DOI: http://dx.doi.org/10.5769/C2011008

Utilização da Computação Distribuída para o Armazenamento e Indexação de Dados Forenses

Marcelo Antonio da Silva[1], and Romualdo Alves Pereira Júnior[2]
(1) Serviço de Perícias de Informática, Departamento de Polícia Federal, Brasília-DF, Brasil,
marcelosilva.mas@dpf.gov.br
(2) Chefe da Divisão de Informática, Agência Espacial Brasileira, Brasília-DF, Brasil,
romualdo.pereira@aeb.gov.br

Abstract — **This paper presents the use of distributed computing to perform the process of storage and indexing of data resulting from a forensic analysis.**

Key-words* — *distributed computing, data forensics, information retrieval, distributed file system, distributed indexing

Resumo — **Este trabalho apresenta a utilização da computação distribuída para realizar o processo de armazenamento e indexação de dados resultantes de uma análise forense.**

Palavras-chave* — *computação distribuída; dados forenses; recuperação da informação; sistema de arquivos distribuído; indexação distribuída

1. Introdução

Este trabalho apresenta um sistema distribuído que foi construído para realizar o armazenamento e a indexação dos dados resultantes de uma análise forense [1].

Atualmente a quantidade de dados forenses a analisar é cada vez maior [2]. Isto é decorrente tanto do constante crescimento da capacidade dos dispositivos de armazenamento computacional quanto da maior popularização destes equipamentos. Em um caso que envolva dezenas de dispositivos de armazenamento computacional, realizar a análise correlacionada neste volume de dados é uma tarefa com alto custo computacional [3].

Este custo computacional é elevado em virtude da necessidade de um espaço de armazenamento adequado para estes dados, além da alta demanda computacional que o processo de indexação requer. O processo de indexação é fundamental para uma posterior análise nestes dados forenses [4].

O sistema distribuído apresentado neste trabalho utiliza um sistema de arquivos distribuído para prover um espaço de armazenamento para os dados forenses com escalabilidade, disponibilidade e tolerância a falhas. Este mesmo sistema distribuído também realiza a indexação dos dados forenses. Serão apresentados os cenários que foram elaborados e utilizados para testar o comportamento do sistema de armazenamento e indexação distribuída de dados forenses.

Este trabalho possibilita que, posteriormente, possam ser desenvolvidas poderosas interfaces para pesquisa em um grande volume de dados. Este volume de dados estará armazenado em um mecanismo com recursos de escalabilidade, disponibilidade e tolerância a falhas, além de poder ser indexado com uma maior eficiência.

O trabalho está organizado em cinco partes: I – Introdução. II - Ferramentas, onde são apresentadas as ferramentas de interpretação de arquivos, indexação de arquivos, computação em grade e ferramentas de análise de dados forenses. III – Arquitetura do Sistema Distribuído, onde descrevemos em detalhes o fluxo de trabalho nas etapas de fornecimento de dados, prova de conceito e mecanismo de pesquisa; o projeto do sistema; e os processos distribuídos relacionados ao processamento, armazenamento e indexação. IV – Cenários de Teste para a realização de *benchmarking*. V – Conclusão, onde tecemos as considerações finais sobre o trabalho realizado e os resultados obtidos.

2. Ferramentas

Para o desenvolvimento do presente trabalho foi realizada uma pesquisa em ferramentas e algoritmos atuais e eficientes para serem utilizados. Esta pesquisa foi visando à reutilização e ao aperfeiçoamento de métodos e técnicas já existentes.

Na análise de cada ferramenta buscou-se verificar as que tivessem as características necessárias para a reutilização e combinação com outras ferramentas. Foram priorizadas as soluções com código-fonte aberto para possibilitar os aperfeiçoamentos e personalizações que fossem necessárias.

A. Interpretação de Arquivos

Para realizar a interpretação dos diferentes tipos de arquivos presentes nos dados forenses coletados foi selecionada a ferramenta *Apache Tika* [5]. Esta ferramenta tem código-fonte aberto, utiliza uma expansível técnica de interpretação de arquivos e possui uma comunidade de usuários ampla e ativa.

O *Apache Tika* é um *framework* desenvolvido na linguagem Java com o objetivo de ler o conteúdo de diferentes tipos de

arquivos. Este *framework* possui uma arquitetura capaz de adicionar *plugins* para possibilitar a leitura de novos tipos de arquivos. Isto possibilita a flexibilidade na expansão desta arquitetura. Atualmente mais de 200 tipos de arquivos podem ser corretamente interpretados neste ambiente, recuperando seu conteúdo e formatação [6].

B. Indexação de Arquivos

Para realizar a indexação de arquivos foi selecionada a ferramenta *Apache Lucene* [7]. Esta ferramenta tem código-fonte aberto, utiliza modernas técnicas de recuperação da informação e possui uma ampla comunidade de usuários.

O *Apache Lucene* é uma biblioteca, desenvolvida na linguagem Java, com o objetivo de criar e realizar pesquisas em índices, possibilitando a rápida localização de conteúdo em dados não estruturados. Para isto o *Lucene* utiliza modernas técnicas e algoritmos de recuperação da informação, além de estruturas de dados performáticas e bem documentadas para a operacionalização de seu funcionamento [8].

C. Computação em Grade

Para realizar a atividade de computação distribuída foi selecionada a ferramenta *Apache Hadoop* [16]**.** Esta ferramenta tem código-fonte aberto, utiliza técnicas de computação distribuída, implementando o algoritmo *MapReduce*, e possui uma comunidade de usuários ampla e ativa.

Para realizar o gerenciamento do ambiente distribuído e fornecer a estrutura lógica para a criação de tarefas paralelas o *Hadoop* implementa o algoritmo *MapReduce* [9].

1) Processamento Distribuído - MapReduce

O *MapReduce* é um modelo de programação para poder processar e gerenciar o processamento distribuído de determinado problema computacional. A abstração utilizada neste algoritmo foi baseada nas primitivas *map* e *reduce* da linguagem Lisp e de diversas outras linguagens funcionais [10].

Essencialmente este modelo de programação permite que usuários escrevam componentes *map/reduce* em um código com o estilo de códigos de linguagens funcionais. Estes componentes são organizados em um determinado fluxo de processamento para especificar seu paralelismo. O ambiente de execução do *MapReduce* monitora e gerencia a execução destes componentes em um ambiente distribuído, controlando os problemas típicos de computação distribuída, como: paralelismo, comunicação entre processos através de uma rede de computadores, balanceamento de carga e tolerância a falhas. Desta forma, o usuário foca mais na lógica de resolução do problema computacional do que com os detalhes de funcionamento de um sistema distribuído [11].

A função *Map* pega um par chave/valor como entrada e produz uma lista com funções chave/valor como saída. O tipo dos dados de chave/valor de saída são, tipicamente, distintos dos de entrada. As chaves geradas pelas funções *Map* também são chamadas de chaves intermediárias.

$$map: (chave_1, valor_1) \rightarrow lista(chave_2, valor_2) \quad (1)$$

A função *Reduce* pega a chave e a lista dos valores associados a esta chave como entrada e gera uma lista com novos valores de saída.

$$reduce: (chave_2, lista(valor_2)) \rightarrow lista(valor_3) \quad (2)$$

Um aplicativo *MapReduce* é executado de forma paralela em duas fases. Na primeira fase, todas as funções *Map* podem ser executadas de forma independente uma da outra. Na segunda fase, as funções *Reduce* são executadas na medida em que as funções *Map* vão gerando os seus resultados de saída. Todas as funções *Reduce* podem ser executadas de forma independente uma da outra. Esta execução independente, tanto da função *Map* quanto da função *Reduce*, permite que o problema computacional possa ser processado de forma paralela. Diversos problemas computacionais podem ser resolvidos utilizando-se deste método de programação. Exemplos de problemas computacionais já resolvidos utilizando-se deste método são: criação de índices invertidos, ordenação de dados, processamento de imagens e mineração de dados [9].

O ambiente de execução *MapReduce* realiza diversas típicas de um ambiente distribuído, como: a divisão dos dados de entrada para que cada função *Map* possa processar uma parte destes dados, o balanceamento de carga através da seleção de quais computadores executarão a função *map* ou a função *reduce*, controle de falhas nas máquinas que executam as funções, gerenciamento da comunicação entre as máquinas do ambiente distribuído e ordenamento das chaves intermediárias na medida em que as funções *Map* são executadas. A Fig. 1 ilustra o funcionamento geral do *MapReduce*.

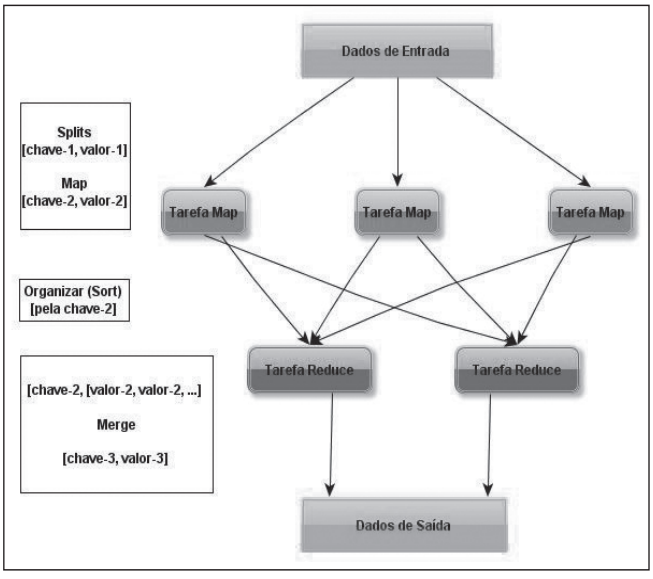

Figura 1: Gráfico esquemático da funcionalidade do *MapReduce*

No gráfico da Fig. 1 é ilustrado um conjunto de dados de entrada sendo distribuído para três tarefas *Map*. Cada conjunto de dados de entrada pode ser divido de diversas formas e deve ser fornecida para cada tarefa Map uma estrutura de dados na forma *chave/valor*. O usuário pode implementar o método de divisão que for mais apropriado para sua aplicação. Cada conjunto *chave/valor* que é fornecido para a função *Map* é chamado de *Split*. Após cada tarefa *Map* receber os dados (*Splits*) de entrada no formato $chave_1/valor_1$, ela realiza o processamento destes dados e gera os dados de saída no formato $chave_2/valor_2$.

Após as tarefas *Map* gerarem diversos resultados de saída no formato $chave_2/valor_2$ o ambiente de execução do *MapReduce* agrupa estas chaves intermediárias de acordo com a quantidade de tarefas *Reduce* existentes. O nome dado a este agrupamento chama-se particionamento. Os valores agrupados de cada chave são fornecidos em um formato de lista $chave_2/lista(valor_2)$, conforme foi ilustrado na Equação 3. A fórmula padrão de particionamento é:

$$particionamento: \quad hash(chave_2) \mod R \qquad (3)$$

Na Equação 3 *R* é a quantidade de tarefas *Reduce* presentes no processamento e *hash* é uma função *hash (SHA-1* ou *MD-5*) [12]. Isto tende a resultar em partições bem balanceadas para distribuir a tarefas *Reduce*. Esta fórmula também pode ser personalizada pelo usuário.

Cada tarefa *Reduce* receberá uma estrutura de dados no formato *chave/valor*, onde *valor* será uma lista dos valores agrupados para cada *chave*. Isto está exemplificado na Fig. 1 como *[chave-2, valor-1, valor-2, ...]*, ilustrando os diversos valores que foram agrupados pela *chave-2*. Finalmente cada tarefa *Reduce* realiza o seu processamento nos dados recebidos gerando os dados de saída no formato *[chave-3, valor-3]*.

A Fig. 2, contém um exemplo de implementação do algoritmo *MapReduce* que permite visualizar o seu comportamento para resolver determinado problema computacional. No caso deste exemplo o problema computacional a ser resolvido é contar quantas palavras diferentes existem em todos os arquivos que forem fornecidos.

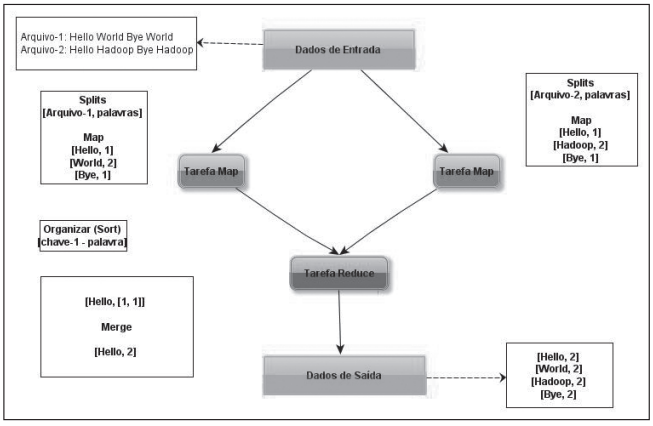

Figura 2: Exemplo de funcionamento do *MapReduce*

No exemplo da Fig. 2 são fornecidos dois arquivos: *(Arquivo-1 e Arquivo-2)*. O ambiente *MapReduce* divide estes dados de entradas em dois *Splits*. Cada *Split* contém o nome do arquivo e todas as palavras contidas no interior do arquivo. Cada tarefa *Map* recebe estes dados de entrada e gera como saída uma estrutura de dados no formato *Palavra/Quantidade*. A saída de cada tarefa *Map* representa a quantidade de cada palavra encontrada. Posteriormente o ambiente *MapReduce* organiza os dados intermediários gerados pela tarefa *Map*, onde agrupa em partições cada chave gerada e a lista de valores resultantes. Como exemplo, para a palavra *Hello*, que foi contata 01 unidade em cada tarefa *Map*, será gerada uma estrutura no formato: *Hello/[1, 1]*. Esta estrutura contém a palavra encontrada e uma lista com o somatório das ocorrências desta palavra que cada tarefa *Map* calculou. Finalmente a tarefa *Reduce* recebe os dados de entrada no formato indicado e finaliza o somatório das palavras contidas nos arquivos fornecidos.

O exemplo de implementação do algoritmo *MapReduce* ilustrado na Fig. 2 ilustra como uma tarefa computacional pode ser representada através de funções *map* e *reduce* para resolver um problema computacional de forma paralela. O processamento realizado por cada tarefa *Map* pode ser executado em máquinas diferentes e de forma paralela. O mesmo também ocorre com a tarefa *Reduce* que na medida em que as tarefas *Map* forem gerando seus resultados também pode executar de forma paralela com outras tarefas *Reduce*, por ventura existente.

Na arquitetura de controle do *MapReduce* uma máquina do ambiente distribuído deve assumir a tarefa de Nó Mestre. O papel deste processo é manter uma estrutura de dados para cada tarefa *Map* e *Reduce*, mantendo dados como: seus estados (inativo, processando e finalizado) e sua identidade. De posse destes dados, este Nó Mestre gerencia o processo de balanceamento de carga, distribuindo as tarefas para a máquina mais apropriada.

Para prover tolerância a falhas o Nó Mestre se comunica com as outras máquinas periodicamente. Se um determinado processo não enviar informação por certo período de tempo é configurado que houve falha nas tarefas que ele processa. Todas as tarefas que ele estava executando voltam ao estado inicial e o Nó Mestre seleciona outro nó com o status inativo para executar a tarefa. Isto proporciona um mecanismo de tolerância a falhas transparente para o usuário que implementa tarefas *map* e tarefas *reduce*.

O ambiente *MapReduce* implementa um método para facilitar a localidade de dados [13]. A largura de banda é um recurso caro em um ambiente distribuído, principalmente quando se processa uma grande quantidade de dados. Desta forma, minimizar a necessidade de transferência de dados na rede é um recurso altamente desejável no ambiente distribuído. A implementação padrão do *MapReduce*, feita pela empresa *Google* [14], utiliza o sistema de arquivos distribuído *GFS – Google File System* [14]. O GFS divide cada arquivo em blocos de 64 MB e armazena diversas cópias (03

por padrão) de cada bloco em diferentes máquinas. O Nó Mestre, quando vai alocar uma tarefa *map* ou *reduce* para ser executada, tenta localizar uma máquina disponível que possui uma réplica dos dados a processar. Se isto falhar ele tenta alocar a máquina mais próxima. A utilização de um sistema de arquivo distribuído no *MapReduce* não é obrigatória mas possibilita que a localidade de dados seja maximizada no ambiente distribuído.

Diversas tarefas *map* e tarefas *reduce* podem ser executadas em cada computador do ambiente distribuído, dependendo da configuração de cada máquina. Quanto maior a quantidade de processos *map* e processos *reduce*, melhor será o balanceamento de cargas realizado e o procedimento de recuperação em caso de falhas.

Outro recurso útil do ambiente *MapReduce* é a existência de contadores distribuídos, que podem ser utilizados para realizar relatórios de comportamento e depuração das funcionalidades criadas.

O *Hadoop* implementa e possibilita a utilização das principais características do *MapReduce*. As que serão destacadas no presente trabalho são:

- A criação de tarefas *Map* e tarefas *Reduce* para possibilitar a resolução de um problema computacional em um ambiente distribuído. Estas funcionalidades são escritas por padrão na linguagem Java, mas também podem ser escritas em outras linguagens, como: Ruby, Python e C++;
- Existência de um Nó Mestre, chamado de *JobTracker* que é responsável por monitorar e controlar todo o processamento distribuído do ambiente MapReduce. Através deste controle os mecanismos de balanceamento de carga, disponibilidade, localidade de dados e tolerância a falhas são implementados, de acordo com o procedimento padrão do *MapReduce*;
- Existência de um sistema de arquivos distribuído padrão para ser utilizado como unidade de armazenamento de dados, o HDFS. Além de recursos inerentes a este tipo de sistema, conforme será descrito na próxima seção, isto possibilita a maximização da localidade de dados no processamento de uma tarefa distribuída;
- Existência de interfaces de monitoramento e controle da execução do ambiente distribuído e de cada tarefa submetida a processamento;
- Possibilidade de personalização das diversas atividades desempenhadas pelo ambiente *MapReduce*, como: divisão do volume de dados a ser processado, método de particionamento das chaves intermediárias e estruturas de dados de cada fase de processamento.

Desta forma, para processar um trabalho, o ambiente do *Hadoop* primeiramente divide o volume de dados de entrada em diversos pedaços, denominados *Splits*. Cada *Split* é processado independentemente e paralelamente em tarefas *Map*. O ambiente ordena os dados gerados pelas tarefas *Map* e encaminha estes dados para as tarefas *Reduce*. Por padrão, tanto a entrada quanto a saída destas tarefas são gravadas no sistema de arquivos configurado. O ambiente de execução do *Hadoop* mantém o controle da alocação de tarefas entre os processos *Map* e *Reduce*, realizando o monitoramento do processamento das atividades e realizando a re-execução de determinadas tarefas em caso de falhas [18].

Quando o sistema de arquivos definido no *Hadoop* é um sistema de arquivos distribuído, como o HDFS, cada nó do ambiente distribuído realiza tanto o armazenamento dos dados quanto o processamento dos mesmos. Esta configuração permite que o ambiente de execução do *Hadoop* possa alocar tarefas para os nós que possuem armazenados os dados que serão processados. Isto resulta em um baixo consumo de largura de banda da rede, através da implementação do conceito de localidade de dados.

Para implementar o algoritmo *MapReduce* o *Hadoop* mantém dois tipos de processos principais no ambiente distribuído. Um destes processos chama-se *JobTracker*. Este processo implementa as atribuições do Nó Mestre do algoritmo *MapReduce* e, quando ativo, fica presente

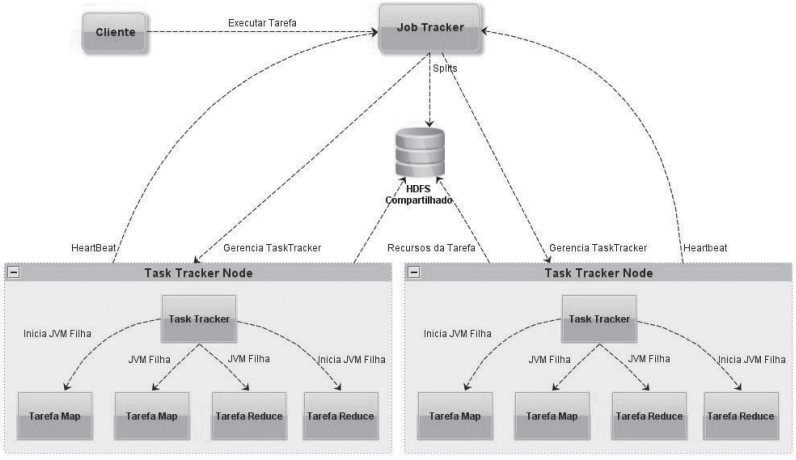

Figura 3: Implementação do *MapReduce* pelo *Hadoop*

em apenas 01 nó do ambiente. O outro tipo processo se chama *TaskTracker* e tem a responsabilidade de gerenciar o processamento das tarefas *Map* e *Reduce*. Desta forma, o processo do tipo *TaskTracker* está presente em cada nó de processamento do ambiente distribuído. A Fig. 3, mostra de forma mais detalhada como o *Hadoop* implementa o processamento distribuído do *MapReduce*.

Conforme ilustrado na Fig.3, primeiramente um processo cliente solicita a execução de um determinado processamento distribuído para o processo Nó Mestre, que se chama *JobTracker*. Neste momento o processo cliente fornece a localização das classes que implementarão as atividades de processamento distribuído (tarefas *Map* e tarefas *Reduce*). O processo *JobTracker* mantém o controle de todos os nós do ambiente distribuído e realiza a inicialização de cada nó para realizar o processamento. Neste processo de inicialização, os nós de processamento recebem o código de implementação das tarefas *Map* e *Reduce*, além das informações de localização de cada dado que será processado, denominado *Split*. O processo de nome *TaskTracker* é responsável por controlar toda a atividade de processamento que o nó deve realizar. Para o processamento das tarefas, o processo *TaskTracker* inicializa e gerencia processos filhos, que executarão as tarefas *Map* e *Reduce* que foram implementadas. O *TaskTracker*, periodicamente, encaminha ao *JobTracker* mensagens contendo os dados de progresso das atividades de estão sendo realizadas. Estas mensagens são denominadas *Heartbeat*. Geralmente, os dados de cada *Split* que o processo *JobTracker* seleciona para processamento já se encontram no nó do *TaskTracker*. Caso o *TaskTracker* necessite de mais dados que não estão em seu nó ele realiza a cópia dos dados através do sistema de arquivos distribuído configurado. De posse das mensagens de controle que o *TaskTracker* encaminha, as *Heartbeats*, o processo *JobTracker* tem as informações necessárias para realizar o monitoramento e gerenciamento das atividades realizadas por todos os nós do ambiente distribuído.

2) Armazenamento Distribuído – HDFS

A arquitetura *MapReduce* recomenda que seja utilizado um sistema de arquivos distribuído para possibilitar a localidade de dados. O projeto *Apache Hadoop* desenvolveu seu próprio sistema de arquivos distribuído, chamado de *Hadoop Distributed File System – HDFS*. O HDFS foi fortemente baseado no funcionamento do GFS. Apesar de o HDFS ser o sistema de arquivos distribuído padrão para o *Hadoop* a sua utilização não é obrigatória. Qualquer outro sistema de arquivos distribuído pode ser utilizado ou mesmo um sistema de arquivos não distribuído como o NTFS ou FAT32, apesar de não ser recomendado em ambiente de produção. A implementação padrão do GFS e do *MapReduce* pela *Google* é utilizando a linguagem C++, enquanto a implementação do *Hadoop* e do HDFS é através da linguagem Java.

O sistema de arquivos distribuído selecionado no presente trabalho foi o *Hadoop Distributed File System – HDFS* [19]. Esta ferramenta tem código-fonte aberto, utiliza modernas técnicas de implementação de um sistema de arquivos distribuído e possui uma comunidade de usuários ampla e ativa. Além disto, possui forte integração com o *Apache Hadoop* facilitando assim a integração das duas soluções.

O HDFS é um sistema de arquivos desenvolvido para armazenar arquivos com megabytes, gigabytes ou mesmo terabytes de tamanho. A arquitetura do HDFS foi concebida para aplicações destinadas a escrever uma vez o arquivo e poder ler diversas vezes, ou seja, para aplicações com alto fluxo de dados. Nestas aplicações um grande volume de dados é escrito uma vez e posteriormente várias leituras e análises são realizadas nestes dados. O HDFS não necessita de poderosos recursos computacionais para operar, ele pode ser construído com um conjunto de máquinas comuns e heterogêneas. Além disto, o HDFS implementa recursos de balanceamento de cargas, disponibilidade, localidade de dados e tolerância a falhas [19].

Um conceito fundamental do HDFS é o conceito de bloco de dados que ele utiliza. Discos rígidos e sistemas de arquivos tradicionais, como o NTFS e o FAT32, também possuem o conceito de bloco de dados, que é a menor unidade de alocação de dados. O tamanho tradicional de bloco de dados em discos rígidos é 512 bytes. Os sistemas tradicionais de arquivos possuem um número múltiplo deste valor, sendo, em geral, 4 KB. Já no HDFS o tamanho padrão do bloco de dados é 64 MB. Como em um sistema de arquivo tradicional, um arquivo no HDFS é dividido em diversos blocos, sendo que isto é transparente para o usuário que utiliza o sistema. No HDFS, quando um dado não ocupa todo o tamanho de um bloco o espaço em disco não é perdido. Além disto, o bloco de dados de um arquivo pode ser armazenado em diversos computadores integrantes da rede do sistema de arquivos distribuído.

O conceito de blocos traz grandes benefícios para o HDFS, provendo simplicidade nas tarefas que ele deve desempenhar. Não existe um tamanho máximo de arquivo para o HDFS, este tamanho depende apenas da quantidade e capacidade de máquinas e de discos rígidos que integram a rede do sistema de arquivos distribuído.

Os blocos permitem que o HDFS implemente um eficiente método de replicação de dados. Quando um bloco de dados é gravado em uma máquina, esta máquina replica este mesmo bloco para outra máquina. A outra máquina também pode replicar o bloco para uma terceira máquina e assim por diante. A quantidade de réplicas que cada bloco terá é definida por um parâmetro chamado fator de replicação. O valor padrão do fator de replicação é três e pode ser definido para cada arquivo presente no sistema de arquivos. O processo de réplica de blocos que o HDFS implementa é chamado de réplica pipeline, pois uma vez que o bloco é copiado para uma máquina, automaticamente esta máquina copia para outra e assim por diante, realizando todas estas cópias em praticamente o mesmo tempo.

A réplica de blocos permite que o HDFS implemente os recursos de tolerância a falhas, disponibilidade, integridade

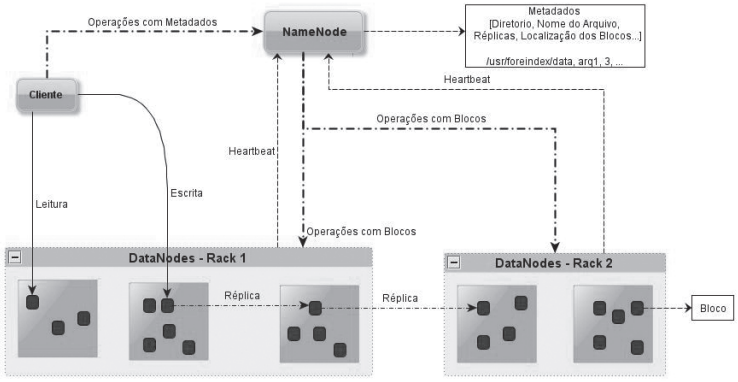

Figura 4: Funcionamento geral do HDFS

e localidade de dados. Quando uma determinada máquina do ambiente falhar o HDFS pode utilizar os blocos do arquivo contidos em outras máquinas para fornecer os dados do arquivo. Isto proporciona um eficiente mecanismo de tolerância a falhas que são tão comuns em um ambiente com diversas máquinas heterogêneas, fazendo com que o sistema de arquivos tenha um bom grau de disponibilidade. Como os blocos são divididos em diversas máquinas do ambiente, o processamento de determinada tarefa pode ser implementado na máquina que já possui o bloco, diminuindo consideravelmente o volume de dados que trafega na rede. Além disto, cada bloco possui um código de checagem de integridade que é calculado no momento em que o bloco é criado. Sempre que um bloco é lido este código é verificado, se houver um erro este bloco é considerado corrompido e um bloco de outra máquina será utilizado.

Para implementar estas funcionalidade o HDFS possui dois tipos de processos principais, uma chama-se *NameNode* e o outro *DataNode*. O *NameNode* gerencia o sistema de arquivos distribuído. Para isto ele armazena a árvore de diretórios do sistema e todos os metadados de arquivos e diretórios. Esta informação é mantida tanto na memória RAM quanto armazenada consistentemente no sistema de arquivos da máquina que implementa o *NameNode*. O *DataNode* é o processo que implementa o armazenamento do bloco de dados propriamente dito. Com exceção da máquina que implementa o *NameNode* em um ambiente HDFS todas as outras máquinas implementam um processo *DataNode*.

O *NameNode* conhece todos os *DataNodes* do ambiente que possuem dados armazenados, contudo ele não armazena persistentemente a localização dos blocos de dados. Estas informações são enviadas dinamicamente pelos *DataNodes* na medida que eles são inicializados.

O sistema HDFS é acessado através do *NameNode*. O *NameNode* realiza o monitoramento e gerenciamento do sistema de arquivos distribuído, além de regular o acesso dos arquivos pelos clientes. Os *DataNodes* gerenciam o armazenamento dos blocos de dados em cada nó do cluster. O *NameNode* realiza operações do sistema de arquivos como os processos de abertura, fechamento e troca de nomes de arquivos e diretórios. Ele também determina quais blocos de dados cada *DataNode* irá armazenar. Os *DataNodes* são responsáveis por servir requisições de leitura e escrita de dados no sistema de arquivos. Os *DataNodes* são responsáveis pelas operações de criação e exclusão de blocos de dados, assim como o processo de replicação de acordo com as instruções recebidas do *NameNode*.

A Fig. 4 ilustra o funcionamento geral do *Hadoop Distributed File System - HDFS*.

Neste diagrama é ilustrado que o cliente solicita a realização de operações de metadados como criação, exclusão e leitura de arquivos ao *NameNode*. O *NameNode* possui armazenado os metadados do arquivo, como: o diretório, o nome do arquivo, a quantidade de réplica dos blocos e a localização dos blocos de cada arquivo. De posse destas informações o *NameNode* pode gerenciar o processo de gravação de arquivos, de leitura, de réplica, controle de falhas e disponibilidade do ambiente. Os *DataNodes* periodicamente encaminham ao *NameNode* mensagens indicando o status de armazenamento e a quantidade de blocos de dados que eles armazenam. Estas mensagens recebem o nome de *Heartbeat*. O *NameNode* encaminha a localização dos *DataNodes* que possuem os blocos de dados que o cliente solicitou. Os clientes se conectam diretamente com os *DataNodes* para realizar o processo de leitura e de escrita de dados. Quando um bloco de dados é copiado para um *DataNode* este bloco é automaticamente replicado para outro *DataNode*, de acordo com o fator de replicação fornecido pelo *NameNode*.

D. Forenses

O sistema distribuído apresentado neste trabalho realiza a indexação dos arquivos ativos que forem disponibilizados em seu ambiente de armazenamento e processamento. Para realizar o processamento forense como: extração de arquivos apagados, eliminação ou seleção de arquivos já conhecidos e reconhecimento de formatos de dados com extensões trocadas são utilizadas ferramentas forenses já existentes.

Qualquer ferramenta forense pode ser utilizada para implementar esta tarefa disponibilização de arquivos para o sistema distribuído de armazenamento e indexação. As

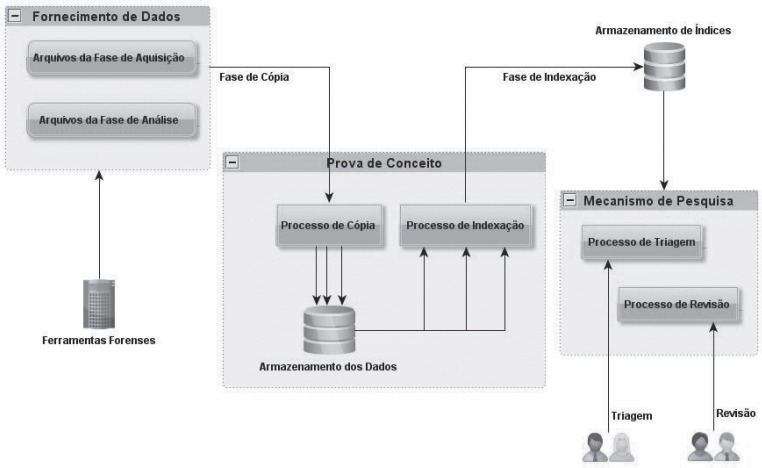

Figura 5: Fluxo de trabalho do sistema distribuído elaborado

ferramentas selecionadas e utilizadas no presente trabalho foram: o *Forensic ToolKit* – FTK, *EnCase Forensic* e o *Sleuth Kit* – TSK [20, 21, 22].

3. Arquitetura do Sistema Distribuído

Este capítulo apresenta a arquitetura da prova de conceito proposta com as personalizações que foram necessárias em cada ferramenta se contemplar a finalidade do presente trabalho.

A. Fluxo de Trabalho

Nesta seção será apresentado o fluxo de trabalho do sistema distribuído elaborado. Serão descritas as atividades que são realizadas em cada fase do processo de análise de dados forenses utilizando computação distribuída.

O escopo do presente trabalho é utilizar a computação distribuída para poder prover espaço de armazenamento para os dados e melhorar eficiência computacional no processo de indexação. Para isto é necessário a existência de um volume de dados forenses a indexar. Após o processo de indexação poderá ser realizada pesquisas e análises diversas nos dados já indexados. A Fig. 5 ilustra o fluxo de trabalho do sistema elaborado.

1) Fornecimento de Dados

Na esquerda do fluxo de trabalho da Fig. 5 estão ilustrados os processos responsáveis pelo fornecimento dos dados forenses para a prova de conceito. Os dados forenses que são fornecidos para a prova de conceito podem ser provenientes tanto da fase de aquisição de dados quanto da fase de análise de dados do processo pericial. Quando os arquivos são provenientes da fase de aquisição de dados, todos os arquivos ativos e apagados que puderam ser recuperados das mídias de armazenamento computacional estarão disponíveis. Quando os arquivos são resultantes da fase de análise, serão fornecidos apenas os arquivos mais relevantes para determinado caso, pois já foi realizado um processo de análise pericial anterior. As fases de aquisição de arquivos e análise forense são realizadas por ferramentas periciais disponíveis no mercado como o *Forensic ToolKit* – FTK, *EnCase Forensic* ou o *Sleuth Kit* – TSK [20, 21, 22].

2) Prova de Conceito

O agrupamento central da Fig. 5 ilustra que o trabalho a ser realizado pela prova de conceito é desempenhado em duas fases. A primeira fase é a fase de cópia. Nesta fase os dados forenses recuperados pelo processo de fornecimento de dados são copiados para o ambiente distribuído da prova de conceito. A segunda fase é a fase de indexação. Na fase de indexação os dados forenses disponíveis no ambiente da prova de conceito são indexados. O índice criado é gravado em um espaço de armazenamento para posteriormente ser utilizado por um mecanismo de pesquisa, que poderá realizar a pesquisa e análise nos dados forenses já indexados. Tanto o processo de cópia dos dados quanto o processo de indexação é realizado de forma distribuída. O local de armazenamento dos dados forenses se encontra no próprio ambiente distribuído da prova de conceito. O local de armazenamento do índice criado se encontra em um espaço externo ao ambiente distribuído.

3) Mecanismo de Pesquisa

O processo de indexação possibilita a realização posterior de um mecanismo de pesquisa. Este mecanismo está ilustrado no agrupamento da direita da Fig. 5. Neste mecanismo de pesquisa podem ser realizados dois processos principais, um processo de triagem e outro de revisão dos dados. O processo de triagem realiza uma pesquisa nos dados adquiridos e indexados podendo verificar quais são os dados relevantes para determinado caso. O processo de revisão realiza uma pesquisa e análise nos dados resultantes de uma análise pericial e já indexados, podendo obter mais informações sobre determinado caso. Tanto o processo de triagem quanto o de revisão possibilitam uma análise correlacionada das informações presentes nas diversas mídias de armazenamento computacional copiadas para o ambiente da prova de conceito.

O processo de fornecimento de dados e o processo de pesquisa dos dados estão fora o escopo principal do sistema elaborado. O escopo principal deste sistema é realizar a indexação dos dados de forma mais eficiente e prover um meio de armazenamento escalável e com tolerância a falhas para os dados forenses. Contudo, a prova de conceito apresenta interfaces que possibilitam a integração com os processos de fornecimento e pesquisa nos dados forenses. No

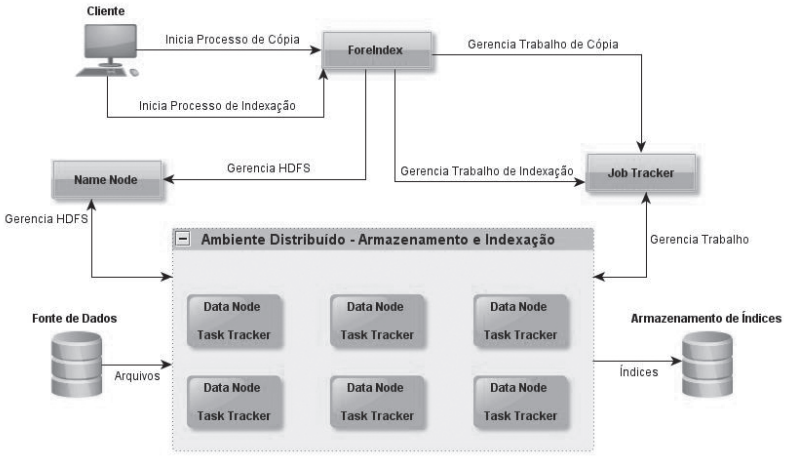

Figura 6: Projeto do sistema distribuído elaborado

presente trabalho foi desenvolvido um módulo para simular as fases de fornecimento e pesquisa dos dados, de forma a avaliar o trabalho realizado.

B. Projeto do Sistema

Nesta seção será apresentado o projeto do sistema elaborado. Os diversos componentes que integram este projeto serão analisados, demonstrando o funcionamento geral da arquitetura. A Fig. 6 ilustra o projeto do sistema distribuído elaborado.

A classe de nome *ForeIndex* é a classe responsável por controlar os dois processos que serão realizados pela prova de conceito, ou seja, o processo de cópia de dados forenses e o processo de indexação. Um processo cliente aciona a classe *ForeIndex* para iniciar o processo de cópia ou o processo de indexação, fornecendo os parâmetros necessários para iniciar cada uma destas atividades.

A classe *ForeIndex* se comunica tanto com o *JobTracker* quanto com o *NameNode* para poder realizar o processamento distribuído das atividades de cópia e indexação. O *NameNode* gerencia o sistema de arquivos distribuído. O *JobTracker* gerencia todo o processamento distribuído que será realizado de acordo com o algoritmo *MapReduce*. Tanto o *JobTracker* quanto o *NameNode* são componentes do *Apache Hadoop* e todas as funcionalidades desta solução está disponível no ambiente do sistema elaborado. O agrupamento central da Fig. 6 ilustra o sistema distribuído que foi construído. Cada nó deste ambiente distribuído contém um *TaskTracker* e um *DataNode*. Isto significa que cada nó do ambiente distribuído possui um processo responsável por realizar o processamento *MapReduce*, o *TaskTracker*, e outro processo responsável por armazenar blocos de dados do sistema de arquivos distribuído, o *DataNode*. Desta forma, o ambiente distribuído da prova de conceito realiza tanto o armazenamento quanto o processamento dos dados forenses.

Este ambiente distribuído se comunica com duas fontes externas de dados, uma que contém os dados forenses que serão copiados para o ambiente e a outra que armazenará os índices que forem criados. O armazenamento dos índices em um ambiente externo de armazenamento contribui para um melhor desempenho do sistema. Pois o processo de gravação do índice não impactará o processo de leitura dos dados que estarão sendo lidos em outro ambiente de armazenamento. Também irá acelerar o processo de pesquisa, pois um índice é acessado de forma randômica e o acesso randômico a arquivos não é tão eficiente no HDFS, que foi criado para o acesso de dados através de um fluxo de transmissão contínuo [23].

C. Processos

Esta seção descreverá os processos realizados pelo sistema elaborado. Será detalhado o funcionamento dos processos bem como as personalizações que foram necessárias nas ferramentas utilizadas para realizar cada atividade.

1) Processamento Distribuído

Todos os processos descritos na presente seção utilizam o mecanismo de processamento distribuído do *Apache Hadoop*, que é o algoritmo *MapReduce* já descrito anteriormente. Todas as funcionalidades deste algoritmo estão presentes em cada processo descrito a seguir.

2) Armazenamento Distribuído – Processo de Cópia

Esta subseção descreverá o processo distribuído de cópia de arquivos. Será detalhado o funcionamento deste processo bem como as personalizações que foram necessárias nas ferramentas utilizadas para realizar esta atividade. Serão também analisados os requisitos para a cópia de dados forenses e as características do processo distribuído que contemplou estes requisitos.

Um conjunto de dados forenses geralmente possui algumas particularidades que requerem tratamento diferenciado no processamento distribuído gerenciado pelo *Apache Hadoop*. O volume dados forenses a analisar está cada vez maior [2]. Com isso a quantidade e tipo de dados forenses a analisar

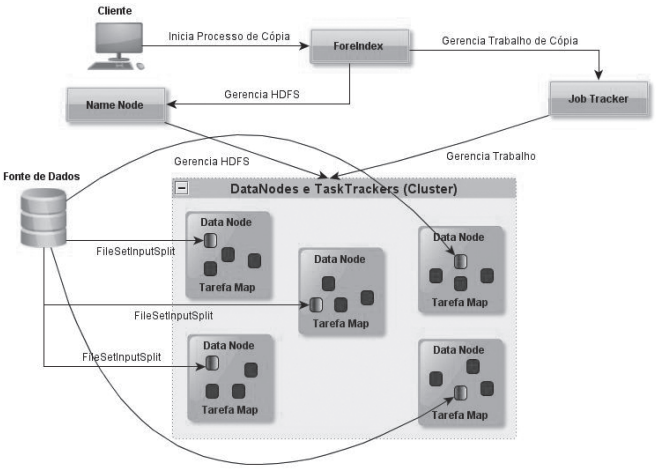

Figura 7: Processo de cópia distribuída de arquivos

tem crescido. Estes arquivos, em geral, possuem um tamanho menor que o tamanho de um bloco de dados do HDFS, que tem um tamanho padrão de 64 MB. Este tamanho do bloco tem o objetivo de melhorar a performance do ambiente [16]. Além disto, a maioria dos arquivos não podem ser divididos para serem processados, conforme é o procedimento padrão no *Apache Hadoop*. Os arquivos possuem diferentes formatos e precisam ser lidos por inteiro de forma a realizar uma interpretação correta do conteúdo de cada arquivo.

Desta forma, é necessário fazer uma personalização no *Hadoop* de forma a processar o arquivo completo em cada nó de processamento, além de apenas partes do arquivo. Para realizar a interpretação correta do conteúdo de muitos arquivos é necessário que seus dados sejam lidos do início ao fim.

Além disto, para poder tornar o processamento mais eficiente no ambiente distribuído do *Hadoop* é necessário que cada arquivo tenha pelo menos o tamanho do bloco de dados, que é 64 MB. Muitos arquivos com tamanhos menores do que este torna o processamento *MapReduce* ineficiente pois muitos recursos computacionais serão alocados para processar um pequeno volume de dados [16].

Para solucionar estas limitações foi desenvolvida uma classe chamada *FileSetInputSplit*. Esta classe é responsável por personalizar o processo de leitura do volume de dados a processar, realizando a divisão dos dados para os diversos nós do ambiente distribuído. Ao invés de os arquivos serem divididos em pedaços e distribuídos para o ambiente distribuído o que esta classe faz é realizar um agrupamento dos arquivos. Se o arquivo tiver um tamanho menor que o tamanho do bloco de dados ele é agrupado com outros arquivos até atingir o tamanho do bloco de dados. Se o arquivo for maior que o tamanho do bloco de dados ele não será agrupado com outros arquivos.

Cada arquivo ou grupo de arquivos é agrupado em um objeto chamado de *FileSet*. Este objeto é um tipo especial de classe que estende uma classe padrão do *Hadoop* que se chama *SequenceFile*. O *SequenceFile* é um tipo de dados que o *Hadoop* fornece para estender os tipos de dados já existentes. Desta forma, a classe *FileSetInputSplit* será utilizada para criar diversos objetos *FileSets*. Os objetos *FileSets* são tipos especiais de arquivos, do tipo *SequenceFile*, que terão agrupados um ou mais arquivos. O tamanho mínimo de cada *FileSet* será o tamanho do bloco de dados HDFS, por padrão 64 MB. Isto melhorará o desempenho do processamento destes dados. A Fig. 7 ilustra como é implementado o processo de cópia distribuída de arquivos.

Para poder ser realizada a indexação dos dados forenses é necessário que estes dados estejam no sistema de arquivos distribuído do ambiente do sistema elaborado. A fase de cópia de arquivos é responsável por copiar os dados de uma fonte de dados externa para o ambiente distribuído criado. A classe *ForeIndex* controla o processo de cópia distribuída. O processo cliente fornece para a classe *ForeIndex* os dados necessários para iniciar o processo, como a origem da fonte de dados e o diretório de destino do HDFS para o qual os dados serão copiados.

A classe *ForeIndex* se comunica com o *JobTracker* para poder distribuir o processamento que será realizado em cada nó do ambiente distribuído. Cada nó do ambiente distribuído realiza, de forma paralela, o processo de cópia, copiando uma determinada parcela do volume de dados. Isto é implementado através de uma tarefa *Map* que cada nó possui, seguindo as diretrizes do algoritmo *MapReduce*. Cada tarefa *Map* do ambiente distribuído utiliza a classe *FileSetInputSplit* para recuperar os arquivos que serão copiados. A classe *FileSetInputSplit* fornecerá os arquivos agrupados em objetos *FileSets* que terão no mínimo o tamanho do bloco HDFS. Estes blocos de dados serão gravados de forma distribuída conforme o funcionamento padrão do HDFS. A classe *ForeIndex* também se comunica com o *NameNode* para que este gerencie todo o processo de cópia de arquivos no ambiente distribuído. O *NameNode* fornece a localização de cada *DataNode* no ambiente distribuído, a quantidade de réplicas que cada bloco de dados terá e monitorará se os dados serão corretamente gravados e replicados nos *DataNodes*.

Um *DataNode* é um processo sendo executado em determinado nó do ambiente distribuído. Neste mesmo nó também existe uma tarefa *Map* sendo executada, demonstrando que este nó é tanto uma unidade de armazenamento quanto de processamento de dados. Um *DateNode* armazena diversos blocos de dados. Estes blocos de dados são objetos do tipo *FileSet*. Os *FileSets* são tipos especiais de arquivos que agrupam outros arquivos de forma que seu tamanho seja no mínimo o tamanho de um bloco HDFS.

As personalizações realizadas na fase de cópia dos dados visaram a utilização do *Hadoop* de forma a melhorar sua performance. Esta personalização permitiu que os arquivos não sejam divididos no processo de cópia e nem de leitura dos dados, além de buscar utilizar um tamanho performático para cada arquivo baseado no tamanho do bloco HDFS. Conforme será ilustrado nos cenários de teste esta melhoria na performance refletirá tanto na fase de cópia de dados quanto na fase de indexação.

3) Indexação Distribuída

Esta subseção descreverá o processo indexação distribuída de arquivos. Será detalhado o funcionamento deste processo bem como as personalizações que foram necessárias nas ferramentas utilizadas para realizar esta atividade. Serão também analisados os requisitos para a indexação de dados forenses e as características do processo distribuído que contemplou estes requisitos.

Para realizar a indexação de um volume de dados forenses, este volume de dados precisa estar facilmente acessível. O HDFS possibilita que estes dados sejam gravados em um sistema de arquivos distribuído com recursos de escalabilidade, disponibilidade e tolerância a falhas. A fase de cópia de arquivos realiza o processo de cópia de uma fonte externa de dados para o HDFS. Neste processo de cópia os dados são gravados em objetos do tipo *FileSet* que facilitarão a leitura dos mesmos, uma vez que cada arquivo deste possui pelo menos o tamanho de um bloco HDFS. Este procedimento facilitará a ocorrência da localidade de dados, onde os dados que serão indexados já estarão fisicamente em cada nó de processamento, diminuindo consideravelmente a largura de banda utilizada e melhorando a performance do processo de indexação.

Confirma ilustrado na Fig. 6, um processo cliente aciona a classe *ForeIndex* para iniciar o processo de indexação. Este processo fornece os parâmetros necessários, como a localização dos arquivos a serem indexados, que estão no HDFS, e o local onde os índices serão gravados. A classe *ForeIndex* se comunica com o processo *JobTracker* para gerenciar toda a atividade de processamento distribuído da indexação. O processo *JobTracker* se comunicará com os processos *TaskTracker*, presentes em cada nó, para a execução das atividades de processamento distribuído. Conforme descrito no funcionamento geral do *MapReduce*, os processos *TaskTracker* gerenciarão as tarefas *Map* e as tarefas *Reduce* que estarão sendo executadas em seu ambiente e reportarão o progresso destas atividades ao processo *JobTracker*. A classe *ForeIndex* também se comunica com o *NameNode* para ter acesso aos dados que estão gravados no HDFS. O processo *NameNode* contém o mapeamento da localização de cada *DataNode* que armazena os dados solicitados, que estarão distribuídos em diversos blocos. O *NameNode* procurará ao máximo implementar a localidade de dados, indicando os blocos que já estejam gravados em cada nó de processamento para a realização da indexação.

Na Fig. 8 estão ilustradas as fases *Map* e *Reduce* da indexação.

Na fase que será executada pelas tarefas *Map* um pedaço (*Split*) do volume de dados a indexar será encaminhado para cada nó de processamento. Os *Splits* estarão no formato de objetos *FileSet* que terão o tamanho mínimo de um tamanho de bloco HDFS. O ambiente *MapReduce* sempre que possível fornecerá o *Split* que já se encontra gravado no próprio nó que a tarefa *Map* executa, diminuindo a transferência de dados pela rede. De posse deste objeto *FileSet* a tarefa *Map* realizará todo o processo de indexação. Para realizar a indexação

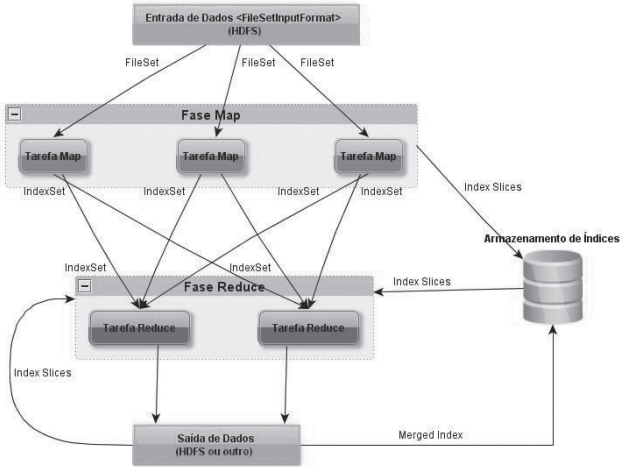

Figura 8: Fluxo de atividades da indexação distribuída

a tarefa *Map* extrai todos os arquivos que estão agrupados no objeto *FileSet*, interpreta estes arquivos extraindo os textos relevantes de seu conteúdo, utilizando o *Tika,* e cria a estrutura de dados que será o índice, utilizando o *Lucene*. Esta atividade é realizada utilizando fluxos de processamento em *pipeline* permitindo que ao mesmo tempo em que o arquivo é lido, o mesmo é interpretado e indexado, diminuindo assim a necessidade de leituras adicionais em disco.

Cada índice criado pela tarefa *Map* é agrupado em um objeto chamado *IndexSet*. Todos os índices criados por determinada tarefa *Map* são agrupados e gravados em um ambiente de armazenamento externo. Este agrupamento recebe o nome de *IndexSlice*. Quando as atividades *Map* finalizam a sua execução existem diversos objetos *IndexSlices* criados, de acordo com a quantidade de tarefas *Map* existentes. Estes objetos são índices inteiramente funcionais. Um processo de pesquisa no volume de dados forenses indexados já pode ser realizado utilizando-se destes índices, onde os mesmos serão pesquisados seqüencialmente para retornar o resultado da pesquisa.

Para poder melhorar a performance na pesquisa, pode ser realizado um agrupamento de todos os *IndexSlices*. Este processo de agrupamento se chama *Merge* e é executado por tarefas *Reduce*. As tarefas *Reduce* recebem uma quantidade de *IndexSlices* e realiza o *merge* destes índices. O processamento das tarefas *Reduce* disponíveis pode ser executado quantas vezes for necessário para agrupar todos os *IndexSlices* em um índice único. Alternativamente, as tarefas *Reduce* podem também receber diretamente os objetos *IndexSet* que são gerados pelas tarefas *Map* para serem agrupados. O índice criado é gravado em um local de armazenamento externo.

A utilização do ambiente *MapReduce*, com as personalizações descritas acima, possibilita uma maior eficiência no processamento de indexação distribuída. O volume de dados forenses agrupado em objetos *FileSet* pode ser paralelamente indexado utilizando a localidade de dados para melhorar o desempenho. A utilização de fluxo de processamento em *pipeline* também melhora a performance de cada tarefa *Map*. O oneroso processo computacional de criação de índices é dividido em diversas tarefas paralelas que processarão um pedaço do volume de dados de cada vez. Além disto, o agrupamento (*merge*) dos diversos pedaços de índices criados também será realizado de forma paralela e distribuída.

4. Cenário de Testes Utilizados

Para a realização dos testes foi selecionada uma massa de dados resultante de uma análise pericial de um órgão do governo federal do Brasil. Esta massa de dados foi selecionada por ser representativa de um conjunto de dados normalmente encontrados em servidores de arquivos e computadores de usuários. A tabela 1 demonstra as características dos dados forenses que compõem a massa de dados selecionada. Esta massa de dados será utilizada para testar tanto o processo de cópia quanto o processo de indexação.

Tabela 1: Massa de Dados do Cenário de Teste

Parâmetro	Valor
Quantidade de arquivos	2.274.796
Tamanho total dos arquivos	280 GB
Formato dos arquivos	.txt, .xls(x), .xls, .doc(x), .rtf, .msg

A. Primeiro Cenário de Teste

Nesta seção será apresentado o cenário de teste criado para realizar o processo de cópia e indexação da massa de dados forenses selecionada utilizando-se 01 computador e sem processamento distribuído. A tabela 2 contém a descrição dos componentes utilizados neste teste.

Tabela 2: Características do Primeiro Cenário de Teste

Parâmetro	Valor
Quantidade de computadores	01
Configuração do computador	Intel Core-2 Quad, 2.66 GHz 04 GB de RAM 01 HD SATA-II, 320 GB, ST332020AS
Fonte de dados	01 HD SATA-II, 1,5 TB, ST150095687
Sistemas Operacionais	Microsoft Windows 7 openSUSE 11.4 (Linux 2.6)

Foram realizados testes nos sistemas operacionais Windows 7 e Linux 2.6. Os resultados obtidos com os testes realizados neste cenário estão descritos na tabela 3.

Tabela 3: Resultados do Primeiro Cenário de Teste

Parâmetro	Valor
Tempo de Cópia – Windows	12h10min
Tempo de Indexação – Windows	26h05min
Tempo de Cópia – Linux	11h30min
Tempo de Indexação - Linux	25h15min

B. Segundo Cenário de Teste

Nesta seção será apresentado o cenário de teste criado para realizar o processo de cópia e indexação da massa de dados forenses selecionada utilizando-se 12 computadores, com processamento distribuído. A tabela 4 contém a descrição dos componentes utilizados neste teste.

Tabela 4: Características do Segundo Cenário de Teste

Parâmetro	Valor
Quantidade de computadores	12

Parâmetro	Valor
Configuração de cada computador	Intel Core-2 Quad, 2.66 GHz 04 GB de RAM 01 HD SATA-II, 320 GB, ST332020AS
Sistema Operacional	openSUSE 11.4 (Linux 2.6)
Rede local	Ethernet 1 Gbps
Tamanho do bloco HDFS	64 MB
Fonte de dados	2 HDs SATA-II, 320 GB, ST332020AS (sem RAID)
Configuração do ambiente	1 Namenode, 1 JobTracker e 10 Workers

A tabela 5 descreve os resultados obtidos neste cenário.

Tabela 5: Resultados do Segundo Cenário de Teste

Parâmetro	Valor
Tempo de Cópia	3h25min
Tempo de Indexação	0h25min

C. Terceiro Cenário de Teste

Nesta seção será apresentado o cenário de teste criado para realizar o processo de cópia e indexação da massa de dados forenses selecionada utilizando-se 12 computadores, com processamento distribuído. A diferença do cenário anterior é que neste caso a fonte de dados é formada utilizando-se um RAID-1. A tabela 6 contém a descrição dos componentes utilizados neste teste.

Tabela 6: Características do Terceiro Cenário de Teste

Parâmetro	Valor
Quantidade de computadores	12
Configuração de cada computador	Intel Core-2 Quad, 2.66 GHz 04 GB de RAM 01 HD SATA-II, 320 GB, ST332020AS
Sistema Operacional	openSUSE 11.4 (Linux 2.6)
Rede local	Ethernet 1 Gbps
Tamanho do bloco HDFS	64 MB
Fonte de dados	2 HDs SATA-II, 320 GB, ST332020AS (configurados em RAID-1)
Configuração do ambiente	1 Namenode, 1 JobTracker e 10 Workers

Os resultados obtidos com os testes realizados neste cenário estão descritos na tabela 7.

Tabela 7: Resultados do Terceiro Cenário de Teste

Parâmetro	Valor
Tempo de Cópia	1h50min
Tempo de Indexação	0h25min

D. Análise dos Resultados Obtidos

Nesta seção será apresentada uma consolidação dos resultados obtidos em cada cenário de teste elaborado. Também será realizada uma análise em como a mudança nas variáveis de cada cenário de teste impactou o resultado final, expondo também o funcionamento da prova de conceito elaborada.

Tabela 8: Resultados dos Cenário de Teste Realizados

Cenário de Teste	Tempo de Cópia	Tempo de Indexação
Primeiro (01 computador)	12h10min	26h05min
Segundo (12 computadores, sem RAID)	3h25min	0h25min
Terceiro (12 computadores, RAID-1)	1h50min	0h25min

De acordo com o exposto na Tabela 8, o tempo de cópia dos 2.274.796 arquivos de dados forenses foi de 12 horas e 10 minutos, utilizando o sistema operacional Windows 7. Além disto, o tempo de indexação destes dados, utilizando este mesmo ambiente foi de 26 horas e 5 minutos. Isto demonstra o gargalo que estas duas atividades representam no processo pericial, através do exemplo deste cenário demonstrativo.

A demora no processo de cópia, utilizando o sistema operacional, se deve principalmente às atividades de gerenciamento do sistema de arquivos e de cópia física dos dados, utilizando apenas um disco rígido com diversos arquivos de tamanho reduzido.

Utilizando o sistema distribuído elaborado para realizar o processo de cópia, observou-se um ganho expressivo no tempo gasto nesta atividade. Nos dois cenários de teste com sistema distribuído, foi utilizado um *cluster* que foi montado com 12 computadores. Como fonte de dados foi utilizado um computador com 2 discos rígidos, sendo que no primeiro cenário não foi utilizado RAID e no segundo foi utilizado RAID-1. O tempo de cópia no primeiro cenário com o sistema distribuído foi de 3 horas e 25 minutos e no segundo foi de 1 hora e 50 minutos. Como o processo de cópia é realizado de forma distribuída entre diferentes computadores, o tempo gasto pelo sistema operacional para alocar os arquivos que serão copiados não foi relevante. O gasto de tempo predominante foi com a atividade de cópia dos dados em si. Com isto observou-se uma diminuição de 75% do tempo gasto no primeiro cenário (com 1 computador). Utilizando o RAID-1 observou-se uma diminuição de 84% com o tempo gasto no primeiro cenário (com 1 computador). Na configuração em RAID-1 os dados estão duplicados

nos dois discos rígidos, possibilitando um paralelismo no tempo de cópia. Desta forma, os cenários de testes utilizados demonstram os benefícios que a computação distribuída proporcionam no tempo de cópia dos dados.

O processo de indexação é o principal gargalo analisado, onde no primeiro cenário de teste (com 1 computador) levou 26 horas e 05 minutos para ser finalizado. Utilizando-se do sistema distribuído elaborado verificou-se a diminuição de impactante no tempo gasto para a indexação, onde levou apenas 25 minutos para concluir este processamento.

A impactante diminuição no tempo de processamento da indexação foi devido a um conjunto de fatores:

- A utilização de um sistema distribuído com 12 máquinas, onde ao invés do processo ser feito por apenas 1 computador, pôde ser realizado por 12 computadores;
- A utilização de um sistema de arquivos distribuído, que proporcionou a localidade dos dados. Ou seja, na maioria dos casos, o bloco de dados a ser processado já se encontrava na própria máquina, não necessitando de um processo de cópia em rede;
- As personalizações que foram implementadas, maximizando a utilização do sistema distribuído no contexto da indexação de dados forenses. Onde, os arquivos foram agrupados em conjuntos do tamanho de um bloco de dados HDFS, maximizando a performance do ambiente distribuído de processamento. Além disto, a utilização do *pipeline* no processamento da indexação acelerou a atividade de criação dos índices propriamente ditos.

O tempo gasto na indexação do segundo cenário de teste foi o mesmo que o tempo gasto no terceiro cenário de teste devido a única diferente ser a configuração da fonte de dados. Esta configuração influiu apenas no processo de cópia dos dados para o ambiente distribuído. No momento da indexação, os dados já estavam disponíveis no sistema de arquivos distribuído.

5. Conclusão

O presente trabalho apresentou um sistema distribuído que foi elaborado para realizar o armazenamento e a indexação dos dados forenses.

Foi exposta a necessidade de um método mais eficiente para tratar com dois importantes gargalos em uma análise pericial de dados correlacionados que é a necessidade de um espaço de armazenamento adequado e uma maior eficiência no método de indexação.

Foi demonstrado o funcionamento das principais ferramentas que foram utilizadas para a construção do sistema distribuído. Especial destaque foi dado para o algoritmo *MapReduce*, para a forma como o *Hadoop* implementa este algoritmo e sobre o funcionamento do sistema de arquivos distribuído HDFS.

Mediante a análise do comportamento padrão das ferramentas selecionadas, foram apresentadas as personalizações que foram necessárias nestas ferramentas para contemplar as necessidades de armazenamento e indexação dos dados forenses. O fluxo de implementação das tarefas *Map* e *Reduce* que foram criadas foi analisado, buscando demonstrar os objetivos de ganho de performance.

Através dos cenários de testes elaborados e utilizados ficou evidenciado os impactantes ganhos de performance que as fases de cópia e indexação de dados ganham com a utilização de um ambiente distribuído.

O sistema distribuído apresentado neste trabalho proporciona um mecanismo para armazenar os dados forenses com escalabilidade, disponibilidade e tolerância a falhas. Além disto, apresenta a funcionalidade de realizar a indexação destes dados forenses com consideráveis ganhos de performance.

Este sistema foi elaborado com a finalidade de armazenar e indexar os dados forenses, mas importantes acréscimos podem ser realizados através da utilização de sua API para a construção de poderosas interfaces capazes de realizar a correlação do grande volume de dados previamente indexados. Isto possibilitará uma maior celeridade no processo de apuração de uma infração penal que envolva a análise de um grande volume de dados.

Como trabalhos futuros, estamos preparando novos cenários de teste, com o uso de 1, 8, 16, 32, 64 e 128 computadores compondo o cluster, uma vez que a nossa infra-estrutura em breve estará assim configurada.

Referências

[1] Coulouris, G., Dollimore, J., Kindberg, T. Sistemas Distribuídos - Conceitos e Projeto. São Paulo/SP: Bookamn, 2007.

[2] Walter, C. Kryder's Law. Scientific American Magazine. August 2005 Issue.

[3] Bezzera, C. C. Relatório Estatístico de Atividades do Sistema Nacional de Criminalística. Brasília/DF: Diretoria Técnico-Científica, Polícia Federal Brasileira, 2011.

[4] Galvão, R. K. Computer Forensics with Sleuth Kit and The Autopsy Forensic Browser. The International Journal of Forensic Computer Science (ICoFCS), 2006 Issue.

[5] Tika. Apache Tika. Acesso em 08 de 07 de 2011, disponível em Sítio do Projeto Apache Tika: http://tika.apache.org/, 2011.

[6] Mattmann, C. A., Zitting, J. L.. Tika in Action. Stamford, CT, Estados Unidos: Manning Publications Co, 2010.

[7] Lucene, 2011. Apache Lucene. Acesso em 08 de 07 de 2011, disponível em Sítio do Projeto Apache Lucene: http://lucene.apache.org/

[8] Manning, C., Raghavan, P., Shutze, H. Introduction to Information Retrieval. Cambridge, Inglaterra: Cambridge University Press, 2008.

[9] Dean, J., Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation, pp. 137-150, 2004. San Francisco, CA, Estados Unidos.

[10] Smith, D. C. MLISP User's Manual. Califórnia, Estados Unidos: Departamento de Ciência da Computação - Universidade de Stanford, 1969.

[11] Buyya, R., Broberg, J., & Goscinski, A. M. Cloud Computing: Principles and Paradigms. Nova Jersey, Estados Unidos: Wiley, 2011.

[12] Stallings, W. Autenticação de Mensagens e Funções de Hash. In: W. Stallings, Criptografia e Segurança de Redes - Princípios e Práticas, pp. 226-251, 2008. São Paulo: Pearson Prentice Hall.

[13] Bernam, F., Fox, G. C., Hey, A. J. Grid Computing - Making the Global Infrastructure a Reality. Nova Jersey, Estados Unidos: Wiley, 2005.
[14] Google. Site Oficial do Google Brasil. Acesso em 12 de 07 de 2011, disponível em http://www.google.com.br, 2011.
[15] Ghemawat, S., Gobioff, H., Leun, S.-T. The Google File System. 9th Symposium on Operating Principles, pp. 29-43, 2003. Nova York, Estados Unidos.
[16] Hadoop. Apache Hadoop. Acesso em 08 de 07 de 2011, disponível em Sítio do Projeto Apache Hadoop: http://hadoop.apache.org/, 2011.
[17] Nutch. Apache Hadoop. Acesso em 08 de 07 de 2011, disponível em Sítio do Projeto Apache Nutch: http://nutch.apache.org/, 2011.
[18] White, T. Hadoop, The Definitive Guide. Georgia, Estados Unidos: O'Reilly Media, 2009.
[19] HDFS. Acesso em 08 de 07 de 2011, disponível em Sítio do Projeto HDFS: http://hadoop.apache.org/hdfs/, 2011.
[20] AccessData. Site Oficial do FTK. Acesso em 19 de 07 de 2011, disponível em http://accessdata.com/products/computer-forensics/ftk, 2011.
[21] Guidance. Site Oficial do EnCase Forensic. Acesso em 19 de 07 de 2011, disponível em http://www.guidancesoftware.com/forensic.htm, 2011.
[22] SleuthKit. Site Oficial do Sleuth Kit. Acesso em 19 de 07 de 2011, disponível em http://www.sleuthkit.org/, 2011.
[23] McCandless, M., Hatcher, E., Gospodnetic, O. Lucene in Action. Stamford, CT, Estados Unidos: Manning Publications Co, 2010.

Marcelo Antonio da Silva. Mestrando do Curso de Informática Forense e Segurança da Informação pela Universidade de Brasília (UnB), linha de pesquisa em Informática Forense, Recuperação da Informação e Computação Distribuída. Graduação em Ciência da Computação com ênfase em Engenharia de Software pela Universidade Católica de Brasília (UCB) em 2001. Trabalhou com engenharia de software em órgãos públicos e privados desde 1997. Desde 2005 é Perito Criminal Federal do Departamento de Polícia Federal. Atualmente lotado no Serviço de Perícias de Informática do Instituto Nacional de Criminalística, em Brasília.

Romualdo Alves Pereira Júnior. Doutorando do Curso de Ciência da Informação da Universidade de Brasília (UnB), Linha de Pesquisa em Arquitetura da Informação. Mestrado em Informática pela Universidade Federal da Paraíba (UFPb), Especialização em Engenharia de Software pela Universidade Católica de Brasília (UCB) e Graduação em Tecnologia de Processamento de Dados pela Universidade de Brasília (UnB). É o líder do Grupo de Pesquisa "Sistemas Solo para Estações de Recepção de Satélites", certificado pelo Instituto Nacional de Pesquisas Espaciais - INPE, desde 2000. Atualmente, é Analista em C&T Senior da Agência Espacial Brasileira (AEB), Chefe da Divisão de Informática. Atua nas áreas de Ground Station, Microsatellites, Bancos de Dados Avançados, Desenvolvimento de Soluções para a Web, Inteligência Organizacional, Gestão do Conhecimento e Descoberta de Conhecimento em Bancos de Dados.