Soluções para processamento paralelo e distribuído de dados

Ilustração de Oliver Munday

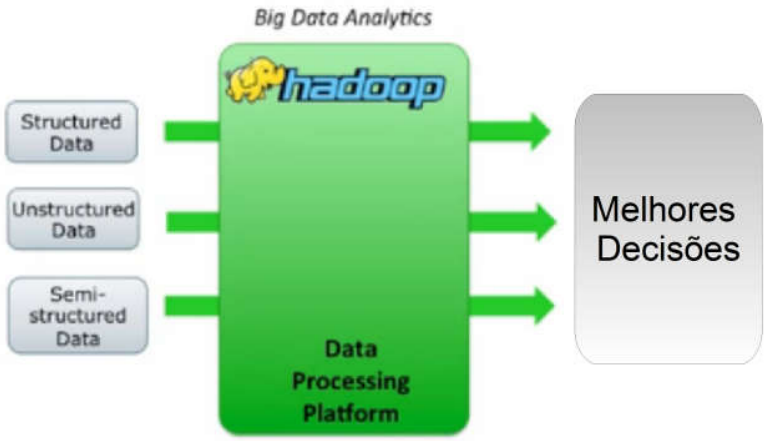
Soluções Processameto paralelo e distribuído de de dados

Hadoop

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

Hadoop - Introdução

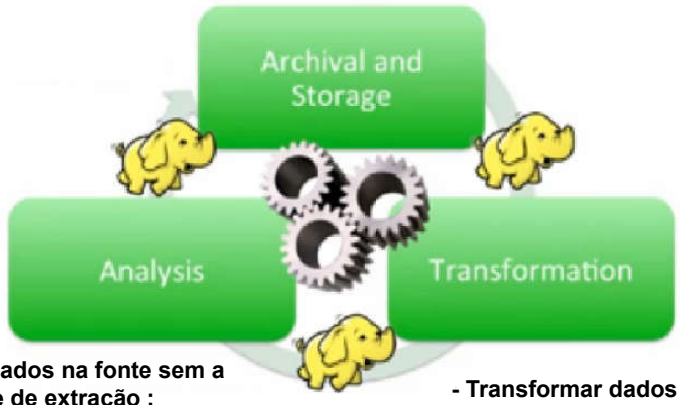
Visão Geral



Hadoop - Introdução

Visão Geral

- Reter dados históricos;
- Reter formatos intermediários;



- Explorar dados na fonte sem a necessidade de extração ;
- Execução de processos de transformação no fonte;

- Transformar dados não estruturados em estruturados em tempo de execução ;

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 - *Operational Data Store* - Acesso a dados interativo	- *Data Discovery* - 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

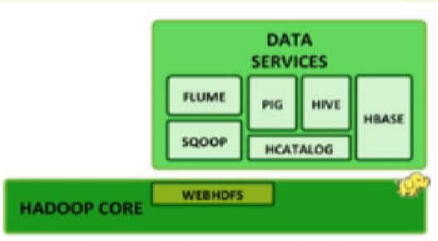
- Framework para execução de processamento paralelos em múltiplos nós de trabalho para posteriormente combinar os resultados;

YARN(Hadoop 2.0)

- Gerenciamento da execução das aplicações no cluster;

Hadoop - Introdução

Hadoop Componentes – Data Services

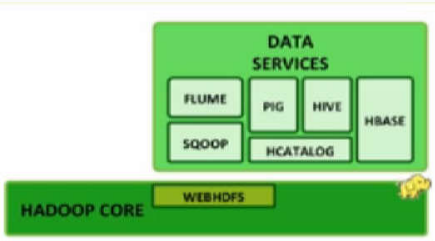


WEBHDFS

- 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

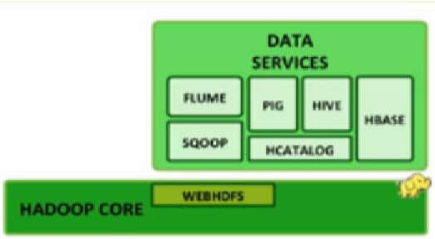


Flume

- 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

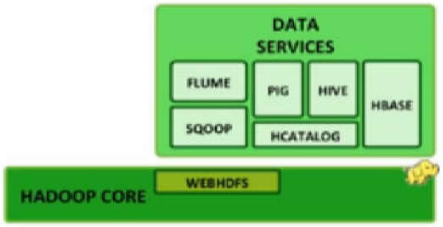


Sqoop

- 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

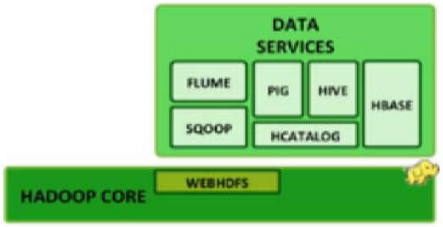


PIG

- 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

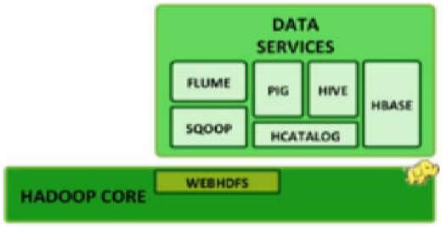


Hive

- Interface SQL para o Hadoop que possibilita sumarização 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

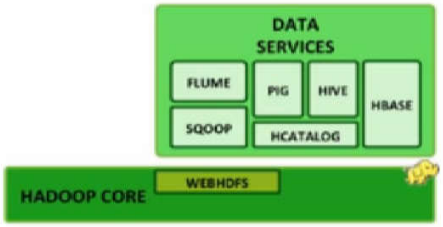


HCatalog

- 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

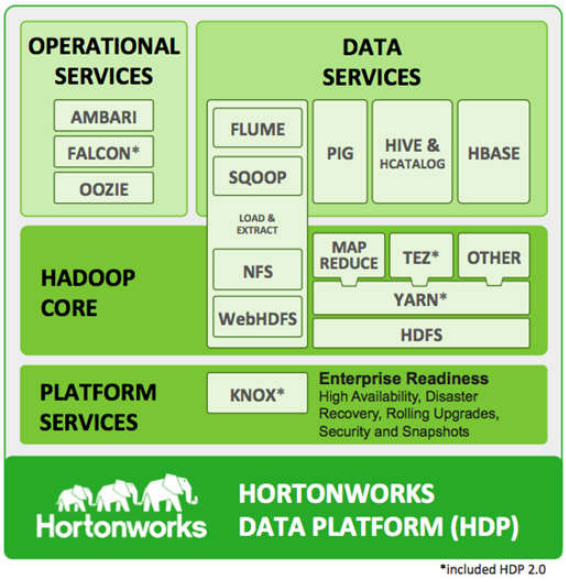


HBASE

- 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;

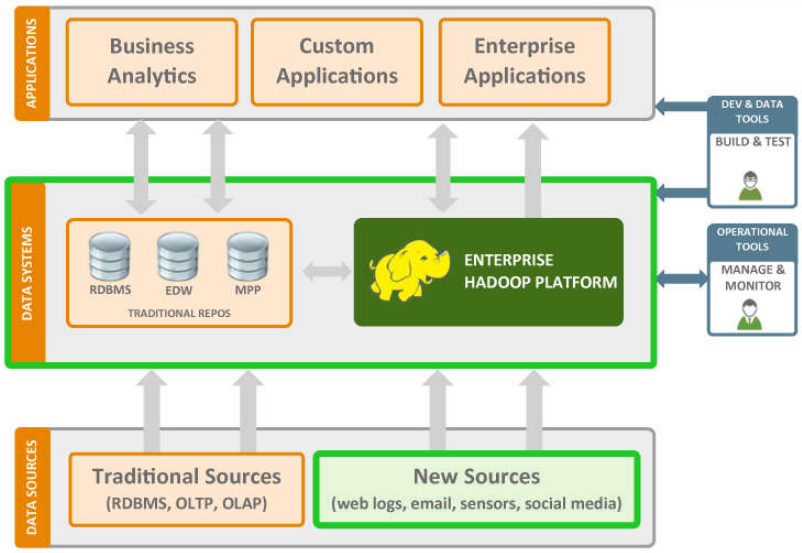
Hadoop - Introdução

Hadoop - Hortonworks



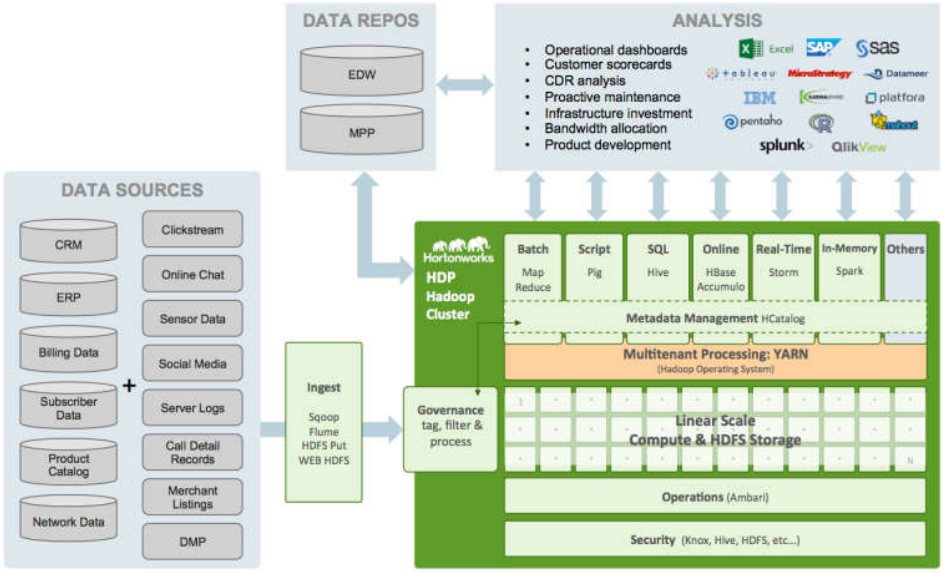
Hadoop - Introdução

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);

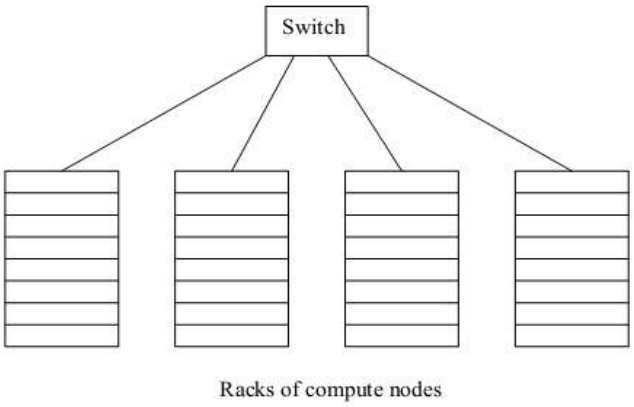
- A Web e o *Big Data* exigem processamento intensivo, mas em outra estrutura de hardware:

  - 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:

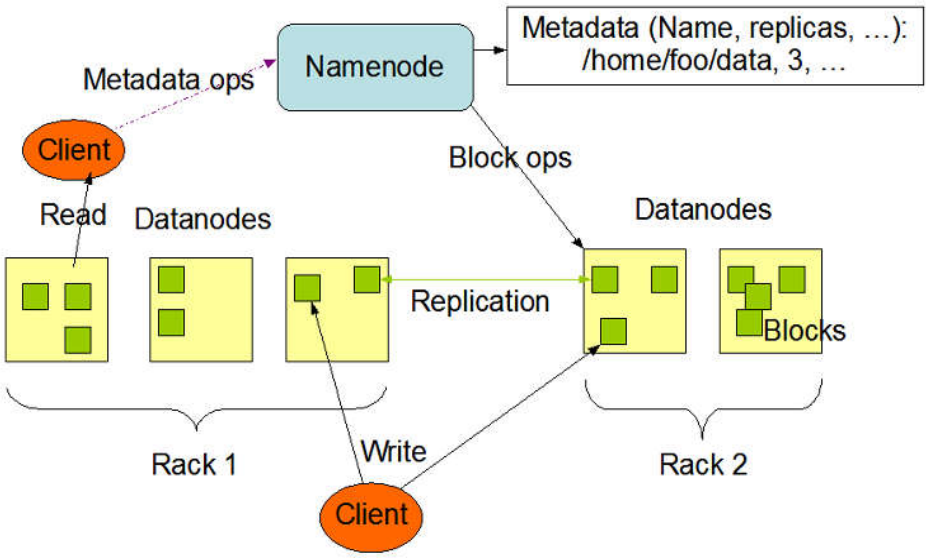
- Gerencia o sistema de arquivos(réplicas, blocos, nós e *racks*): abrir, fechar, renomear arquivos;
- Gerencia o acesso dos clientes ao arquivos;

- 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*:



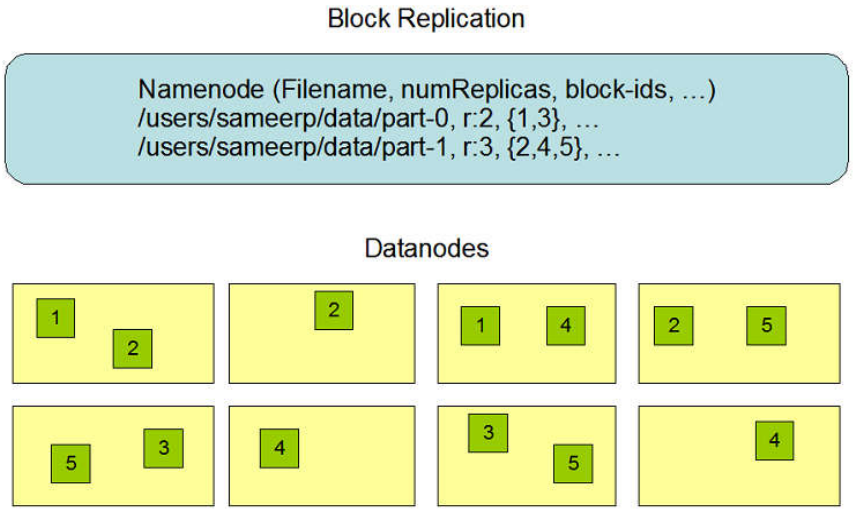
Hadoop – Componentes principais: HDFS

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(fora 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:



Hadoop – Componentes principais: HDFS

Características DFS:

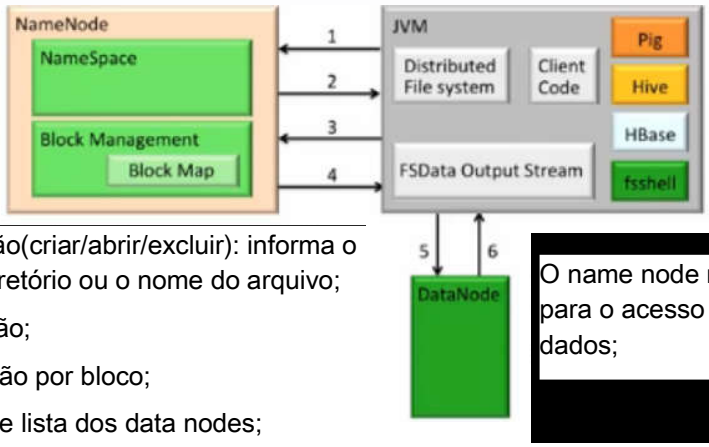
- O '*name node'* é replicado para outra estrutura no *DFS* e também possui *log* de alterações;

- 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



1-Requisição(criar/abrir/excluir): informa o nome do diretório ou o nome do arquivo;

2- Aprovação;

3- Requisição por bloco;

4- Block Id e lista dos data nodes;

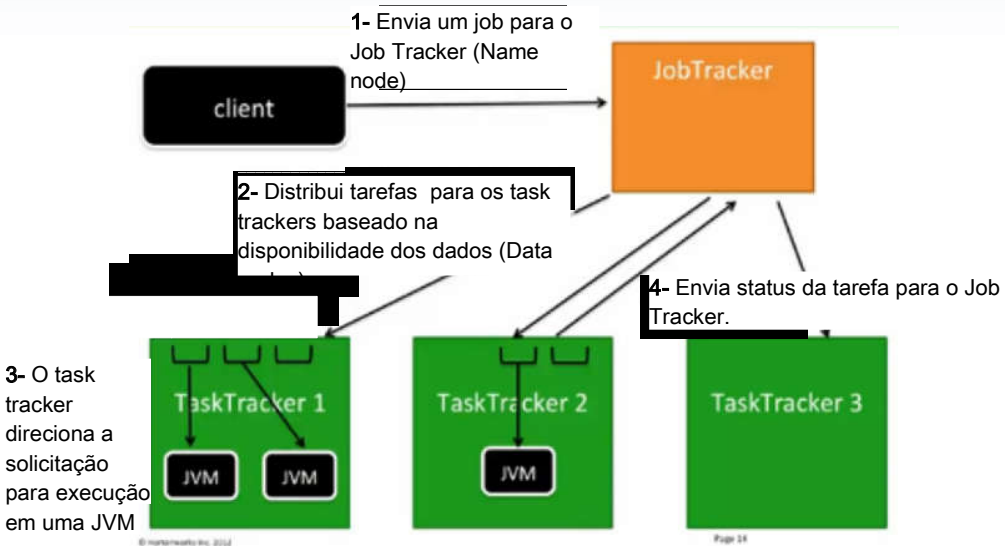
5- Operação no data node(leitura, escrita ou exclusão);

6- Retorno

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

Hadoop – Componentes principais: HDFS

Hadoop - Organização de backend Hadoop



1- Envia um job para o Job Tracker (Name node)

2- Distribui tarefas para os task trackers baseado na disponibilidade dos dados (Data

3- O task tracker direciona a solicitação para execução em uma JVM

4- Envia status da tarefa para o Job Tracker.

Hadoop – Componentes principais: HDFS

Implementações DFS:

- '*Google File System*' (*GFS*), implementação original e primeira a fica '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

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 principio '*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 M*ap* existe um ou mais blocos que serão processados; As tarefas *Map* vão transformar o dado em um estrutura chave valor ou tuplas;

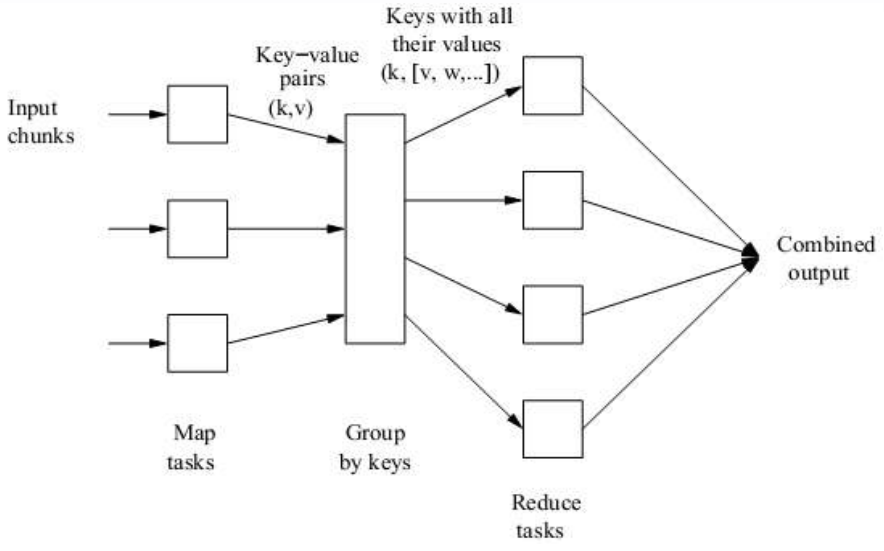
- As estruturas chave valor são coletadas pelo controlador *master* e ordenadas pelas suas chaves;

- 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: $w_1, w_2, .., w_n$;
- A seguinte estrutura chave valor será criada:

<u>Sugestões para melhorar a função *Map* ???</u>

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:

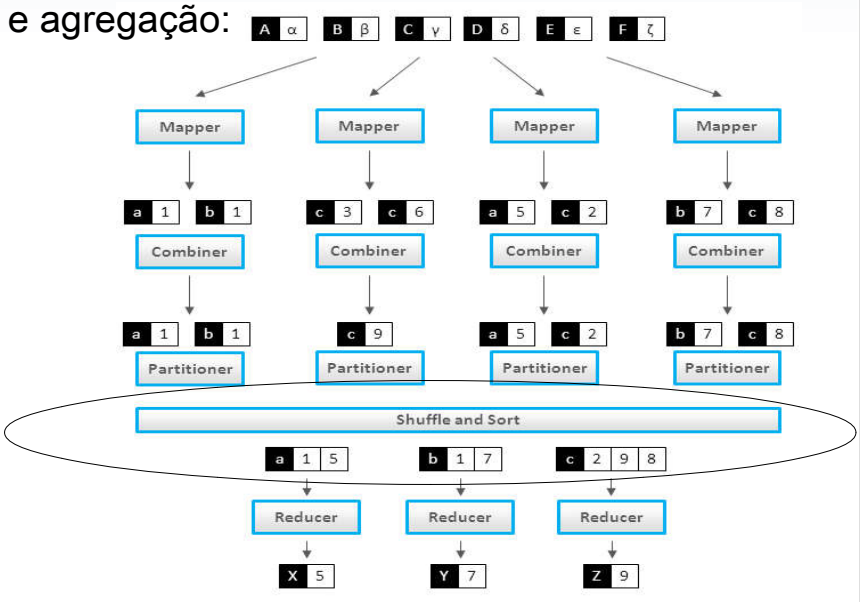
- O "*Name node*" controla este processo:

  - 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 destinas 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 $(k,[v_1,v_2,..,v_n])$ oriundos de várias tarefas *map* na forma $(k,v_1)(k,v_2)..(k,v_n)$;

- A tarefa reduce deve combinar os valores de alguma forma;

Hadoop – Componentes principais: Map Reduce

Detalhes de funcionamento:

Tarefas *Reduce*:

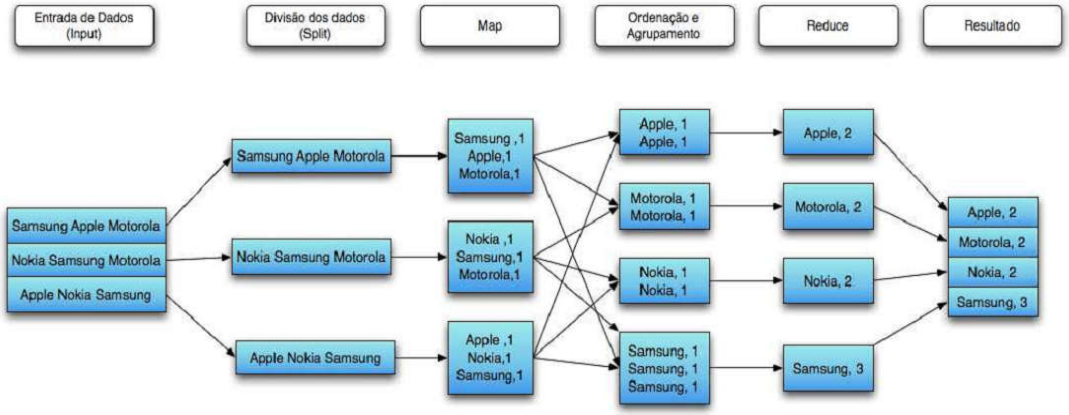
Exemplo clássico - Contar o número de palavras em um coleção de documentos:

- 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:

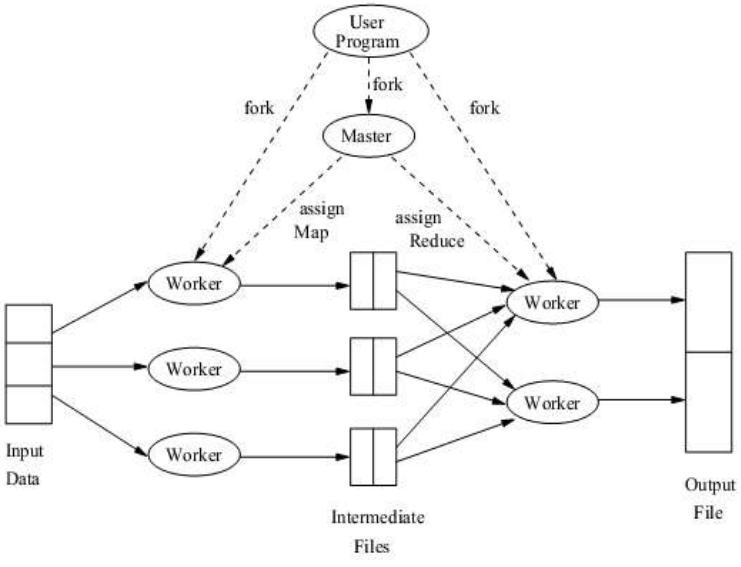
Exemplo clássico:



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:
       continue # Volta para o começo do laço
    print "teste"
```

Funções:

```
def funcao( p1, p2="Valor Padrao" ):
    print "p1: '%s'  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, valor ):
        self.atributo = valor
        self.__metodo_privado()
    def __metodo_privado( self ):
        print "chamando metodo privado"
class B:
    atributo = 2
    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()
c = C()
d = D()
print a.atributo   # imprime 1
print b.atributo   # imprime 2
print c.atributo   # imprime 1: herança múlt. (A,B) A sobrepõe-se a B
print d.atributo   # imprime 2: herança múlt. (B,A) B sobrepõe-se a A
```

Módulos e Espaço de Nomes:

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

# importa símbolos 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':
```

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),...])

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