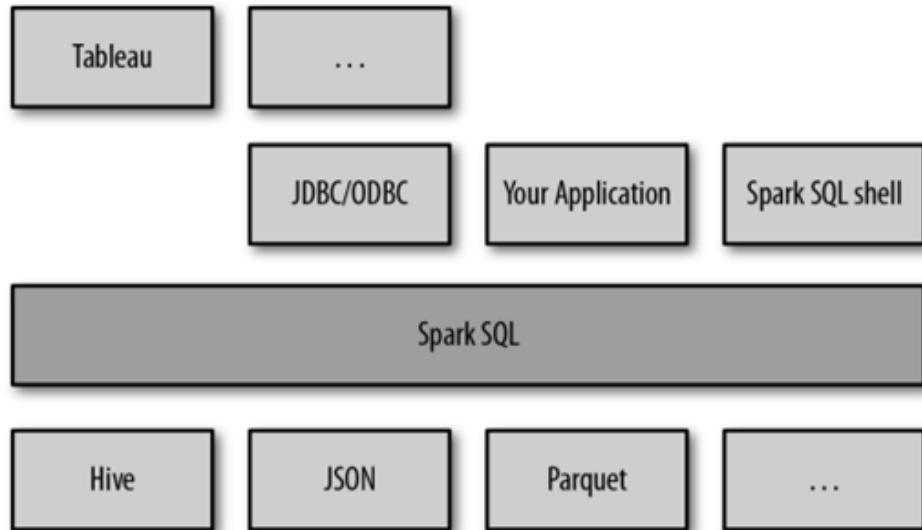
Spark – SQL

Introdução

- Os dados carregados com SQL ainda são RDD's;
- No entanto, do tipo SchemaRDD;
- SchemaRDD:
 - Conjunto de objetos do tipo *Row*;
 - SchemaRDD conhece seu esquema (campos);
 - Apesar de ser um RDD, possui otimizações internas para lidar com este tipo de dado;
 - Possui novas operações que não existem nos outros RDD's
 - Podem ser carregados a partir de várias fontes externas

Spark – SQL

Introdução



Spark – SQL

Introdução

- Pode possuir forte integração com o Hive (recomenda-se o seu uso com esta integração habilitada);
- No entanto, do tipo SchemaRDD;
- SchemaRDD:
 - Conjunto de objetos do tipo *Row*,
 - SchemaRDD conhece seu esquema (campos);
 - Apesar de ser um RDD, possui otimizações internas para lidar com este tipo de dado;
 - Possui novas operações que não existem nos outros RDD's
 - Podem ser carregados a partir de várias fontes externas

Spark – SQL

Exemplo inicial

- O exemplo abaixo mostra como é uma conexão utilizando o pyspark:

```
from pyspark.sql import HiveContext, Row # ou
from pyspark.sql import SQLContext, Row

hiveCtx = HiveContext(sc)
arqinput = hiveCtx.jsonFile('/user/hue/pessoasval.json')
arqinput.registerTempTable("pessoas")

allPessoas = hiveCtx.sql("""SELECT * FROM pessoas ORDER BY 1 LIMIT 10""")
allPessoas.collect()
```

Spark – SQL

SQL

- Para fazer consultas SQL utilize o método Sql() do HiveContext ou SQLContext;
- Antes da consulta os dados precisam ser 'conhecidos' pelo SparkSQL, no caso anterior, usamos um arquivo json;
- Uma tabela temporária é registrada com um metadado com nome;
- Consultas SQL pode ser feitas nesta estrutura;

Spark – SQL

SchemaRDD – versão Spark 1.x

- São similares a tabelas em bancos de dados relacionais;
- Os resultados das operações de carga dos dados ou SQL retornam SchemaRDD's;
- Internamente os SchemaRDD's são um conjunto de objetos do tipo Row;
- Cada objeto do tipo Row é uma especie de vetor com campos definidos pelo Schema;
- Como o SchemaRDD é ainda um RDD, todos os métodos utilizados para o RDD podem ser usados no SchemaRDD

Spark – SQL

SchemaRDD – versão Spark 1.x

- As tabelas temporárias registradas pelo método registerTempTable são alocadas temporariamente e assim que o Driver program termina elas são liberadas;
- Em python cada uma linhas pode ser acessada:

```
row[i]
row.datafieldname # veja que datafieldname é o nome real da coluna

#Exemplo
topTweetText = topTweets.map(lambda row: row.text)
```

Spark – SQL

SchemaRDD – versão Spark 1.x

- Tipos de dados:

Spark SQL/HiveQL type	Scala type	Java type	Python
TINYINT	Byte	Byte/byte	int/long (in range of –128 to 127)
SMALLINT	Short	Short/short	int/long (in range of –32768 to 32767)
INT	Int	Int/int	int or long
BIGINT	Long	Long/long	long
FLOAT	Float	Float/float	float
DOUBLE	Double	Double/double	float
DECIMAL	Scala.math.BigDecimal	Java.math.BigDecimal	decimal.Decimal

Spark – SQL

SchemaRDD – versão Spark 1.x

- Tipos de dados:

Spark SQL/HiveQL type	Scala type	Java type	Python
STRING	String	String	string
BINARY	Array[Byte]	byte[]	bytearray
BOOLEAN	Boolean	Boolean/boolean	bool
TIMESTAMP	java.sql.Timestamp	java.sql.Timestamp	datetime.datetime
ARRAY<DATA_TYPE>	Seq	List	list, tuple, or array
MAP<KEY_TYPE, VAL_TYPE>	Map	Map	dict
STRUCT<COL1: COL1_TYPE, ...>	Row	Row	Row

Spark – SQL

SchemaRDD – versão Spark 1.x

- A partir do Spark SQL 1.3 o schemaRDD foi renomeado para DataFrame (nesta versão não herda mais diretamente do RDD, mas possui os métodos que o mesmo tem)

Spark – SQL

Caching

- Como o SchemaRDD conhece os tipos de dados, o processo de carregar os dados para a memória é mais otimizada;
- Utiliza uma forma colunar de armazenamento dos dados;
- O método *cacheTable*("nometabela") garante que o método otimizado é utilizado ao invés do objeto completo;
- Os dados ficam na memória durante toda a execução do *Driver program*;
- Da mesma forma que os RDD, faz sentido manter em memória se for utilizado várias vezes;

Spark – SQL

Caching

- As tabelas também pode ser armazenadas na memória por comandos HIVE QL :

```
CACHE TABLE tablename  
UNCACHE TABLE tablename
```

-Este comando é mais utilizado com comando enviados por clientes, através de um JDBC *Server*, por exemplo ;

- As tabelas (*SchemaRDD* ou *DataFrames*) são apresentados com um RDD típico no Spark UI



HDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size in Tachyon	Size on Disk
Physical Volume	Memory Deserialized (Le Replicated)	2	100%	1001.7 kB	0.0 B	0.0 B
Physical Volume	Memory Serialized (Le Replicated)	2	100%	1001.7 kB	0.0 B	0.0 B
Physical Volume	Memory Deserialized (Le Shuffled)	2	100%	1001.7 kB	0.0 B	0.0 B
Physical Volume	Memory Serialized (Le Shuffled)	2	100%	1001.7 kB	0.0 B	0.0 B

Spark – SQL

Carregando e Salvando Dados

- As fontes para carga de dados são: tabelas *Hive*, arquivos *Json* e arquivos Parquet;
- Os RDD's existentes também pode ser transformados em SchemaRDD's
- A recomendação de uso do SparkSQL é quando existem vários cálculos a serem feitos na mesma estrutura (média, desvio padrão, soma, quantidade e etc....). Melhor Desempenho;
- O processo de junção do SQL também simplifica algumas tarefas;

Spark – SQL

Carregando e Salvando Dados

Hive

- Quando lê usando o Hive consegue utilizar todos os formatos de arquivos suportados pelo Hive (arquivos textos, Parquet, ORC);
- Para conectar o Spark SQL com o Hive é necessário prover uma configuração. Em essência: copiar o arquivo *hive-site.xml* para o *./conf* do Spark;

-Exemplo Python:

```
from pyspark.sql import HiveContext  
  
hiveCtx = HiveContext(sc)  
rows = hiveCtx.sql("SELECT key, value FROM mytable")  
keys = rows.map(lambda row: row[0])
```

Spark – SQL

Carregando e Salvando Dados

Parquet

- É um estrutura de arquivos orientado a coluna e bastante utilizada no Hadoop;
- Suporta todos os tipos do Spark SQL

-Exemplo Python:

```
rows = hiveCtx.parquetFile(parquetFile)
tbl = rows.registerTempTable("people")
pandaFriends = hiveCtx.sql("SELECT name FROM people WHERE favouriteAnimal = \"panda\"")
pandaFriends.saveAsParquetFile("hdfs://...")
```

Spark – SQL

Carregando e Salvando Dados

Json

- É necessário que os arquivos tenham o mesmo esquema (caso de uso: diretórios com milhões de arquivos json com o mesmo esquema);
- Spark SQL infere o esquema e permite o acesso aos nomes dos campos ;

-Exemplo Python:

```
{"name": "Sparky The Bear", "lovesPandas": true, "knows": {"friends": ["holden"]}}
```

```
root
  |-- knows: struct (nullable = true)
  |   |-- friends: array (nullable = true)
  |       |-- element: string (containsNull = false)
  |-- lovesPandas: boolean (nullable = true)
  |-- name: string (nullable = true)
```

```
printSchema()
```

Spark – SQL

Carregando e Salvando Dados

RDD

- A partir de um RDD é possível criar também uma schemaRDD;
- Exemplo Python:

```
happyPeopleRDD = sc.parallelize([Row(name="holden", favouriteBeverage="coffee")])
happyPeopleSchemaRDD = hiveCtx.inferSchema(happyPeopleRDD)
happyPeopleSchemaRDD.registerTempTable("happy_people")
```

Spark – SQL

Acessando dados

JDBC/ODBC

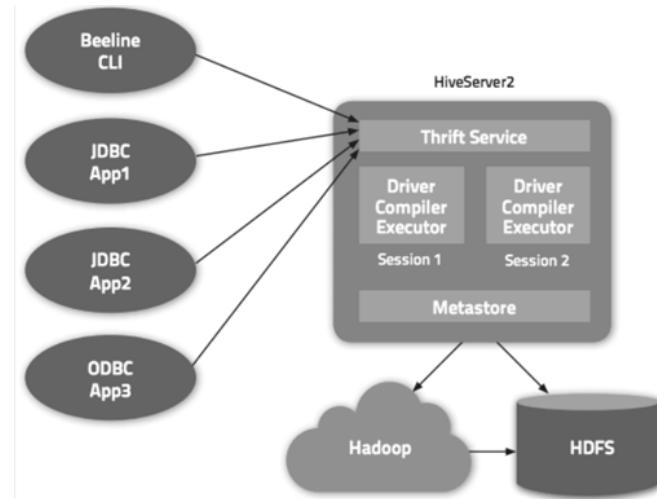
- Provê conexão JDBC para uso, por exemplo, por ferramentas de clássicas de BI ou Data Discovery;
- Servidor JDBC executa como um processo de *background* (é um *Driver program*);
- Permite conexão de vários usuários e compartilha as estruturas de memória entre os mesmos;
- O servidor Spark SQL JDBC corresponde o HiveServer2 no Hive. Esta conexão JDBC precisa do Hive habilitado para que possa funcionar;

Spark – SQL

Acessando dados

JDBC/ODBC

HiveServer2



Spark – SQL

Acessando dados

JDBC/ODBC

- O servidor pode ser iniciado em *sbin/start-thriftserver.sh*;

```
./sbin/start-thriftserver.sh --master sparkMaster
```
- Também pode ser usado o Beeline um cliente que realiza faz conexões com o servidor JDBC;
- O servidor JDBC gera arquivos de Log que pode ajudar na resolução de algum tipo de problema;

Spark – SQL

Acessando dados

JDBC/ODBC

- Muitos outros clientes também podem fazer conexão via *driver* ODBC;
- O Driver ODBC do Spark SQL é produzido pela Simba®;
- Adicionalmente se a ferramenta possui conexão com o Hive (Hive ODBC HortonWorks) ele poderá se conectar ao Spark SQL

Spark – Algumas Ações e Transformações

Prática

- Utilizando o Spark-sql

Curso: Ciência de dados e Big Data

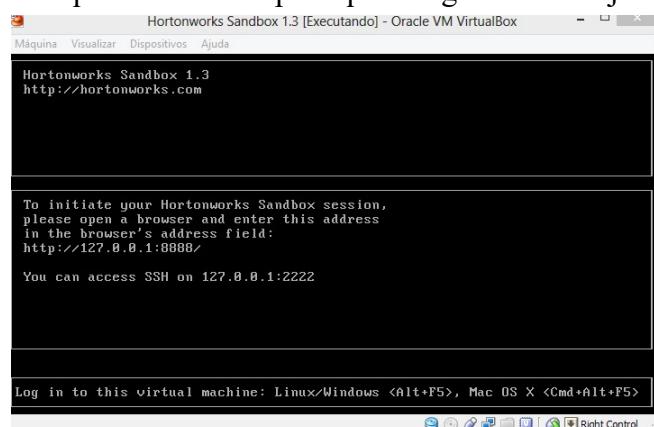
Professor: Cláudio Lúcio

Atividade: Utilizando o Spark-sql

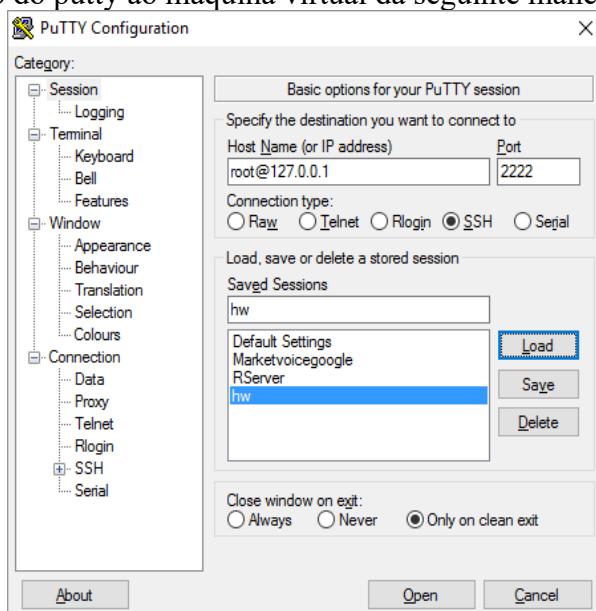
- 1) Para acesso ao Shell do Spark vamos usar a versão do Spark que vem na máquina virtual do HortonWorks:

Esta é a versão 2.3.2 do produto e pode ser obtida em:
<http://hortonworks.com/products/hortonworks-sandbox/>

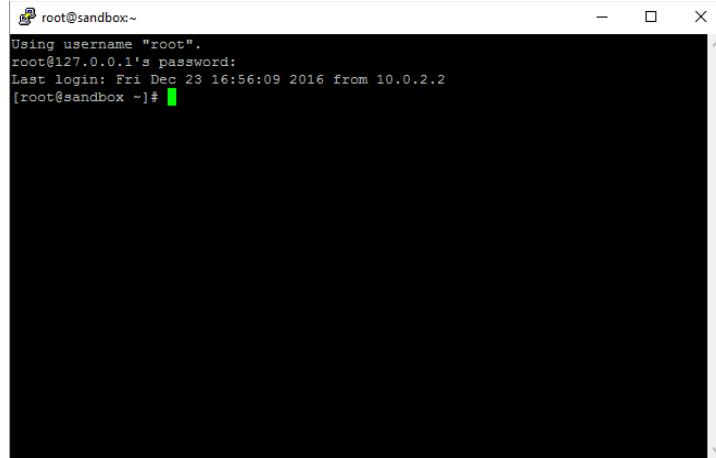
- 2) Inicialize a máquina virtual. Espere que a seguinte tela seja exibida:



- 3) A recomendação é usar um cliente para acesso SSH ao servidor do spark. Baixe a ferramenta putty.exe;
- 4) Configure o acesso do putty ao máquina virtual da seguinte maneira:



- 5) Clique em “Open” e então digite a senha do Root: hadoop



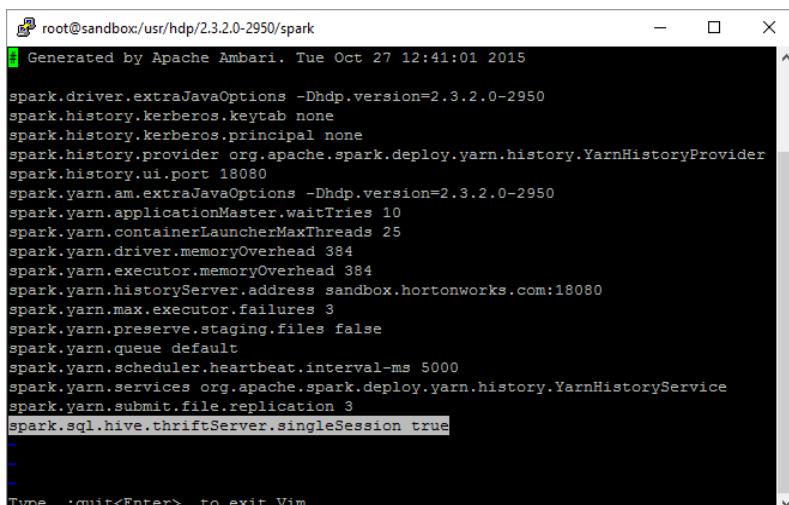
```
root@sandbox:~  
Using username "root".  
root@127.0.0.1's password:  
Last login: Fri Dec 23 16:56:09 2016 from 10.0.2.2  
[root@sandbox ~]#
```

- 6) Vamos fazer um alteração para compartilhar as tabelas criadas entre os usuários no spark

```
cd /usr/hdp/2.3.2.0-2950/spark  
vi conf/spark-defaults.conf  
Aperte a tecla INSERT
```

Insira a última linha conforme abaixo:

```
spark.sql.hive.thriftServer.singleSession true
```



```
root@sandbox:/usr/hdp/2.3.2.0-2950/spark  
Generated by Apache Ambari. Tue Oct 27 12:41:01 2015  
  
spark.driver.extraJavaOptions -Dhdp.version=2.3.2.0-2950  
spark.history.kerberos.keytab none  
spark.history.kerberos.principal none  
spark.history.provider org.apache.spark.deploy.yarn.history.YarnHistoryProvider  
spark.history.ui.port 18080  
spark.yarn.am.extraJavaOptions -Dhdp.version=2.3.2.0-2950  
spark.yarn.applicationMaster.waitTries 10  
spark.yarn.containerLauncherMaxThreads 25  
spark.yarn.driver.memoryOverhead 384  
spark.yarn.executor.memoryOverhead 384  
spark.yarn.historyServer.address sandbox.hortonworks.com:18080  
spark.yarn.max.executor.failures 3  
spark.yarn.preserve.staging.files false  
spark.yarn.queue default  
spark.yarn.scheduler.heartbeat.interval-ms 5000  
spark.yarn.services org.apache.spark.deploy.yarn.history.YarnHistoryService  
spark.yarn.submit.file.replication 3  
spark.sql.hive.thriftServer.singleSession true  
.  
.  
.  
Type :quit<Enter> to exit Vim
```

Aperte a tecla ESC

Aperte a tecla ":" e digite "wq" e Aperte a tecla ENTER

Agora vamos ajustar os acessos aos diretórios do HDFS:

```
su hdfs  
hadoop fs -chmod -R 777 /user/hive/  
hadoop fs -chmod -R 777 /apps/hive/
```

- 7) Vamos interagir com a interface spark-sql do Spark, para tal vamos acessar o bin/spark-sql:

- 8) Digite os seguintes comandos:

```
cd /usr/hdp/2.3.2.0-2950/spark  
bin/spark-sql --packages com.databricks:spark-csv_2.10:1.1.0
```

```

root@sandbox:/usr/hdp/2.3.2.0-2950/spark
:: resolving dependencies :: org.apache.spark#spark-submit-parent;1.0
  confs: [default]
  found com.databricks#spark-csv_2.10;1.1.0 in central
  found org.apache.commons#commons-csv;1.1 in central
  found com.univocity#univocity-parsers;1.5.1 in central
:: resolution report :: resolve 353ms :: artifacts dl 12ms
  :: modules in use:
  com.databricks#spark-csv_2.10;1.1.0 from central in [default]
  com.univocity#univocity-parsers;1.5.1 from central in [default]
  org.apache.commons#commons-csv;1.1 from central in [default]
  -----
  |           |         modules      ||   artifacts   |
  |       conf    | number| search|dwnlded|evicted|| number|dwnlded|
  -----|-----|-----|-----|-----||-----|-----|
  | default   |   3   |  0   |  0   |  0   ||  3   |  0   |
  -----
:: retrieving :: org.apache.spark#spark-submit-parent
  confs: [default]
  0 artifacts copied, 3 already retrieved (0kB/9ms)
SET spark.sql.hive.thriftServer.singleSession=true
SET spark.sql.hive.thriftServer.singleSession=true
SET spark.sql.hive.version=0.13.1
SET spark.sql.hive.version=0.13.1
spark-sql>

```

Caso ocorra um erro no download dos pacotes acima, vamos ter que executar os seguintes passos:

```

su root ou exit
cd /etc/spark/2.3.2.0-2950/0/
mkdir lib
cd lib

wget http://central.maven.org/maven2/com/databricks/spark-csv_2.10/1.5.0/spark-csv_2.10-1.5.0.jar

wget http://central.maven.org/maven2/org/apache/commons/commons-csv/1.2/commons-csv-1.2.jar

su hdfs
spark-sql --jars spark-csv_2.10-1.5.0.jar,commons-csv-1.2.jar

```

- 9) Veja que estamos usando também a conexão com o Hive. Desta forma conseguiremos acessar as tabelas que já estão no HIVE;
- 10) Experimente os comandos:

```

show tables;
SET spark.sql.hive.version=0.13.1
SET spark.sql.hive.version=0.13.1
spark-sql> show tables;
sample_07      false
sample_08      false
Time taken: 1.184 seconds, Fetched 2 row(s)
spark-sql>

```

Veja que o segundo parâmetro retornado indica se a tabela é temporária.

```

desc sample_07;
Time taken: 1.184 seconds, Fetched 2 row(s)
spark-sql> desc sample_07;
code      string  NULL
description  string  NULL
total_emp    int    NULL
salary      int    NULL
Time taken: 1.737 seconds, Fetched 4 row(s)
spark-sql>

```

```

select distinct total_emp,salary from sample_07;

```

```

120060 46520
5250 55490
499640 80460
20270 72150
103320 62500
100820 24000
1390260 23920
6620 78200
33950 42190
28890 145210
67700 73240
47210 97170
152870 87550
15780 37680
Time taken: 3.749 seconds, Fetched 823 row(s)
spark-sql> 
```

- 11) Observe que ainda não usamos o Spark SQL, usamos apenas o Hive;
- 12) Vamos trabalhar com o SparkSQL
- 13) Vamos recriar a estrutura do json utilizado anteriormente, mas agora com um *data frame*, para isto vamos usar o seguinte comando:

```

CREATE TEMPORARY TABLE pessoaval USING org.apache.spark.sql.json
OPTIONS (path '/user/hue/pessoaval.json');
spark-sql> CREATE TEMPORARY TABLE pessoaval USING org.apache.spark.sql.json OPT
IONS (path '/user/hue/pessoaval.json');
Time taken: 0.238 seconds
spark-sql> [root@sandbox spark]# 
```

- 14) Veja, agora, o comando para exibir as tabelas:
show tables;

```

IONS (path '/user/hue/pessoaval.json');
Time taken: 0.662 seconds
spark-sql> show tables;
pessoaval      true
sample_07      false
sample_08      false
Time taken: 0.087 seconds, Fetched 3 row(s)
spark-sql> 
```

- 15) Vamos consultar como o spark inferiu a estrutura do Json:
desc pessoaval;

```

sample_08      false
Time taken: 0.087 seconds, Fetched 3 row(s)
spark-sql> desc pessoaval;
dat      string
id      bigint
val      bigint
Time taken: 0.572 seconds, Fetched 3 row(s)
spark-sql> 
```

```
select * from pessoaval;
```

```

val    bigint
Time taken: 0.572 seconds, Fetched 3 row(s)
spark-sql> select * from pessoaval;
12/01/2006      1      45
04/06/2009      2      53
18/01/2012      4     345
12/01/2016      5     435
08/09/2015      8     2003
12/11/2000     12     100
12/01/2006     45     200
12/01/2006     13    99999
12/03/2006      1     405
04/09/2009      2     503
01/10/2012      4     35
12/12/2016      5     45
01/01/2015      8     23
02/01/2002     12     10
12/12/2006     45     20
12/01/2007     13    99999
Time taken: 0.23 seconds, Fetched 16 row(s)
spark-sql> 

```

16) Vamos agora usar fazer um upload de um arquivo CSV para o HDFS: pessoas.csv

The screenshot shows the Hue File Browser interface at the URL localhost:8000/filebrowser/view/user/hue. The browser has a green header bar with various icons. Below it is a toolbar with buttons for Rename, Move, Copy, Change Permissions, Download, Delete, New, and Upload. The main area shows a file tree under the path /user/hue. The files listed are ., .., jobs, oozie, pessoas.csv (highlighted in green), and pessoaval.json. A message at the bottom indicates 4 items per page, showing 1 to 4 of 4 items, page 1 of 1.

17) Acesse agora o UI SparkSQL e veja os jobs executados

The screenshot shows the Spark UI at the URL localhost:4040/jobs/. The UI has a top navigation bar with tabs for Jobs, Stages, Storage, Environment, and Executors. The main content area is titled "Spark Jobs". It displays the total uptime (18 min), scheduling mode (FIFO), and the number of completed jobs (7). A "Completed Jobs (7)" section lists the following details:

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
6	show tables processCmd at CliDriver.java:423	2017/01/04 12:50:24	44 ms	1/1	1/1
5	cache table pessoaval processCmd at CliDriver.java:423	2017/01/04 12:49:48	41 ms	1/1	1/1
4	cache table pessoaval processCmd at CliDriver.java:423	2017/01/04 12:49:48	0.5 s	2/2	3/3
3	show tables processCmd at CliDriver.java:423	2017/01/04 12:49:28	58 ms	1/1	1/1
2	show tables processCmd at CliDriver.java:423	2017/01/04 12:48:15	58 ms	1/1	1/1
1	CREATE TEMPORARY TABLE pessoaval USING org.apache.spark.sql.json OPTIONS (path ... processCmd at CliDriver.java:423	2017/01/04 12:48:13	76 ms	1/1	1/1
0	CREATE TEMPORARY TABLE pessoaval USING org.apache.spark.sql.json OPTIONS (path ... processCmd at CliDriver.java:423	2017/01/04 12:48:12	0.5 s	1/1	2/2

18) Veja a tabela que esta na aba 'Storage':

The screenshot shows the Spark UI interface. At the top, there are tabs for Jobs, Stages, Storage, Environment, and Executors. The Storage tab is selected. Below the tabs, there is a table with one row:

RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size in ExternalBlockStore	Size on Disk
in-memory table pessoasaval	Memory Deserialized 1x Replicated	2	100%	912.0 B	0.0 B	0.0 B

19) Vamos agora usar fazer um upload de um arquivo CSV para o HDFS: pessoas.csv

The screenshot shows the Hue File Browser interface. At the top, there is a toolbar with various icons for file operations like Rename, Move, Copy, Change Permissions, Download, Delete, New, and Upload. Below the toolbar, the current path is shown as /user/hue. The main area displays a file listing:

Type	Name	Size	User	Group	Permissions	Date
..			hue	hdfs	drwxrwxrwx	January 04, 2017 11:12 AM
..			hdfs	hdfs	drwxr-xr-x	October 27, 2015 06:22 AM
jobsub			hue	hdfs	drwxrwxrwx	October 27, 2015 05:55 AM
oozie			hue	hdfs	drwxrwxrwx	October 27, 2015 05:55 AM
pessoas.csv		135 bytes	hue	hdfs	-rw-r--r--	January 04, 2017 11:12 AM
pessoasaval.json		689 bytes	hue	hdfs	-rw-r--r--	January 04, 2017 03:54 AM

At the bottom of the browser, there is a message: "Show 45 items per page. Showing 1 to 4 of 4 items, page 1 of 1." and the URL localhost:8000/filebrowser/view/user/hue#.

20) Vamos então criar esta tabela neste fluxo de dados do spark-sql:

```
CREATE TEMPORARY TABLE pessoas USING com.databricks.spark.csv
OPTIONS (path "/user/hue/pessoas.csv", header "true");
```

21) Veja os campos da tabela:

```
desc pessoas;
Time taken: 0.252 seconds
spark-sql> desc pessoas;
+-----+-----+
| id   | string |
| nome | string |
| idade| string |
| gen  | string |
+-----+-----+
Time taken: 0.17 seconds, Fetched 4 row(s)
spark-sql>
```

22) Agora vamos criar uma consulta envolvendo as duas tabelas

```
select nome,idade, gen, sum(val) as val from pessoas inner join
pessoasaval on pessoas.id=pessoasaval.id group by nome,idade, gen;
>
> select nome,idade, gen, sum(val) as val from pessoas inner join pessoasaval on pessoas.id=pessoasaval.id group by nome,idade, gen;
Albert 28      M      380
Simone 18      T      2026
Teste   38      T      199998
Gloria  43      F      556
Laura   33      F      480
Marta   45      F      110
Jairo   82      M      220
Bob     45      M      450
Time taken: 4.079 seconds, Fetched 8 row(s)
spark-sql> [root@sandbox spark]#
```

23) Agora vamos criar uma consulta envolvendo as duas tabelas e salvando em CSV no HDFS:

```
CREATE TABLE totpessoasval1 USING com.databricks.spark.csv OPTIONS  
(path "/user/hive/exemplo") AS select nome,idade, gen, sum(val) as  
val from pessoas inner join pessoasval on pessoas.id=pessoasval.id  
group by nome,idade, gen;
```

Soluções para processamento paralelo e distribuído de dados

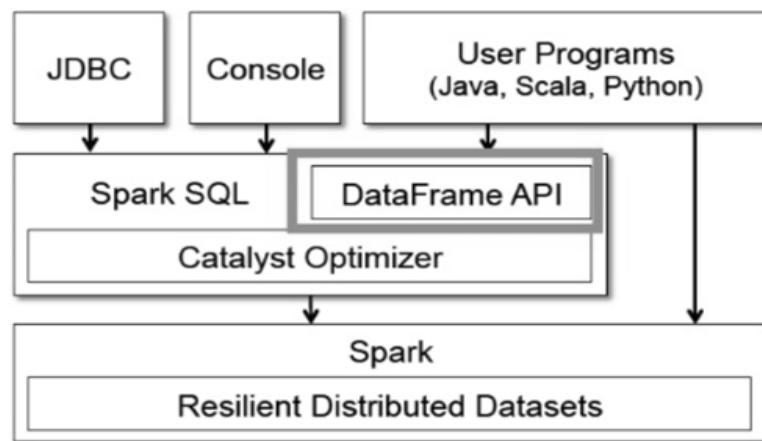
DataFrame - API



Ilustração de Oliver Munday

DataFrame - API

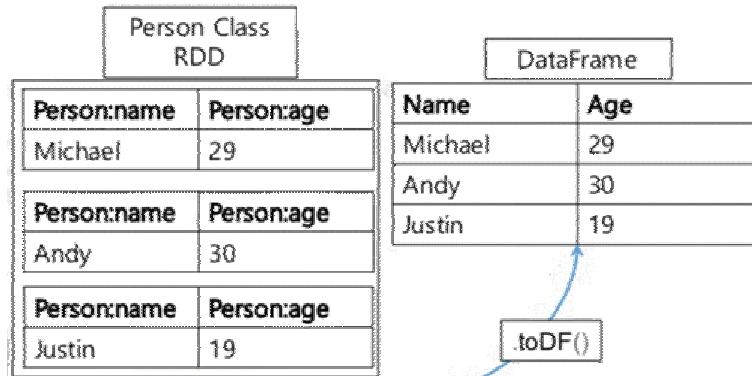
DataFrame API



DataFrame - API

DataFrame API

- Diferenças RDD e DataFrame



Spark – SQL

Alguma operações comuns

- Tipo de dados das colunas

`DF.printSchema()`

- Obtenção das primeiras linhas do data frame

Parâmetro com o número de linhas

`DF.head(5)`

- Número de linhas

`DF.count()`

Número de colunas

`len(DF.columns)`

Spark – SQL

Alguma operações comuns

- Estatísticas descritivas para colunas numéricas

```
DF.describe()
```

- Mostrando somente algumas colunas

```
DF.select('Col1', 'Col2').show(5)
```

- Quantidade de elementos distintos em uma coluna

```
DF.select('Col1').distinct().count()
```

- Geração de tabelas cruzadas(crosstab)

```
DF.crosstab('Col1', 'Col2').show()
```

Spark – SQL

Alguma operações comuns

- Eliminado linhas duplicadas em um data frame

```
DF.select('Col1', 'Col2').dropDuplicates()  
.show()
```

- Excluindo linhas com valores nulos

```
DF.dropna()
```

- Preenchendo valores nulos com outros valores

```
DF.fillna(-1).show(2)
```

- Filtra linhas

```
DF.filter(DF.Col1 > 15000)
```

Spark – SQL

Alguma operações comuns

- Agrupamentos por colunas

```
DF.groupby('Col1').agg({'Val':  
    'sum'}).show()  
DF.groupby('Col1').count().show()
```

- Amostragem

- Parâmetros :

Reposição = True ou False

Fração = 0.2 significa 20% das linhas do data frame

seed = para permitir que os resultados sejam
reproduzidos

```
DF.sample(False, 0.2, 1234)
```

Spark – SQL

Alguma operações comuns

- Aplicando operações map para uma coluna

```
DF.select('Col1').map(lambda x: (x,1))
```

- Ordenação da tabela base em uma coluna

```
DF.orderBy(DF.Col1.desc()).show(10)
```

- Adicionando colunas calculadas

- Parâmetros :

Nome da coluna = nova ou existente

Expressão

```
DF.withColumn('New_Col1', DF.Col1 *  
0.2).select('Col1','New_Col1').show(5)
```

Curso: Ciência de dados e Big Data

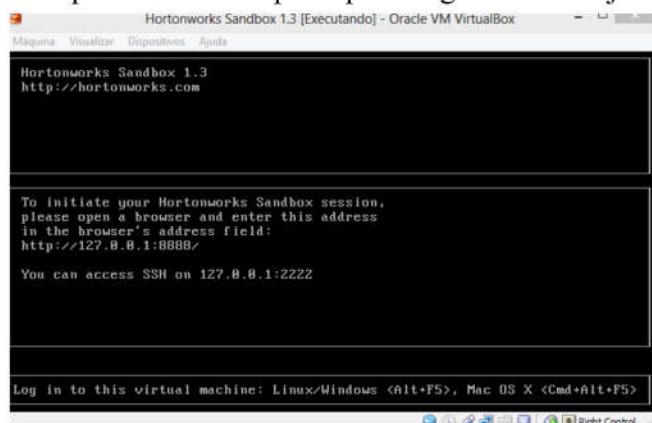
Professor: Cláudio Lúcio

Atividade Verificando processos e RDD's

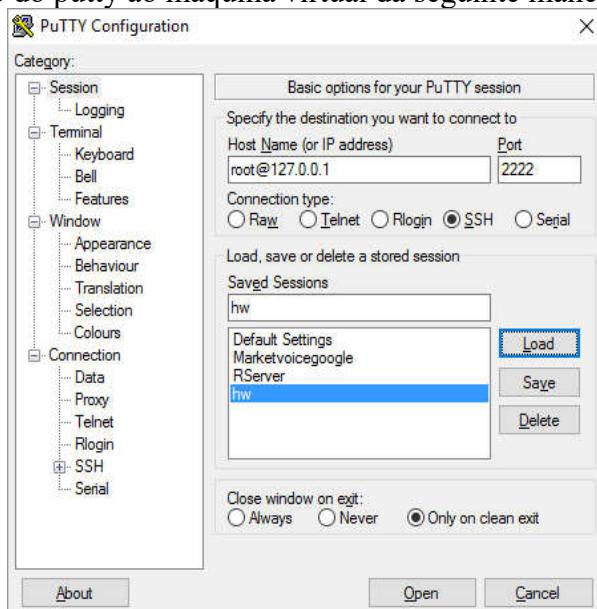
- 1) Para acesso ao Shell do Spark vamos usar a versão do Spark que vem na máquina virtual do HortonWorks:

Esta é a versão 2.3.2 do produto e pode ser obtida em:
<http://hortonworks.com/products/hortonworks-sandbox/>

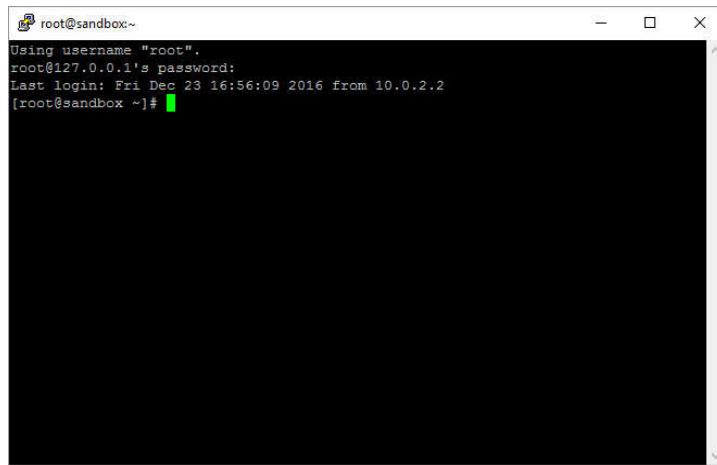
- 2) Inicialize a máquina virtual. Espere que a seguinte tela seja exibida:



- 3) A recomendação é usar um cliente para acesso SSH ao servidor do spark. Baixe a ferramenta putty.exe;
- 4) Configure o acesso do putty ao máquina virtual da seguinte maneira:

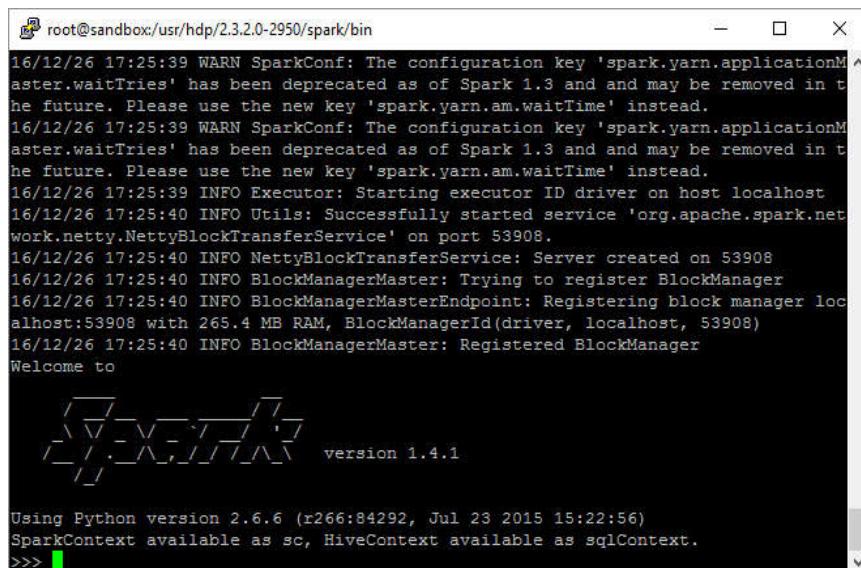


- 5) Clique em “Open” e então digite a senha do Root: hadoop



- 6) Primeiramente vamos interagir com a interface python para o Spark, para tal vamos acessar o bin/pyspark:
- 7) Digite os seguintes comandos:

```
cd /usr/hdp/2.3.2.0-2950/spark/bin
pyspark
```



- 8) Veja que estamos com a versão 1.4.1 do Spark;
- 9) Vamos preparar algumas estruturas e transformá-las em DataFrame:

```
pessoas = []
pessoas.append({'id':1,'nome':'Bob', 'idade':45, 'gen':'M'})
pessoas.append({'id':2,'nome':'Gloria', 'idade':43, 'gen':'F'})
pessoas.append({'id':4,'nome':'Albert', 'idade':28, 'gen':'M'})
pessoas.append({'id':5,'nome':'Laura', 'idade':33, 'gen':'F'})
pessoas.append({'id':8,'nome':'Simone', 'idade':18, 'gen':'T'})
pessoas.append({'id':12,'nome':'Marta', 'idade':45, 'gen':'F'})
pessoas.append({'id':45,'nome':'Jairo', 'idade':82, 'gen':'M'})
pessoas.append({'id':13,'nome':'Teste', 'idade':38, 'gen':'T'})
pessoasRdd=sc.parallelize(pessoas)

import json
pessoasValFile = sc.textFile("/user/hue/pessoasval.json")
pessoasValRdd = pessoasValFile.flatMap(lambda arq : (json.loads(arq)))
pessoasValRdd.collect()

pessoasDF = pessoasRdd.toDF()
pessoasValDF = pessoasValRdd.toDF()
```

- 10) Qual a estrutura dos dataframes?

```
pessoasDF.printSchema()
pessoasValDF.printSchema()
```

11) Verifique o número de linhas de cada dataframe?

```
pessoasDF.count()  
pessoasValDF.count()
```

12) Estatísticas descritivas sobre a tabela de valores das pessoas?

```
pessoasValDF.select('val').describe().show()
```

13) Quantidade de Id's distintos na tabela de valores?

```
pessoasValDF.select('id').distinct().count()
```

14) Geração da tabela cruzada de idade por gênero:

```
pessoasDF.crosstab('idade', 'gen').show()
```

15) Filtros diversos (gênero, idade e valores):

```
pessoasDF.filter(pessoasDF.gen == 'M').show()  
pessoasDF.filter(pessoasDF.idade > 30).show()  
pessoasValDF.filter((pessoasValDF.val > 10) & (pessoasValDF.val < 10000)).show()
```

16) Valor total somado para cada um dos anos (DataFrame Val):

```
pessoasValDFYear = pessoasValDF.withColumn('Ano', pessoasValDF.dat.substr(7, 11)).groupby('Ano').agg({'val': 'sum'})  
  
pessoasValDFYear.show()
```

Curso: Ciência de dados e Big Data

Professor: Cláudio Lúcio

Trabalho prático de DataFrame no Spark

Faça o download dos arquivos TRAB2.zip

Neste arquivo temos dados de promoções e vendas realizadas em um rede de lojas

Este é um trabalho guida para cada um dos itens abaixo solicitados você deve apresentar o código spark (SQL, PySpark, Scala) utilizado.

Colunas contidas no arquivo:

1. Faça a importação do arquivo TXT
2. Apresente o esquema da estrutura importada
3. Verifique a quantidade de linhas da estrutura importada

Resultado(s) : 550068

4. Faça uma análise estatística descritiva(básica(média, desvio padrão, quantidade, min, max) para a coluna Purchase (transforme a coluna purchase para o tipo de dados inteiro)

Resultado(s) :

	PurchaseI
count	550068
mean	9263.96
stddev	5023.06
min	12
max	23961

5. Verifique o número de produtos distintos contidos nesta base de dados

Resultado(s) : 3631

6. Verifique o número de pessoas que compraram os produtos por idade cruzados por gênero

Possíveis Resultado(s) :

Age_Gender	F	M
0-17	5083	10019
46-50	13199	32502
18-25	24628	75032
36-45	27170	82843
55+	5083	16421
51-55	9894	28607
26-35	50752	168835

7. Crie outro data frame em que o valor da compras(purchase) foi maior que 20.000 e depois exiba o número de linhas:

Resultado(s) : 9869

8. Crie outro data frame em que o valor da compras(purchase) foi maior que 20.000 e menor que 45000 somente para as mulheres. Depois exiba o número de linhas:

Resultado(s) : 12691

1.

9. Encontre a média de compras (purchase) de cada uma das faixas de idade:

Resultado(s) :

Age	AVG(PurchaseI)
51-55	9534.80
46-50	9208.62
0-17	8933.46
36-45	9331.35
26-35	9252.69
55+	9336.28
18-25	9169.66

10. Encontre a média de compras(purchase) de cada um por genero:

Resultado(s) :

Gender	AVG(PurchaseI)
F	8734.56
M	9437.52

11. Faça uma simulação gerando um novo dataframe com a média de compras por idade aumentada em 10%:

Resultado(s) :

Age	AVG(PurchaseI)	Pur_10_percent
51-55	9534.80	10488.28
46-50	9208.62	10129.48
0-17	8933.46	9826.81
36-45	9331.35	10264.48
26-35	9252.69	10177.95
55+	9336.28	10269.90
18-25	9169.66	10086.62