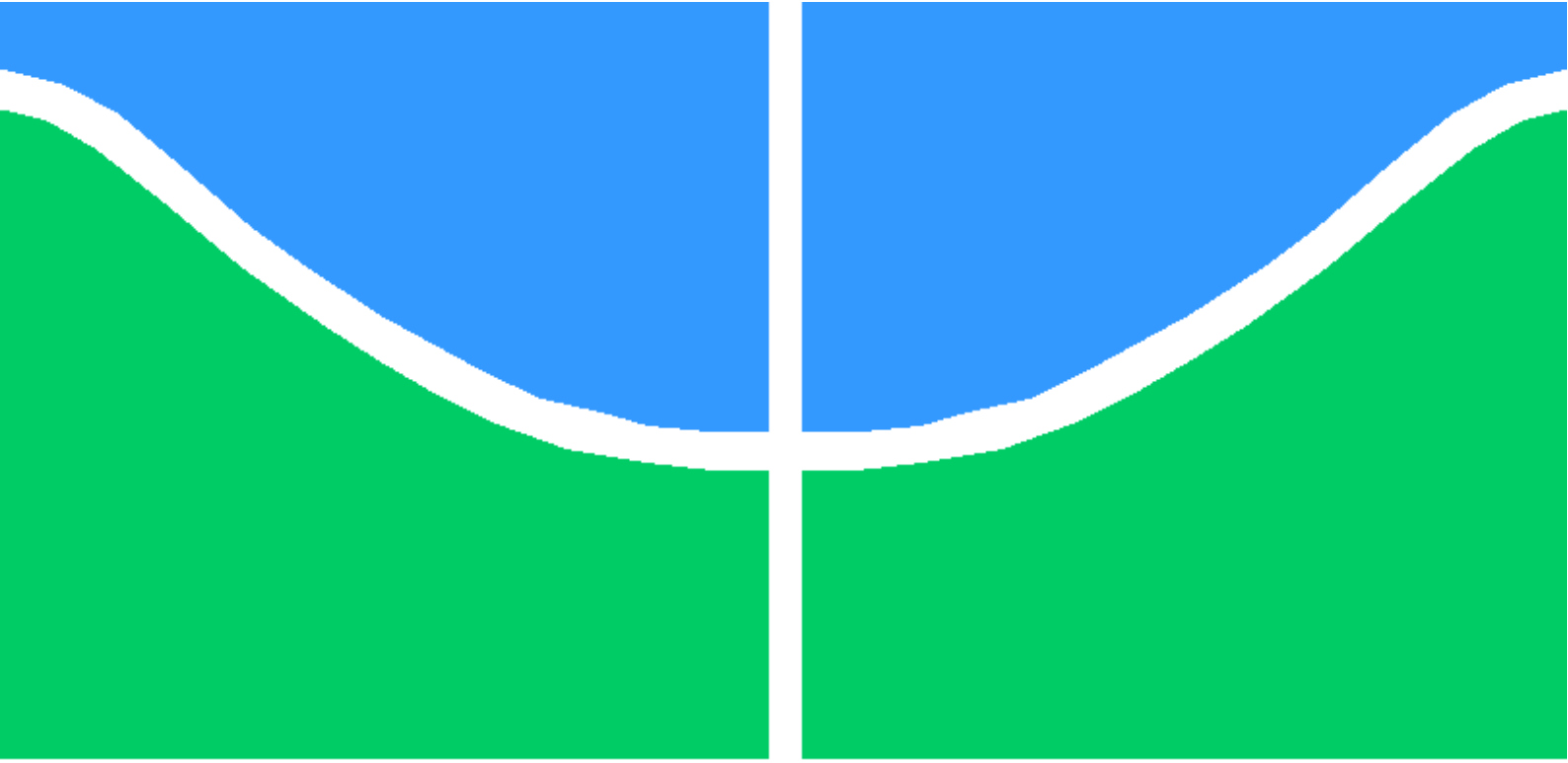

TRABALHO DE GRADUAÇÃO

MODELAGEM DO SENTIMENTO NEGATIVO EM OPINIÕES ACERCA DE SERVIÇOS DE TELECOMUNICAÇÕES EMITIDAS POR USUÁRIOS DO TWITTER

Bruno Monteiro Pimentel de Alencar

Brasília, Julho de 2015

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

MODELAGEM DO SENTIMENTO NEGATIVO EM OPINIÕES ACERCA DE SERVIÇOS DE TELECOMUNICAÇÕES EMITIDAS POR USUÁRIOS DO TWITTER

Bruno Monteiro Pimentel de Alencar

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Dr. Rafael Timóteo de Sousa Jr, ENE/UnB _____
Orientador

Prof. Fábio Mesquita Buiati, PPGEE/UnB _____
Examinador Interno

Prof. MSc. Fábio Lúcio Lopes Mendonça _____
Examinador Externo

Dedicatória

Dedico este trabalho primeiramente aos meus pais, Aderson e Nildete, que me deram plena liberdade e confiança para seguir através destes anos. Dedico também a todos que estiveram presentes durante o trabalho e toda a difícil etapa de graduação.

Bruno Monteiro Pimentel de Alencar

Agradecimentos

Agradeço a todos os meus amigos, familiares e professores que durante o percurso da graduação contribuíram para o meu amadurecimento pessoal, acadêmico e profissional, preenchendo este caminho com boas recordações. Agradeço a Deus por tornar tudo possível, sempre iluminando o meu caminho e as pessoas que passam por ele.

Bruno Monteiro Pimentel de Alencar

RESUMO

Este trabalho apresenta um estudo da modelagem sentimental — mapeamento das satisfações — dos usuários sobre as operadoras de telecomunicações no Brasil através da rede social *Twitter*. Com reclamações ou opiniões publicadas abertamente no microblog, é possível traçar e obter informações valiosas de como está a situação de cada prestadora, assim como, dos serviços por elas prestado. A proposta principal do trabalho é desenvolver uma plataforma, com ferramentas de *Big Data* (*Apache Hadoop*, *Apache Flume* e *MapReduce*), que possa coletar, armazenar e processar esses dados segmentando-os pela hora de coleta, operadora e serviço. Ou seja, com isso pode-se descrever o comportamento dos usuários, prever possíveis eventos na área de telecomunicações e mais ainda: constatar possíveis falhas nos sistemas das prestadoras. É utilizado no ambiente ferramentas que suportam grandes volumes de dados, justamente para a possível implementação do sistema para coletar informações da rede social sobre as operadoras a nível mundial.

ABSTRACT

This graduation projet present a study of sentimental modeling — satisfacation mapping — of users of telecomunication providers in Brazil through the social network Twitter. With complaints or opinions published openly in the microblogging, it can ben traced and obtained valuable information as is the situation of each provider, as well as the services they provided. The main goal of this project is to develop a plataform, with Big Data tools (Apache Hadoop, Apache Flume e MapReduce), to collect, store and process the data segmenting them by the time of collection, provider and service, i.e., it can describe the behavior of the users, anticipate possible events in telecommunications and more: find possible flaws in the systems providers. Is is used tools with can handle large volume of data, just for the possible implementation of the system to collect information on the social network operatores worldwide.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Abreviações

API	Aplication Programming Interface
CPU	Central Processing Unit
E/S	Entrada e saída
EC2	Amazon Elastic Computer Cloud
IP	Internet Protocol
GB	Gigabyte
GPS	Global Positioning System
HDD	Hard Drive Disk
HDFS	Hadoop Distributed File System
HiveQL	Hive Query Language
HTTP	Hypertext Transfer Protocol
JSON	Javascript Object Notation
KB	Kilobyte
MB	Megabyte
PDA	Personal Digital Assistant
PHP	Hypertext Preprocessor
RAM	Random Access Memory
RESTful	Representational State Transfer
SAC	Serviço de atendimento ao consumidor
SNN	Secondary NameNode
SQL	Structured Query Language
SSD	Solid State Disk
TCP	Transmission Control Protocol
TI	Tecnologia da Informação
WEB	Word Wide Web
XML	Extensible Markup Language

1 INTRODUÇÃO

Temos hoje uma grande concorrência entre as operadoras de telecomunicação no Brasil, tanto para oferecer um serviço melhor ao cliente existente quanto pela busca incessante por novos usuários. Entretanto, o Brasil possui poucas dessas empresas e elas precisam obter lucro para se sustentarem. Nesse cenário, parece existir um resultado geral dos serviços de telecomunicações que não agrada a uma importante parcela de clientes e em consequência as reclamações são tornadas públicas.

Com o advento da *Internet* e das redes sociais, os clientes começaram a buscar novos meios para comunicar informações sobre os serviços e reclamar da empresa ou do serviço prestado. Ou seja, os usuários viram nas redes sociais um ambiente livre onde podem publicar uma reclamação — ou elogio — de serviços que estão contratando. Avaliar tais situações é o foco do presente trabalho.

1.1 OBJETIVOS

Neste trabalho, consideramos o problema da construção de um ambiente de captura de dados escalável e eficiente que possa coletar grandes massas de dados. Podemos traçar etapas a serem analisadas: a criação de um ambiente de baixo custo, a coleta, a armazenamento e a manipulação dos dados capturados.

Além disso, tem-se como objetivos secundários o estudo e segmentação dos dados por operadoras, serviços e o tempo. Com isso, busca-se nesse trabalho desenvolver uma nova ferramenta para que usuários visualizem em "tempo real" informações de cada operadora e possam tomar decisões, conforme os dados apresentados.

1.2 MOTIVAÇÃO

As empresas de telecomunicação são citadas a cada segundo nas redes sociais. Com isso, podemos utilizar essas informações públicas para traçar e modelar sentimentos dos clientes. Quanto mais informação estiver disponível, mais segmentado pode ser esse mapeamento e, o mais importante, mais próximas da realidade podem ser essas informações.

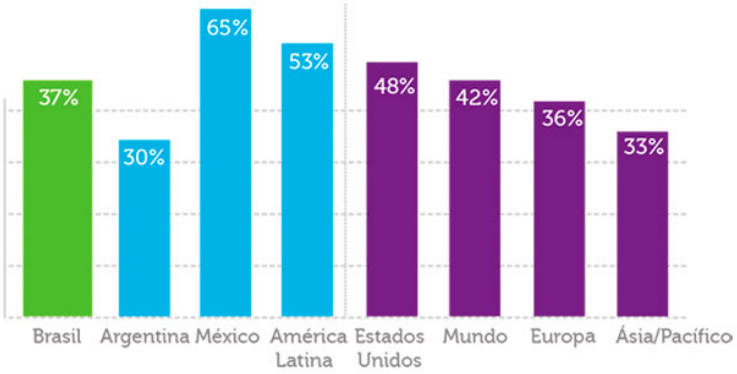

Fig. 1.1: Percentual de empresas citadas no *Twitter* (BURSON-MARSTELLER, 2010)

De acordo com um estudo da *Burson Marsteller* (BURSON-MARSTELLER, 2010), a América Latina em 2010 possuía 53% das empresas citadas no *Twitter*, acima da média mundial. Isso significa que as pessoas estão dispostas a interagir.

O serviço de atendimento ao consumidor (SAC) também está evoluindo e virando tendência no Brasil. Entre várias comunicações tradicionais via WEB (*e-mail*, formulários de contato), as empresas estão disponibilizando ao consumidor as redes sociais. O *Twitter* tem se mostrado o meio mais rápido de contato entre consumidores e companhias.

As empresas, que utilizam o *Twitter* como mais um canal para atender o cliente, acabam também investindo em estratégias de marketing. Uma mensagem de um cliente insatisfeito pode se propagar na rede social, caso haja uma demora na resposta ao consumidor, comprometendo assim a imagem da empresa. As organizações frequentemente querem neutralizar, rapidamente, a insatisfação de um cliente e, neste caso, se feito de forma inteligente, pode até funcionar a favor da empresa.

Um estudo realizado pela *E.life* (E.LIFE, 2014), divulgado em 15 de março de 2013, reuniu cerca de 521 mil *tweets* feitos no microblog durante março de 2012 a março de 2013. Os *tweets* foram coletados com base na existência da "*hashtag*" *#fail*, que deu o nome ao estudo: "Estudo *#fail*". O estudo constatou que cerca de 15 mil *tweets* faziam referências negativas a empresas de telecomunicações, ou seja, cerca de 3% do total.

As principais reclamações foram da qualidade dos serviços, falhas de sinal e da extrema dificuldade para cancelamento de contas.

No presente trabalho, é buscado a captura e avaliação das reclamações no Twitter, de modo a obter uma síntese dos sentimentos negativos relativos a serviços de telecomunicação no Brasil.

Este estudo objetiva contribuir com a implantação de uma ferramenta que possa por meio da análise de dados, identificar a emoção dos clientes expressa de forma escrita sobre as operadoras de telecomunicação no Brasil em "tempo real", baseado no monitoramento da rede social *Twitter*.

Embora o "Estudo *#fail*" citado demonstre a insatisfação dos usuários, a coleta de dados no cenário brasileiro não é adequada. A *hashtag* "*#fail*" é um termo associado às reclamações de consumidores no *Twitter*. Porém, os consumidores não a utilizam frequentemente. Se o mesmo estudo for feito sem adaptações ao Brasil, muitos dados serão perdidos e não são contabilizados por se tratar de uma captura superficial, não apontando para o volume total de reclamações.

Deste modo, haverá uma coleta com grande volume de dados, mais segmentações e maior precisão. Em comparação ao estudo da *E.life*, o trabalho utiliza mais palavras chaves e existe o relacionamento entre elas. A previsão de *tweets* que tenham conteúdo com sentimento de insatisfação é de, aproximadamente, 10% a 15%.

1.3 JUSTIFICATIVA

Conhecer as emoções dos atuais clientes das operadoras e proceder de acordo com dados reais representa uma forma de evitar prejuízos e insatisfação futura. Com essas informações coletadas podemos separar fatos de opiniões e emoções, classificar opiniões como positivas, negativas ou neutras, classificar os dados sob determinado ponto de vista e, além disso, identificar os textos de acordo com as diferentes emoções encontradas.

Este estudo pretende abordar o estudo das ferramentas que serão utilizadas, assim como todo o processo de criação do ambiente, da captura de dados e da manipulação. Uma proposta altamente aplicável, flexível e escalável. Flexível por se encaixar no contexto de diversos usuários para obter informações, assim como a própria empresa traçar metas para diminuir opiniões negativas nas redes sociais. Escalável por ser uma solução preparada para manipular uma porção crescente de dados.

1.4 METODOLOGIA DO TRABALHO

O projeto foi elaborado em quatro partes. Primeiramente partiu-se para um levantamento e pesquisa bibliográfica sobre todas as ferramentas e tecnologias aqui referenciadas. A seguir uma pesquisa exploratória sobre as técnicas de captura de dados nas redes sociais cujas características mais se adequam às necessidades que motivaram o trabalho.

Definindo as ferramentas e sistemas a serem utilizados, desenvolveu um estudo para determinar os filtros que serão utilizados para a obtenção dos dados específicos. Em seguida, um estudo do que pode realmente ser importante e quais os segmentos irão ser traçados.

A diante, é feita a implementação do ambiente, assim como é demonstrado a utilização do sistema. Os modos de armazenamento das informações e processamento são abordados de forma

teórica e prática.

Por fim, são apresentados os resultados de todo o caso real abordado, junto com as recomendações para melhor implantação do sistema e resoluções de problemas encontrados. Os resultados estão disponíveis em um sítio aberto a comunidade, acessando o link <http://telecomnasredes.com.br>.

1.5 ORGANIZAÇÃO DO TRABALHO

O estudo encontra-se estruturado em capítulos que buscam de maneira lógica guiar o leitor e, de forma didática, expor a pesquisa realizada.

No capítulo 2 traz os conceitos e fundamentações teóricas para o entendimento da dissertação, tendo como foco: a explicação de conceitos relativo a manipulação de grandes volumes de dados, sistemas distribuídos, bancos de dados e a explicação de como foram construídas as ferramentas e o funcionamento das mesmas.

No capítulo seguinte, são feitos os estudos relativos ao projeto prático. O que é necessário e como foram feitas as capturas e segmentações. Além disso, é demonstrado como se comporta o sistema com as funções para processamento. O armazenamento em bancos de dados relacionais também são abordadas neste capítulo.

No capítulo 4, a implementação do ambiente é tratada. Desde a construção ao funcionamento do sistema é abordada, assim como executado os testes de performance e parte física utilizada. Também é demonstrado as consultas que podem ser feitas sobre os dados para obter informações relevantes.

No capítulo 5 é apresentado os resultados. As informações coletadas em sua forma bruta, sem nenhum tipo de manipulação e posteriormente os dados processados e segmentados. Também é feita a avaliação dos resultados, para aferir se as proporções estão de acordo com os estudos e previsões. Neste capítulo também, é abordado os problemas encontrados, assim como suas possíveis soluções.

Por fim, no capítulo 6, há a conclusão do trabalho, onde é informado os setores e plataformas que podem ser utilizados no sistema. Recomendações para o projeto, e possíveis melhorias ao sistema também são abordados neste capítulo. Os trabalhos relacionados também são tratados.

2 CONCEITOS E FUNDAMENTAÇÃO TEÓRICA

2.1 BIG DATA

A presença cada vez maior da *internet* no dia a dia da sociedade faz com que até as mais simples ações cotidianas de seus cidadãos sejam registradas. Como resultado, 2.5 quintilhões de *bytes* de dados são gerados diariamente, oriundo de imagens digitais, vídeos, sensores inteligentes, transações eletrônicas, sinais de sistemas de posicionamento globais (GPS), entre outros (EATON et al., 2012). O aumento nesse volume é justificado, em partes, pelo maior acesso as tecnologias digitais e ao uso cada vez mais frequente que se tem feito delas.

A partir disso, as organizações tiveram que se adaptar e analisar todas as informações disponíveis em tempo real, aplicando técnicas de modelos estatísticos para ter a disposição informações para prognosticar o futuro e tomar decisões.

No entanto, além do grande volume de dados, uma dificuldade em se trabalhar com esses dados pode ser ainda mais complicada de ser resolvida. Na maioria das vezes, a informação contida nesses dados se revela de uma forma não coerente e estruturada, sendo necessário quase sempre que a organização lide com a informação contida em formatos diferentes. Imagens, tabelas, vídeos, textos, comentários, entre outros, devem ser não analisados como dados mas interpretados a ponto de se tornarem úteis a propósitos distintos.

A forma como lidar com essa variedade colossal de dados e, principalmente, de informação é um desafio ímpar para as empresas interessadas em explorar as oportunidades que tal informação pode trazer. Outro problema para as empresas contornarem é que o consumidor costuma mudar muito rapidamente suas necessidades de consumo. Para que a análise citada acima seja eficaz, ou seja, para que as vulnerabilidades dos consumidores sejam exploradas no momento certo, se faz necessaria grande velocidade na interpretação de informação.

Sendo variável a relevância e o prazo de validade de cada informação conseguida pela análise de dados, torna-se imprescindível classificar de forma correta o que deve ser ou não armazenado, posto que a capacidade de armazenamento é limitada e que quanto maior a quantidade de informações guardadas mais difícil será o acesso a essa informação quando esta se fizer útil.

O *Twitter*, plataforma digital de microblogging, por exemplo, possui uma rica base de dados sobre seus usuários. Somente em 2012, foram mais de 400 milhões de *tweets* enviados por usuários de todo o mundo (MAYER-SCHONBERGER; CUKIER, 2013). Ou seja, podemos considerar uma rica base de dados sobre seus usuários. Como toda essa informação publicado na rede social, conseguimos informar sobre interações dos usuários, humores e opiniões. Muitas empresas têm utilizado a ferramenta para analisar o nível de satisfação dos clientes, sucesso de

campanhas publicitárias e outros.

Big Data refere-se então ao conjunto de dados que não conseguem mais ser facilmente manipulados ou analisados pelas ferramentas, métodos e infraestrutura tradicionais.

2.1.1 Os 4 Vs

O conjunto de dados deve compartilhar pelo menos 3 características: grande volume, velocidade e variedade (HURWITZ et al., 2013). Alguns autores ainda citam a quarta característica: veracidade, que representa tanto a credibilidade dos dados quanto a adequação para uma determinado público (SATHI, 2012).

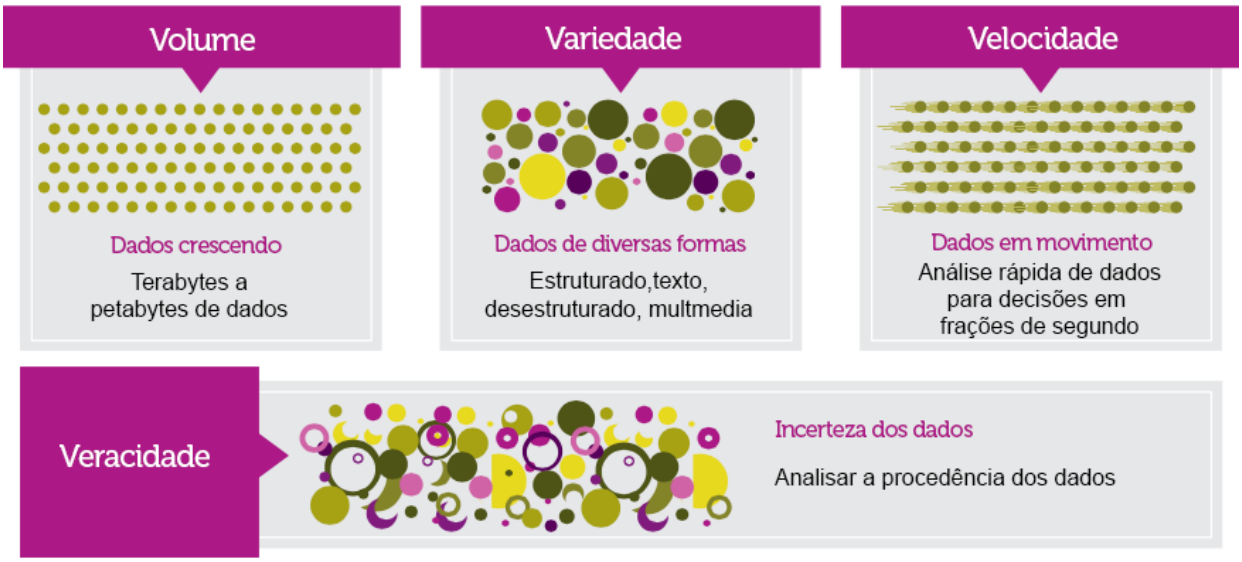

Fig. 2.1: As 4 características do Big Data (SCHROECK et al., 2013, modificado)

Essas características são fundamentais para visualizar e entender a natureza dos dados e as plataformas de *software* disponíveis para explorá-las.

O volume refere a quantidade de informações digitais que são produzidas pelos usuários e/ou processos de aplicações. Segundo a revista Fortune (FORTUNE, 2012, p. 163), nós criamos até 2003, 5 *exabytes* de dados digitais. Já em 2011, a mesma quantidade era criado em 2 dias. Em 2013, o período esperado era de apenas 10 minutos. É previsto, que em 2016, tratando-se somente de dados provenientes de celulares e PDAs, cerca de 10.8 exa*bytes* por mês, já que há um aumento significativo de compartilhamento de vídeos e fotos (MALIK, 2012).

Os benefícios obtidos da habilidade de processar grandes volumes de informação é a maior atrativo para a análise utilizando *Big Data*. O volume apresenta um dos maiores desafios para estruturas de tecnologia da informações (TI), já que necessita de um armazenamento escalável e um ambiente distribuído para as consultas.

Os processos *Big Data* em relação ao volume têm como objetivo encontrar novos padrões e tendências, onde as ferramentas atuais não conseguem trabalhar satisfatoriamente com tamanha quantidade de informação (PETRY; VILICIC, 2013, p. 74-75). Tratando de empresas e organizações, estas possuem grandes quantidades de dados armazenados, mas sem a capacidade de processar e obter informações valiosas. Assumindo que esta grande quantidade de dados seja superior ao que os bancos de dados convencionais conseguem trabalhar, o processamento dos dados se reduz basicamente a arquiteturas de processos paralelos ou soluções de *Apache Hadoop* desenvolvido para paralelizar o processamento de dados em diversos nós, com objetivo de aumentar o poder computacional e diminuir a latência (HURWITZ et al., 2013, p. 112).

Há dois aspectos referentes a velocidade, um que trata do fluxo de dados, ou seja, da vazão e o outro representa a latência. Para analisar todos esses dados, a infraestrutura almeja sempre melhores caminhos e processamentos paralelos. Com a velocidade, as informações são criadas, selecionadas e alocadas com uma rapidez maior do que os sistemas tradicionais. A consulta em tempo real se tornou-se a grande necessidade e exigência das áreas de análises de dados das grandes empresas e setores como o setor público, privado, trabalhadores e clientes.

Quem possuir esse poder de processamento rápido o bastante para utilizar a informação quase que instantaneamente ganha vantagens competitivas. Essa rapidez que chamamos do outro aspecto que tipifica a velocidade, a latência. Antigamente, era utilizado ambientes em que guardavam os dados e depois era confeccionado os relatórios. Hoje, com a tecnologia do *Big Data*, o conceito de dados em movimento (data-in-motion) é inserido e a resposta deve ser rápida o bastante para o objetivo desejado (SATHI, 2012).

Esse grande volume de dados em sua grande maioria não possui estruturação, são informações provindas de fontes variadas de dados, como por exemplo, fotos, músicas, mensagens de celular, entre outras. Antigamente, quando a tecnologia de Data Warehouse foi introduzida, o desafio foi representar todo o conteúdo em um formato padrão. As fontes de informação incluem, além de dados estruturados, fotos, vídeos, sons e todo e qualquer texto desestruturado.

Com *Big Data*, os horizontes foram expandidos, podendo haver uma integração dos dados e tecnologias de análise, descobrindo assim um novo padrão por meio do confronto de dados. O uso comum do processamento através do *Big Data* é utilizar estes dados desestruturados e extrair significados ordenados, para análise ou para a inserção estruturada de uma aplicação (DUMBILL et al., 2012);

Muitos dos dados são provenientes de fontes não confiáveis, nos quais, podem sofrer diversos tipos de problemas. Um deles é de precisão, na qual os dados confrontados nada tem a ver com o objetivo ou público desejado. A veracidade representa ambos a credibilidade dos dados como a adequação para dado público. Segundo Sathi (SATHI, 2012), a coerência dos dados para certa audiência não necessariamente sirva para outra. Devemos pensar na adequação e quanto a veracidade pode ser compartilhada com o público específico.

2.1.2 Personagens do Big Data e os tipos de dados

Existem duas funções de grande importância no mundo *Big Data*: os seus personagens e os mais diversificados tipos de dados. Com o advento da tecnologia e a era onde todos tem acesso a dispositivos, gerou um ambiente onde os indivíduos de qualquer classe social pode atuar como um gerador de conteúdo.

Tratando primeiramente dos personagens, a Fig. 2.2 ilustra as suas funções no ecossistema *Big Data*.

Fig. 2.2: Personagens do Big Data

A grande revolução no campo da informação advém da telefonia móvel em conjunto com os serviços disponíveis na *internet*, onde os usuários passaram a ser grandes contribuintes na geração de tráfico de informações (WEF, 2012). Um outro personagem no meio *Big Data* é o setor privado, pois é onde concentra o grande montante das informações transacionais como dados provenientes de resultados das suas aplicações, dos seus clientes e de seus produtos.

Temos também o papel fundamental do setor público, que da mesma forma possui enormes volumes de dados que são extraídos de diversas áreas como saúde, economia, censos, educação, entre outros.

Já se tratando dos tipos de dados, devemos definir o que seria dados estruturados e desestruturados. O termo dado estruturado geralmente refere-se aos dados que possuem tamanho e formato fixo. Exemplos disto são números, datas e grupos de palavras e números. Especialistas no campo de análise de dados concordam que esses tipos de dados compõem cerca de 20% de todo os dados que são gerados no mundo, e são eles que normalmente lidamos (HURWITZ et al., 2013).

Nesse tipo de fontes, separamos em duas categorias:

- **Dados de máquina**: refere-se aos dados gerados por máquinas e computadores sem a intervenção humana, como dados de sensores que podem ser transmitidos à receptores e analisados, dados de log de WEB, em que os servidores, aplicações e redes quando em funcionamento capturam qualquer tipo de dados provenientes de suas atividades. Esse tipo de fonte pode gerar um grande volume de informação útil, visto que é diretamente ligada ao usuário. Além desses dados, podemos citar dados financeiros e de vendas;
- **Dados de pessoas**: dados gerados por humanos, com interação de computadores e máquinas. Incluem dados de entrada, ou seja, todos os dados que o usuário pode inserir em uma máquina, como seu nome, idade, localidade, dentre outros. Esse tipo de informação é útil para entender o comportamento dos clientes. Além desses possuem dados de jogos, de fluxo e outros.

Se cerca de 20% dos dados gerados no mundo são estruturados, os outros 80% são de dados desestruturados. Nesses dados as informações não seguem um formato específico. O termo refere-se a dados que o conteúdo não é organizado em matrizes de atributos ou valores (BERMAN, 2013). São, na maioria, em forma de texto, como mensagens de *e-mail*, *tweets*, documentos, dentre outros.

Para obter valor de informação desses textos, é necessária alguma estrutura. Isso envolve traduzir o texto para determinada linguagem, analisar o texto em sentenças, extrair e normalizar termos conceituais contidos nas sentenças, mapear os termos para uma nomenclatura padrão, anotar os termos com códigos de uma ou mais nomenclatura padrão, extrair e padronizar os valores encontrados classificando classes de dados pertencentes a classificação do sistema, definindo os dados encontrados para armazenamento e recuperação do sistema e indexando os dados no sistema.

Big Data deve lidar com esses dados desestruturados de forma que retorne os melhores resultados possíveis com os recursos disponíveis (BERMAN, 2013). Outros exemplos de dados desestruturados é observado na figura abaixo:

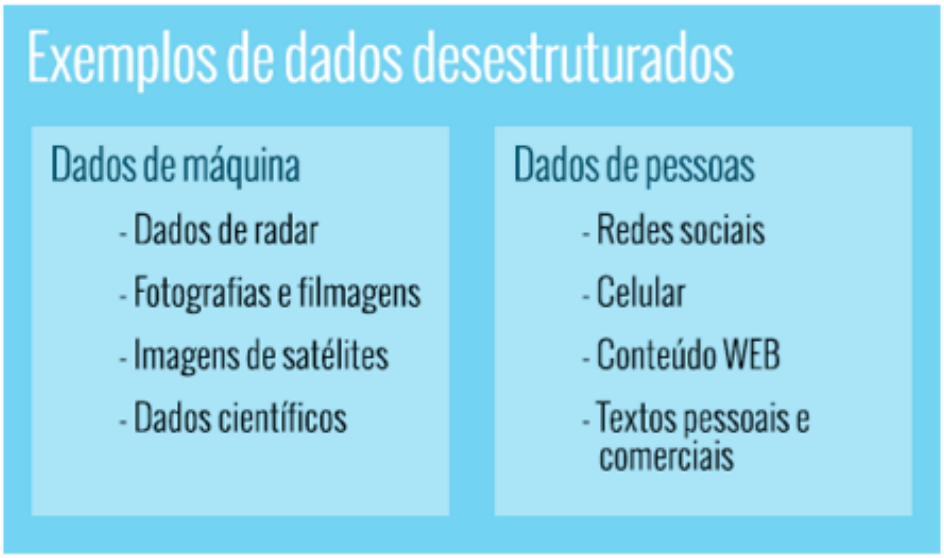

Fig. 2.3: Exemplos de dados desestruturados

Alguns autores ainda citam que a maioria desses dados desestruturados são pelo menos semiestruturados. Segundo Franks (FRANKS, 2012), dados semiestruturados ou multiestruturados são aqueles que possuem um fluxo lógico e um formato que pode ser entendido, porém não intuitivo para os humanos. Para se ler um dado semiestruturado afim de analisá-lo não é simples como especificar um tamanho fixo de formato. Para isso, é necessário empregar regras complexas que determinam, dinamicamente, como proceder depois de ler cada pedaço de informação (FRANKS, 2012). Um exemplo de um dado semiestruturado são os logs provindos de sistemas WEB:

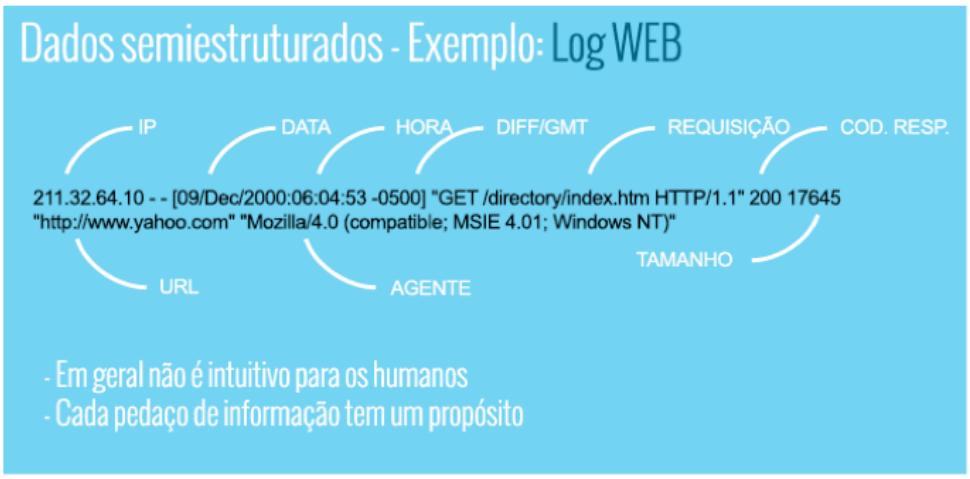

Fig. 2.4: Exemplos de dados semiestruturados

2.2 SISTEMAS DISTRIBUÍDOS E HADOOP

Quando o volume de dados ultrapassa a capacidade de armazenamento de uma máquina física, é necessário que haja a divisão desses dados em diversas outras máquinas (WHITE, 2012). Porém construir máquinas ou servidores cada vez maiores não é, necessariamente, a melhor solução para os problemas de larga escala. Um meio alternativo que tem ganhado popularidade nos últimos anos é construir várias máquinas "*low-end*" — sem muito desempenho/performance — que iremos chamá-las de *nós* e as unir como um simples *sistema distribuído* ("*scale-out*") (LAM, 2011).

Desde que esse sistema seja baseado em rede, todo o estudo e programação das redes são implementadas, deixando os sistemas distribuídos mais complexos do que os sistemas tradicionais de arquivos. Para comprovar essa complexidade, White cita um exemplo onde diz que um dos maiores desafios dos sistemas distribuídos é torná-los tolerantes a falha sem sofrer com perda de dados (WHITE, 2012).

Fig. 2.5: Desafios dos sistemas distribuídos

Não existe um simples modelo de sistema distribuído, porque os recursos podem ser distribuídos de várias formas. Pode-se distribuir, por exemplo, várias aplicações na mesma máquina física e usar serviço de mensagens para habilitar a comunicação e a transferência de informação entre eles. Também é possível possuir várias máquinas diferentes, cada uma com seu próprio recurso, que conseguem trabalhar juntas para resolver um problema.

Os avanços em *hardware* e *software* revolucionaram a indústria de gestão de dado de certa forma que a inovação e a demanda aumentaram o poder de processamento, armazenamento e diminuíram o custo do *hardware*. Segundo Hurwitz (HURWITZ et al., 2013), novos *softwares*

conseguem tirar vantagem do *hardware*, automatizando os processos, como o balanceamento de carga e a otimização em grandes *clusters* de nós. Esses *softwares* aprenderam como lidar com as requisições e conseguem determinar o nível de performance requerido para determinado problema.

Hadoop é um *open source framework* destinado a gravação e execução de aplicações distribuídas, tendo como principal objetivo o processamento de forma barata, de grande quantidade de dados (DUMBILL et al., 2012). Segundo Jason, Dean e Edward (RUTHERGLEN; WAMPLER; CAPRIOLO, 2012), o ecossistema do *Hadoop* surge como um meio de trabalhar com conjuntos grandes de dados de modo a prover o melhor custo-benefício.

De acordo com Lam (LAM, 2011), as premissas básicas do *Hadoop* são:

- **Acessível**: *Hadoop* é executado em *clusters* de máquinas "*low-end*" ou em serviços de *cloud-computing*;
- **Robusto**: Como as aplicações serão executadas em máquinas sem muito valor, o *Hadoop* é desenvolvido com o pressuposto de falhas de *hardware* frequentes;
- **Escalável**: *Hadoop* é escalável linearmente para gerenciar grande volume de dados adicionando mais nós aos *clusters*.
- **Simples**: *Hadoop* permite aos usuários programar rapidamente e eficientemente códigos paralelos.

Em um *cluster* utilizando *Hadoop*, teremos máquinas em paralelo que armazena e processam grande volume de dados, e clientes que se conectam para enviar as requisições e trabalhos.

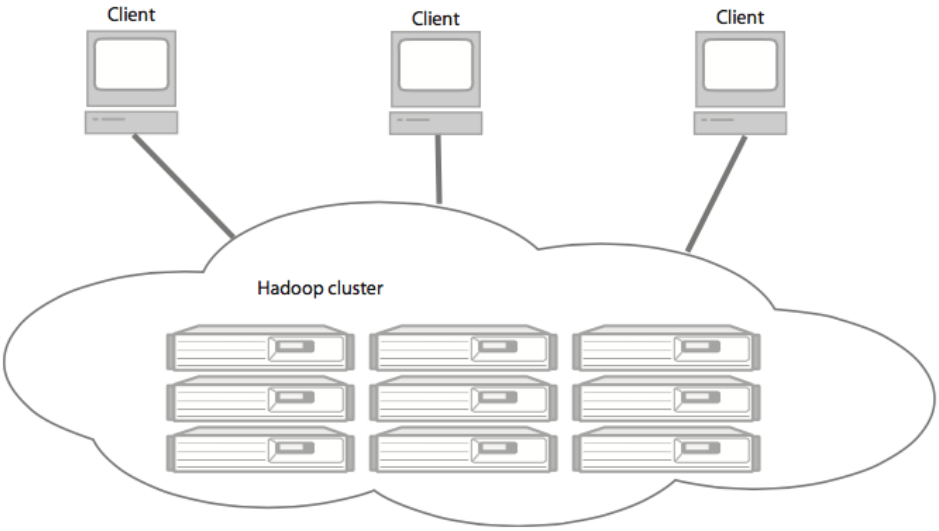

Fig. 2.6: Ilustração de um *Hadoop cluster* (LAM, 2011)

A Fig. 2.6 ilustra como é o ambiente do *Hadoop*. Ou seja, os computadores clientes enviam as requisições para a nuvem ou para o *cluster Hadoop* e, posteriormente, obtém os resultados. O processamento ocorre dentro desta nuvem e vários clientes podem submeter requisições, podendo estar até em locais diferentes.

Para armazenar os dados, o *Hadoop* utiliza seu próprio sistema de arquivos distribuído — *Hadoop Distributed File System* (HDFS). Tal sistema disponibiliza os dados para o processamento nos nós. Servidores no *cluster* do *Hadoop* podem falhar, no entando, o processamento de dados não é interrompido. O HDFS garante que os dados sejam replicados com redundância através do *cluster* (DUMBILL et al., 2012).

2.2.1 HDFS

O HDFS foi desenvolvido para armazenar grandes volumes de dados — *gigabytes*, *terabytes* ou *petabytes* com tolerância a falhas. Falhas de *hardware* são comuns, considerando que uma instância podem consistir de várias máquinas. A detecção das falhas e a rápida recuperação são os objetivos do núcleo da arquitetura do HDFS (APACHE, 2014).

O tamanho do bloco do HDFS é bem maior do que os blocos de sistemas de arquivo tradicionais. Um bloco é o mínimo de dados que o sistema pode ler ou escrever. Um sistema de arquivos com um simples disco que armazena seus dados por blocos, para obter sua eficiência deve manipular os dados com a finalidade que tenha o mesmo tamanho ou múltiplos do tamanho bloco. No entanto, isso é, geralmente, transparente para o usuário que está simplesmente lendo e gravando arquivo de qualquer tamanho.

De acordo com White (WHITE, 2012), no HDFS o bloco é de 64 *megabytes* por padrão. Similar aos sistemas de disco simples, os arquivos no HDFS são quebrados em porções do tamanho do bloco e são armazenados em unidades independentes. Comparados aos blocos de discos padrões, tais arquivos são grandes visando minimizar o custo de procura — "*seek-time*". Definindo o tamanho grande o suficiente, o tempo de transferência dos dados do disco pode ser significantemente maior que o tempo de procura para começar um novo bloco, mas o tempo de transferência de um arquivo grande que tenha múltiplos blocos, chamado de de *taxa de transferência do disco*. A ênfase da arquitetura HDFS é de prover uma alta taxa, ao invés de baixa latência de acesso aos dados, que será abordado na seção 2.2.2.

O HDFS foi otimizado baseado na ideia em que o padrão eficiente de dados é a de escrever uma vez, — *write-once* — e ler várias vezes, — *read-many-times* — chamado de *padrões de acesso via streaming* (WHITE, 2012). Não há restrições do que pode ser armazenado no sistema HDFS. Dados podem ser desestruturados, estruturados e semiestruturados, diferentemente dos bancos de dados relacionais que necessitam que os dados sejam estruturados e as informações desses dados definidos antes da armazenagem dos dados.

Em um *cluster* totalmente configurado, executando o *Hadoop*, deve haver programas residentes chamados de "*daemons*", executando nos diferentes nós da rede. Alguns existem em somente uma máquina e outros, em diversas máquinas (LAM, 2011).

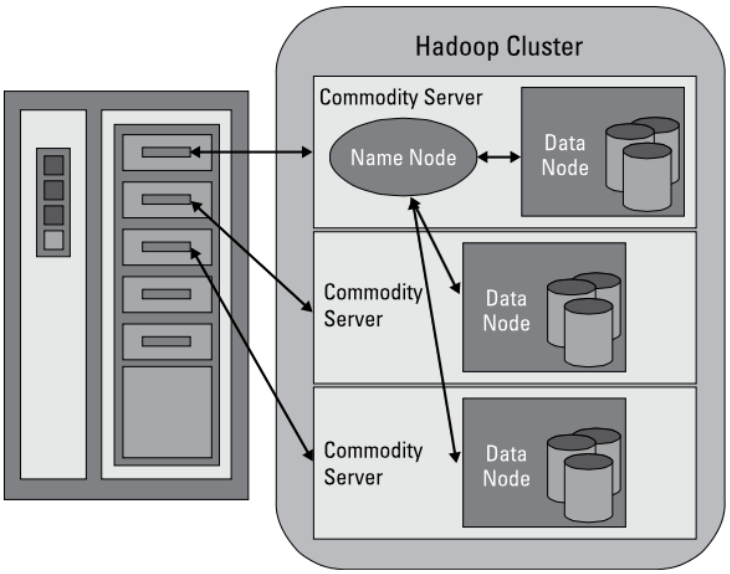

Fig. 2.7: *Daemons* relacionados com o HDFS (HURWITZ et al., 2013)

De acordo com a Fig. 2.7, no contexto do HDFS, existem dois tipos de *daemons*, operando no modelo *master-slave* ou *master-worker*: o "*NameNode*" e múltiplos "*DataNodes*":

- ***NameNode*** (master): local onde é mantido registro de onde os dados são fisicamente armazenado (HURWITZ et al., 2013). Isto é, mantém, de forma persistente no disco local, toda a árvore do sistema de arquivos e dos metadados de todos os arquivos e diretórios, bem como informações gerais da integridade do sistema de arquivos. É o programa residente mais vital para todo o ecossistema.
- ***DataNodes*** (workers): onde executam as solicitações do *NameNode* — leitura e escrita dos blocos HDFS para o sistema de arquivos locais. Reportam constantemente para o *NameNode* a lista de blocos que estão armazenando (WHITE, 2012).

Esses metadados são definidos como "dados sobre o dado", ou seja, provê informações adicionais dos blocos. O *NameNode* guarda também informações de qual *DataNode* estão armazenandos os blocos de um determinado arquivo. Porém, nesse caso, de acordo com White (WHITE, 2012), não são armazenados persistentemente, visto que a informação é reconstruída dos *DataNodes* quando o sistema é iniciado. Além disso, é no *NameNode* que é direcionado para os daemons workers (*DataNodes*) o pedido para executar tarefas de entrada e saída (E/S).

Normalmente o nó que "hospeda" o *NameNode* não armazena nenhum dado do usuário ou executa nenhum processo proveniente da tarefa de mapear e reduzir, visto que o *NameNode*

utiliza bastante memória e gerencia as entradas e saídas intensamente. Caso ocorrer a execução desses processos, pode haver a redução da carga de trabalho da máquina (*workload*).

Um ponto negativo abordado por Lam, é que o *NameNode* é um simples ponto de falha do *cluster Hadoop* (LAM, 2011). Ao contrário dos outros daemons, se algum nó tiver problemas por falhas de *software* ou *hardware*, o *cluster* continuará a funcionar ou será reiniciado rapidamente. No caso do *NameNode*, isso não ocorre visto que sem o mesmo, o sistema de arquivos não pode ser utilizado. Em fato, se a máquina que hospeda o *NameNode* for apagada, todos os arquivos no sistema de arquivos serão perdidos, já que não existe outra forma de reconstruir os arquivos a partir dos blocos nos *DataNodes* (WHITE, 2012).

Uma possível solução para esse problema com os *NameNode* é a utilização de um segundo *NameNode* chamado de *NameNode Secundário* (SNN). Esse *daemon* assistente é responsável por monitora o estado do *cluster*. Semelhante ao *NameNode*, é recomendado ser hospedado em uma máquina que não possui outros daemons sendo executados. O SNN difere do *NameNode* principal pelo fato que este processo não recebe ou grava nenhuma modificação em tempo real para o HDFS. Ao invés disso, comunica ao *NameNode* para gravar o estado dos metadados em intervalos definidos. Desde modo, os estados gravados minimizam o tempo inoperante e a perda de dados.

O ponto de falha citado acima não ocorre com os *DataNodes* porque o HDFS suporta a capacidade de criação de "*pipelines*". Um *pipeline* é uma conexão entre múltiplos *DataNodes* para a replicação dos dados buscando a redundância de dados, e será melhor abordado a seguir.

Cada *DataNode* envia, periodicamente, uma mensagem para o *NameNode* com intuito de avisar que está operacional. Essa mensagem é chamada de "*Heartbeat*". Como normalmente os *DataNodes* estão em diferentes máquinas, falhas na rede podem causar perda de conectividade com o *NameNode*. Nesse caso o *NameNode* detecta a falta dessas mensagens e sinaliza o *DataNode*. A partir disso o *NameNode* não encaminha mais nenhum pedido de E/S. Cada dado que estava registrado naquele *DataNode* não ficará mais disponível no sistema HDFS (VENNER, 2009).

Com a falha desse DataNode, outros do mesmo podem ficar congestionados. O HDFS tem a capacidade de rebalanceamento em que tem como objetivo balancear os nós de dados baseado no estado dos discos locais. O rebalanceamento pode ser considerado eficaz, mas não possui grande inteligência embutida. Como Hurwitz cita (HURWITZ et al., 2013), você não pode criar padrões de acessos e ter o rebalanceamento otimizado para estas condições.

O objetivo do *cluster Hadoop* é um processamento paralelo de dados. Para atingir isso é necessárias todas as máquinas disponíveis para trabalhar nos dados de uma vez. Na escrita de informações no HDFS, como citado anteriormente, o cliente quebra o arquivo de dados em vários blocos pequenos e insere-os em diferentes nós do *cluster*. Quanto mais blocos tiver, mais máquinas estarão disponíveis para trabalhar naquela informação em paralelo. Ao mesmo tempo,

essas máquinas podem estar dispostas a falhar, então para evitar perda de dados é informado, manualmente, na configuração do sistema de arquivos quantas replicações os blocos de arquivos terão. De acordo com (APACHE, 2014), a configuração padrão de replicação de blocos é definido como 3. Ou seja, existiram 3 cópias de cada bloco no *cluster*.

Para cada bloco do arquivo, o cliente consulta o *NameNode* e recebe a lista de *DataNodes* que devem ter a cópia deste bloco. Ou seja, o *NameNode* não faz parte do caminho do dado, visto que ao receber a lista, o cliente escreve diretamente o bloco no nó via *DataNode*, que por sua vez replica-o em outros *DataNodes*. O ciclo é repetido para os blocos restantes. O *NameNode* somente provê o "mapa" de onde a informação está e onde a informação deverá estar via metadados.

De acordo com a Apache (APACHE, 2014), para manter esse mapa o *NameNode* utiliza log transacionais chamado de "*EditLog*" que guarda persistentemente toda alteração que ocorre no metadado do sistema de arquivos. Esse log transacional é armazenado no sistema de arquivos local do sistema operacional (OS). Todo o ambiente do sistema de arquivos HDFS, incluindo o mapeamento dos blocos dos arquivos e propriedades do sistema é guardado em um arquivo chamado de "*FsImage*", que por sua vez também é mantido no sistema de arquivo local do OS.

Quando o *NameNode* é inicializado, os arquivos *FsImage* e *EditLog* são lidos do disco, são aplicadas todas as transações provenientes do *EditLog* para uma representação na memória da *FsImage*, armazenando uma nova versão dentro de uma nova *FsImage*. Nessa etapa, pode ser truncado o *EditLog* antigo, já que as transações foram aplicadas em uma nova *FsImage* persistente. Esse processo é chamado de "*checkpoint*". Nas versões atuais do *Hadoop*, esse tipo de processo é executado somente quando o *NameNode* é inicializado (VENNER, 2009).

Como foi apontado, o *NameNode* mantém a imagem de todo o ambiente do sistema de arquivos e o mapa de blocos de arquivos — "*blockmap*" — na memória. Por isso é importante a utilização de máquinas dedicadas com maiores memórias para *daemons* essenciais do sistema.

A escrita de informações nos *DataNodes* é feita com a armazenagem dos dados HDFS em arquivos, que por sua vez são armazenados em seu sistema de arquivos locais. Ainda de acordo com a Apache, os *DataNodes* não possuem conhecimento sobre os arquivos HDFS. Armazenam-os através de blocos em arquivos separados e em diretórios diferentes (utilizando funções heurísticas para determinar o número ideal de arquivos por diretório). Esse tipo de arquitetura é implementada por existir sistemas de arquivos que não suportam eficientemente números grandes de arquivos em um simples diretório. A Fig. 2.8 demonstra o processo de escrita de um dado no HDFS:

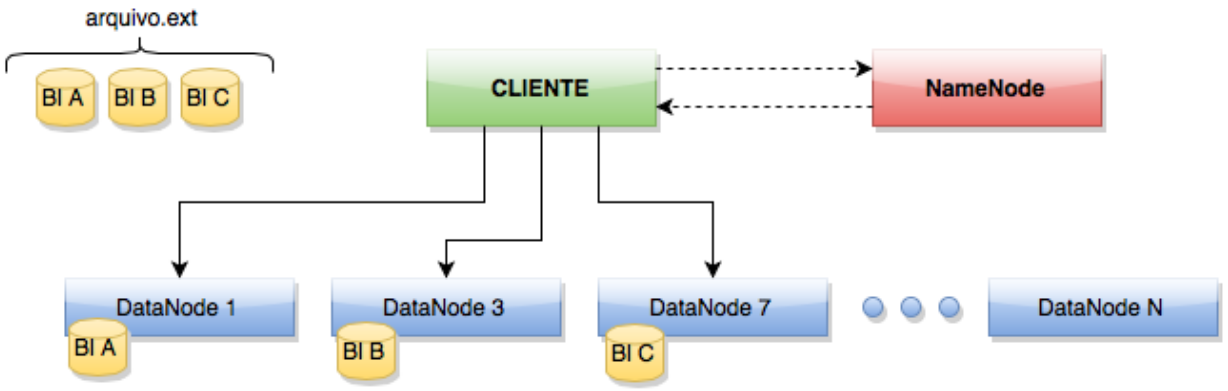

Fig. 2.8: Processo de escrita no HDFS

Essa escrita porém não é de forma totalmente paralela aos *DataNodes*, como possa ser entendido pela figura. Como já foi citado, o *Hadoop* implementa a escrita através de *pipeline*, ou seja, o cliente tem a função de escrever somente no primeiro *DataNode*, que por sua vez tem a responsabilidade de replicar o bloco para o próximo *DataNode* (APACHE, 2014). Isso significa que enquanto o *DataNode* estiver recebendo informações em blocos, ao mesmo tempo ele estará enviado uma cópia do dado para o próximo no *pipeline*.

Quando os *DataNodes* recebem com sucesso o bloco, cada um envia uma mensagem de sucesso ao *NameNode* e para o *DataNode* que lhe enviou o dado. O cliente então recebe uma mensagem de sucesso, com isso informa ao *NameNode* que o bloco foi escrito com sucesso, o qual, por sua vez atualiza o metadado referente aquele arquivo com a localização do bloco. O cliente assim está disponível para recomeçar o processo novamente com o próximo bloco de dados. Na gravação de pedaços de 64MB, o cliente divide esses pedaços em fragmentos menores de 4KB e envia-os ao primeiro *DataNode*, demonstrado na figura Fig. 2.9.

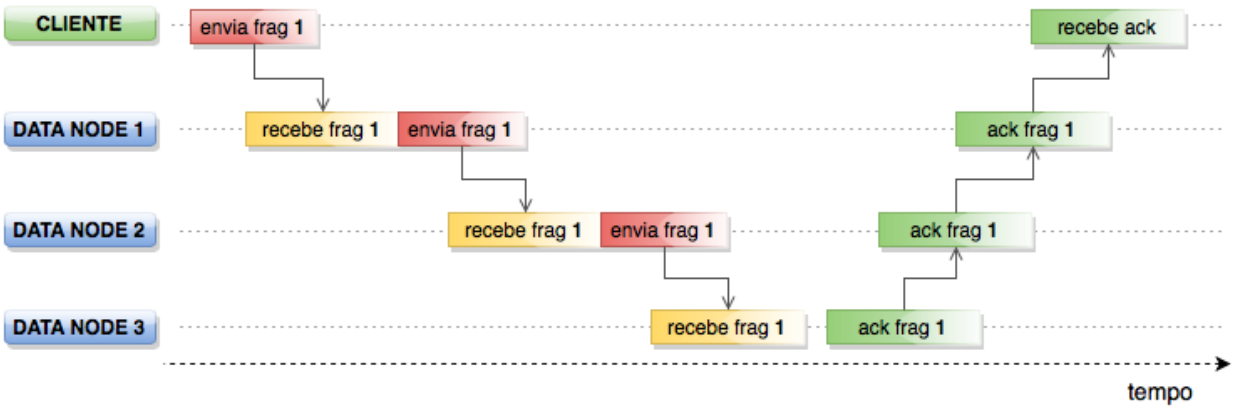

Fig. 2.9: Escrita no HDFS de um fragmento

Na figura, é mostrado o paralelismo de escrita de somente um *pipeline*. Porém, normalmente há diferentes *pipelines* para escrever diferentes fragmentos, visto que o *DataNode* pode

liberar memória do fragmento 2, receber o fragmento 3 do cliente e enviar a confirmação do fragmento 1 ao mesmo tempo. A Fig. 2.10 mostra como diferentes pipelines trabalham em paralelo:

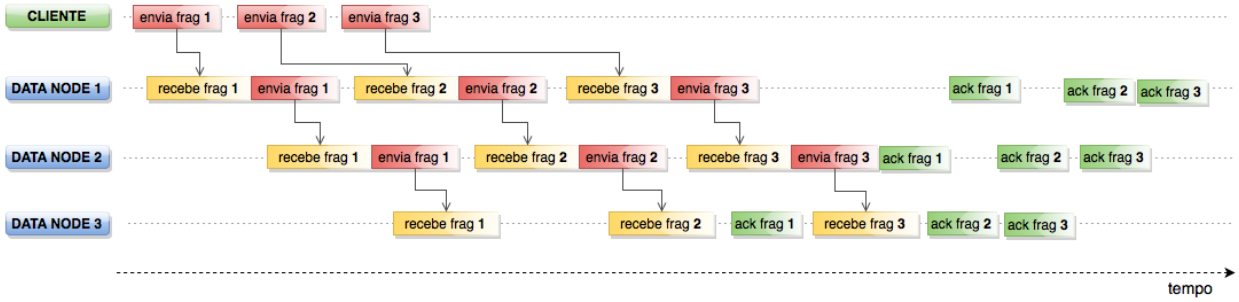

Fig. 2.10: Escrita no HDFS de vários fragmentos

Com o uso da escrita via pipeline, há um consumo mais eficiente da largura de banda para o cliente, já que não é preciso escrever três cópias em três diferentes *DataNodes*, e as operações são executadas concorrentemente. O consumo da banda é mais balanceado no uso do pipeline. Além disso, o cliente mantêm uma janela muito menor para gravar quais blocos que a réplica está sendo enviada e quais blocos estão aguardando confirmações de escrita com sucesso.

Muitas otimizações podem ser implementadas com o processamento em paralelo, como o uso do "*piggybacking*", que ao invés de enviar pacotes de confirmações ("*acknowledgement*") em quadros individuais, é enviado a confirmação no próprio quadro de dados.

Esta comunicação dos nós do *cluster* é feita através de protocolos HDFS, que estão no topo da pilha de protocolos de controle de transmissão sobre o protocolo de rede (TCP/IP). O cliente estabelece a conexão para uma porta TCP configurável na máquina que hospeda o *NameNode*, e os *DataNodes* conversam com o *NameNode* através de outro tipo de protocolo (APACHE, 2014).

2.2.2 Desvantagens no uso do HDFS

Como todo sistema possui desvantagens, no caso do HDFS não é diferente. É importante examinar as aplicações que não funcionam muito bem utilizando o HDFS. Ou seja, são características que o sistema de arquivo não se encaixa atualmente, e caso se encaixem irão ser relativamente ineficientes:

- **Latência no acesso aos dados**: a latência é um problema em qualquer aspecto da computação, incluindo comunicações, gerenciamento de dados, performance de sistemas e mais. As aplicações que necessitam de acesso aos dados com baixa latência não funcionam muito bem, visto que o HDFS foi otimizado para entregar uma grande volume de dados, mesmo que possa sacrificar a baixa-latência;

- **Arquivos pequenos**: para armazenar os arquivos, diretórios e blocos, o HDFS reserva cerca de 150 *bytes* de metadado de cada um na memória, ou seja o limite de arquivos que

o sistema de arquivos pode suportar é definido pelo tamanho da memória do *NameNode* (WHITE, 2012);

- **Múltiplos usuários escrevendo e modificações arbitrárias em arquivos**: arquivos no HDFS podem ser escritos por um simples usuário. As gravações são sempre feitas no final do arquivo, ou seja, não há suporte para múltiplos usuários gravando ou para modificações em locais arbitrários no arquivo.

2.2.3 Bancos de dados estruturados e o Hadoop

Nas aplicações atuais o *Hadoop* oferece um maior suporte a dados desestruturados. Por isso que as linguagens de busca estruturada — *structured query language* (SQL) — não são recomendadas para a utilização com o Hadoop. Elas são projetadas para dados estruturados, e nessa perspectiva, *Hadoop* oferece um paradigma mais genérico que o SQL. Se tratando de somente dados estruturados, SQL e *Hadoop* podem ser complementares, ou seja, nesse caso a linguagem pode ser implementada em cima do mecanismo de execução do *Hadoop* (LAM, 2011).

Nessas dimensões, podemos focar em alguns pontos chaves das diferenças entre bancos de dados estruturados e o *Hadoop*:

- **Escalabilidade horizontal invés de vertical**: bancos de dados relacionais são mais desenvolvidos para serem escalados verticalmente, ou seja, para executar um grande banco de dados é necessário de uma máquina com grandes recursos. Além de que escalar bancos de dados relacionais é caro e pode chegar a um ponto em que os recursos não serão grandes o bastante para o volume de dados. Uma máquina com o poder de processamento de 4 vezes o de uma máquina convencional é muito mais caro do que 4 máquinas convencionais em *cluster*. Adicionando mais recursos em uma escalabilidade horizontal significa adicionar mais nós ao *cluster*.

- **Pares chave/valor invés de tabelas relacionais**: o princípio de banco de dados relacionais é que as informações são armazenadas de forma estruturada dentro de tabelas, que por sua vez é organizada por *schemas*. Nesse cenário, muitas aplicações não se encaixam por se tratar de dados desestruturados ou semiestruturados, como documento de texto, imagens e arquivos XML. *Hadoop* usa a arquitetura de *pares chave/valor* como sua unidade básica de dados, que é flexível o bastante para trabalhar com os tipos de informações mais desestruturados. Ou seja, no *Hadoop* as informações podem ser originadas de várias formas, mas são transformadas em pares chave/valor.

- **Programação funcional invés de buscas declarativas**: *SQL* é basicamente uma linguagem declarativa, ou seja, há a consulta dos dados indicando o resultado e o banco de dados é o responsável de como irá fazer essa consulta e retornar. Já com o *Hadoop*, é utilizado o

MapReduce, que será melhor estudado a seguir, na seção 2.3. Basicamente, é especificado as etapas de processamento dos dados, ou seja você possui scripts e códigos, em contraste ao SQL onde temos instruções de consulta. Dessa forma conseguimos com o MapReduce processar dados de forma mais genérica.

- **Processamento off-line invés de transações on-line**: o *Hadoop* foi desenvolvido para trabalhar com processamento off-line e análise de grande volume de dados, ou seja, não foi otimizado para leituras aleatórias e escritas de poucos registros que temos no processamento de transações on-line. Como já foi dito, o melhor uso do Hadoop é quando o tipo de dado armazenado segue a premissa de gravação uma única vez e várias leituras.

Com o amadurecimento do *Hadoop*, foi sendo desenvolvido novos componentes e ferramentas para aumentar sua usabilidade e funcionalidade. Para melhor alcançar o processamento de dados foi necessário além de ferramentas, novas abordagens tecnológicas. *MapReduce* é uma dessas abordagens.

2.3 MAPREDUCE

MapReduce é um *software framework* criado pelo Google, que possibilita aos desenvolvedores escrever programas que possam processar grandes quantidades de dados desestruturados em paralelo diante vários grupos de processadores (DEAN; GHERMAWAT, 2008). Ou seja, a principal função é possibilitar a divisão e execução de pesquisas sobre um conjunto de dados, de modo que seja paralelo em diversos nós. De acordo com Hurwitz (HURWITZ et al., 2013), a distribuição do trabalho deve ser realizada em paralelo por 3 razões:

- O processo deve poder se expandir e contrair automaticamente;
- O processo deve continuar mesmo com falhas na rede ou em sistemas individuais;
- Desenvolvedores que utilizam o *MapReduce* devem ser capazes de desenvolver serviços que são fáceis para outros desenvolvedores, até porque esta abordagem deve ser independente de onde os dados estão e aonde que serão processados.

MapReduce em seu significado técnico, é um paradigma computacional que consiste em 2 passos principais: mapear e reduzir (*map* e *reduce*):

- **Função *Map***: O primeiro contato e transformação das informações, onde cada registro pode ser processado em paralelo;
- **Função *Reduce***: É o passo da agregação ou sumarização, em que todas as associações dos registros devem ser processados juntos por uma simples entidade (VENNER, 2009).

Os desenvolvedores entenderam a importância do reuso, por isso o mapeamento das informações se tornou o coração da tecnologia de processamento de dados. Basicamente, a função map utiliza-se dos documentos de entrada para mapear pares chave/valor. No mapeamento, é aplicado uma função em cada elemento de um conjunto de dados e tem como produto um novo conjunto sem modificar os dados originais. Esse novo conjunto é particionado e cada partição é agrupada e ordenada pela chave.

Já a função de redução, tem como entrada, a saída das funções de mapeamento, reduzindo o conjunto de dados. É executado uma vez para cada chave, em sequência ordenada, com o conjunto de valores que compartilham a mesma chave. No final do processo, a função retorna os valores baseado na tarefa que foi executada. Para Jason, Dean e Edward (RUTHERGLEN; WAMPLER; CAPRIOLO, 2012), o objetivo da redução é transformar o conjunto de dados em um valor, como por exemplo somar ou determinar a média de uma conjunto de números ou de outro conjunto. Um par final de chave-valor é retornado pela função de redução. Caso a implementação não precise da etapa de redução, não é necessário o desenvolvimento da mesma.

O conceito central do MapReduce é a possibilidade da utilização de algoritmos capazes de processar grandes quantidade de dados, já que as operações são independentes (HURWITZ et al., 2013).

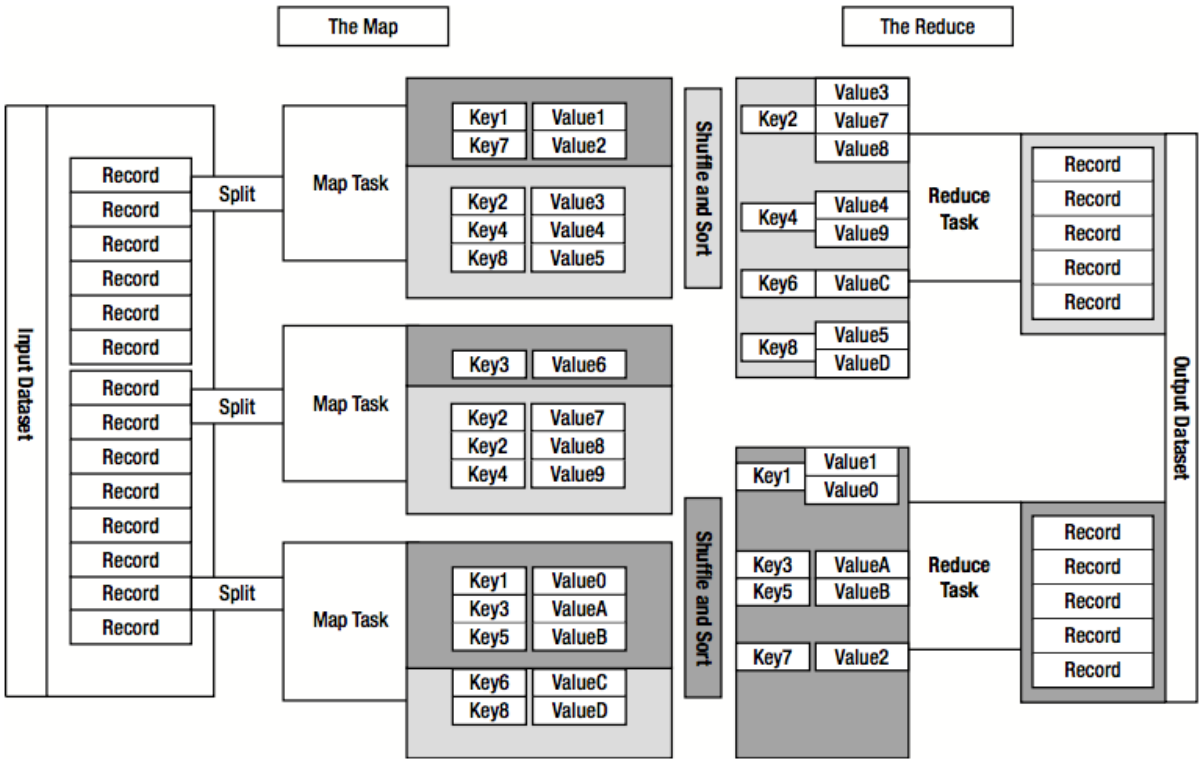

Fig. 2.11: O modelo MapReduce (VENNER, 2009)

Como podemos analisar na Fig. 2.11, as tarefas de map (*map task*) e reduce (*reduce task*) são separadas. Os dados são divididos em pedaços menores, processados independentemente, e recolocados juntos no final. A função map pode ser executada sem uma ordem e em qualquer nó do *cluster*.

De acordo com Hurwitz (HURWITZ et al., 2013), alguns comportamentos do MapReduce devem ser analisadas com maior profundidade. A função de redução só pode ser iniciada quando toda a tarefa de mapeamento for completada. Ou seja, essas tarefas são priorizadas de acordo com o número de nós no *cluster*. Se possuir mais tarefas do que nós, o *framework* irá gerenciar as map tasks até completarem. Posteriormente as tarefas de redução irão ser gerenciadas da mesma forma até o processo terminar — quando as a execução de redução executar com sucesso. O mesmo é citado por Jason, Dean e Edward (RUTHERGLEN; WAMPLER; CAPRIOLO, 2012), o *Hadoop* determina como decompor o trabalho em diferentes e individuais tarefas de mapeamento e redução. É gerenciado e executado de acordo com os recursos disponíveis. Ou seja, é decidido para onde enviar uma tarefa específica no *cluster*.

Para que os processos sejam executados simultaneamente, é necessário que haja mecanismos de sincronização. O *framework* entende em qual função está e guarda as informações do que está sendo executado e quando. Quando toda a função de mapeamento é finalizada, a redução é iniciada, enquanto os dados são copiados através da rede e ordenados. Para que um processamento eficiente ocorra, o código de mapeamento é relocado para os diversos nós em que os dados serão processados. O MapReduce oferece 2 processos que podem gerenciar os trabalhos:

- ***JobTracker*** (master): faz a gerência das tarefas, ou seja, provê controle e monitoramento do trabalho e a coordenação na distribuição das tarefas para os nós que executam o TaskTracker;
- ***TaskTracker*** (workers): gerencia a execução individual das funções map e reduce em um nó de um *cluster*.

Geralmente existe um processo *JobTracker* em um *cluster*, normalmente no nó *master*, e vários *TaskTracker* nos diversos nós *slaves*. Como no processo *DataNode* visto na seção 2.2, Venner (VENNER, 2009) cita que o processo *JobTracker* é um ponto de falha crítico, diferentemente do *TaskTracker*. A maioria das implementações do *MapReduce* possuem uma manipulação de erros robusta e tolerância a falhas. O *framework* é capaz de reconhecer alguma falha e criar a correção necessária (HURWITZ et al., 2013).

Uma vantagens de possuir esta arquitetura de processos é evidenciada por Venner (VENNER, 2009). O autor cita que pode-se adicionar novos nós *TaskTracker* no *cluster* enquanto uma tarefa está sendo executada e o trabalho será dividido para essas novas máquinas.

Uma arquitetura baseada em *racks* é ideal para a construção de um *cluster Hadoop*, com cada *daemon* essencial em um diferente *rack*.

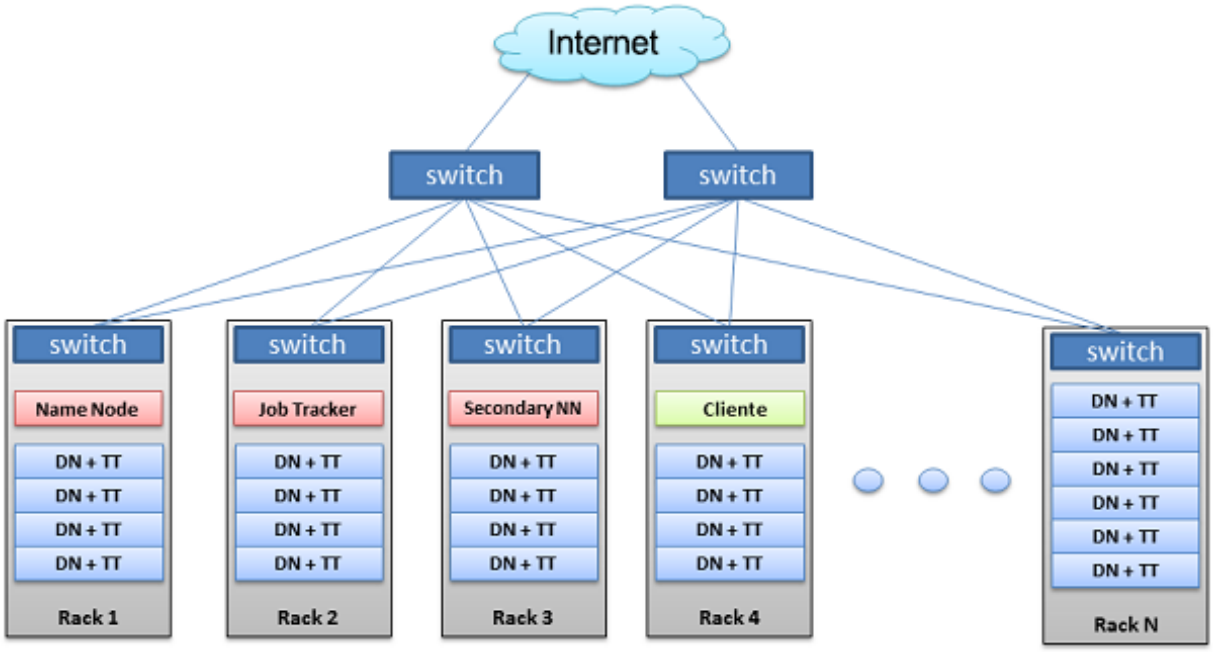

Fig. 2.12: *cluster* ideal: uso de *racks* com *daemons* independentes

A figura Fig. 2.12 demonstra esta arquitetura. Cada *rack* possui um *daemon* vital (*NameNode*, *JobTracker* ou *NameNode* Secundário) ou o cliente independente, e seus *DataNodes/TaskTrackers* para armazenamento e processamento. Além disso, cada *rack* é interligado à 2 *switches*, criando assim redundância caso um *switch* venha a falhar. A maioria dos servidores são nós escravos, com muito espaço em disco e poder de processamento e memória moderados. Algumas máquinas serão nós principais (master) que podem ter uma configuração diferente favorecendo memória e unidades de processamento (CPU), com menos espaço em disco.

Assim como acontece na comunicação entre cliente com o *NameNode* do HDFS, já mencionado, o mesmo é implementado no *MapReduce*. Uma porta TCP é definida para o *JobTracker*, que irá gerenciar os trabalhos.

2.4 OBTENÇÃO E PROCESSAMENTO DA INFORMAÇÃO

Existem diversas formas de integrar os dados ao *Hadoop*. As informações são carregadas no HDFS, processadas e então recuperada. Ferramentas tradicionais de integração não são rápidas suficientes para mover grandes quantidades de dados em tempo para entregar resultados para análises em tempo real. As ferramentas atuais que mais são utilizadas no *Hadoop* para obtenção de dados são:

- **Sqoop**: ferramenta desenvolvida para importar dados de banco de dados relacionais para o *Hadoop*;

- ***Flume***: ferramenta desenvolvida para importar fluxos de *streaming* de dados diretamente no HDFS, será abordado a seguir.

Na essência do *Hadoop* os dados são inseridos no sistema de arquivos HDFS, processados e então consumidos. Mais do que escrever aplicações para mover dados para o HDFS, existem ferramentas que conseguem obter os dados e inserir neste sistema.

No escopo deste trabalho daremos ênfase ao *Apache Flume*, visto que será o único método que será utilizado na obtenção dos dados.

2.4.1 Apache Flume

Apache Flume é uma ferramenta para obter e inserir grande quantidade de fluxo de dados para o HDFS. Uma forma comum utilizada é de coletar dados de log de um sistema ou um grupo de nós, e agregar ao HDFS para uma análise posterior.

O *Flume* foi desenvolvido para prover escalabilidade, ou seja, pode ser adicionado mais recursos ao sistema que a ferramenta irá continuar a gerenciar a grande quantidade de dados de forma eficiente (HURWITZ et al., 2013). Além disso, suporta uma grande variedade de fontes de dados que podem ser integrados, como por exemplo o "*tail*" — um túnel de informação para que os dados de arquivos locais sejam consumidos, syslog e o Apache log4j que permite aplicações Java escrever eventos para arquivos no HDFS via *Flume*.

O conceito de agentes é abordado no *Flume* e demonstrado na Fig. 2.13:

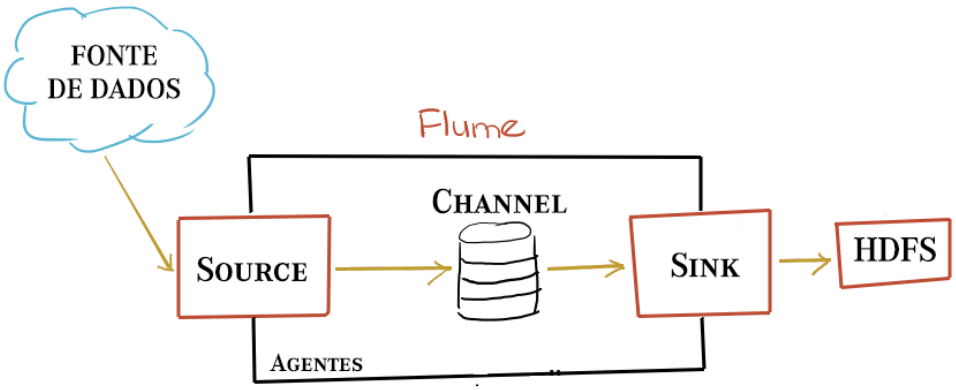

Fig. 2.13: Funcionamento do *Flume* (SRIPATI, 2013, modificado)

As fontes, coletores e canais intermediários são tipos diferentes de agentes. As fontes podem capturar e transferir os dados para diferentes canais, que por sua vez enviam as informações para coletores diferentes. *Flume* já vem com um conjunto de fontes, canais e coletores. Novos agentes podem ser implementados estendendo a base de classes do *Flume*.

Neste trabalho será utilizado o *Flume* como ferramenta principal para obtenção dos dados provenientes do *Twitter*. A implementação da ferramenta e como a captura será executada é abordado em 3.

2.4.2 Processamento

Para o processamento de dados como foi mencionado é utilizado o *MapReduce*, porém o cliente deve iniciar o trabalho a partir de algum script de mapeamento e redução. Um trabalho do *Hadoop — Hadoop job —* é enviado pelo cliente em forma de executável (normalmente arquivos compilados java utilizando a "*Java MapReduce API*") e a configuração enviada para o gerenciador de recursos ("*ResourceManager*"), que assume a responsabilidade da distribuição do executável e da configuração para os nós escravos, agendando e monitorando-os para prover informações de status e de diagnóstico de como está o andamento do trabalho enviado.

Usualmente são executados ferramentas acima da camada do *MapReduce* que transcrevem as operações desejadas para scripts de mapeamento e redução. Uma dessas ferramentas é o *Apache Hive* que provê um ambiente para efetuar consultas direto nos arquivos armazenados no HDFS.

Hive oferece um rico conjunto de ferramentas em múltiplas linguagens para executar análises direto no sistema de arquivos HDFS (VENNER, 2009). Hive é amplamente utilizado por oferecer suporte a consultas com linguagem próxima ao SQL, chamada de *Hive Query Language* (HiveQL), sem que haja o contato com funções de mapeamento e redução. Ou seja, com o Hive temos acesso a dados estruturados de forma SQL e uma análise sofisticada de *Big Data* com o *MapReduce* (HURWITZ et al., 2013).

Porém, o Hive não foi desenvolvido para respostas rápidas das consultas. Dependendo da complexidade pode demorar minutos ou até mesmo horas. Portanto o Hive não é aconselhado para consultas que requerem comportamentos em tempo real. É usualmente utilizado para mineração de dados e análises profundas off-line (PROKOPP, 2013).

Um outro meio de iniciar aplicações de MapReduce é a utilização de uma outra API, chamada de "*Hadoop Streaming*", que oferece aos usuários a realização de jobs com qualquer executável (como por exemplo utilitários shell, scripts c, python, php...) como funções de mapeamento e redução (APACHE, 2014).

A utilização do *Hadoop Streaming* se da com a leitura da entrada padrão do sistema operacional ("*stdin*"), linha por linha, pelos executáveis *mapper* e *reducer*. Também é emitido as respostas de cada um na saída padrão ("stdout"). Ou seja, este utilitário permite a construção de funções flexíveis em relação a linguagem.

A diferença entre a *Java MapReduce* API com o *Hadoop Streaming* se dá principalmente na implementação do processamento. Na tradicional Java API, o mecanismo é de processar cada

registro, um por vez. Ou seja, é implementado para executar o método de mapeamento para cada registro. Mas com a Streaming API, a função de map pode controlar o processamento da entrada de dados. Também pode ler e processar múltiplas linhas ao mesmo tempo. Essa leitura também pode ser implementada no Java API, porém com ajuda de outro mecanismo como a utilização de variáveis de instanciação para acumular múltiplos registros e então processá-los (APACHE, 2014).

3 AMBIENTE DE CAPTURA

3.1 CAPTURA E ARMAZENAMENTO DOS DADOS

No escopo deste trabalho iremos montar um ambiente de captura e processamento dos dados para segmentar o mapeamento sentimental dos clientes das operadoras de telecomunicações. Para isto utilizaremos a rede social *Twitter* e ferramentas de captura de dados em tempo real para obter as informações.

3.1.1 Twitter Streaming API

Tweets são a essência do *Twitter*, e enquanto for disponibilizado 140 caracteres de texto associado a atualização do usuário, haverá algum metadado incluído. O *Twitter* possui 2 tipos de interface de programação para aplicação — *Application Programming Interface* (API) — que provê acesso aos *tweets* para o público:

- ***Representational State Transfer* (RESTful) API**: provê acesso programática de leitura e escrita de dados do *Twitter*, utilizado para conduzir buscas simples;
- ***Streaming* API**: provê monitoração ou processamento de *tweets* em tempo real.

Existem várias bibliotecas de inúmeras linguagens de programação que implementam a RESTful API, porém como a intenção desse trabalho é a obtenção do fluxo em tempo real, iremos utilizar a *Streaming* API.

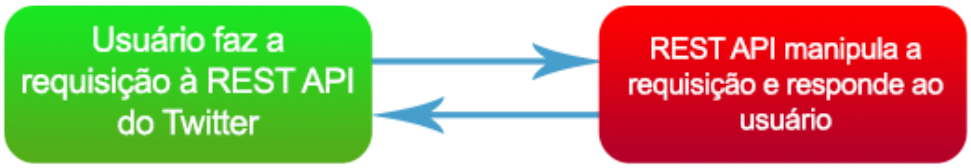

Fig. 3.1: Funcionamento da RESTful API (TWITTER, 2015, modificado)

A *Streaming* API disponibiliza acesso ao fluxo de dados de *tweets* do *Twitter* com baixa latência. Uma implementação típica do uso desta API é do cliente ser notificado por mensagens indicando *tweets* ou outros eventos ocorridos sem que exista sobrecarga associado a requisição de um destino final (endpoint) de um RESTful API.

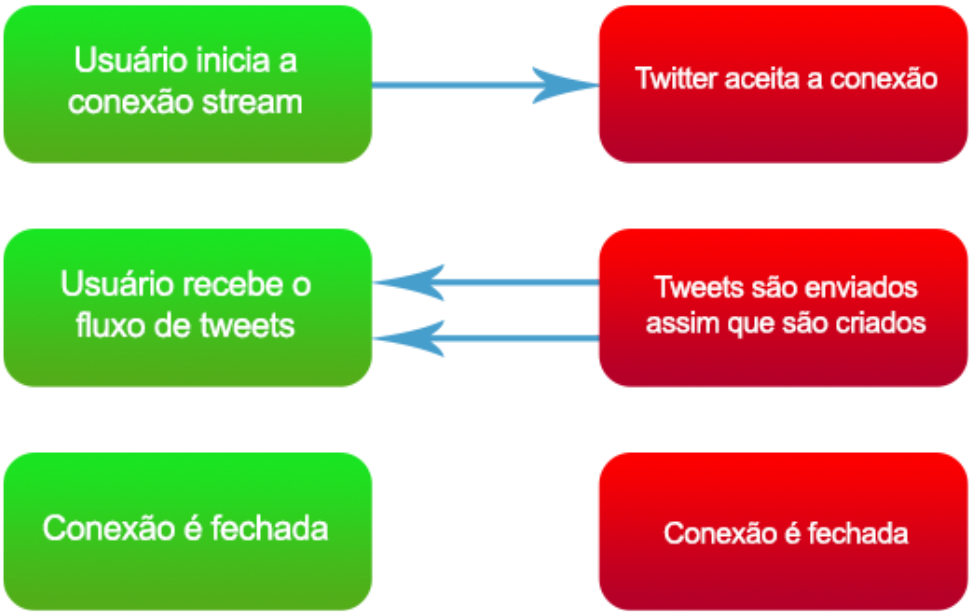

Fig. 3.2: Funcionamento da Streaming API (TWITTER, 2015, modificado)

O *Twitter* disponibiliza 1% de todos os *tweets* disponíveis em tempo real através de uma técnica de amostragem aleatória que representa a uma determinada quantidade de *tweets*. Como o intuito deste trabalho é mapear o sentimento dos clientes com as operadoras de telecomunicação, iremos fazer o uso de somente esses 1%. Portanto, o *Streaming* API provê uma forma de obter informações próximo ao tempo real.

O *Twitter* disponibiliza 3 endpoints para a *Streaming* API:

- **Fluxo público (*Public Streams*)**: indicado para seguir usuários ou tópicos específicos e mineração de dados;
- **Fluxo de usuário (*User Stream*)**: contém todos as informações e dados correspondente de um único usuário do *Twitter*;
- **Fluxo de site (*Site Streams*)**: é a versão multiusuário do fluxo de usuário, ou seja, é indicado para serviços que necessitam conectar ao *Twitter* por múltiplos usuários.

Utilizaremos o fluxo público, onde irá prover os dados em tempo real. Para a conexão entre a Streaming API do *Twitter* e o sistema de arquivos do *Hadoop* iremos utilizar o *Apache Flume*, já visto na seção 2.4.1.

A resposta das APIs do *Twitter* retorna as informações em notação de objeto javascript, — *javascript Object Notation* (JSON) — com todos os dados relativos daquela informação. É um formato leve e amplamente utilizado em APIs de diversos serviços na WEB, especialmente quando há o interesse de obter uma alternativa ao formato de linguagem de marcação extensível — Extensible Markup Language (XML).

```
{
  "statuses": [
    {
      "favorited":false,
      "created_at":"Mon Sep 24 03:35:21 +0000
      2012",
      "id_str":"250075927172759552",
      "in_reply_to_user_id_str":null,
      "contributors":null,
      "text":"@gvtoficial minha internet não esta
      funcionando. o que devo fazer?",
      "metadata":{…},
      "retweet_count":0,
      "in_reply_to_status_id_str":null,
      "id":250075927172759552,
      "geo":null,
      "in_reply_to_user_id":null,
      "user":{
        "id":137238150,
        "screen_name":"@brnmonteirop"
      },
      "source":"<a>Twitter for Mac</a>",
    },
```


Fig. 3.3: Exemplo de resposta das requisições para a API

Um exemplo de uma resposta pode ser visto na Fig. 3.3. Com esse tipo de formato conseguimos, via programação, analisar ("*parse*") e obter informações específicas. Neste projeto utilizaremos principalmente as informações provenientes do campo "*text*" e "*created_at*".

3.1.2 Captura de dados

Utilizando o *Apache Flume* e a *Streaming* API do *Twitter* conseguimos capturar os dados e inseri-los no sistema de arquivo. A Fig. 3.4 ilustra bem como será o funcionamento da coleta de informação do *Twitter*:

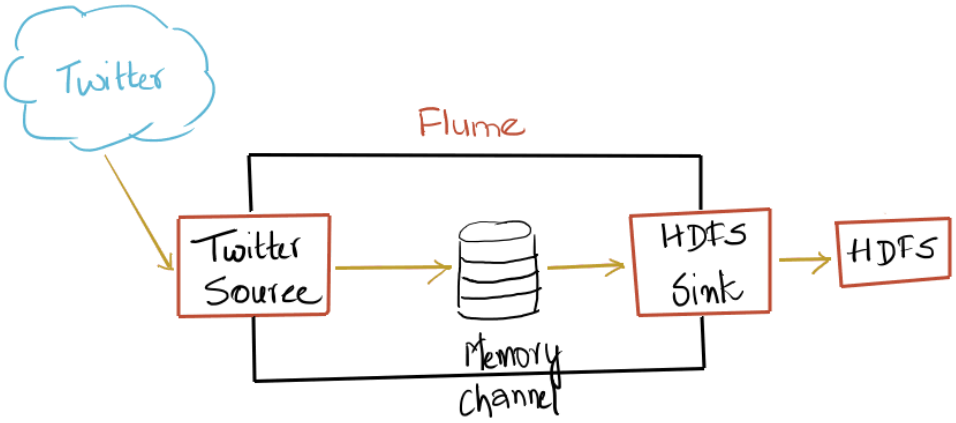

Fig. 3.4: Captura de *tweets* pelo *Flume* (SRIPATI, 2013)

Através do agente fonte (*Twitter Source*) os *tweets* serão recebidos, e então transferidos para a canal da memória (*Memory Channel*) do sistema até o tamanho do arquivo ser suportado pelo sistema de arquivo. Após atingir o tamanho mínimo, as informações dos *tweets* em formato JSON são concatenados e criado um único arquivo. Este arquivo é transferido para o coletor HDFS (HDFS *Sink*), que envia os dados para o HDFS propriamente dito. O sistema permanece neste loop até que haja uma interrupção.

Como o intuito é obter as informações das operadoras de telecomunicação do Brasil, iremos filtrar a busca dos *tweets* para palavras chaves que contenham o nome do usuário que as operadoras utilizam para responder os *tweets* dos clientes que entram em contato. Esses nomes de usuários são identificados na tabela 3.1.

Operadora	**Usuários no *Twitter***
1. Claro	@clarobrasil
2. GVT	@gvtoficial & @gvt_suporte
3. NET	@netoficial & @netatende
4. Oi	@digaoi
5. Tim Brasil	@timbrasil
6. Vivo	@vivoemrede

Tabela 3.1: Lista das principais operadoras de telecomunicação do Brasil com seus respectivos nomes de usuários no *Twitter*.

A partir da identificação dos nomes dos usuários, conseguimos obter os *tweets* que contenham essas palavras e inserir os arquivos no HDFS.

Como dito na seção 2.2, caso o ambiente possua mais de um nó em *cluster*, esses arquivos são replicados e armazenados nos diversos nós para fins de redundância.

3.2 PROCESSAMENTO DOS DADOS

Para o processamento dos dados é necessário que os arquivos já estejam no sistema de arquivos do *Hadoop* (HDFS). Com isso teremos o recurso principal para conduzir o estudo para segmentação das informações.

Os dados coletados, como já abordado, são em formato JSON e com várias informações que não entra no escopo deste trabalho. A principal informação que irá ser feito o estudo será o próprio tweet de 140 caracteres, onde o filtro será aplicado.

3.2.1 Estudo da segmentação

Para a segmentação, iremos filtrar os *tweets* já capturados para que se encaixem em algum destino. Caso contrário o tweet é desprezado, porém continuando no sistema de arquivo para futuras análises.

Fig. 3.5: Segmentação do tweet

Como a Fig. 3.5 ilustra, o tweet primeiramente será segmentado por operadora, e depois pelo tipo de serviço.

O estudo de como é feito as segmentações é baseado em amostragens iniciais de *tweets* em diversos momentos e retirado as palavras chaves que mais se relacionam ao determinado filtro.

Os tipos de serviços oferecidos pelas operadoras serão filtrados da seguinte forma:

Serviços	**Palavras chaves**
Telefonia	telefonia, fixo, celular, tel, cel, sinal movel, sinal de celular
Internet	internet, inet, conexao, net, 3g, sinal de internet, virtua, 4g
Atendimento	atendimento, call center, cancelamento, contato
TV	tv, televisao
Marketing	loja, oferta

Tabela 3.2: Serviços que serão segmentado com as palavras chaves procuradas

Neste caso, quanto mais palavras chaves inserirmos nessa tabela, mais *tweets* iremos obter. Como os *tweets* em forma original estarão no sistema de arquivos, podemos alterar a tabela, adicionando ou removendo palavras chaves para aumentar a precisão.

Para a descoberta do teor negativo do *tweet*, analisamos uma grande quantidade de *tweets*

e retiramos as palavras negativas com maior frequência: “ruim”, “cancelar”, “cancelem”, “horrivel”:

Satisfação	Palavras chaves
Negativa	ruim, cancelar, cancelem, horrivel, manutencao, resolvam, lenta, instavel, lerdo, lerda, nao funciona, injusto, ligacao cai, sem sistema, nao consigo, parou, complicado, tentando, persiste, decepciona, corta, dificil, sinal cai, tv trava, pior, caiu, absurdo, falha, lixo, sem internet, sem telefone, sem net, sem servico, espera, limite, vergonha, problema, anatel, chateacao, desprezo, cortar, indisponivel, nunca atende, descaso, sem atendimento, fora do ar & gírias + palavras de baixo calão.

Tabela 3.3: Palavras chaves que serão procuradas para segmentar por satisfação do cliente

A partir disso conseguimos extrair de que operadora o *tweet* está relacionado, de qual serviço daquela operadora e se possui informações negativas. *tweets* que não entrarem nessas segmentações não serão processados, porém ficarão no sistema de arquivos para futuras modificações nas palavras chaves, e aprimoramento do algoritmo desenvolvido.

3.2.2 Mapear e reduzir

Com o *cluster Hadoop* já montado, os arquivos de *tweets* em sua forma original no sistema de arquivos HDFS, o próximo passo é montar o algoritmo para segmentar, mapear os *tweets* e reduzir.

O estudo da segmentação foi vista na seção anterior, e é inserida no conceito de mapeamento. Ou seja, o *tweet* é analisado, segmentado e as informações pertinentes são exportadas para o processo de redução.

O resultado do mapeamento da informação do *tweet* é visto na Fig. 3.6, que iremos chamar de “*token*”.

timestamp da hora do tweet - operadora - serviço - satisfação

Fig. 3.6: Resultado do mapeamento

Ou seja, com o algoritmo de mapeamento será possível extrair de cada *tweet* essas informações resumidas, de acordo com nosso interesse.

Com esses tokens sendo exportados para o algoritmo de redução, é feita a contagem dos mesmos e o resultado exportado para o banco de dados relacional para futuras análises do resumo, resumido na Fig. 3.7.

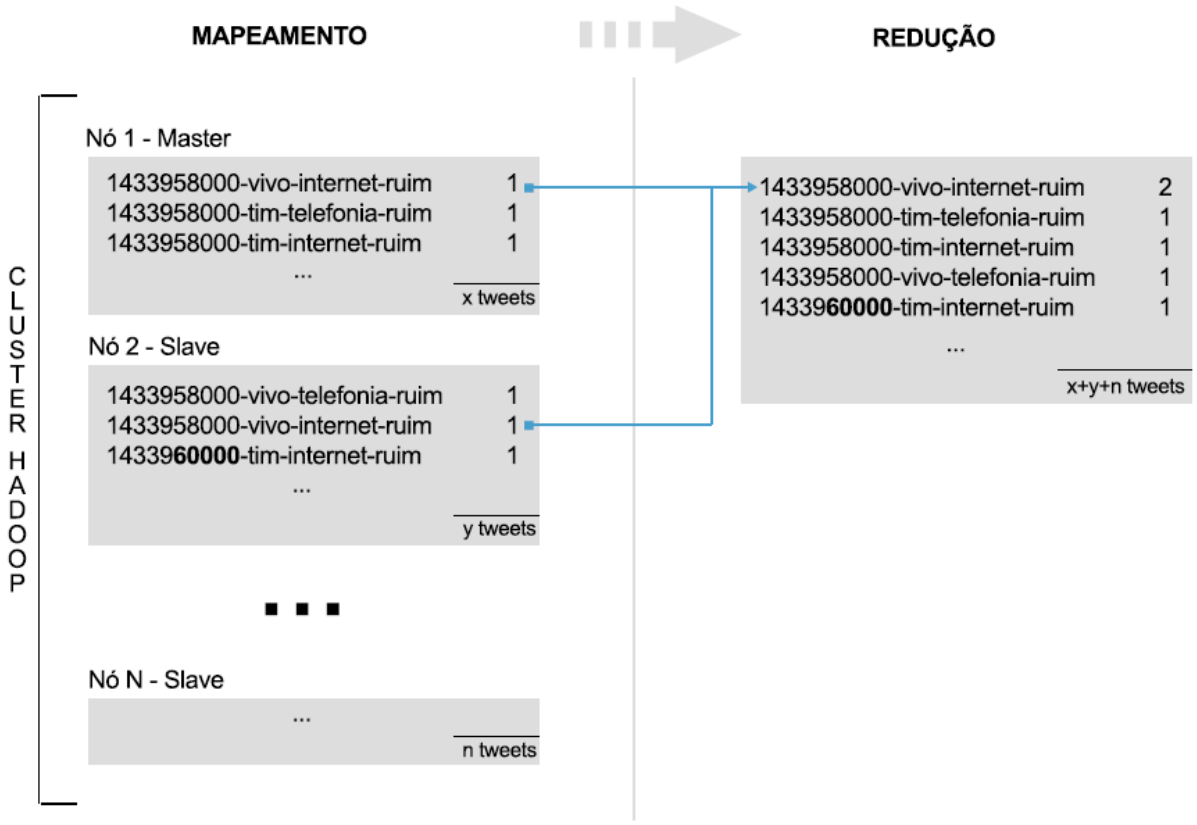

Fig. 3.7: Exemplo do mapeamento e redução dos tokens

O apanhado será inserido no banco de dados a cada 10 minutos, ou seja, o job do *Hadoop* será executado nesta janela temporal. Porém o *Flume* executará em tempo contínuo, recebendo os *tweets* em tempo real. Esta janela foi definida pela previsão de dados que seria capturado e realmente processado. Somente o processamento da informação pode levar muitos minutos. Foi estabelecido a janela em 10 minutos para que as funções de mapeamento e redução não interfira no próximo processo.

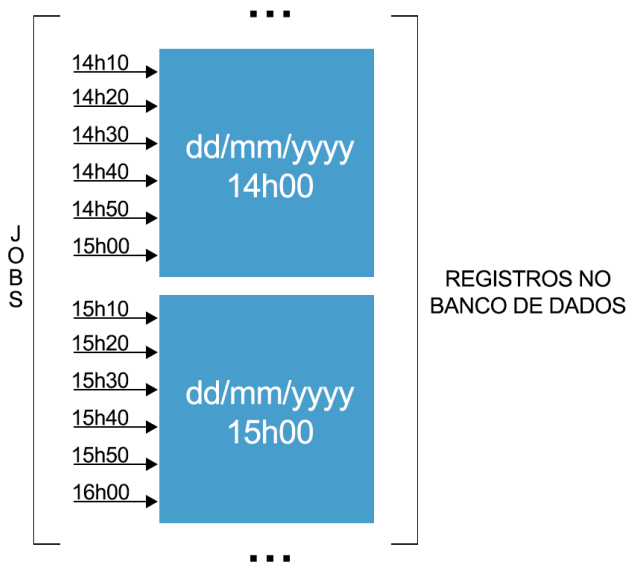

Fig. 3.8: Mapeamento das execução do *Hadoop* em registros no banco de dados

O *timestamp* dos *tweets* será resumido somente a data e a hora do dado, ou seja, caso o tweet tenha sido criado no dia 05/06/2015 às 10h40, o *timestamp* será formatado para às 10h. Portanto, os tokens serão baseado nas horas, como explicado na Fig. 3.8.

3.2.3 Banco de dados SQL

Para armazenar a síntese das informações reduzidas (data, operadora e serviço), utilizaremos um banco de dados relacional SQL.

Como os *tokens* serão baseado nas horas e temos 6 operadoras com 5 serviços, por dia poderemos ter no máximo 720 registros (24x6x5). Cada operadora pode ter *tweet* de 5 serviços a cada hora.

Campos
data *
operadora *
servico *
qntTweets
positivo

Tabela 3.4: Estrutura da tabela SQL para armazenar os dados resumo

A tabela 3.4 mostra os campos que serão utilizados para a construção da tabela no bando de dados MySQL. Os campos assinalados com * são os campos primários, onde serão únicos. Ou seja, a cada hora do dia uma operadora pode ter somente 1 registro de cada serviço.

A cada execução do trabalho de mapeamento e redução, será inserido o registro. Como a execução será com uma janela de 10 minutos, e os resumos são com uma janela de 1h, o

campo “qntTweets” será incrementado com a quantidade de *tweets* segmentados, caso o registro da data/operadora/serviço já possuir.

3.3 MODELAGEM DO PROCESSO

A Fig. 3.9 mostra a modelagem do processo da captura dos *tweets* até a inclusão no banco relacional SQL:

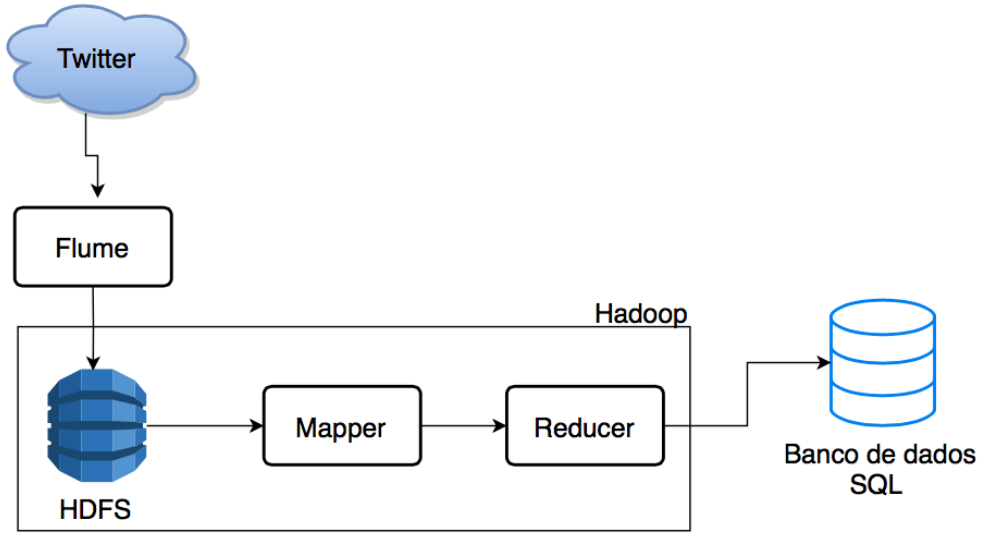

Fig. 3.9: Modelagem do processo

O *Twitter* disponibiliza os *tweets* em tempo real ao *Apache Flume*, que agrega os *tweets* em formato JSON e insere-os no sistema de arquivos do *Hadoop* (HDFS). Para o consumo e análise dos *tweets*, é executado a função de mapeamento, e em seguida a de redução.

Como o intuito deste trabalho é inserir o resumo das informações em um banco de dados relacional SQL, não é necessário a interligação da função de redução com o sistema de arquivos HDFS. Ou seja, no final do processo não é gravado nada além dos logs e arquivos temporários no sistema de arquivo. O próprio algoritmo da função de redução é responsável pela inserção no banco de dados.

Na Fig. 3.10 podemos observar a modelagem das ações executadas pelo projeto resumidamente.

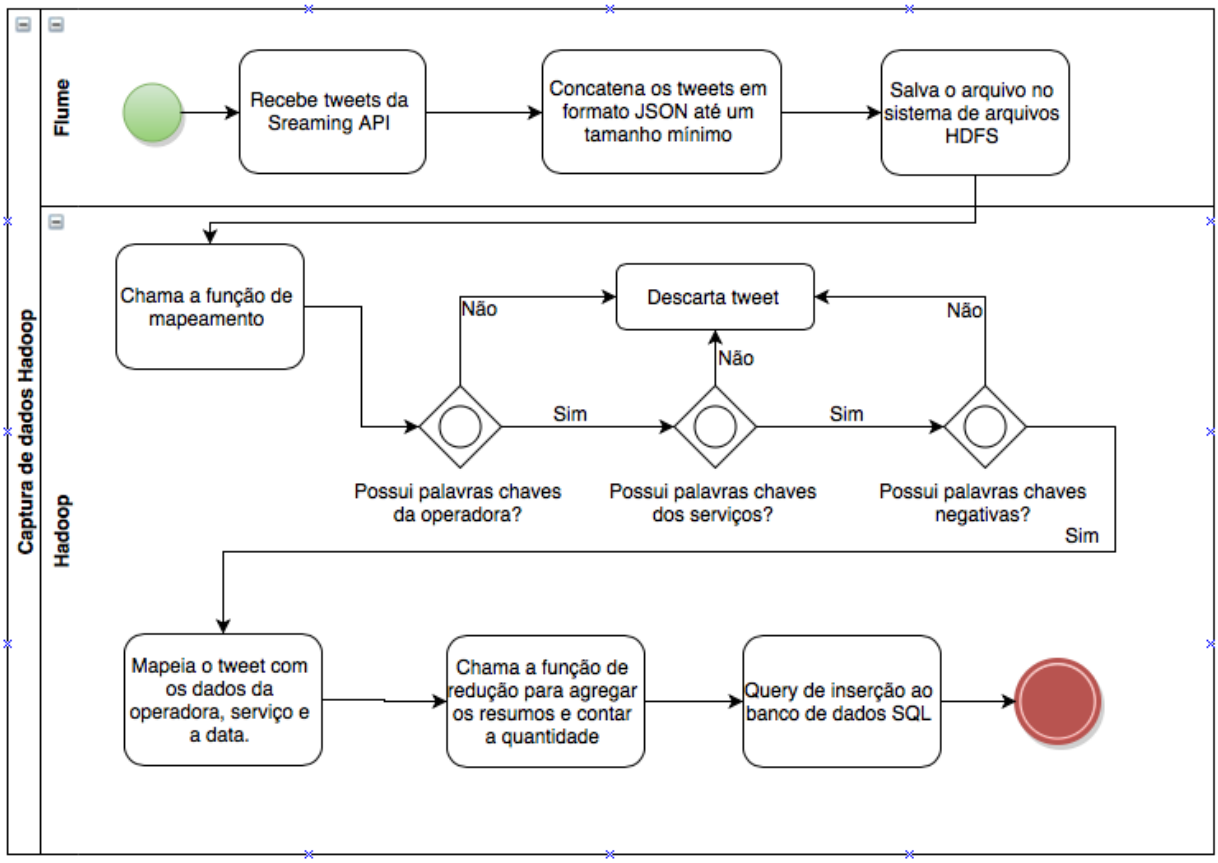

Fig. 3.10: Modelagem do processo das ações

3.4 RESULTADOS ESPERADOS

Com a captura de dados obteremos os *tweets* em sua forma bruta e sem tratamento, porém em formato JSON o que torna simples a análise e a extração de informação. Com a manipulação e segmentação temos os tokens com o resumo dos *tweets* e a quantidade de cada data/operadora/serviço.

A partir disso temos como resultado um banco de dados de fácil consulta de dados resumidos por hora, além dos *tweets* capturados que estarão armazenados no sistema de arquivos para qualquer outra consulta ou implementação. Através do Hive — visto em 2.4.2 — podemos fazer consultas SQL direto no sistema de arquivos, nos *tweets* brutos capturados.

Através desse banco de dados abre-se uma gama de possibilidades de análises e implementações para visualização dos dados.

4 IMPLEMENTAÇÃO

4.1 IMPLEMENTAÇÃO A NÍVEL DE HARDWARE

Um dos objetivos deste projeto é a construção de um ambiente de baixo custo, mas ao mesmo tempo sendo eficiente em suas tarefas.

O *Apache Hadoop* não indica de forma explícita os requisitos mínimos para o funcionamento do sistema. Como o Java é necessário para o funcionamento do sistema, é através dele que as especificações mínimas para a construção do sistema são definidas.

A utilização de máquinas virtuais é ideal para o propósito do trabalho. A criação deste um ambiente de baixo custo é formado por duas máquinas virtuais na nuvem, com arquitetura simples. Podemos chamar de máquinas commodity, visto que além da arquitetura simples citado, são máquinas virtuais simples com um custo acessível.

A escolha de duas máquinas virtuais se dá pelo fato da quantidade de dados que serão coletados e processados. Foi feito um estudo inicial para determinar e prever o volume de dados e o poder de processamento necessário para um modelo ideal do trabalho. Foi concluído que uma máquina virtual é ideal, porém no escopo do projeto é utilizado duas com intuito de demonstrar o uso real do *cluster*, do paralelismo do processamento e da redundância do sistema de arquivos HDFS.

Novas máquinas podem ser adicionadas ao *cluster* sem nenhum problema, deixando-o mais robusto e rápido. No projeto atual ambas as máquinas virtuais são configuradas com a mesma configuração, descrito na tabela 4.1.

Processador	2 Core Virtual
Memória RAM	4 GB
Capacidade de armazenamento	30GB SSD Disk
Rede	1Gbps = 1000 Mbps

Tabela 4.1: Configuração da máquina virtual que é utilizada

Conforme mencionado, a escolha da configuração se da pelos mesmos motivos que o a escolha de número de nós no *cluster*. A partir do estudo da quantidade de informação que é capturada e processada, foi concluído que, para o projeto em ambiente de produção, a configuração ideal pode ser representada pela tabela 4.2. Com essa configuração, pode-se aumentar a portabilidade, visto que a máquina não necessariamente seja utilizada somente para as atividades do projeto.

Processador	1 Core Virtual
Memória RAM	1 GB
Capacidade de armazenamento	30GB SSD Disk
Rede	1Gbps = 1000 Mbps

Tabela 4.2: Configuração ideal da máquina virtual no projeto

Um detalhe na configuração das máquinas virtuais é o uso de discos SSD, que tornam a leitura e gravação dos arquivos mais rápidos que discos rígidos HDD, amplamente utilizado em *clusters*. Para a implementação do *cluster* foi utilizada a topologia de rede visto na Fig. 4.1.

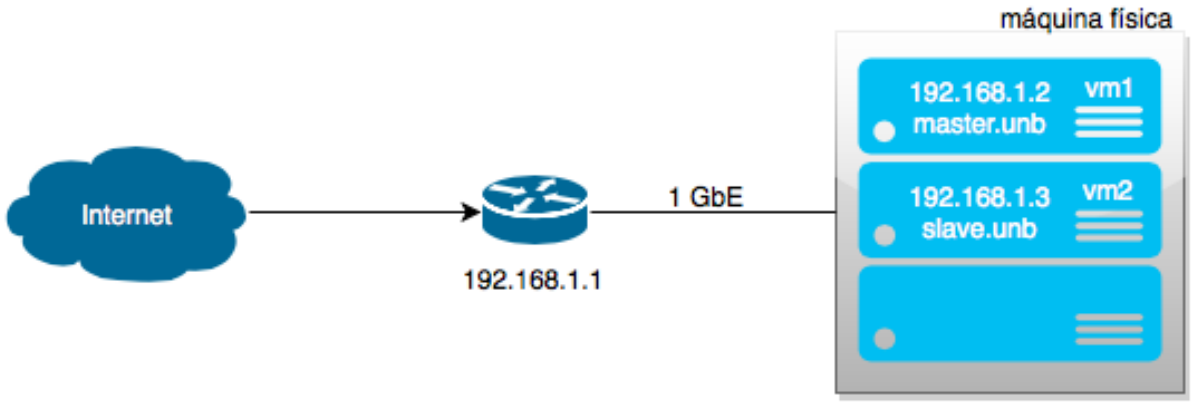

Fig. 4.1: Topologia da rede implementada

A velocidade da rede é de muita importância, visto que é através dela que serão transferidos os dados para processamento, além disso, é por onde irá passar os porções de dados que serão armazenados em cada nó do *cluster*.

Uma das máquinas é configurada como a "*Master*", onde além de distribuir as tarefas aos outros nós, também é utilizada para o processamento de dados. Além disso, é nesta máquina que é feito a captura dos dados via *Flume*, descrito na seção 3.

Com os dois nós formando o *cluster Hadoop*, é obtido uma capacidade de armazenamento do volume HDFS de 60 GB.

Para testes de escrita no sistema de arquivos, foi feito uma amostragem de dados, formando amostras de 500MB de dados reais, ou seja, com *tweets* coletados. A partir desses arquivos foram feitos testes com 2GB, 4GB, 6GB, 8GB, 10GB, 15GB e 20GB. O resultado da escrita no HDFS pode ser visto na Fig. 4.2:

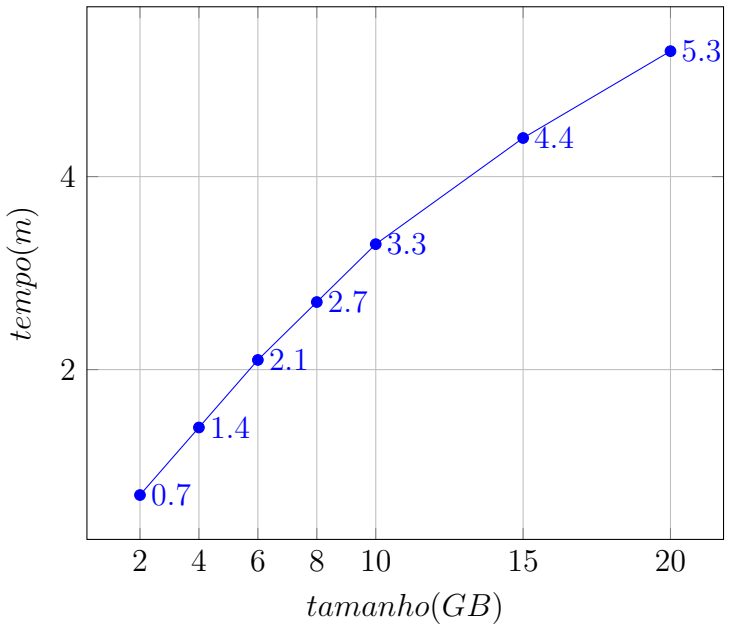

Fig. 4.2: Tempo para escrita de arquivos no HDFS

Podemos observar que o tempo de escrita no HDFS aumenta proporcional ao tamanho total dos arquivos, de forma praticamente linear. Foi analisado também que mesmo com mais máquinas no *cluster* o tempo para escrita é semelhante a de somente uma, pelo fato da escrita ser através de *pipeline*, visto na seção 2.2.1.

Para experimentos de performance de processamento, foi feito o teste com as mesmas amostras, com 1 máquina virtual e 2, demonstrado no gráfico da Fig. 4.3:

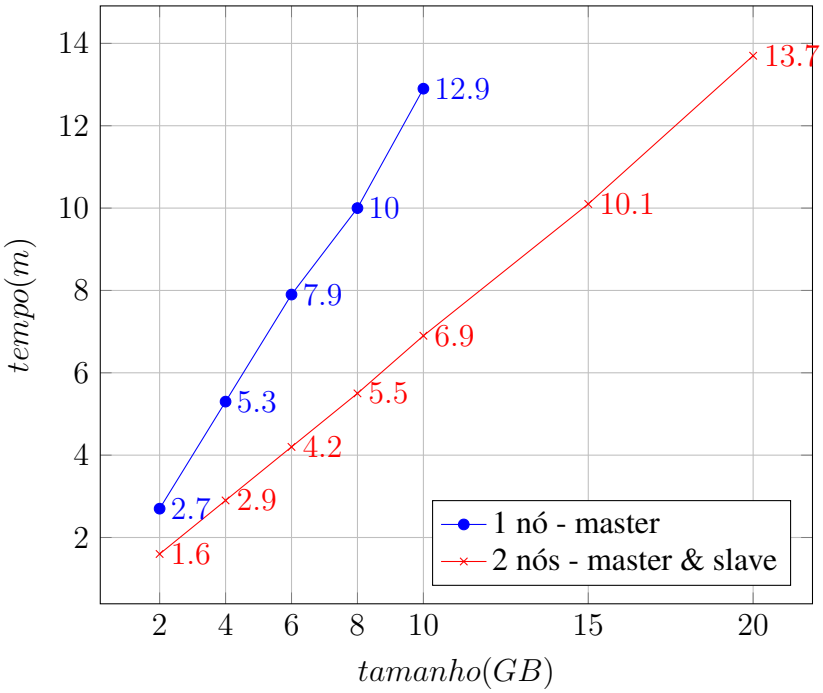

Fig. 4.3: Tempo de processamento de amostra de arquivos

Esse experimento mostra de como a inclusão de nós de processamento (com *TaskTrackers* em execução) podem diminuir o tempo de processamento dos arquivos. Os arquivos de tamanho 15GB e 20GB não foram utilizados no teste com somente 1 nó visto que ultrapassava o espaço em disco, já que o arquivo é copiado para o HDFS.

4.2 IMPLEMENTAÇÃO A NÍVEL DE APLICAÇÃO E SERVIÇOS

As versões do sistema operacional, *softwares* e ferramentas envolvidas com o projeto são listadas na tabela 4.3.

Debian	6.0 “squeeze” 64bits
Java	OpenJDK 1.6.0_35
Apache Hadoop	1.2.1
Apache Flume	1.5.2
Apache HTTP Server	2.2.16
PHP	5.3.3
MySQL	5.1.73

Tabela 4.3: Versões das aplicações

A utilização do *Debian* 6.0 foi definida por ter suporte ao *hardware* utilizado, além de características de estabilidade e segurança possui disponibilidade de pacotes e recursos. O *Hadoop* também suporta outras distribuições do *linux*, como *CentOS* e *Ubuntu*, mas como preferência pessoal e conhecimento já adquirido, foi escolhido o *Debian*.

A Fig. 4.4 demonstra o *cluster* construído com seus respectivos daemons.

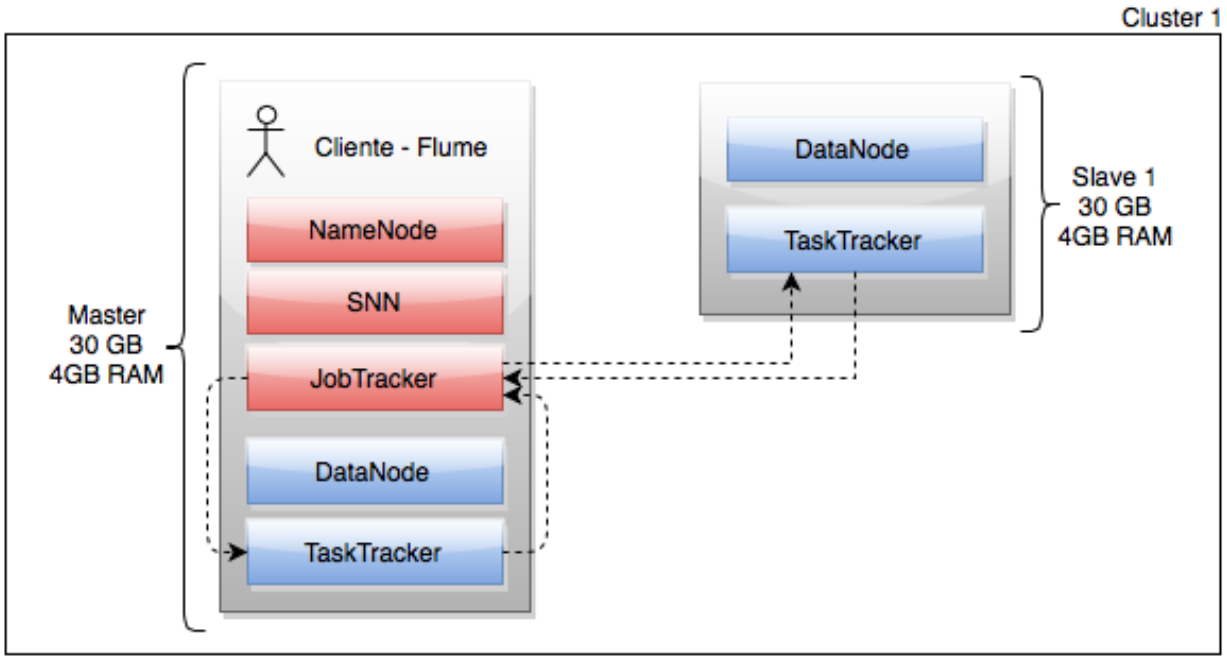

Fig. 4.4: *cluster* construído para o projeto com seus respectivo daemons

Para a comunicação entre o cliente e os daemons *NameNode* e *JobTracker*, mencionado nas seções 2.2.1 e 2.3, é designado portas TCP para cada um desses daemons, essas configurações são armazenadas nos arquivos *core-site.xml* e *mapred-site.xml*, na pasta da instalação do *Hadoop*. No projeto é utilizado as portas 54310 para o NameNode e 54311 para o *JobTracker*, ambos na máquina master. Ou seja, para gerenciar os arquivos no sistema de arquivos HDFS, a comunicação inicial deve ser feita para o *NameNode* hospedado na máquina master, na porta especificada.

O mesmo acontece para novos "*Hadoop jobs*", a requisição deve ser feita para a porta 54311, rodando o JobTracker na máquina master.

Como já foi mencionado, o *Twitter* disponibiliza suas APIs para o acesso aos dados, porém é necessário o prévio cadastro da aplicação que irá consumir esses dados. Com esse cadastro, o *Twitter* informa as chaves de acesso à API. No projeto foi utilizado a versão 1.1 da *Streaming* API do *Twitter*.

Na configuração do *Flume*, é necessário definir a fonte de dados (*Twitter*), o canal (memória) e o coletor (HDFS). Como a fonte de dados é o *Twitter*, as chaves de acesso e as palavras chaves da procura de *tweets* são inseridas, vide tabela 3.1. Com o canal definido na memória, é importante definir a capacidade dela que o *Flume* pode utilizar. Já nas configurações do coletor, é configurado o caminho onde serão salvos os arquivos de saída do *Flume*. Os arquivos serão salvos direto no sistema de arquivos HDFS, portanto o caminho será o mesmo: <hdfs://{IP-da-maquina-master}:{porta-do-NameNode}/{caminho-da-pasta}>. Com a captura, o próprio *Flume* gerencia os arquivos, aguardando com que os mesmos obtenham o tamanho

mínimo para então serem escritos no HDFS.

Para a implementação das funções de mapeamento e redução, é utilizado scripts desenvolvidos em linguagem de preprocessamento de texto com ligações — *hypertext preprocessor* (PHP). Como o intuito posterior do trabalho é prover um ambiente WEB que possa consumir os resumos do que foi processado, a linguagem PHP se adequa a esse escopo, assim como a escolha do banco de dados MySQL. A função de mapeamento e redução são implementadas conforme visto na seção 3.2.2 e disponibilizadas no apêndice deste trabalho.

Foi construído uma estrutura de pastas no sistema HDFS que permite o armazenamento independente dos arquivos a serem processados, que estão sendo processados e os que já foram processados. Ou seja, pode ser encontrado de forma simplificada os arquivos já processados, para análises mais profundas ou utilização para reconstrução do banco de dados.

A Fig. 4.5 ilustra a estrutura da tabela relacional. A query para a criação da tabela se encontra no apêndice do trabalho.

```
mysql> DESCRIBE resumo;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| data      | datetime    | NO   | PRI | NULL    |       |
| operadora | varchar(10) | NO   | PRI |         |       |
| servico   | varchar(10) | NO   | PRI |         |       |
| qntTweets | smallint(3) | NO   |     | NULL    |       |
| positivo  | tinyint(1)  | NO   |     | 0       |       |
+-----------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Fig. 4.5: Estrutura da tabela MySQL

Com a construção do ambiente na nuvem mencionado, juntamente com o armazenamento distribuído HDFS, o sistema se torna eficiente e tolerante a falhas. Juntamente com o processamento paralelo do *MapReduce*, o torna mais tolerante e robusto. Além de ser um *cluster* escalável, de baixo custo, previsto nos objetivos do trabalho. Desse modo a coleta, armazenamento e processamento de dados sãs realizadas. A gerência dos dados é facilitada por existirem interfaces WEB provenientes do próprio *Hadoop* que permitem a visualização dos dados armazenados no sistema de arquivos e informações básicas do status de cada nó no *cluster*. Em configurações padrões do *Hadoop*, a interface WEB do *NameNode* pode ser acessado em <http://{IP-da-Maquina-master}:50070>. Na página também é listado todos os DataNodes no *cluster* e estatísticas básicas (APACHE, 2014). O importante desta interface WEB é ter disponível de fácil acesso um sumário do *cluster*, incluindo informações sobre a capacidade total e a utilizada no HDFS. Além de informar também os nós hospedando *DataNodes* que não estão funcionando.

A mesma coisa acontece com o *JobTracker*, que possui sua própria interface WEB para verificação dos "Hadoop jobs", assim como o progresso dos mesmos. O endereço padrão é <http://{IP-da-maquina-hospedeira-do-JobTracker}:50030>. Também provê os acessos de *log*

das máquinas que estão rodando o *Hadoop*.

No atual trabalho, os dados coletados e processados são consumidos a partir de uma interface WEB construída para disponibilizar informações em gráficos, para consultas antes de possíveis decisões. Essas pesquisas se dão nos tokens armazenados no banco de dados relacional MySQL, já citado na seção 3.2.3. Em uma pesquisa superficial, já pode ser retirado dos dados informações correlacionadas das operadoras, o tempo e o serviço. Análises mais profundas conseguem informações mais relevantes a cada operadora, e possíveis previsões de comportamento no cenário.

Como os dados são salvos no HDFS e além disso os tokens exportados para o banco de dados SQL, há a possibilidade de efetuar consultas direto nos arquivos HDFS. Utilizando o Apache Hive é possível obter informações adicionais que não foram implementadas nas funções de mapeamento, já desenvolvidas. Como por exemplo, verificar a existência de *tweets* com outras palavras chaves, ou verificar qual usuário tem mais *tweets* direcionados a operadoras. O processo da consulta através do Hive normalmente demora alguns minutos por se tratar de uma transformação de uma consulta SQL para funções de mapeamento e redução. Além disso todos os arquivos são consumidos. Ou seja, quanto maior for a quantidade de arquivos existentes, mais demorado será o processo. Mesmo assim, caso o *cluster* possuir mais nós para o trabalho, o tempo de processamento diminui, conforme o gráfico da Fig. 4.3.

4.3 CONSULTAS AOS DADOS

Em ambos os casos citados — utilização do MySQL e Apache Hive —, as consultas são feitas em linguagem SQL, amplamente difundida no cenário computacional, com documentação robusta e comunidade bastante ativa. No caso do Apache Hive, as consultas são feitas de uma vertente do SQL, chamada de HiveQL, que possui algumas novas funções e a retirada de outras do SQL. Com testes feitos, foi constatado que o HiveQL possui menor robustez em relação ao SQL que podemos consultar por exemplo no MySQL. Ou seja, possui consultas que não são possíveis serem feitas no SQL, ou algum resultado esperado, como por exemplo funções de data. Foi concluído também que o HiveQL não suporta o truncamento de dados de uma tabela, ou seja, não é possível obter partes de informações de um campo, além de agrupamentos de campos utilizando o “join” entre tabelas.

Na tabela 4.4 é definida algumas consultas ao banco de dados MySQL — onde está armazenado os dados referentes ao sentimento negativo dos usuários.

Descrição	Consulta SQL
Obter quantidade de *tweets* por dia	`SELECT DATE(data) data, SUM(qntTweets) total FROM resumo GROUP BY DATE(data);`
Obter quantidade de *tweets* por hora	`SELECT HOUR(data) hora, SUM(qntTweets) total FROM resumo GROUP BY HOUR(data);`
Obter quantidade de *tweets* por operadora	`SELECT operadora, SUM(qntTweets) total FROM resumo GROUP BY operadora;`

Tabela 4.4: Exemplos de consultas no banco de dados relacional

Com esses tipos de consultas é possível obter informações mais do que básicas, com análise mais detalhada é possível observar o comportamento dos usuários por determinada operadora.

5 RESULTADOS

5.1 DADOS CAPTURADOS E PROCESSADOS

Com a coleta de dados do *Apache Flume*, arquivos com conteúdo JSON, visto na seção 3.3, são criados no HDFS. Esse conteúdo são os *tweets* brutos, ou seja, sem nenhum tipo de processamento. Esses arquivos servem para serem consumidos pelas funções "*mapper*" e "*reducer*" e posteriormente para futuras análises diretos nos *tweets*. Portanto é de grande importância o armazenamento destes arquivos. Qualquer tipo de problema que tiver no banco de dados relacional, ou com atualizações do sistema deste trabalho para novas versões incluindo novas palavras chaves, as funções de mapeamento e redução podem ser executadas novamente em todos os arquivos, construindo ou reconstruindo o banco de dados.

FlumeData.1432055960942.json	file	7.58 KB	1	64 MB	2015-05-19 14:19	rw-r--r--	hduser	supergroup
FlumeData.1432056025420.json	file	2.42 KB	1	64 MB	2015-05-19 14:20	rw-r--r--	hduser	supergroup
FlumeData.1432056090548.json	file	4.48 KB	1	64 MB	2015-05-19 14:21	rw-r--r--	hduser	supergroup
FlumeData.1432056146084.json	file	3.47 KB	1	64 MB	2015-05-19 14:22	rw-r--r--	hduser	supergroup

Fig. 5.1: Arquivos com o conteúdo sem processamento

Na Fig. 5.1 é mostrado os arquivos gerados pelo *Flume*, no navegador de arquivos da interface WEB do *Hadoop*. O nome do arquivo e a extensão fica a critério do desenvolvedor, visto que o importante é o conteúdo. É definido o nome do arquivo deste modo para que os arquivos não tenha sobreposição — utilização de timestamp — e a extensão para informar que o conteúdo do arquivo é em formato JSON, vide Fig. 3.3.

Portanto, a cada vez que o *Apache Flume* coletar as informações do *Twitter* e os dados possuírem um tamanho mínimo, é criado um arquivo visto na figura. Ou seja, a média de *tweets* por arquivo é variável. No arquivo log do *Flume* é possível verificar em tempo real a obtenção dos *tweets* e o gerenciamento dos arquivos funcionando. Esse log será utilizado para disponibilizar na interface WEB criada os *tweets* que estão sendo indexados no momento.

Com o mapeamento dos *tweets* pela função "*map*", os tokens são gerados e enviados para a função "*reducer*", que os insere no banco de dados. Os dados no banco de dados ficam armazenados de acordo com a Fig. 5.2.

```
mysql> SELECT * FROM resumo ORDER BY data ASC LIMIT 1000,10;
+---------------------+-----------+------------+-----------+----------+
| data                | operadora | servico    | qntTweets | positivo |
+---------------------+-----------+------------+-----------+----------+
| 2015-05-16 18:00:00 | vivo      | internet   |         2 |        0 |
| 2015-05-16 18:00:00 | vivo      | telefonia  |         2 |        0 |
| 2015-05-16 19:00:00 | claro     | telefonia  |         1 |        0 |
| 2015-05-16 19:00:00 | gvt       | internet   |         1 |        0 |
| 2015-05-16 19:00:00 | gvt       | telefonia  |         1 |        0 |
| 2015-05-16 19:00:00 | oi        | atendiment |         1 |        0 |
| 2015-05-16 19:00:00 | oi        | internet   |         1 |        0 |
| 2015-05-16 19:00:00 | vivo      | telefonia  |         1 |        0 |
| 2015-05-16 20:00:00 | claro     | internet   |         1 |        0 |
| 2015-05-16 20:00:00 | claro     | telefonia  |         2 |        0 |
+---------------------+-----------+------------+-----------+----------+
10 rows in set (0.00 sec)
```

Fig. 5.2: Resultado final do processamento dos dados

Analisando os 2 primeiros registros do resultado da consulta mostrado na figura, no dia 16/05/2015 das 18h às 19h houveram 2 *tweets* com sentimento negativo para a operadora Vivo, especificamente pelo serviço de *internet*. Nesse mesmo horário teve 2 *tweets* para a mesma operadora, porém para o serviço de telefonia. Como foi visto nas seções anteriores, o sentimento negativo do usuário se dá pela existência das palavras chaves definidas na tabela 3.3. O campo "positivo", no final da tabela é relacionado a satisfação da pessoa, todos os registros são definidos para "0", por se tratar de uma satisfação negativa.

Um tópico importante a ser abordado no processamento dos dados é que não são processados *tweets* provenientes de "*retweets*" (ato em que o usuário posta um tweet de outro usuário). Ou seja, mesmo que o sistema armazena os *retweets*, o sistema somente irá processar os únicos *tweets*.

5.2 AVALIAÇÃO DOS RESULTADOS

Foi feito análise nos dados coletados e processados, e foi observado que cerca de 12% dos *tweets* únicos capturados possuem sentimento negativo. Este número mostra que realmente os usuários utilizam as redes sociais para fazer uma reclamação de serviços prestados das operadoras.

Esse número está em concordância com o estudo inicial, onde foi previsto que cerca de 10% a 15% de *tweets* seriam de usuários que buscam um serviço que atinjam suas expectativas mas contém insatisfações. Ainda mais, este número pode ser maior. É definido que para caracterizar um sentimento negativo, deve haver palavras chaves que tracem o perfil de insatisfação. Com o aumento da lista de palavras chaves o número de *tweets* caracterizados como negativos aumenta consideravelmente.

Um outro ponto importante para ser abordado são os falso-positivos. Na segmentação, o sistema implementado pode obter *tweets* e indicar como uma insatisfação, mesmo não possuindo

realmente algo que caracterize. O uso das palavras chaves de insatisfação talvez não possam ser utilizadas para inferir algo do serviço prestado e nem da operadora. Portanto, o trabalho tem resultado exclusivamente acadêmico, para melhorias futuras como a construção de um sistema inteligente que aprenda o comportamento dos usuários e consiga encaixar o tweet no tipo de satisfação. A construção de um algoritmo que consiga obter um índice de satisfação utilizando uma base de dados de comportamentos previstos pode ser um bom exemplo de uma nova atualização da ferramenta. Um sistema neste cenário torna cada vez mais preciso, ilustrando o exato comportamento dos usuários.

Os *tweets* considerados para este cálculo não contabilizam os "*retweets*". Cerca de 60% dos *tweets* brutos coletados são provenientes de *retweets*. Esse alto número de *retweets* se dá pelo fato em que os usuários famosos do *Twitter* publicam algo sobre alguma operadora e os seus seguidores concordam com o que foi comentado e fazem a publicação da mesma mensagem.

Os resultados foram de acordo com o que foi analisado inicialmente e atingem os objetivos traçados no projeto. Problemas foram encontrados, e serão abordados a seguir.

O primeiro problema é a quantidade de *tweets* que o *Twitter* disponibiliza na *Streaming* API, mencionado na seção 3.1.1. A disponibilização de APIs abertas ao público tem como vantagem a promoção da inovação, ampliando a base tecnológica, os serviços e as informações. Oferecendo os dados, o *Twitter* permite que desenvolvedores possam criar produtos, plataformas e interfaces sem a necessidade de expor os dados originais. Porém, o problema encontrado é que o *Twitter* provê somente amostras dos *tweets* que estão ocorrendo. A porcentagem atual do total de *tweets* que são consumidos pela *Streaming* API é bastante variável, baseado nos critérios de busca e do tráfego atual.

Estudos estimam que utilizando a *Streaming* API, os dados consumidos podem ser de 1% a 40% do total de *tweets* em tempo real (MORSTATTER J. PFEFFER; CARLEY, 2013). A razão dessa limitação se dá pelo simples fato que o *Twitter* não possui, atualmente, uma infraestrutura que possa suportar o consumo real de todos os *tweets*. Além disso, o mesmo não tem interesse a esse tipo de abertura a nenhum custo. A maior consequência disso é que os sistemas podem ficar imprecisos. Mas, de acordo com Russel (RUSSEL, 2014, pag 391), em horários com picos de *tweets*, a velocidade das publicações podem ser de mil *tweets* por segundo. Ou seja, mesmo com 1%, a taxa de *tweets* disponibilizados tendem a ser grandes.

Para contornar esse problema, existe um meio de acesso final que o *Twitter* disponibiliza. Chama-se "*Twitter* Firehose". O Firehose de fato é bastante similar ao *Streaming* API, onde é enviado as informações próximo ao tempo real. A diferença principal é que o *Firehose* garante a entrega de 100% dos *tweets* que encaixam nos critérios das buscas. Além disso, limitações de utilização são retiradas, mas em contra partida o custo para ter acesso a este serviço é altíssimo.

Um outro ponto a ser abordado é a utilização do *Hadoop* e suas ferramentas, com a quantidade de dados observado. O HDFS não foi projetado para arquivos pequenos. Para cada novo

arquivo lido, o cliente conversa com o *NameNode*, que informa a localização dos blocos de informação do determinado arquivo, e daí o cliente consegue obter o dado do *DataNode*. No melhor caso, o cliente faz este procedimento uma única vez, e observa que o arquivo está na mesma máquina que está executando. Porém, normalmente os blocos estão espalhados entre os nós do *cluster*. Ou seja, o fluxo de dados deve ser transmitido pela rede. Com isso, a velocidade fica limitada pela E/S da rede, que pode ser muito menor do que um acesso direto ao disco.

Ou pode atingir o pior caso, em que o overhead que existe quando há uma conversa com o *NameNode* se torna significante. Com arquivos de 1KB, a quantidade de overhead que é transferida praticamente é a mesma da informação útil. Ou seja, com o tamanho do bloco muito pequeno, terá mais blocos e por consequência mais overhead. Adicionado a isso existe a possibilidade do *NameNode* ficar sobrecarregado por todas essas requisições de diferentes lugares. Ou seja, se tornar um gargalo no sistema (WHITE, 2009).

Como foi dito na seção 2.2.1, uma diferença entre o HDFS e outros sistemas de arquivos é que o HDFS armazena grandes blocos (normalmente de 64MB) para que o acesso ao disco seja o menor possível. Comparado com estes outros sistemas de arquivos, o tamanho do bloco é na ordem de KB. Por esse motivo que é dito que o HDFS é bom para processamento e armazenagem com arquivos grandes.

Um modo de evitar este tipo de problema é utilizar o HDFS realmente com arquivos grandes. Como o *Flume* grava os arquivos em pequenos pedaços, pode ser feito a junção de todos os arquivos em somente um com tamanho grande. Ou mais simples ainda, configurar o *Flume* para somente gravar o arquivo quando chegar a um determinado tamanho. O problema dessa última solução é o estouro de memória já que a captura é armazenada na memória da máquina até que chegue ao tamanho especificado. Com arquivos maiores, é obtido melhor performance, por menos overhead e além disso, a vantagem de leituras sequenciais no disco.

Apesar disso, para o trabalho atual, não é substancial a utilização das práticas ideais, visto que o intuito é o consumo dos dados para análises superficiais. É recomendado a utilização das recomendações em situações em que é necessário o monitoramento em tempo real durante eventos específicos ou situações críticas.

Com os dados capturados, armazenados e processados, pode-se por exemplo verificar qual operadora tem usuários insatisfeitos no serviço prestado de telefonia em todo período coletado, observado na Fig. 5.3.

```
mysql> SELECT operadora, SUM(qntTweets) tweets FROM resumo WHERE
servico = 'telefonia' GROUP BY(operadora) ORDER BY tweets DESC;
+-----------+--------+
| operadora | tweets |
+-----------+--------+
| oi        |   1637 |
| vivo      |   1411 |
| claro     |   1281 |
| net       |    719 |
| tim       |    605 |
| gvt       |    450 |
+-----------+--------+
6 rows in set (0.01 sec)
```

Fig. 5.3: Quantidade de *tweets* com insatisfação do serviço de telefonia

Uma forma de verificar se as informações coletadas são de fato condizentes com a realidade, pode-se confrontar os dados apresentados com os dados disponibilizados pelas agências reguladoras destes serviços.

A real inteligência para se obter as informações valiosas dos dados é basicamente na armazenagem desse grande volume de dados, e além disso, no próprio processamento. Extrair informações relevantes e com isso prever resultados de alta precisão são os objetivos e tendências.

A abordagem da análise em tempo real também se torna imprescindível. Dificilmente um sistema é realmente em tempo real. Há atrasos por diversos serviços e transmissão. O que vemos é próximo ao tempo real, por isso esta atribuição quando há pequenos atrasos ou janelas. No caso do projeto do trabalho, existe claramente um pequeno atrasado na disponibilização das informações pelo *Twitter*. Isso se dá pela política da própria instituição, para que haja um controle interno. Existe também o tempo de processamento das informações no sistema.

Primeiramente é abordado o tempo gasto até que exista dados com o tamanho mínimo para que o *Apache Flume*, explicado na seção 2.4.1, escreva no sistema de arquivos HDFS. Dependendo da consulta ao *Twitter* esse atraso se torna grande, mesmo com a implementação de timeouts — o arquivo é escrito com um tamanho menor.

Um outro ponto é a consulta aos dados coletados. No trabalho, foi definido uma janela de 10 minutos para que as funções de mapeamento e redução sejam executadas sobre os arquivos coletados. No pior caso, uma consulta direto no banco de dados relacional pode retornar dados que ainda não foram inseridos por estes 10 minutos.

Para obter dados mais próximos ao tempo real, pode-se utilizar o *Apache Hive*, abordado na seção 2.4.2. Com ele é possível buscar informações direto nos arquivos no HDFS e retornar o resultado de acordo com o que está armazenado naquele momento. Porém como a consulta é traduzida para funções de *map* e *reducer*, existe mais um tempo de atraso, sem falar nas próprias execuções das funções, que dependendo da consulta e do tamanho das informações armazenadas pode durar minutos ou até horas. Por esse motivo o Hive não é implementado no projeto para a visualização dos dados na interface WEB.

Uma das premissas do projeto é a construção de um ambiente de baixo custo. A insfraestrutura utilizada reforça a ideia principal do *Hadoop*: utilização de máquinas commodities em conjunto para aumentar o armazenamento com redundância e o poder de processamento. Um ambiente ideal para esse tipo de implementação *Hadoop* é a de nuvem. Inicialmente o volume da dados pode ser pequeno, com a nuvem é atingido o fator escalabilidade e elasticidade. O operador do sistema pode começar com a configuração mínima, e de acordo com o desempenho, o volume de dados, e o tempo gasto, ampliar a configuração sem esforço e prejuízo à aplicação. Empresas como a *Amazon*, *Digital Ocean* e *Linode* oferecem este de serviço de nuvem por custos muito baixo. A Amazon por exemplo, além de oferecer o serviço de nuvem (*EC2*) — o operador do sistema monta seu ambiente, oferece já o serviço de armazenagem e processamento de dados, sem que o operador tenha conhecimentos profundos sobre *Hadoop*, e ainda mais, do *Linux*.

6 CONCLUSÃO

A modelagem sentimental dos usuários via rede social permite provar mais uma opção para os usuários analisar e comparar as prestadoras de serviço de telecomunicações. Esse estudo mostra o comportamento dos usuários no aspecto de reclamações nas redes sociais. Ou seja, consegue mostrar por exemplo os momentos que há picos de reclamações.

Para as operadoras, o estudo oferece uma ferramenta para que consigam observar como os serviços prestados estão sendo mencionados na rede. Uma prestadora pode aproveitar desses dados em "tempo real" para traçar estratégias até descobrir possíveis falhas em sua infraestrutura — pico de reclamações na última hora. E mais do que isso, pode observar se uma nova estratégia está trazendo alguma melhoria no rendimento da empresa.

Com o amadurecimento do mercado, as análises sociais ganham destaque e grandes investimentos. Podemos observar isso através do atendimento ao consumidor por meio das mídias sociais. Ou seja, as instituições estão buscando reduzir custos de suporte com monitoração e atendimento nas redes. Isto traz diversos benefícios para a empresa. Melhorar engajamento com consumidores, identificar gargalos, descobrir novas ideias, identificar e mitigar problemas com o lançamento de novos produtos são alguns exemplos.

A modelagem sentimental será um elemento indispensável para as empresas, que querem medir a opinião dos usuários, interpretar e recomendar ações baseadas em sinais do mercado. A habilidade de coletar, interpretar e agir em tempo real representará uma imensa vantagem competitiva. Aliado a isso o grande volume de dados, que será um dos grandes impactos da transição.

O ambiente deve ser flexível e escalável. Ou seja, além de poder de processamento, devem estar preparados para mudanças de padrões e estratégias. No trabalho, foi implementado o sistema para mapear as emoções dos usuários sobre as operadoras de telecomunicação. Mas se tratando de serviços e produtos pode ser flexível por exemplo para companhias aéreas. Ademais, o ambiente construído pode ser implementável em outros "setores", como por exemplo de lugares. Lugares para conhecer, hospedar, alimentar. Ou seja, mapeia os locais e constrói segmentações de acordo com as palavras chaves. Esse tipo de informação é útil tanto para quem é o consumidor quanto a empresa.

Um outro tipo de "setor" que pode ser implementado é o de eventos. Um exemplo clássico que pode ser citado é o das eleições. Os usuários podem identificar as opiniões e emoções de outros sobre determinado candidato, ou até mesmo como anda o desempenho da campanha eleitoral.

A grande contribuição deste estudo está na ferramenta desenvolvida para que os usuários consigam obter informações referente as operadoras de telecomunicação. É escolhido ferramentas

e *frameworks* utilizados para grandes volumes de dados (*terabytes*, *petabytes*) para demonstrar o seu funcionamento, mesmo para pequenas quantidade de dados. A coleta de dados, armazenagem e processamento, pode ser feito por diversas plataformas e linguagem. Porém sob o ponto de vista operacional, oferecendo capacidade, flexibilidade, escalabilidade e além de tudo, redundância as ferramentas utilizadas obtém vantagens em cenários específicos.

6.1 TRABALHOS FUTUROS E RELACIONADOS

O estudo se limitou à insatisfação dos usuários publicada nos *Twitter*, mas vale ressaltar que o microblog também serve para um *feedback* positivo dos consumidores. Um possível trabalho futuro é a modelagem mais profunda das emoções dos usuários. Ou seja, realmente separar as opiniões e emoções, descobrir a polaridade das informações, identificar textos emotivos ou sentimentais, classificando assim os dados sob um desses ponto de vista.

Além disso, uma implementação que contorne os problemas encontrados, mencionados nos resultados do projeto (5.2), é de suma importância, visto que com a resolução dos problemas é possível construir um sistema mais eficiente que ultrapasse as barreiras limitantes, como a dificuldade de conciliar os dados, tecnologia deficiente, excesso de dados e a falta de compreensão.

Existe um trabalho relacionado com o atual projeto, que visa a análise dos dados coletados. Ou seja, a inteligência para encontrar oportunidades, combater as falhas e prever situações a partir dos dados coletados através deste trabalho. Além disso, construir formas e modelos visuais que possam correlacionar os dados processados.

O resultado dos dados capturados e processados pode ser encontrado no sítio <http://telecomnasredes.com.br>.

REFERÊNCIAS BIBLIOGRÁFICAS

APACHE. *HDFS Users Guide*. 2014. Disponível em: <https://hadoop.apache.org/docs/r2.6.0/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html>. Acesso em: 12 jun 2015.

BERMAN, J. J. *Principles of big data*. [S.l.]: Elsevier and Morgan Kaufmann, 2013.

BURSON-MARSTELLER. *The Global Social Media Check-up 2010*. 2010. Disponível em: <http://www.burson-marsteller.com/what-we-do/our-thinking/the-global-social-media-check-up-2010-2/>. Acesso em: 08 jun. 2015.

DEAN, J.; GHERMAWAT, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, v. 51, p. 107–113, 2008.

DUMBILL, E. et al. *Big Data Now: 2012 Edition*. [S.l.]: O'Reilly Media, Inc., 2012.

EATON, C. et al. *Understanding Big Data. Analytics for enterprise class hadoop and streaming data*. Nova York: McGraw Hill, 2012.

E.LIFE. *Operadoras são bicampeãs de reclamações no Twitter*. 2014. Disponível em: <http://veja.abril.com.br/noticia/vida-digital/operadoras-sao-bicampeas-de-reclamacoes-no-twitter/>. Acesso em: 01 jul. 2015.

FORTUNE. What data says about us. *Fortune*, 2012. Disponível em: <http://fortune.com/2012/09/10/what-data-says-about-us/>. Acesso em: 27 jun 2015.

FRANKS, B. *Taming the big data tidal wave*. Hoboken, New Jersey: John Wiley and Sons, Inc., 2012.

HURWITZ, J. et al. *Big data for dummies*. [S.l.]: Wiley, 2013.

LAM, C. *Hadoop in Action*. [S.l.]: Manning Publications Co., 2011.

MALIK, O. *Statshot: How mobile data traffic will grow by 2016*. 2012. Disponível em: <https://gigaom.com/2012/08/23/global-mobile-data-forecast/>. Acesso em: 15 abr. 2015.

MAYER-SCHONBERGER, V.; CUKIER, K. *Big Data: life, work and think*. Nova York: Houghton Mifflin Harcourt, 2013.

MORSTATTER J. PFEFFER, H. L. F.; CARLEY, K. Is the sample good enough? comparing data from twitter's streaming api with twitter's firehose. *ICWSM*, 2013.

PETRY, A.; VILICIC, F. A era do big data e dos algoritmos está mudando o mundo. *Revista Veja*, Maio 2013.

PROKOPP, C. *The Free Hive Book*. 2013. Disponível em: <http://www.semantikoz.com/blog/the-free-apache-hive-book/>. Acesso em: 09 jul. 2015.

RUSSEL, M. A. *Mining the Social Web*. [S.l.]: O'Reilly Media, Inc., 2014.

RUTHERGLEN, J.; WAMPLER, D.; CAPRIOLO, E. *Programming Hive*. [S.l.]: O'Reilly Media, Inc., 2012.

SATHI, A. *Big data analytics*. [S.l.]: MC Press, 2012.

SCHROECK, M. et al. Analytics: The real-world use of big data. *IBM Global Business Services*, 2013.

SRIPATI, P. *Analyse Tweets using Flume, Hadoop and Hive*. 2013. Disponível em: <http://www.thecloudavenue.com/2013/03/analyse-tweets-using-flume-hadoop-and.html>. Acesso em: 24 jun. 2015.

TWITTER. *The Streaming APIs*. 2015. Disponível em: <https://dev.twitter.com/streaming/overview>. Acesso em: 23 mai. 2015.

VENNER, J. *Pro Hadoop*. [S.l.]: Apress, 2009.

WEF. Committed to improving the state of the world. In: WORLD ECONOMIC FORUM. Genebra, 2012.

WHITE, T. *The Small Files Problem*. 2009. Disponível em: <http://blog.cloudera.com/blog/2009/02/the-small-files-problem/>. Acesso em: 29 jun. 2015.

WHITE, T. *Hadoop: The Definitive Guide, Third Edition*. [S.l.]: O'Reilly Media, Inc., 2012.

ANEXOS

I. INSTALAÇÃO

I.1 MASTER

Para a instalação do ambiente no master executar os seguintes comandos no terminal (de preferência adicionar a um arquivo .sh, editar a permisão do arquivo para executável e executar o arquivo).

```
#!/bin/bash
#Instalação no master

#=====================================================
#Definição das variáveis
#Definir a quantidade de maquinas
export NUMERO_MAQUINAS=2

#Inserir o hostname da maquina
export MASTER_HOSTNAME="master.unb"
export MASTER_IP="10.128.86.158"

#Inserir o hostename do slave
export SLAVE_HOSTNAME="slave.unb"
export SLAVE_IP="10.128.86.157"

export ROOT_PASSWORD="pfg2"

#Definir as palavras chaves
export PALAVRAS_CHAVES="vivoemrede, timbrasil, clarobrasil, digaoi, gvtoficial,
    gvt_suporte, netoficial"

#Definir as versões, url e locais do Hadoop
export HADOOP_VER="hadoop-1.2.1"
export HADOOP_URL="http://www.us.apache.org/dist/hadoop/core/${HADOOP_VER}/${
    HADOOP_VER}.tar.gz"
export HADOOP_DIR="/usr/local/hadoop"

#Definir as versões, url e locais do Flume
export FLUME_VER="1.5.2"
export FLUME_FOLDER="apache-flume-${FLUME_VER}-bin"
export FLUME_FILE="${FLUME_FOLDER}.tar.gz"
export FLUME_URL="http://archive.apache.org/dist/flume/${FLUME_VER}/${FLUME_FILE}"
export FLUME_SOURCE_URL="https://github.com/cloudera/cdh-twitter-example.git"
export FLUME_DIR="${HADOOP_DIR}/flume"

#Definir local do java
export JAVA="/usr/lib/jvm/java-6-openjdk"

#Definir a pasta de trabalho do HDFS
export HDFS_SAIDA_DIR="/user/hduser/tmp"
```

```
#=====================================================

#Alterar a senha do root
echo "Alterar password do root: (alterar para pfg2):"
passwd

#=====================================================

#Configurar timezone para o Brasil
echo "America/Sao_Paulo" > /etc/timezone
dpkg-reconfigure -f noninteractive tzdata

#=====================================================

#Adicionar repositórios para baixar as ferramentas
cat <<EOF >> /etc/apt/sources.list
deb http://ppa.launchpad.net/natecarlson/maven3/ubuntu precise main
deb-src http://ppa.launchpad.net/natecarlson/maven3/ubuntu precise main
EOF

#=====================================================

#Atualizar pacotes
apt-get --assume-yes --force-yes update
apt-get --assume-yes --force-yes upgrade
apt-get --assume-yes --force-yes install default-jdk git maven3 apache2 mysql-server
    php5 php-pear php5-mysql php5-curl

#Configuração
ln -s /usr/share/maven3/bin/mvn /usr/bin/mvn

#No passo do mysql, inserir a senha como pfg2
mysql_secure_installation

#=====================================================

#Criacao do banco de dados e usuario mysql
mysql -u root -p'pfg2' <<EOF
CREATE DATABASE tweets_db;
CREATE USER tweets_usr;
GRANT USAGE ON *.* to tweets_usr@localhost identified by 'pfg2';
GRANT ALL PRIVILEGES ON tweets_db.* to tweets_usr@localhost;
FLUSH PRIVILEGES;
USE tweets_db;
CREATE TABLE resumo (
  data datetime NOT NULL,
  operadora varchar(10) NOT NULL DEFAULT '',
  servico varchar(10) NOT NULL DEFAULT '',
  qntTweets smallint(3) NOT NULL,
  positivo tinyint(1) NOT NULL DEFAULT 0,
  PRIMARY KEY (data,operadora,servico)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE tweets (
  id bigint(20) NOT NULL,
  PRIMARY KEY(id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
EOF
```

```
#=====================================================

#Reiniciar servico do apache2
service apache2 restart

#=====================================================

#Adicionar usuário dedicado Hadoop (Usuário: hduser | Grupo: hadoop)
addgroup hadoop
adduser --ingroup hadoop --disabled-password --gecos "" hduser

#=====================================================

#Adicionar hosts no /etc/hosts
cat <<EOF > /etc/hosts
127.0.0.1   localhost
$MASTER_IP  $MASTER_HOSTNAME
$SLAVE_IP   $SLAVE_HOSTNAME
EOF

#=====================================================

#Inserir variaveis e funções no arquivo de inicialização do bash do usuário criado (
    hduser)
cat <<EOF | tee -a /home/hduser/.profile /root/.bashrc /home/hduser/.bashrc &> /dev/
    null

export HADOOP_PREFIX=$HADOOP_DIR
export JAVA_HOME=$JAVA


export HADOOP_VER=$HADOOP_VER
export HADOOP_URL=$HADOOP_URL
export HADOOP_DIR=$HADOOP_DIR

export FLUME_VER=$FLUME_VER
export FLUME_FOLDER=$FLUME_FOLDER
export FLUME_FILE=$FLUME_FILE
export FLUME_URL=$FLUME_URL
export FLUME_DIR=$FLUME_DIR

export JAVA=$JAVA

export LC_ALL=pt_BR.utf8
export LANG="$LC_ALL"
export PATH=\$PATH:\$FLUME_DIR/bin:\$HADOOP_DIR/bin


unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"

lzohead () {
    hadoop fs -cat $1 | lzop -dc | head -1000 | less
}
```

```
EOF

#====================================================

#Configurar SSH para o hduser
#Logar como hduser e executar os comandos
su -l hduser << 'EOF'

rm -rf ~/.ssh
mkdir ~/.ssh
ssh-keygen -t rsa -P "" -f ~/.ssh/id_rsa
cat $HOME/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

#Testar configuração SSH com private/shared keys
ssh $(hostname) -o 'ConnectionAttempts=1' true
exit
EOF

#====================================================

#Instalação do Hadoop
cd /tmp
wget $HADOOP_URL
tar xzf $HADOOP_VER.tar.gz
mv $HADOOP_VER $HADOOP_DIR
cd $HADOOP_DIR
chown -R hduser:hadoop .

#Adicionar a variável do java no ambiente do hadoop
cat <<EOF >> $HADOOP_DIR/conf/hadoop-env.sh
export JAVA_HOME=$JAVA
EOF

#Criando os arquivos de configuração do hadoop
rm -rf /app/hadoop
mkdir -p /app/hadoop/tmp
chown hduser:hadoop /app/hadoop/tmp

#core-site.xml (informacoes do HDFS)
#Aqui é definido a porta onde o HDFS será executado
cat <<EOF > $HADOOP_DIR/conf/core-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
</property>

<property>
  <name>fs.default.name</name>
  <value>hdfs://${MASTER_HOSTNAME}:54310</value>
  <description>The name of the default file system.  A URI whose
  scheme and authority determine the FileSystem implementation.  The
```

```
  uris scheme determines the config property (fs.SCHEME.impl) naming
  the FileSystem implementation class.  The uris authority is used to
  determine the host, port, etc. for a filesystem.</description>
</property>

</configuration>
EOF

#mapred-site.xml
#Informações do MapReduce. Em qual porta o JobTracker irá ser executado
cat <<EOF > $HADOOP_DIR/conf/mapred-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
  <name>mapred.job.tracker</name>
  <value>${MASTER_HOSTNAME}:54311</value>
  <description>The host and port that the MapReduce job tracker runs
  at.  If "local", then jobs are run in-process as a single map
  and reduce task.
  </description>
</property>

</configuration>
EOF

#hdfs-site.xml
#Configuração da quantidade de replicações um bloco de arquivos terá. Normalmente o
    número é definido como a quantiadde de máquinas
cat <<EOF > $HADOOP_DIR/conf/hdfs-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
  <name>dfs.replication</name>
  <value>${NUMERO_MAQUINAS}</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
  </description>
</property>

</configuration>
EOF

#=====================================================

#Instalação do Flume
wget $FLUME_URL -P /tmp
tar xzf /tmp/$FLUME_FILE -C /tmp
mv /tmp/$FLUME_FOLDER $FLUME_DIR
git clone $FLUME_SOURCE_URL /tmp/cdh
cd /tmp/cdh/flume-sources
```

```
mvn package
cp target/flume-sources-1.0-SNAPSHOT.jar $FLUME_DIR/

#flume-env.sh
#Adicionar variaveis no ambiente do flume
cat <<EOF > $FLUME_DIR/conf/flume-env.sh
JAVA_HOME=$JAVA
FLUME_HOME=$FLUME_DIR
FLUME_CONF_DIR=${FLUME_DIR}/conf
FLUME_CLASSPATH=${FLUME_DIR}/flume-sources-1.0-SNAPSHOT.jar
EOF

#flume.conf
#Configurações principais do flume
#Inserir aqui as chaves disponibilizadas pelo Twitter
cat <<EOF > $FLUME_DIR/conf/flume.conf
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS

TwitterAgent.sources.Twitter.type = com.cloudera.flume.source.TwitterSource
TwitterAgent.sources.Twitter.channels = MemChannel
TwitterAgent.sources.Twitter.consumerKey = 2A0V02WuDcfr2Fz1tolQeOQMx
TwitterAgent.sources.Twitter.consumerSecret =
    YHO14pSx8SslheD7NwcTwfz5O98r1H4FpdD4wzQyo8whEV882Y
TwitterAgent.sources.Twitter.accessToken = 39406016-
    fx535vCZJ8zDvjcLqoqcEJseHPVQn2VGwUHYzOSv7
TwitterAgent.sources.Twitter.accessTokenSecret =
    RUNxAMVUscDTlrWBs09viuMNrF93Hfd9wLIZ3m33LlzNB

TwitterAgent.sources.Twitter.keywords = ${PALAVRAS_CHAVES}

TwitterAgent.sinks.HDFS.channel = MemChannel
TwitterAgent.sinks.HDFS.type = hdfs
TwitterAgent.sinks.HDFS.hdfs.path = hdfs://${MASTER_HOSTNAME}:54310${HDFS_SAIDA_DIR}
TwitterAgent.sinks.HDFS.hdfs.fileSuffix = .json
TwitterAgent.sinks.HDFS.hdfs.fileType = DataStream
TwitterAgent.sinks.HDFS.hdfs.writeFormat = Text
TwitterAgent.sinks.HDFS.hdfs.batchSize = 1000
TwitterAgent.sinks.HDFS.hdfs.rollSize = 0
TwitterAgent.sinks.HDFS.hdfs.rollCount = 10000

TwitterAgent.channels.MemChannel.type = memory
TwitterAgent.channels.MemChannel.capacity = 10000
TwitterAgent.channels.MemChannel.transactionCapacity = 100
EOF

#=====================================================

#Adicionar configuracoes da maquina master
cat <<EOF > $HADOOP_DIR/conf/masters
$MASTER_HOSTNAME
EOF

#Adicionar configuracões das maquinas slave (no caso a master também executará
    trabalhos)
cat <<EOF > $HADOOP_DIR/conf/slaves
```

```
$MASTER_HOSTNAME
EOF

#=====================================================

#Criação da pasta de script
mkdir /home/hduser/scripts

#=====================================================

#Modificar proprietadade das pastas para hduser
chown -R hduser:hadoop /home/hduser/*
chown -R hduser:hadoop $HADOOP_DIR

#=====================================================

#Formatar o HDFS, logado no usuario hduser
su -l hduser <<'EOF'
hadoop namenode -format
EOF
```

Efetuando esses passos, o Hadoop estará instalado na máquina master.

I.2 SLAVE

Para a instalação do ambiente na máquina slave, os passos a serem seguidos são semelhantes ao da instalação do master, com a adição e remoção de alguns comandos.

```
#!/bin/bash
#Instalação no master

#=====================================================
#Definição das variáveis
#Definir a quantidade de maquinas
export NUMERO_MAQUINAS=2

#Inserir o hostname da maquina
export MASTER_HOSTNAME="master.unb"
export MASTER_IP="10.128.86.158"

#Inserir o hostename do slave
export SLAVE_HOSTNAME="slave.unb"
export SLAVE_IP="10.128.86.157"

export ROOT_PASSWORD="pfg2"

#Definir as versões, url e locais do Hadoop
export HADOOP_VER="hadoop-1.2.1"
export HADOOP_URL="http://www.us.apache.org/dist/hadoop/core/${HADOOP_VER}/${
    HADOOP_VER}.tar.gz"
export HADOOP_DIR="/usr/local/hadoop"

#Definir local do java
```

```
export JAVA="/usr/lib/jvm/java-6-openjdk"

#Definir a pasta de trabalho do HDFS
export HDFS_SAIDA_DIR="/user/hduser/tmp"


#=====================================================

#Alterar a senha do root
echo "Alterar password do root: (alterar para pfg2):"
passwd

#=====================================================

#Configurar timezone para o Brasil
echo "America/Sao_Paulo" > /etc/timezone
dpkg-reconfigure -f noninteractive tzdata

#=====================================================

#Adicionar repositórios para baixar as ferramentas
cat <<EOF >> /etc/apt/sources.list
deb http://ppa.launchpad.net/natecarlson/maven3/ubuntu precise main
deb-src http://ppa.launchpad.net/natecarlson/maven3/ubuntu precise main
EOF

#=====================================================

#Atualizar pacotes
apt-get --assume-yes --force-yes update
apt-get --assume-yes --force-yes upgrade
apt-get --assume-yes --force-yes install default-jdk apache2 mysql-server php5 php-
    pear php5-mysql php5-curl

#=====================================================

#Adicionar usuário dedicado Hadoop (Usuário: hduser | Grupo: hadoop)
addgroup hadoop
adduser --ingroup hadoop --disabled-password --gecos "" hduser

#=====================================================

#Adicionar hosts no /etc/hosts
cat <<EOF > /etc/hosts
127.0.0.1   localhost
$MASTER_IP  $MASTER_HOSTNAME
$SLAVE_IP   $SLAVE_HOSTNAME
EOF

#=====================================================

#Inserir variaveis e funções no arquivo de inicialização do bash do usuário criado (
    hduser)
cat <<EOF | tee -a /home/hduser/.profile /root/.bashrc /home/hduser/.bashrc &> /dev/
    null

export HADOOP_PREFIX=$HADOOP_DIR
```

```
export JAVA_HOME=$JAVA


export HADOOP_VER=$HADOOP_VER
export HADOOP_URL=$HADOOP_URL
export HADOOP_DIR=$HADOOP_DIR

export JAVA=$JAVA

export LC_ALL=pt_BR.utf8
export LANG="$LC_ALL"
export PATH=\$PATH:\$FLUME_DIR/bin:\$HADOOP_DIR/bin


unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"

lzohead () {
    hadoop fs -cat $1 | lzop -dc | head -1000 | less
}
EOF

#=====================================================

#Configurar SSH para o hduser
#Logar como hduser e executar os comandos
su -l hduser << 'EOF'

rm -rf ~/.ssh
mkdir ~/.ssh

#Conectar com o master e inserir a chave do master nas chaves autorizadas
scp root@${MASTER_HOSTNAME}:/home/hduser/.ssh/id_rsa.pub ~/.ssh/authorized_keys
ssh root@${MASTER_HOSTNAME} "echo $(hostname) >> ${HADOOP_DIR}/conf/slaves"

ssh-keygen -t rsa -P "" -f ~/.ssh/id_rsa
cat $HOME/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

#Testar configuração SSH com private/shared keys
ssh $(hostname) -o 'ConnectionAttempts=1' true
exit
EOF

#=====================================================

#Instalação do Hadoop
cd /tmp
wget $HADOOP_URL
tar xzf $HADOOP_VER.tar.gz
mv $HADOOP_VER $HADOOP_DIR
cd $HADOOP_DIR
chown -R hduser:hadoop .


#Adicionar a variável do java no ambiente do hadoop
```

```
cat <<EOF >> $HADOOP_DIR/conf/hadoop-env.sh
export JAVA_HOME=$JAVA
EOF

#Criando os arquivos de configuração do hadoop
rm -rf /app/hadoop
mkdir -p /app/hadoop/tmp
chown hduser:hadoop /app/hadoop/tmp

#core-site.xml (informacoes do HDFS)
#Aqui é definido a porta onde o HDFS será executado
cat <<EOF > $HADOOP_DIR/conf/core-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
  <description>A base for other temporary directories.</description>
</property>

<property>
  <name>fs.default.name</name>
  <value>hdfs://${MASTER_HOSTNAME}:54310</value>
  <description>The name of the default file system.  A URI whose
  scheme and authority determine the FileSystem implementation.  The
  uris scheme determines the config property (fs.SCHEME.impl) naming
  the FileSystem implementation class.  The uris authority is used to
  determine the host, port, etc. for a filesystem.</description>
</property>

</configuration>
EOF

#mapred-site.xml
#Informações do MapReduce. Em qual porta o JobTracker irá ser executado
cat <<EOF > $HADOOP_DIR/conf/mapred-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
  <name>mapred.job.tracker</name>
  <value>${MASTER_HOSTNAME}:54311</value>
  <description>The host and port that the MapReduce job tracker runs
  at.  If "local", then jobs are run in-process as a single map
  and reduce task.
  </description>
</property>

</configuration>
EOF

#hdfs-site.xml
```

```
#Configuração da quantidade de replicações um bloco de arquivos terá. Normalmente o
    número é definido como a quantiadde de máquinas
cat <<EOF > $HADOOP_DIR/conf/hdfs-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>
  <name>dfs.replication</name>
  <value>${NUMERO_MAQUINAS}</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
  </description>
</property>

</configuration>
EOF

#=====================================================

#Adicionar configuracoes da maquina master
cat <<EOF > $HADOOP_DIR/conf/masters
$MASTER_HOSTNAME
EOF

#=====================================================

#Adicionar configuracões das maquinas slave (no caso a master também executará
    trabalhos)
cat <<EOF > $HADOOP_DIR/conf/slaves
$MASTER_HOSTNAME
$SLAVE_HOSTNAME
EOF

#=====================================================

#Criação da pasta de script
mkdir /home/hduser/scripts

#=====================================================

#Modificar proprietadade das pastas para hduser
chown -R hduser:hadoop /home/hduser/*
chown -R hduser:hadoop $HADOOP_DIR
```

II. CAPTURA DOS DADOS

Para a coleta dos dados o sistema de arquivos HDFS deve está em funcionamento. Para isso, é executado no **master** os comandos:

```
#Na maquina master
#Se estiver logado como root, logar como hduser
su - hduser
/usr/local/hadoop/bin/start-dfs.sh
```

Com isso, os daemons responsáveis pelo HDFS serão inicializados. Como o master já possui as configurações do Slave, tentará via SSH, conectar ao Slave e iniciar os daemons no mesmo. Após o comando, para verificar os daemons rodando no Master e no Slave, deve ser executado o seguinte comando:

```
#Comando para verificar os daemons rodando
jps
```

No Master deverá estar rodando o **NameNode**, **Secondary NameNode** e **DataNode**. No Slave deverá estar rodando somente o **DataNode**.

Já com o sistema de arquivos inicializado, o Apache Flume pode ser executado:

```
#!/bin/bash
#Verificar se esta logado com hduser

#su - hduser
/usr/local/hadoop/flume/bin/flume-ng agent \
--conf /usr/local/hadoop/flume/conf/ \
-f /usr/local/hadoop/flume/conf/flume.conf \
-Dflume.root.logger=DEBUG,console \
-n TwitterAgent
```

A partir deste comando será possível ver as informações de conexão e coleta dos dados proveniente do twitter. Adicionando a instrução "*nohup*" no começo do comando e a instrução "*&*" no final comando, o processo ficará como daemon, ou seja, será executado em segundo plano.

III. PROCESSAMENTO

No processamento, inicialmente definimos as funções de map e reducer. Os scripts são implementados na linguagem PHP.

III.1 FUNÇÃO MAPPER - MAPPER.PHP

O arquivo de mapeamento deve estar em "*/home/hduser/scripts/mapper.php*", com permissão para execução ("*chmod +x*").

```
#!/usr/bin/php

<?php
//mapper.php
mb_internal_encoding('UTF-8');
date_default_timezone_set('America/Sao_Paulo');

//SEGMENTACAO POR OPERADORA
$OPERADORAS['tim'][]    = 'timbrasil';
$OPERADORAS['vivo'][]   = 'vivoemrede';
$OPERADORAS['claro'][]  = 'clarobrasil';
$OPERADORAS['oi'][]     = 'digaoi';
$OPERADORAS['gvt'][]    = 'gvtoficial';
$OPERADORAS['gvt'][]    = 'gvt_suporte';
$OPERADORAS['net'][]    = 'netoficial';
$OPERADORAS['net'][]    = 'netatende';


//SEGMENTACAO POR SERVICO
$SERVICOS['telefonia'][]    = 'telefonia';
$SERVICOS['telefonia'][]    = 'fixo';
$SERVICOS['telefonia'][]    = 'celular';
$SERVICOS['telefonia'][]    = 'tel';
$SERVICOS['telefonia'][]    = 'cel';
$SERVICOS['telefonia'][]    = 'sinal movel';
$SERVICOS['telefonia'][]    = 'sinal de celular';
$SERVICOS['internet'][]     = 'internet';
$SERVICOS['internet'][]     = 'inet';
$SERVICOS['internet'][]     = 'conexao';
$SERVICOS['internet'][]     = 'net';
$SERVICOS['internet'][]     = '3g';
$SERVICOS['internet'][]     = 'sinal de internet';
$SERVICOS['internet'][]     = 'virtua';
$SERVICOS['internet'][]     = '4g';
$SERVICOS['atendimento'][]  = 'atendimento';
$SERVICOS['atendimento'][]  = 'call center';
$SERVICOS['atendimento'][]  = 'cancelamento';
$SERVICOS['atendimento'][]  = 'contato';
$SERVICOS['tv'][]           = 'tv';
```

```
$SERVICOS['tv'][]           = 'televisao';
$SERVICOS['marketing'][]    = 'loja';
$SERVICOS['marketing'][]    = 'oferta';
$SERVICOS['marketing'][]    = 'oferta';


//SEGMENTACAO POR SATISFACAO
$SATISFACOES['ruim'][] = 'ruim';
$SATISFACOES['ruim'][] = 'cancelar';
$SATISFACOES['ruim'][] = 'cancelem';
$SATISFACOES['ruim'][] = 'horrivel';
$SATISFACOES['ruim'][] = 'manutencao';
$SATISFACOES['ruim'][] = 'resolvam';
$SATISFACOES['ruim'][] = 'lenta';
$SATISFACOES['ruim'][] = 'instavel';
$SATISFACOES['ruim'][] = 'lerdo';
$SATISFACOES['ruim'][] = 'lerda';
$SATISFACOES['ruim'][] = 'nao funciona';
$SATISFACOES['ruim'][] = 'injusto';
$SATISFACOES['ruim'][] = 'ligacao cai';
$SATISFACOES['ruim'][] = 'ligacao so cai';
$SATISFACOES['ruim'][] = 'sem sistema';
$SATISFACOES['ruim'][] = 'nao consigo';
$SATISFACOES['ruim'][] = 'nao conseguindo';
$SATISFACOES['ruim'][] = 'parou';
$SATISFACOES['ruim'][] = 'porcaria';
$SATISFACOES['ruim'][] = 'complica';
$SATISFACOES['ruim'][] = 'tentando';
$SATISFACOES['ruim'][] = 'resiste';
$SATISFACOES['ruim'][] = 'persiste';
$SATISFACOES['ruim'][] = 'horrorosa';
$SATISFACOES['ruim'][] = 'decepciona';
$SATISFACOES['ruim'][] = 'corta';
$SATISFACOES['ruim'][] = 'dificil';
$SATISFACOES['ruim'][] = 'sinal cai';
$SATISFACOES['ruim'][] = 'funcionar';
$SATISFACOES['ruim'][] = 'tv trava';
$SATISFACOES['ruim'][] = 'pior';
$SATISFACOES['ruim'][] = 'cai';
$SATISFACOES['ruim'][] = 'caiu';
$SATISFACOES['ruim'][] = 'absurdo';
$SATISFACOES['ruim'][] = 'colaborar';
$SATISFACOES['ruim'][] = 'acabando meus creditos';
$SATISFACOES['ruim'][] = 'falha';
$SATISFACOES['ruim'][] = 'lixo';
$SATISFACOES['ruim'][] = 'sem internet';
$SATISFACOES['ruim'][] = 'sem telefone';
$SATISFACOES['ruim'][] = 'sem net';
$SATISFACOES['ruim'][] = 'sem servico';
$SATISFACOES['ruim'][] = 'espera';
$SATISFACOES['ruim'][] = 'limite';
$SATISFACOES['ruim'][] = 'nao consigo';
$SATISFACOES['ruim'][] = 'vergonha';
$SATISFACOES['ruim'][] = 'problema';
$SATISFACOES['ruim'][] = 'anatel';
$SATISFACOES['ruim'][] = 'chateacao';
$SATISFACOES['ruim'][] = 'desprezo';
```

```
$SATISFACOES['ruim'][] = 'pessima decisao';
$SATISFACOES['ruim'][] = 'cansei';
$SATISFACOES['ruim'][] = 'inferno';
$SATISFACOES['ruim'][] = 'ninguem';
$SATISFACOES['ruim'][] = 'cortar';
$SATISFACOES['ruim'][] = 'nada instalado';
$SATISFACOES['ruim'][] = 'abaca minha internet';
$SATISFACOES['ruim'][] = 'ja acabou';
$SATISFACOES['ruim'][] = 'devolve';
$SATISFACOES['ruim'][] = 'indisponivel';
$SATISFACOES['ruim'][] = 'nunca atende';
$SATISFACOES['ruim'][] = 'descaso';
$SATISFACOES['ruim'][] = 'palhacada';
$SATISFACOES['ruim'][] = 'sem atendimento';
$SATISFACOES['ruim'][] = 'fora do ar';


//Conexão com o banco de dados (na maquina slave modificar localhost para o IP do
    master)
$mysqli = new mysqli("localhost", "tweets_usr", "pfg2", "tweets_db");



$resultados = array();

//A cada linha entrando via STDIN...
while (($line = fgets(STDIN)) !== false) {

    //Cria a array
    $resultado = array();

    //Decodifica a linha (tweet) para um objeto (JSON para OBJETO)
    $tweet = json_decode($line);
    //Obtem o texto do tweet
    $texto = $tweet->text;
    //Manipula o texto (retira espacos no comeco e final e coloca todas as letras em
        minusculas)
    $texto = mb_strtolower(trim($texto));
    //Obtem as letras sem acentuacao e letras especiais
    $texto = iconv('UTF-8', 'ASCII//TRANSLIT', $texto);
    $texto = preg_replace("/[^a-z ]/","",$texto);


    //Ignora retweets
    if (strpos($texto, 'rt ') !== false)
        continue;

    if (strpos($texto, 'RT ') !== false)
        continue;


    //Segmentar por operadora
    //Procura no texto se existe alguma palavra com os palavras chaves da operadora
    foreach($OPERADORAS as $operadora => $palavras_chaves) {
        foreach($palavras_chaves as $palavra_chave) {
            if(mb_strtolower($tweet->user->screen_name) == $palavra_chave)
                break 2;
```

```
            if (strpos($texto, $palavra_chave) !== false) {
                $resultado['operadora'] = $operadora;
                break 2;
            }
        }
    }

    //Se não foi encontrado a operadora, descarta o tweet
    if(!isset($resultado['operadora']))
        continue;


    //segmentar por tipo de servico
    //Procura no texto se existe alguma palavra com os palavras chaves dos servicos
    foreach($SERVICOS as $servico => $palavras_chaves) {
        foreach($palavras_chaves as $palavra_chave) {
            if (strpos($texto, $palavra_chave) !== false) {
                $resultado['servico'] = $servico;
                break 2;
            }
        }
    }

    //Se não foi encontrado o servico, descarta o tweet
    if(!isset($resultado['servico']))
        continue;


    //segmentar por satisfacao
    //Procura no texto se existe alguma palavra com os palavras chaves das
        satisfações
    foreach($SATISFACOES as $satisfacao => $palavras_chaves) {
        foreach($palavras_chaves as $palavra_chave) {
            if (strpos($texto, $palavra_chave) !== false) {
                $resultado['satisfacao'] = $satisfacao;
                break 2;
            }
        }
    }

    //Se não foi encontrado o servico, descarta o tweet
    if(!isset($resultado['satisfacao']))
        continue;


    //Obtem o timestamp da data do tweet
    $timestamp = strtotime($tweet->created_at);

    //monta o TOKEN com as informações (Data, operadora e o serviço) e insere no
        vetor
    $resultados[] = strtotime(date('Y-m-d H:00:00', $timestamp)).'-'.$resultado['
        operadora'].'-'.$resultado['servico'];

}

//No final da função mapper e todas as linhas percorridas, é enviado ao STDOUT todos
```

```
    os tokens armazenados no vetor
foreach($resultados as $resultado)
    echo $resultado.PHP_EOL;

?>
```

III.2 FUNÇÃO REDUCER - REDUCER.PHP

O arquivo de redução deve estar em "*/home/hduser/scripts/reducer.php*", com permissão para execução ("*chmod +x*").

```
#!/usr/bin/php
<?
//reducer.php
//Configuração de localidade e encoding
mb_internal_encoding('UTF-8');
date_default_timezone_set('America/Sao_Paulo');

//Inicializa vetor das tokens
$tokens = array();

// Obtem a linha pela entrada STDIN / OBTEM O TOKEN (1 em cada linha)
while (($line = fgets(STDIN)) !== false) {

    //Manipula a linha retirando os espacos do comeco e do final
    $token = trim($line);

    //Se ja existe o mesmo token no vetor principal, incrementa. Se não, adiciona com
        valor 1 (ou seja 1 frequencia)
    if(isset($tokens[$token]))
        $tokens[$token] += 1;
    else
        $tokens[$token] = 1;

}

//Inicializa vetor da query para o banco de dados relacional
$query = array();

//Percorre os tokens com suas quantidades
foreach($tokens as $token => $quantidade) {

    //quebra o token para obter as informações do mesmo
    $insert = explode('-', $token);

    //Manipula a data para que fique somente com a hora (despresar os minutos e
        segundos)
    $insert[0] = date('Y-m-d H:00:00', $insert[0]);

    //Insere a quantidade no vetor
    array_push($insert, $quantidade);
```

```
    //Adiciona no vetor da query, os valores para serem adicionados no banco de dados
    $query[] = '("'.implode('","',$insert).'")';

    //Exibe na saida STDOUT os tokens com a respectiva quantidade
    echo "MASTER", chr(9), $token, chr(9), $quantidade, PHP_EOL;

}

//Junta o vetor separando com virgula para serem inseridos no BD
$query = implode(',', $query);


//Se existir algum token, existirá algo na variavel $query, portanto existe esta
    verificacao
if($query) {

    //Comando para insercao dos valores no banco de dados
    $query = '  INSERT INTO resumo
                    (data, operadora, servico, qntTweets)
                VALUES
                    '.$query.'
                ON DUPLICATE KEY UPDATE
                    qntTweets = qntTweets + VALUES(qntTweets)
    ';


    //Conexão com o banco de dados (na maquina slave modificar localhost para o IP do
        master)
    $mysqli = new mysqli("localhost", "tweets_usr", "pfg2", "tweets_db");


     if ($mysqli->connect_errno)
         exit();

     //Insere os dados no banco de dados relacional
     $query = $mysqli->query($query);



}

?>
```

III.3 MAPEAR E REDUZIR

Para processar os dados já armazenados, inicialmente deve ser inicializado os daemons do MapReduce. Para isso, na máquina **master**, executar o código:

```
#Na maquina master
#Se estiver logado como root, logar como hduser
su - hduser
/usr/local/hadoop/bin/mapred-start.sh
```

Com isso os daemons serão inicializados, tanto no master, quando nas máquinas escravas (se possuir). Na máquina master deverão ser inicializadas os daemons: **JobTracker** e **TaskTracker** (a máquina master também processará os dados). Na máquina escrava, somente o **TaskTracker**.

```
#Comando para verificar os daemons rodando
jps
```

Inicializado o sistema, o processamento das informações é feito pelo comando:

```
#!/bin/bash
#Verificar se esta logado com hduser

#su - hduser
/usr/local/hadoop/bin/hadoop jar /usr/local/hadoop/contrib/streaming/hadoop-streaming
    -1.2.1.jar \
-mapper /home/hduser/scripts/mapper.php \
-reducer /home/hduser/scripts/reducer.php \
-input tmp/*.json \
-output output
```

O sistema comecará o processamento das informações, criando os tokens com as funções de mapeamento, e sendo somados e inseridos no banco de dados relacional através da função de redução.