Apache Hadoop

Bruno Antunes da Silva

Universidade Federal de São Carlos - Campus Sorocaba

brunoasilva88@gmail.com

***Resumo* — Este artigo descreve uma solução para o armazenamento e processamento de grande quantidade de dados através do uso de um sistema distribuído de arquivos proposto pelo Apache Hadoop, um sistema escalável, confiável e tolerante a falhas.**

***Palavras-chave* — Hadoop, HDFS, cloud computing, sistema de arquivos distribuido, escalabilidade.**

I. Introdução

UM fenômeno que ocorre em aplicações web de grande sucesso é o crescimento exponencial da quantidades de dados gerados e que precisam ser armazenados e processados. Grandes sistemas, como Google, Facebook, Twiter entre outros, possuem bases de dados na ordem de petabytes. A solução utilizada para manter a escalabilidade destes sistemas é o processamento distribuído de dados em larga escala.

Inicialmente criado por Doug Cutting, O Apache Hadoop é um sistema de alto desempenho e tolerante a falhas para armazenamento e processamento de dados. É econômico e confiável, o que o torna perfeito para rodar aplicações com grande volume de dados em um conjunto de computadores.

O Apache Hadoop desenvolve uma solução de código aberto, licenciado pela Apache, para computação distribuída, escalável e segura. Possui um framework que proporciona o processamento distribuído de grandes conjuntos de dados em *clusters* de computadores usando um modelo de programação simples. Trata-se de um modelo que tem como objetivo ampliar o processamento de simples servidores para milhares de computadores, os quais oferecem armazenamento e processamento interno.

Ao invés de confiar no *hardware* para proporcionar alta disponibilidade, o Hadoop foi projetado para detectar e manipular falhas na camada de aplicação, portanto, proporciona um serviço de alta disponibilidade no topo de um *cluster* de computadores, os quais, individualmente, podem falhar.

Grandes corporações como Google, Yahoo, Facebook, Cloudera entre outras contribuem com o desenvolvimento do Apache Hadoop.

O Apache Hadoop consiste em dois serviços principais: o Sistema de Arquivos Distribuídos Hadoop (HDFS), que proporciona um armazenamento de dados confiável e o Hadoop MapReduce, responsável pelo processamento concorrente de dados com alto desempenho. Nas próximas sessões, haverá uma explanação de cada um destes subprojetos.

II. Sistema de Arquivos Ditribuído Hadoop (HDFS)

É o componente de sistema de arquivos distribuído do Hadoop, baseado no Google File System. Este sistema armazena os metadados separadamente dos dados da aplicação. Os metadados, como em alguns outros sistemas de arquivos distribuidos, são armazenados em um servidor dedicado, chamado de NameNode (nó de nomes).

Os dados da aplicação, por sua vez, são armazenados em outros servidores chamado de DataNodes (nós de dados). Todos os servidores são conectados e se comunicam utilizando protocolos baseados em TCP.

Ao contrário de alguns outros sistemas de arquivos distribuídos, o HDFS não utliliza nenhum mecanismo de proteção de dados, porém, todos os dados são replicados em diferentes DataNodes para garantir a confiabilidade. Com essa abordagem, a durabilidade dos dados é garantida, além de proporcionar maior largura de banda e mais oportunidade de processamento perto do dado solicitado.

A seguir, serão detalhados a arquitetura do HDFS, os mecanismos de entrada e saída de arquivos e o gerenciamento de replicas.

A. Arquitetura

1) NameNode

Arquivos e diretórios são representados no NameNode por inodes, os quais armazenam as permissões, horário de acessos e modificações, *namespace* (hierarquia de arquivos e diretórios) e espaço ocupado em disco. O conteúdo dos arquivos é dividido em vários blocos, usualmente cento e vinte e oito megabytes, mas o usuário pode especificar seu tamanho em cada arquivo, e estes blocos são distribuidos e replicados em vários DataNodes, geralmente três, porém o usuário pode especificar o número de replicas para cada arquivo, como o tamanho dos blocos.

O NameNode possui a responsabilidade de gerenciar a hierarquia do namespace, mapeando todos os blocos de arquivos em seus respectivos DataNodes (endereço físico dos dados).

Quando um cliente HDFS requer a leitura de um arquivo, primeiramente é enviada a requisição ao NameNode, que

retorna o endereço das réplicas dos blocos que estão mais próximas do cliente.

Para gravar um arquivo, o cliente solicita ao HDFS ao NameNode a alocação de um conjunto de três DataNodes para hospedar as réplicas dos blocos. Em seguida, o cliente escreve os dados paralelamente nos três DataNodes previamente alocados. No modelo atual, cada cluster possui apenas um NameNode e pode possuir milhares de DataNodes, além de dezenas de milhares de clientes HDFS, pelo fato de cada DataNode possui a capacidade de processar multiplas tarefas de forma concorrente.

O HDFS mantém todo o seu namespace na memória RAM. A image de um arquivo é composta pelo conteúdo de seu inode e sua lista de blocos nos DataNodes. É chamado de checkpoint o armazenamento de uma image no sistema de arquivos hospedeiro local. O NameNode é encarregado de armazenar a lista de atualizações de uma image no sistema de arquivos local, esta lista é chamada de *journal*. Para proporcionar a durabilidade dos dados, réplicas dos checkpoints e dos journals são armazenadas em outros servidores. A localização das réplicas pode mudar a qualquer momento, portanto, não está contida entre os elementos do *checkpoint* persistente.

2) DataNodes

Um bloco armazenado em um DataNode é composto por dois arquivos: o primeiro contém o próprio dado do arquivo e o segundo contém seus metadados. O tamanho do arquivo e igual ao espaço real do bloco e não precisa de espaço extra para arredondar para o valor nominal do tamanho de um bloco como em um sistema tradicional de arquivos. Portanto, caso haja um bloco com apenas a metade do tamanho padrão, este necessitará de apenas metade do espaço de um bloco padrão em disco.

O ID do namespace é criado para a instância do sistema de arquivo quando ele é formatado. O ID do namespace é gravado em todas os nós do cluster. Quando um nó possui um ID de outro namespace, não é permitido o seu acesso ao cluster, garantindo a integradade do sistema de arquivos. Um novo nó que foi criado recentemente e ainda não possui um identificador de namespace poderá juntar-se ao cluster e obter seu ID.

Um DataNode indentifica as réplicas de blocos que estão sob sua posse enviando um block report ao NameNode. Um block report contém o indentificador do bloco, o generation stamp e o tamanho de cada bloco que o servidor hospeda. O primeiro block report é enviado imediatamente após o registro do DataNode, a partir desse momento, a cada uma hora, o DataNode envia um novo block report ao NameNode para atualizar a sua lista de onde estão as réplicas de seus arquivos no cluster.

Enquanto o DataNode executa uma operação, a cada três segundos envia um heartbeat ao NameNode para confirma que permanece em funcionamento e suas réplicas continuam disponíveis. Se o NameNode pára de receber os heartbeats de um DataNode, após dez minutos o NameNode assume que este DataNode está fora de serviço e suas réplicas não estão mais disponíveis. Neste caso, o NameNode cria novas réplicas em outros DataNodes para manter o número desejado de réplicas de um arquivo.

Além de confirmar a disponibilidade de um DataNode, seu heartbeat leva consigo informações sobre sua capacidade de armazenamento, espaço em uso e número de dados sendo transferidos no momento. Estes dados são utilizados pelo NameNode para fazer a distribuição e balanceamento dos arquivos.

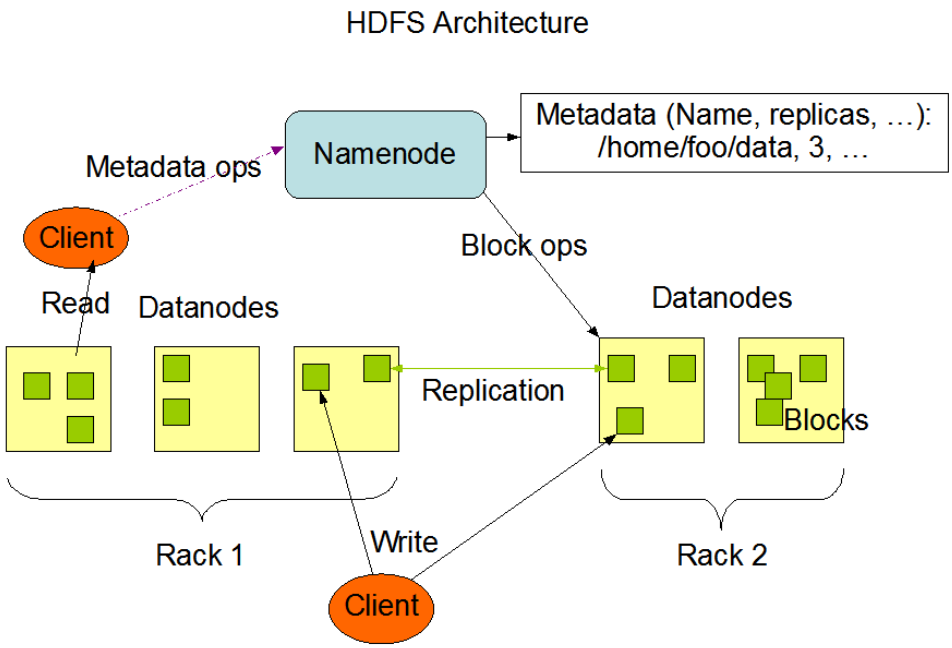

Figura 1: Arquitetura do HDFS

O NameNode não manda mensagens diretas aos DataNodes, apenas responde aos heartbeats para enviar instruções aos DataNodes. Algumas dessas instruções são:

- Replicar blocos para outros nós
- Remover réplica local de um bloco
- Refazer o registro ou desativar um nó
- Enviar um block report imediato

Estes comandos são importantes para manter a integridade do sistema e, portanto, é fundamental manter o envio frequente do heartbeat, mesmo em grandes grupos. O NameNode pode responder a milhares de heartbeats por segundo sem afetar a execução de outros NameNodes.

3) Cliente HDFS

Biblioteca de código que proporciona uma interface ao HDFS para as aplicações que o utilizam. Como a maioria dos sistemas de arquivos, o HDFS suporta operações de leitura, escrita e exclusão de arquivos, além de operações de criar e remover diretórios. O usuário referencia os arquivos e diretórios através de caminhos no *namespace*. Normalmente, a aplicação do usuário não precisa saber que os arquivos e seus metadados estão em diferentes servidores, assim como os blocos possuem várias réplicas.

Para solicitar a leitura de um arquivo, o cliente HDFS primeiramente faz uma requisição ao NameNode por todos os DataNodes que possuam réplica dos blocos do arquivo solicitado. Então, o cliente HDFS faz a requisição diretamente a um DataNode pela transferência do bloco desejado.

Para fazer uma escrita, o cliente HDFS solicita ao NameNode que selecione um DataNode para efetuar o armazenamento do primeiro bloco. Assim que o NameNode escolhe um DataNode, retorna ao cliente HDFS que escreve o bloco e requer ao NameNode um DataNode para armazenar o segundo bloco, e assim sucessivamente até que todos os blocos sejam armazenados. Esta sequencia de operações é representada na figura 2.

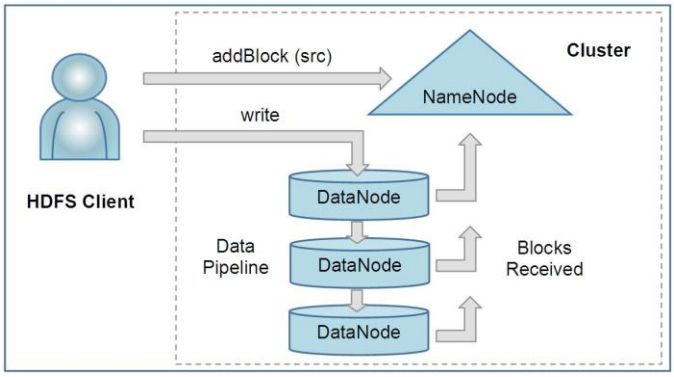

Figura 2: Cliente HDFS

Diferente dos sistemas de arquivos, o HDFS fornece uma API que expõe a localização dos blocos de um arquivo. Com isso, é possível que aplicações como o MapReduce agende tarefas onde os dados estão armazenados, aumentando o desempenho da leitura.

4) Image e journal

Image é metadado do sistema de arquivos que descreve os dados da aplicação em uma hierarquia de arquivos e diretórios. O journal é uma lista das alterações realizadas no sistema de arquivos que é armazenada. Para cada transação iniciada pelo cliente, o journal é atualizado e antes da alteração ser executada pela HDFS cliente. Um arquivo de checkpoint não é nunca alterado. É apenas sobrescrito quando um novo checkpoint é criado durante o reinicio, por uma requisição do administrador ou por um CheckpointNode que será descrito na próxima seção.

Durante sua inicialização, o NameNode cria a image do namespace do checkpoint e realiza as alterações armazenadas no journal até que a image esteja atualizada conforme o ultimo estado do sistema de arquivos. Um novo checkpoint e journal vazios são escritos de volta para os diretórios de armazenamento antes que o NameNode comece executar as tarefas do cliente.

Tanto a image quanto o journal devem ser replicados, pois com a perda ou falha de um desses arquivos, parte das informações do namespace ou sua totalidade será perdida.

5) CheckpointNode

O NameNode em HDFS, além de seu papel principal atendendo a solicitações do cliente, pode executar, alternativamente, um dos outros dois papéis, ou de um CheckpointNode ou de um BackupNode. O papel é especificado na inicialização do nó.

O CheckpointNode periodicamente combina os checkpoint e journals existentes para criar um novo checkpoint e um journal vazio. O CheckpointNode geralmente é executado em um local diferente do NameNode, uma vez que possuem os mesmos requisitos de memória. Ele baixa os arquivos do checkpoint e journal atuais do NameNode, os sincroniza localmente, e retorna o novo checkpoint para o NameNode.

6) BackupNode

Recurso recentemente introduzido no HDFS que, assim como o CheckpointNode, tem como função criar checkpoints periódicos mas, além disso, mantém em memória uma image do namespace do sistema de arquivo que é sempre sincronizada com o estado do atual do NameNode.

O BackupNode recebe o journal das transações do namespace vindo do NameNode ativo, o salva em seu próprio diretório e aplica tais transações em sua própria image do namespace. O NameNode trata o BackupNode como um arquivo journal em seus diretórios. Em caso do NameNode falhar, a image do BackupNode e o checkpoint em disco serão recuperados como o estado do namespace mais recente.

O BackupNode pode ser visto como um NameNode somente leitura. Ele contém todos os metadados do sistema de arquivo, exceto localizações dos blocos. Ele pode realizar todas as operações de um NameNode regular que não impliquem em modificação do namespace ou o conhecimento das localizações dos blocos. O uso de um BackupNode

fornece a opção de executar o NameNode sem armazenamento persistente, delegando a responsabilidade de armazenamento do estado do namespace ao BackupNode.

B. Leitura e escrita de arquivos

Para armazenar um novo arquivo no HDFS, a aplicação cliente tem que primeiramente criar um novo arquivo e escrever o conteúdo nele. Uma vez fechado, este arquivo não poderá ter nenhum dado apagado ou alterado, somente poderá acrescentar mais dados, reabrindo o arquivo para apêndice. O HDFS implementa um modelo de escrita simples e múltipla leitura.

O HDFS cliente que abre um arquivo para leitura possui exclusividade na permissão de escrita, portanto, nenhum outro cliente poderá escrever neste arquivo. Porém, este cliente que possui permissão de escrita deve enviar periodicamente um heartbeat ao NameNode para renovar sua permissão. Esta permissão exclusiva é mantida até que o arquivo seja fechado. O NameNode estabelece dois tipos de timeout para a permissao de escrita: um curto e outro longo. Quando o escritor pára de enviar seus heartbeats, depois de um curto timeout, caso o cliente escritor falhe em fechar o arquivo ou em renovar sua permissão, outro cliente poderá solicitar a permissão de escrita no arquivo. Após um longo timeout (uma hora), o HDFS assume que o cliente não está mais funcionando e fecha o arquivo. A permissão de escrita é exclusiva, porém, não impede nenhum outro cliente de ler o arquivo. Um arquivo poderá ter vários clientes lendo-o simultaneamente.

Quando é necessário acrescentar mais um bloco a um arquivo, o NameNode aloca um novo bloco com um ID de bloco e determina uma lista de DataNodes que irá armazenar as réplicas desse bloco. A ordem destes DataNodes é criada de forma a minimizar a distância da rede criada do cliente até o último DataNode. Por esta rede, os bytes são enviados como uma sequência de pacotes.

Na figura 3, há uma ilustração dos passos para armazenar um bloco no HDFS com três DataNodes. Onde, as linhas verticais representam o cliente e os DataNodes (NDs), linhas finas são mensagens de controle para criar e fechar a transferência, linhas largas representam envio de pacote e linhas pontilhadas a confirmação do recebimento da mensagem (ack).

O intervalo t0 a t1, na figura 3, representa o tempo de estabelecer a conexão, o intervalo t1 a t2, é o tempo de transferência dos dados e o intervalo t2 a t3, é o tempo para fechar o arquivo, portanto, a partir de t3, é garantido que o arquivo estará disponível para consulta.

Ao escrever um arquivo no HDFS, apenas será garantido que o arquivo estará disponível para leitura após o seu fechamento. Para garantir sua disponibilidade antes do seu fechamento é preciso chamar a função hflush, que envia todos os pacotes aos DataNodes imediatamente e aguarda a reposta de todos os DataNodes para retornar.

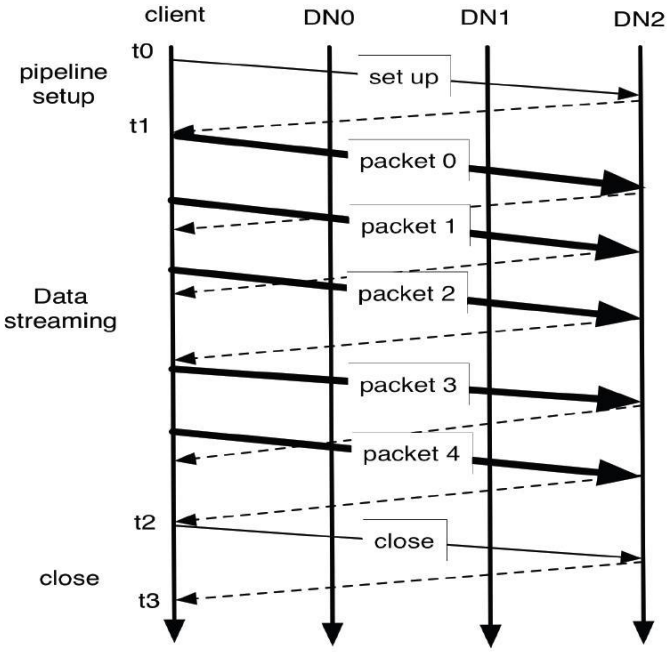

Figura 3: Armazenando um bloco no HDFS

O controle de arquivos corrimpidos no HDFS é feito da seguinte maneira: quando um novo arquivo HDFS é criado, é computado um checksum para cada bloco deste arquivo e este checksum é armazenado por cada DataNode que contém uma réplica do bloco em um arquivo de metadados, separado dos dados do bloco. Quando um cliente recebe um bloco de um DataNode, ele recalcula o checksum do bloco recebido e confere se está igual ao checksum armazenado pelo DataNode. Caso contrário, informa ao NameNode que aquele bloco está corrompido e busca por outra réplica daquele bloco no próximo DataNode da lista de blocos.

C. Distribuição de DataNodes e blocos

Em clusters com um grande número de nós, torna-se ineficiente a conexão destes nós de maneira linear. Neste caso, os nós são divididos em vários racks. Nós de um mesmo rack compartilham um mesmo switch, enquanto os switches de diferentes racks são conectados através de um switch central. Portanto, a largura de banda entre nós de um mesmo rack geralmente é melhor em relação a nós em diferentes racks.

Na figura 4 há uma ilustração de um cluster com dois racks, cada um com três DataNodes.

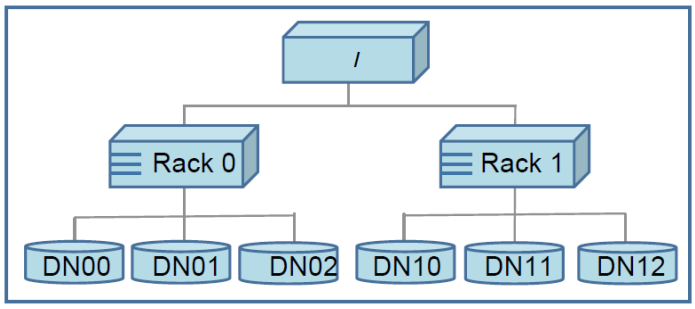

Figura 4: Distribuição dos DataNodes

O HDFS estima a largura de banda entre dois nós através da distância entre eles. Assume-se que a distância de um nó a seu nó pai é um. E, a distancia entre dois nós é a distância deles ao ancestral comum mais próximo.

A distribuição das réplicas dos blocos pode ser programada pelo administrador de acordo com sua necessidade, através de scripts que retornam o endereço de cada bloco e podem decidir em qual rack a réplica será armazenada.

Porém, o HDFS oferece uma política padrão de distribuição de blocos. Esta política pode ser resumida em apenas duas condições:

1. Nenhum DataNode pode conter mais de uma réplica de um mesmo bloco.
2. Nenhum rack contém mais de duas réplicas de um mesmo bloco, uma vez que tenho racks suficiente no cluster.

D. Controle de replicas

O NameNode é o responsável por controlar o número de réplicas de um bloco. Ele detecta a falta ou sobra de réplicas através de um block report enviado por um DataNode.

Quando há sobra de uma réplica, uma réplica tem que ser escolhida para ser excluída. O NameNode prioriza, primeiramente, a exclusão de uma réplica que não cause a diminuição do número de racks e, em seguida, uma réplica que esteja em um DataNode com o menor espaço disponível em disco. A meta é balancear o uso do armazenamento dos DataNodes, sem perder a disponibilidade dos blocos.

Quando há a falta de uma réplica de um bloco, este bloco é posto em uma fila com prioridade, onde quanto menor o número de réplicas, maior a prioridade. Uma thread de fundo, periodicamente, pega o primeiro elemento desta lista de blocos a serem replicados, e decide onde será armazenada a réplica. A politica para a escolha do DataNode que armazenará a nova réplica é similar a utilizada para criar um novo bloco, visando estabelecer réplicas em diferentes nós e racks.

O NameNode também verifica se todas as réplicas de um bloco não se encontram em um mesmo rack, caso positivo, o classifica como se estivesse com falta de réplicas, inserindo-o na lista de blocos a serem replicados. Após o NameNode receber a notificação que a réplica foi criada, o bloco torna-se com sobra de réplicas. O NameNode então decide remover a réplica mais antiga, porque a política de replicação prefere não reduzir o número de racks.

E. Balancer

O Balancer é uma ferramenta que equilibra a utilização do espaço em disco em um cluster HDFS. Necessita de um valor limiar como um parâmetro de entrada, que é uma fração entre zero e um. Um cluster é equilibrado se, para cada DataNode, a utilização do nó (razão entre a quantidade de espaço utilizado no nó e a capacidade total do nó) difere da utilização de todo o cluster (razão entre a quantidade de espaço utilizado no cluster e a capacidade total do cluster) por um valor menor ou igual ao valor limite.

III. HADOOP MAPREDUCE

O Hadoop MapReduce é um framework que permite escrever aplicações que processam grandes quantidades de dados (em ordem de terabytes) em paralelo utilizando enormes clusters de máquinas (milhares de nós) de forma confiável e tolerante a falhas. Mesmo o Hadoop sendo implementado em Java, aplicações MapReduce não necessitam ser, obrigatoriamente, implementadas em Java.

Uma tarefa do MapReduce geralmente divide os dados de entrada em blocos independentes que são processados pelas tarefas *map* de forma totalmente paralela. Em seguida, o framework distribui as saídas dos maps de forma aleatória para as tarefas reduce. Este framework se encarrega de agendar tarefas, monitorá-las e as re-executar em caso de falha.

Geralmente, a execução e o armazenamento são realizados em um mesmo conjunto de nós, ou seja, o MapReduce e o Sistema de Arquivos distribuído (HDFS) rodam juntos no referido conjunto de nós. Com esta configuração, é garantido que as tarefas sejam agendadas, pelo MapReduce, nos nós onde estão armazenadas, implicando em alta largura de banda em todo o cluster.

O MapReduce é composto por um JobTraker (mestre) e um TaskTraker (escravo) em cada nó do cluster. O JobTraker é responsável pelo agendamento, monitoramento e re-execução em caso de falha das tarefas, enquanto o TaskTraker as executa conforme requisitado pelo JobTraker.

As aplicações especificam detalhadamente os locais de entrada e saída e a fonte das funções map e reduce através da implementação de interfaces e/ou classes abstratas apropriadas. A configuração do job é composta por estes parâmetros entre outros.

O job cliente que submete o job (jar, executável, etc) e a configuração ao JobTraker, que se torna o responsável por distribuir as tarefas aos TaskTraker, agendando, monitorando e as re-executando quando preciso, além de retornar informações do status e do diagnóstico para o job cliente.

IV. CONCLUSÃO

A primeira vantagem do Hadoop é ser um software de código aberto. É uma plataforma ideal para consolidação de dados em larga escala. Ele complementa as soluções existentes de gerenciamento de dados com novas ferramentas de análises e de processamento.

É utilizado pelo Facebook para gerenciar seus petabytes de dados, pela Microsoft, pelo Yahoo e pelo Google por diminuir o custo para manter o processamento, armazenamento e acesso a grandes volumes de dados via web.

Esta diminuição do custo está relacionada ao fato de não precisar de um *Data Center* de grande porte onde todo o armazenamento e processamento é concentrado. Em vez disto, o Hadoop permite que o armazenamento e o processamento sejam distribuídos em milhares de computadores de pequeno porte.

Além disso, o MapReduce torna possível dividir grandes

conjuntos de dados em subgrupos, difundir essa informação por milhares de computadores, fazer perguntas às máquinas e receber as respostas. A tecnologia permitiu, por exemplo, que o software de busca do Google funcionasse de forma mais rápida em computadores mais baratos e menos confiáveis, o que significa menos despesas e menor investimento em hardware.

REFERÊNCIAS

[1] Apache Hadoop. http://hadoop.apache.org/
[2] K. Shvachko, H. Kuang, S. Radia, R. Chansler, "The Hadoop Distributed File System", Yahoo!, Sunnyvale, California USA, 2010.
[3] Cloudera. http://www.cloudera.com/
[4] Escalabilidade. http://escalabilidade.com/tag/hadoop/
[5] Yahoo! Developer network. http://developer.yahoo.com/hadoop/
[6] Hadoop Wiki. http://wiki.apache.org/hadoop/ProjectDescription