**QUESTIONÁRIO SOBRE HADOOP – LEITURA DO MATERIAL FORNECIDO**

**ALUNO/GRUPO ; _____**

**_____**

**LEIA O MATERIAL ANTES DE FAZER O EXEMPLO DIDÁTICO.**
**AS QUESTÕES ABAIXO PODEM SER RESPONDIDAS, USANDO, PREFERENCIALMENTE, SUA PRÓPRIAS PALAVRAS, USANDO ESTE PRÓPRIO ARQUIVO E O SEU EDITOR DE TEXTO.**

1. Qual a relação entre Computação Paralela, Big Data e Hadoop ?

   As aplicações **Big Data** fazem da **computação paralela**, o meio para criar soluções capazes de analisar grandes bases de dados, processar seus pesados cálculos, identificar comportamentos e disponibilizar serviços especializados em seus domínios. Apache **Hadoop** é um framework, open source, para o **processamento paralelo e distribuído de grandes quantidades de dados em clusters**.

2. O Apache Hadoop é considerado atualmente uma das melhores ferramentas para processamento de alta demanda de dados. Entre os benefícios de utilizá-lo pode-se destacar os seguintes aspectos importantes: código aberto, economia, robustez, escalabilidade e simplicidade. Explique ligeiramente cada um desses.

   Código aberto: Todo projeto de software livre de sucesso tem por trás uma comunidade ativa. No projeto Apache Hadoop, essa comunidade é composta por diversas empresas e programadores independentes partilhando seus conhecimentos no desenvolvimento de melhorias, funcionalidades e documentação.

   Economia: (a) Com um esforço relativamente pequeno é possível implantar, desenvolver aplicações e executar Hadoop sem gastar com aquisição de licenças e contratação de pessoal especializado. **(b)** A possibilidade de realizar o processamento da sua massa de dados utilizando máquinas e rede convencionais. **(c)** Existe uma outra alternativa econômica dado pela existência de serviços em nuvem, como a *Amazon Elastic MapReduce* (*EMR*), que permite a execução de aplicações Hadoop sem a necessidade de implantar seu próprio cluster de máquinas, alugando um parque virtual ou simplesmente pagando pelo tempo de processamento utilizado.

Robustez: Um outro diferencial do Hadoop é que como ele foi projetado para ser executado em hardware comum, ele já considera a possibilidade de falhas frequentes nesses equipamentos e oferece estratégias de recuperação automática para essas situações. Assim, sua implementação disponibiliza mecanismos como replicação de dados, armazenamento de metadados e informações de processamento, que dão uma maior garantia para que a aplicação continue em execução mesmo na ocorrência de falhas em algum recurso.

Escalabilidade: Enquanto as demais aplicações similares apresentam dificuldade em aumentar a quantidade de máquinas utilizadas no processamento e/ou aumentar o conjunto de dados, precisando em alguns casos até reescrever todo o código-fonte da aplicação, Hadoop permite obter escalabilidade de forma relativamente simples. Mudanças no ambiente implicam em pequenas modificações em um arquivo de configuração. Dessa forma, o trabalho para preparar um ambiente contendo mil máquinas não será muito maior do que se este fosse de dez máquinas. O aumento no volume de dados só fica limitado aos recursos, espaço em disco e capacidade de processamento, disponíveis nos equipamentos do aglomerado, sem a necessidade de alteração da codificação.

Simplicidade: Hadoop retira do desenvolvedor a responsabilidade de gerenciar questões relativas à computação paralela, tais como **tolerância a falhas**, **escalonamento** e **balanceamento de carga**, ficando estas a cargo do próprio *framework*. O Hadoop descreve suas operações apenas por meio das funções de mapeamento (Map) e de junção (Reduce). Dessa forma, o foco pode ser mantido somente na abstração do problema, para que este possa ser processado no *modelo de programação MapReduce*.

## 3. Mesmo sendo uma excelente ferramenta para o processamento de dados em larga escala, existem situações para as quais o Hadoop não é a solução mais adequada, devido a alguma particularidade. Explique.

(a) Único nó mestre: a arquitetura do Hadoop tradicionalmente utiliza várias máquinas para o processamento e armazenamento dos dados, porém, contém apenas um nó mestre. Esse modelo pode se tornar restritivo a medida que essa centralidade causar empecilhos na escalabilidade e ainda criar um ponto crítico, suscetível a falha, uma vez que o nó mestre é vital para o funcionamento da aplicação.

(b) Dificuldade no gerenciamento do aglomerado: um dos maiores problemas sofridos pelos desenvolvedores de computação paralela ocorre no momento de depurar a execução da aplicação e de analisar os logs que estão distribuídos. Infelizmente, com o Hadoop, também ocorrem essas mesmas dificuldades, entretanto, existem subprojetos do ecossistema Hadoop que procuram minimizar esses problemas.

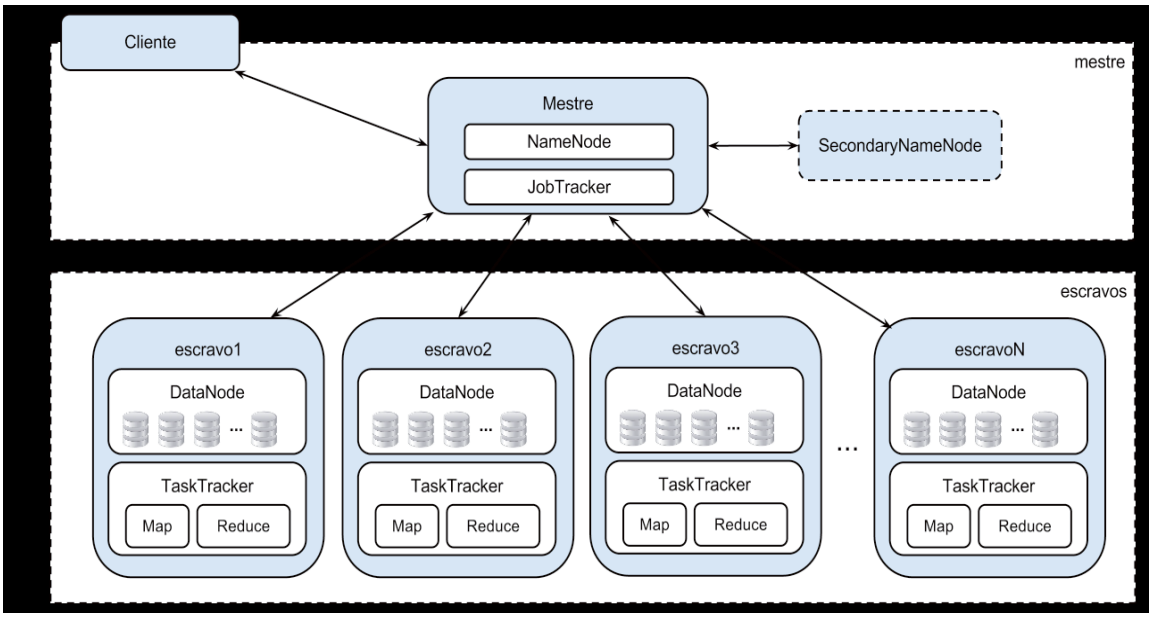
(c) Problemas não paralelizáveis ou com grande dependência entre os dados: esta é uma das maiores dificuldades para toda e qualquer aplicação que requer alto desempenho. Como o princípio básico dos programas paralelos e/ou distribuídos é dividir as tarefas em subtarefas menores, para serem processadas simultaneamente por vários recursos computacionais, a impossibilidade ou a dificuldade na extração da porção paralelizável da aplicação torna-se um grande fator de insucesso. Assim, problemas que não podem ser divididos em problemas menores e problemas com alta dependência entre os seus dados não são bons candidatos para o Hadoop.

Como exemplo, podemos imaginar uma tarefa T sendo dividida em cinco subtarefas t1, t2, t3, t4 e t5. Suponha cada tarefa ti+1 dependente do resultado da tarefa ti. Dessa forma, a dependência entre as subtarefas limita ou elimina a possibilidade de sua execução em paralelo. O tempo total de execução de uma aplicação nesse contexto será equivalente (ou maior) do que se fosse executada sequencialmente; equivalente (ou maior) do que se fosse executada sequencialmente.

(d) Processamento de arquivos pequenos: trabalho pequeno definitivamente não é tarefa para ser executada com base em um arcabouço com foco em larga escala. Os custos adicionais (overhead) impostos pela divisão e junção das tarefas, rotinas e estruturas gerenciais do ambiente e comunicação entre os nós de processamento, certamente irão inviabilizar sua execução.

(e) Problemas com muito processamento em poucos dados: problemas com regras de negócio complexas demais ou com um fluxo de execução extenso em um conjunto de dados não muito grande, também não serão bons candidatos a se tornar aplicações Hadoop.

## 4. O que significa a figura e cada processo mostrada na figura seguinte:




A figura refere-se aos **Componentes do Hadoop**. Num cluster constituído por uma máquina Mestre e várias máquinas escravas, esse ambiente pode ser visualizado através de uma máquina-cliente, dotada de uma interface Web.

Uma execução típica de uma aplicação Hadoop em um cluster utiliza cinco processos diferentes: **NameNode**, **DataNode**, **SecondaryNameNode**, **JobTracker** e **TaskTracker**.

Os componentes **NameNode**, **JobTracker** e **SecondaryNameNode** são únicos para toda a aplicação, enquanto que o **DataNode** e **JobTracker** são instanciados para cada máquina.

**NameNode:** tem como responsabilidade gerenciar os arquivos armazenados no HDFS. Suas funções incluem mapear a localização, realizar a divisão dos arquivos em blocos, encaminhar os blocos aos nós escravos, obter os metadados dos arquivos e controlar a localização de suas réplicas. Como o **NameNode** é constantemente acessado, por questões de desempenho, ele mantém todas as suas informações em memória. Ele integra o sistema HDFS e fica localizado no nó mestre da aplicação, juntamente com o **JobTracker**.

**DataNode:** enquanto o NameNode gerencia os blocos de arquivos, são os **DataNode** que efetivamente realizam o armazenamento dos dados. Como o HDFS é um sistema de arquivos distribuído, é comum a existência de diversas instâncias do **DataNode** em uma aplicação Hadoop, para que eles possam distribuir os blocos de arquivos em diversas máquinas. Um **DataNode** poderá armazenar múltiplos blocos, inclusive de diferentes arquivos. Além de armazenar, eles precisam se reportar constantemente ao **NameNode**, informando quais blocos estão guardando bem como todas as alterações realizadas localmente nesses blocos.

**JobTracker:** assim como o **NameNode**, o **JobTracker** também possui uma função de gerenciamento, porém, nesse caso, o controle é realizado sobre o plano de execução das tarefas a serem processadas pelo ***MapReduce***. Sua função então é designar diferentes nós para processar as tarefas de uma aplicação e monitorá-las enquanto estiverem em execução. Um dos objetivos do monitoramento é, em caso de falha, identificar e reiniciar uma tarefa no mesmo nó ou, em caso de necessidade, em um nó diferente.

**TaskTracker:** processo responsável pela execução de tarefas ***MapReduce***. Assim como os **DataNodes**, uma aplicação Hadoop é composta por diversas instâncias de **TaskTracker**, cada uma em um nó escravo. Um **TaskTracker** executa uma tarefa ***Map*** ou uma tarefa ***Reduce*** designada a ele. Como os **TaskTracker** rodam sobre máquinas virtuais, é possível criar várias máquinas virtuais em uma mesma máquina física, de forma a explorar melhor os recursos computacionais.

**SecondaryNameNode:** utilizado para auxiliar o **NameNode** a manter seu serviço, e ser uma alternativa de recuperação no caso de uma falha do **NameNode**. Sua única função é realizar pontos de checagem (*checkpointing*) do **NameNode** em intervalos pré-definidos, de modo a garantir a sua recuperação e atenuar o seu tempo de reinicialização.

## 5. O que caracteriza as formas de execução do Hadoop ? Explique brevemente cada dessas.

Chuck Lam (Lam, 2010) cita três modos possíveis de execução: **modo local** (standalone mode), **modo pseudo-distribuído** (pseudo-distributed mode) e **modo completamente distribuído** (fully distributed mode). Para alternar entre essas configurações é necessária a edição de três arquivos: *core-site.xml*, *hdfs-site.xml* e *mapred-site.xml*.

**Modo Local** - Hadoop é por padrão configurado para ser executado no modo local. Dessa maneira, se essa for a sua opção escolhida, os parâmetros nos arquivos de configuração não precisam ser alterados. Esse modo é o mais recomendado para a fase de desenvolvimento, onde normalmente ocorre a maior incidência de erros, sendo necessária a realização de vários testes da execução. Nessa configuração, todo o processamento da aplicação é executado apenas na máquina local.

Por isso, não é necessário que os arquivos sejam carregados no HDFS, visto que, os dados não são distribuídos. Dessa forma simplificada, fica mais fácil para o usuário realizar a depuração de seu código, aumentando sua produtividade.

**Modo pseudo-distribuído** - Uma segunda alternativa para executar uma aplicação Hadoop é o modo pseudodistribuído. Nesse modo são aplicadas todas as configurações, semelhantes às necessárias para execução em um aglomerado, entretanto, toda a aplicação é processada em modo local, por isso o termo pseudo-distribuído ou também chamado "cluster" de uma máquina só. Embora não seja executado realmente em paralelo, esse modo permite a sua simulação, pois utiliza todos os processos de uma execução paralela efetiva: **NameNode**, **DataNode**, **JobTracker, TaskTracker** e **SecondaryNameNode**.

**Modo completamente distribuído** - Por fim, o terceiro e último modo de execução é utilizado para o processamento distribuído da aplicação Hadoop em um cluster de computadores real. Nessa opção, como no modo pseudo-distribuído, também é necessário editar os três arquivos de configuração, definindo parâmetros específicos e a localização do **SecondaryNameNode** e dos nós escravos. Todavia, como nesse modo temos diversos computadores, devemos indicar quais máquinas irão efetivamente executar cada componente.

## 6. Descreva, de forma breve, o HDFS do Hadoop.

Sistema de arquivos distribuídos A manipulação dos arquivos em um computador é realizada pelo sistema operacional instalado na máquina por meio de um sistema gerenciador de arquivos.

Esse sistema possui um conjunto de funcionalidades, tais como: *armazenamento*, *organização*, *nomeação*, *recuperação*, *compartilhamento*, *proteção* e *permissão de acesso* aos arquivos.

Todo o gerenciamento deve *ocorrer da forma mais transparente possível aos usuários* (transparência significa, "parece não existir, mas existe") ou seja, o sistema de arquivos deve omitir toda a complexidade arquitetural da sua estrutura não exigindo do seu usuário muito conhecimento para operá-lo. Um sistema de arquivos distribuído possui as mesmas características que um sistema de arquivos convencional, entretanto, deve permitir o armazenamento e o compartilhamento desses arquivos em diversos hardwares diferentes, que normalmente estão interconectados por meio de uma rede.

O sistema de arquivos distribuído também deve prover os mesmos requisitos de transparência a seus usuários, inclusive permitindo uma manipulação remota dos arquivos como se eles estivessem localmente em suas máquinas.

Além disso, deve oferecer um desempenho similar ao de um sistema tradicional e ainda prover escalabilidade.

Existem algumas estruturas de controle exclusivas, que devem ser implementadas em um sistema de arquivos distribuído. Entre essas, podemos citar algumas imprescindíveis para o seu bom funcionamento:

● **Segurança:** no armazenamento e no tráfego das informações, garantindo que o arquivo não seja danificado no momento de sua transferência, no acesso às informações, criando mecanismos de controle de privacidade e gerenciamento de permissões de acesso;

● **Tolerância a falhas:** deve possuir mecanismos que não interrompam o sistema em casos de falhas em algum nó escravo;

● **Integridade:** o sistema deverá controlar as modificações realizadas no arquivo, como por exemplo, alterações de escrita e remoção, permitindo que esse seja modificado somente se o usuário tiver permissão para tal;

● **Consistência:** todos os usuários devem ter a mesma visão do arquivo;

● **Desempenho:** embora possua mais controles a serem tratados que o sistema de arquivos convencional, o desempenho do sistema de arquivos distribuído deve ser alto, uma vez que provavelmente deverá ser acessado por uma maior gama de usuários.

Atualmente há diversas implementações de sistemas de arquivos distribuídos, algumas comerciais e outras de software livre, tais como: **GNU Cluster File System (GlusterFS)** da empresa Red Hat e o e **Google File System (GFS)**.

## 7. O que é o HDFS ?

O ***Hadoop Distributed File System*** *(HDFS)* é um sistema de arquivos distribuído integrado ao arcabouço Hadoop. Esse sistema teve forte inspiração no GFS da Google, entretanto, uma das diferenças deve-se ao fato de que o HDFS é um arcabouço de código aberto, e foi implementado na linguagem Java.

O HDFS também oferece suporte ao armazenamento e ao processamento de grandes volumes de dados em um agrupamento de computadores heterogêneos de baixo custo.

Essa quantidade de dados pode chegar à ordem de *petabytes*, quantidade que não seria possível armazenar em um sistema de arquivos tradicional.

## 8. Por que o Hadoop divide seus arquivos em blocos ? E por que replica esses blocos ?

**Divisão em blocos** - Existem arquivos que devido ao seu grande tamanho, não podem ser armazenados em um único disco rígido. Esta situação fica mais evidente quando nos referimos aos arquivos de aplicações “Big Data”. Uma alternativa nesse caso é criar uma forma de dividir esses grandes arquivos e distribuí-los em um cluster de máquinas.

Embora funcional, essa tarefa seria muito difícil de ser implementada sem a utilização de um recurso específico, como o HDFS. Com esse arcabouço, toda a questão estrutural relativa a

distribuição dos arquivos é feita de forma implícita, devendo apenas que o desenvolvedor aponte corretamente os parâmetros de configuração.

O HDFS adota a estratégia de que antes de armazenar os arquivos, estes sejam submetidos a um procedimento de divisão em uma sequência de blocos de tamanho fixo. O tamanho padrão definido no arcabouço é 64 Mb, podendo ser alterado se necessário. Esse tamanho é muito superior ao dos sistemas de arquivos tradicionais, que usa blocos de 512 bytes. Dessa forma, somente depois de dividido é que esses arquivos são distribuídos para os diversos nós escravos.

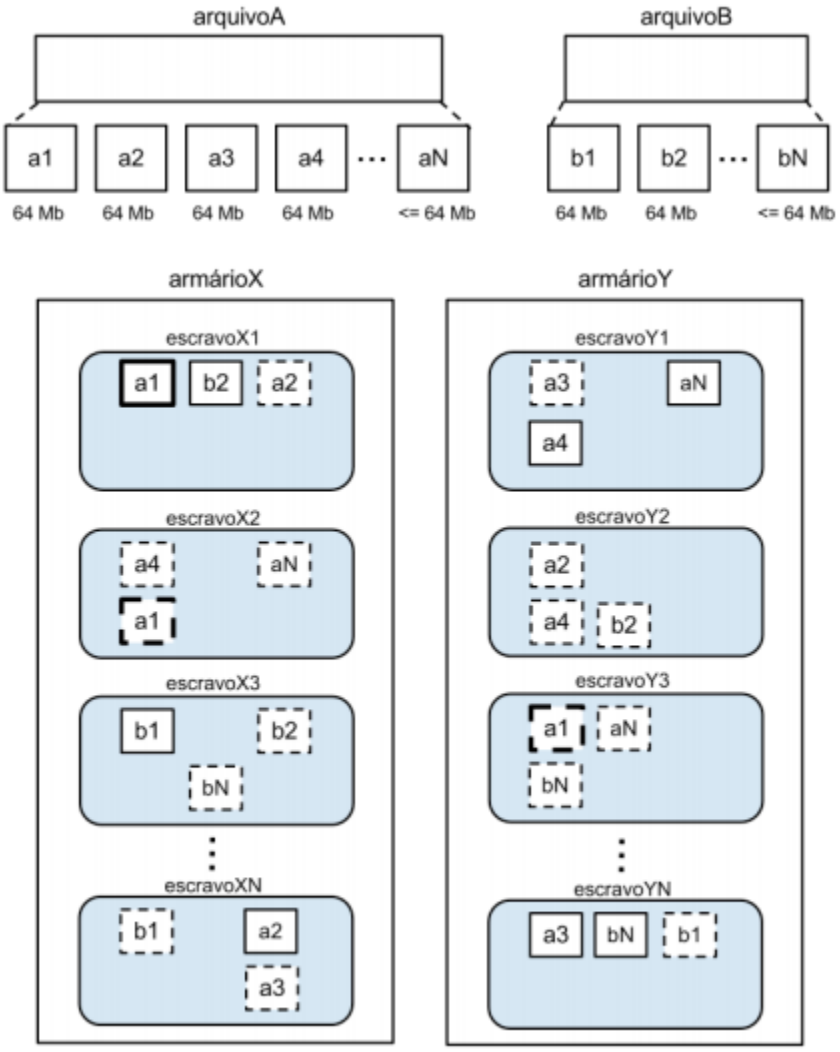



**Figura 3.4. Replicação de blocos de dados**

**Comandos HDFS** - Para iniciar os trabalhos em um cluster Hadoop é necessário formatar o HDFS no intuito de prepará-lo para receber os dados de sua aplicação. Ver e estudar os comandos para o HDFS, quando rodando sua aplicação. Veja no primeiro link já mostrado em aula.

## 9. Como se resume a arquitetura do HDFS ?

**Arquitetura -** O HDFS é implementado sobre a arquitetura mestre/escravo, possuindo no lado mestre uma instância do **NameNode** e em cada escravo uma instância do **DataNode.**

Em um cluster Hadoop podemos ter centenas ou milhares de máquinas escravas, e dessa forma, elas precisam estar dispostas em diversos armários (racks). Nesse texto, a palavra armário é utilizada como um conjunto de máquinas alocadas em um mesmo espaço físico e interligadas por um comutador (switch). Por questões estratégicas, que serão discutidas na próxima seção, o HDFS organiza a armazenagem dos blocos dos arquivos, e suas réplicas, em diferentes máquinas e armários. Assim, mesmo ocorrendo uma falha em um armário inteiro, o dado pode ser recuperado e a aplicação não precisaria ser interrompida.

O **NameNode** é o componente central do HDFS, assim, é recomendável ser implantado em um nó exclusivo, e preferencialmente o nó com melhor desempenho do aglomerado. Ainda por questões de desempenho, o **NameNode** mantém todas suas informações em memória.

Em uma aplicação Hadoop, cada nó escravo contém um **DataNode**, que trabalha em conjunto com um **TaskTracker**, sendo o primeiro para armazenamento e o segundo para processamento dos dados.

## 10. Como funciona o Hadoop MapReduce ?

O **paradigma de programação *MapReduce*** implementado pelo Hadoop se inspira em duas funções simples (***Map*** e ***Reduce***) presentes em diversas linguagens de programação funcionais.

A função ***Map*** recebe uma lista como entrada, e aplicando uma função dada, gera uma nova lista como saída.

A função ***Reduce***, similarmente à função Map, vai receber como entrada uma lista e, em geral, aplicará uma função para que a entrada seja reduzida a um único valor na saída.