**UNIVERSIDADE DE BRASÍLIA**
**FACULDADE DE TECNOLOGIA**
**DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

# UTILIZAÇÃO DA COMPUTAÇÃO DISTRIBUÍDA PARA O ARMAZENAMENTO E INDEXAÇÃO DE DADOS FORENSES

**MARCELO ANTONIO DA SILVA**

**ORIENTADOR: ANDERSON CLAYTON A. NASCIMENTO**

**DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA**
**ÁREA DE CONCENTRAÇÃO INFORMÁTICA FORENSE E SEGURANÇA DA INFORMAÇÃO**

**PUBLICAÇÃO: PPGENE.DM – 091/12**

**BRASÍLIA/DF: FEVEREIRO/2012**

**UNIVERSIDADE DE BRASÍLIA**
**FACULDADE DE TECNOLOGIA**
**DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

# UTILIZAÇÃO DA COMPUTAÇÃO DISTRIBUÍDA PARA O ARMAZENAMENTO E INDEXAÇÃO DE DADOS FORENSES


## MARCELO ANTONIO DA SILVA


DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE PROFISSIONAL EM INFORMÁTICA FORENSE E SEGURANÇA DA INFORMAÇÃO.

APROVADA POR:

---

**ANDERSON CLAYTON A. NASCIMENTO, Doutor, UnB**
**(ORIENTADOR)**

---

**FLAVIO ELIAS GOMES DE DEUS, Doutor, UnB**
**(EXAMINADOR INTERNO)**

---

**DIVANILSON RORIGO DE SOUSA CAMPELO, Doutor, UnB**
**(EXAMINADOR EXTERNO)**

**DATA: BRASÍLIA/DF, 10 DE FEVEREIRO DE 2012**

## FICHA CATALOGRÁFICA

SILVA, MARCELO ANTONIO DA
Utilização da Computação Distribuída para o Armazenamento e Indexação de Dados Forenses [Distrito Federal] 2011.
xxii, 138p., 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2011).

Dissertação de Mestrado – Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. Computação Distribuída
2. Dados Forenses
3. Recuperação da Informação
4. Sistema de Arquivos Distribuído
5. Indexação Distribuída

I. ENE/FT/UnB.
II. Título (Série)

## REFERÊNCIA BIBLIOGRÁFICA

SILVA, MARCELO ANTONIO DA (2011). Utilização da Computação Distribuída para o Armazenamento e Indexação de Dados Forenses. Dissertação de Mestrado, Publicação PPGENE.DM – 091/12, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 138p.

## CESSÃO DE DIREITOS

NOME DO AUTOR: Marcelo Antonio da Silva
TÍTULO DA DISSERTAÇÃO: Utilização da Computação Distribuída para o Armazenamento e Indexação de Dados Forenses.
GRAU/ANO: Mestre/2011.



Marcelo Antonio da Silva

Este trabalho é dedicado à minha amada esposa, Laila Dias da Silva, por seu indescritível companheirismo, incentivo e carinho em todas as fases deste trabalho.

# AGRADECIMENTOS

Primeiramente sou grato a Deus, pelo dom da vida e por sempre iluminar o meu caminho.

Aos meus queridos pais, Divino e Celina, pelos ensinamentos de vida tão preciosos em minha existência.

À minha amada esposa, Laila, por toda sua compreensão, seu carinho, sua presença e seu apoio fundamentais para as horas e horas dedicadas a este trabalho.

Ao meu orientador Prof. Dr. Anderson Clayton A. Nascimento, do Curso de Mestrado em Engenharia Elétrica, área de concentração Informática Forense e Segurança da Informação, Departamento de Engenharia Elétrica, pelo brilhantismo de seu conhecimento e didática exemplar. Seu direcionamento, apoio e confiança foram basilares para o desenvolvimento deste trabalho.

Ao Prof. Romualdo Alves Pereira Júnior, da Agência Espacial Brasileira, pela rica orientação técnica, constante apoio, incentivo, dedicação e amizade essenciais para o desenvolvimento deste trabalho e para o meu desenvolvimento como pesquisador.

Aos bolsistas e funcionários da Agência Espacial Brasileira, Marcelo Holtz, Joesley e Daniel Luchetta, que prestaram um apoio técnico de grande relevância nas ferramentas utilizadas no presente trabalho, além das conversas enriquecedoras, colaboração e amizade.

Aos amigos do SEPINF, especialmente Bruno Werneck, Luciano Kuppens e Linhares, por terem contribuído com idéias essenciais para a realização deste projeto.

Aos organizadores do Mestrado, por terem acreditado neste projeto e se esforçado exemplarmente para proporcionar uma melhoria na qualificação da perícia criminal brasileira.

Finalmente sou grato aos meus irmãos, Márcia e Daniel, por toda a amizade, apoio e presença em minha vida. Em nome de vocês sou grato aos meus amigos, os mais próximos e os mais distantes, que de alguma forma me apoiaram na peleja de realizar este trabalho.

O presente trabalho foi realizado com o apoio do Departamento Polícia Federal – DPF, com recursos do Programa Nacional de Segurança Pública com Cidadania – PRONASCI, do Ministério da Justiça.

# RESUMO

## UTILIZAÇÃO DA COMPUTAÇÃO DISTRIBUÍDA PARA O ARMAZENAMENTO E INDEXAÇÃO DE DADOS FORENSES

**Autor: Marcelo Antonio da Silva**
**Orientador: Anderson Clayton A. Nascimento**
**Co-orientador: Romualdo Alves Pereira Júnior**
**Programa de Pós-graduação em Engenharia Elétrica**
**Brasília, dezembro de 2011**

Este trabalho apresenta um sistema distribuído construído para realizar o armazenamento e a indexação dos dados resultantes de uma análise forense em dispositivo de armazenamento computacional.

Com o passar dos anos, a quantidade de dados forenses a analisar vem se tornando cada vez maior. 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 secundário, realizar a análise forense torna-se uma tarefa com alto custo computacional de processamento, devido ao processo de geração dos índices e a necessidade de um espaço de armazenamento adequado para os dados. A indexação é fundamental para uma posterior análise nos dados forenses.

A solução apresentada 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. Também, realiza a indexação dos dados forenses, através de um eficiente método de distribuição de processamento em grade.

Neste trabalho são apresentados os cenários que foram elaborados e utilizados para testar o comportamento do sistema de armazenamento e indexação distribuída de dados forenses.

# ABSTRACT

## USE OF DISTRIBUTED COMPUTING FOR STORAGE AND INDEXING OF FORENSIC DATA

**Author: Marcelo Antonio da Silva**

**Supervisor: Anderson Clayton A. Nascimento**

**Co-Supervisor: Romualdo Alves Pereira Júnior**

**Programa de Pós-graduação em Engenharia Elétrica**

**Brasília, december of 2011**

This thesis presents a distributed system built to perform storage and indexing of data resulting from a forensic analysis of computer storage device.

Over the years, the amount of forensic data to analyze is becoming ever greater. This is due to both the growing capacity of computer storage devices and the increased popularity of these devices. In a case involving dozens of secondary storage devices, perform forensic analysis becomes a task with high computational cost of processing, due to the index generation process and the need for adequate storage space for data. The indexing is fundamental for further forensics analysis in these data.

The solution presented here uses a distributed file system to provide a storage space for forensic data with scalability, availability and fault tolerance. Also, performs the indexing of forensic data, through an efficient distribution method with grid processing.

This paper presents the scenarios that were developed and used to test the behavior of the system of distributed storage and indexing of forensic data.

# SUMÁRIO

# LISTA DE TABELAS

## LISTA DE FIGURAS

# LISTA DE SÍMBOLOS, NOMENCLATURA E ABREVIAÇÕES

| | |
|---|---|
| **AD LAB** | ACCESSDATA LAB |
| **AES** | ADVANCED ENCRYPTION STANDARD |
| **AFS** | ANDREW FILE SYSTEM |
| **BSP** | BULK SYNCHRONOUS PROGRAMMING |
| **CPAI** | CENTRO DE PESQUISA EM ARQUITETURA DA INFORMAÇÃO |
| **CPF** | CADASTRO DE PESSOAS FÍSICAS |
| **CPU** | CENTRAL PROCESSING UNIT |
| **CUDA** | COMPUTE UNIFIED DEVICE ARCHITECTURE |
| **DES** | DATA ENCRYPTION STANDARD |
| **DNS** | DOMAIN NAME SYSTEM |
| **EC2** | AMAZON ELASTIC COMPUTE CLOUD |
| **EPUB** | ELETRONIC PUBLICATION |
| **E/S** | ENTRADA/SAÍDA |
| **FAT32** | FILE ALLOCATION TABLE - 32 BITS |
| **FTK** | FORENSIC TOOLKIT |
| **FTP** | FILE TRANSFER PROTOCOL |
| **GB** | GIBABYTE - 1 GB = 1024 MB |
| **GFS** | GOOGLE FILE SYSTEM |
| **GHZ** | GIGAHERTZ |
| **GMETAD** | GANGLIA META DAEMON |
| **GMOND** | GANGLIA MONITOR DAEMON |
| **GPFS** | GENERAL PARALLEL FILE SYSTEM |
| **HDFS** | HADOOP DISTRIBUTED FILE SYSTEM |
| **HPC** | HIGH PERFORMANCE COMPUTING |
| **HTML** | HYPERTEXT MARKUP LANGUAGE |
| **IP** | INTERNET PROTOCOL |
| **IR** | INFORMATION RETRIEVAL |
| **ISBN** | INTERNATIONAL STANDARD BOOK NUMBER |
| **JAAV RMI** | JAVA REMOTE METHOD INVOCATION |
| **JAR** | JAVA ARCHIVE |
| **LUSTREFS** | LUSTRE FILE SYSTEM |
| **MB** | MEGABYTES - 1 MB = 1024 KB |

| | |
|---|---|
| **MB/S** | MEGABYTES POR SEGUNDO |
| **MD-5** | MESSAGE DIGEST - VERSION 5 |
| **MPI** | MESSAGE PASSING INTERFACE |
| **NFS** | NETWORK FILE SYSTEM |
| **NTFS** | NEW TECHNOLOGY FILE SYSTEM |
| **OPENCL** | OPEN COMPUTING LANGUAGE |
| **OPENMP** | OPEN MULTI-PROCESSING |
| **PB** | PETABYTE - 1 PTB = 1024 TERABYTES |
| **PDF** | PORTABLE DOCUMENT FORMAT |
| **PVFS** | PARALLEL VIRTUAL FILE SYSTEM |
| **RAM** | RANDOM ACCESS MEMORY |
| **RCFS** | REQUEST FOR COMMENTS |
| **RRDTOOL** | ROUND-ROBIN DATABASE TOOL |
| **RSA** | RON RIVEST, ADI SHAMIR AND LEONARD ADLEMAN |
| **RTF** | RICH TEXT FORMAT |
| **S3** | AMAZON SIMPLE STORAGE SERVICE |
| **SHA-1** | SECURE HASH ALGORITHM – VERSION 1 |
| **SATA** | SERIAL ADVANCED TECHNOLOGY ATTACHMENT |
| **SEPINF** | SERVIÇO DE PERÍCIAS DE INFORMÁTIVA |
| **SGBDR** | SISTEMA GERENCIADOR DE BANCO DE DADOS RELACIONAL |
| **SOAP** | SIMPLE OBJECT ACCESS PROTOCOL |
| **TAR** | TAPE ARCHIVE |
| **TBB** | THREAD BUILDING BLOCKS |
| **TCP** | TRANSMISSION CONTROL PROTOCOL |
| **TLS** | TRANSPORT LAYER SECURITY |
| **TSK** | THE SLEUTH KIT |
| **UDP** | USER DATAGRAM PROTOCOL |
| **UNB** | UNIVERSIDADE DE BRASÍLIA |
| **UPC** | UNIFIED PARALLEL C |
| **VHDL** | VHSIC HARDWARE DESCRIPTION LANGUAGE |
| **VHSIC** | VERY HIGH SPEED INTEGRATED CIRCUITS |
| **XDR** | EXTERNAL DATA REPRESENTATION |
| **XML** | EXTENSIBLE MARKUP LANGUAGE |

# 1. INTRODUÇÃO

Em investigações criminais, para a elucidação de fatos ocorridos, as forças policiais realizam procedimentos de produção de provas. A produção de uma prova robusta é considerada o principal resultado de um bom trabalho elucidativo que uma determinada autoridade policial pode realizar (Lima, 2011).

Juridicamente existem alguns meios de prova. Os mais comumente utilizados são: a prova testemunhal, a prova documental e a prova material. A prova testemunhal decorre de um testemunho de alguém que presenciou determinada situação e realiza o seu depoimento sobre este fato. A prova documental consiste em todos os documentos físicos encontrados em determinado procedimento investigativo. Já a prova material consiste em uma análise técnico-científica, em determinada área de conhecimento, elucidando determinado fato (Lima, 2011).

A perícia forense consiste no procedimento de elaboração de provas materiais, ou seja, de provas técnico-científicas. Já a perícia computacional é uma das áreas da perícia forense que produz provas técnico-científicas através da análise de recursos computacionais. A perícia computacional envolve as fases de aquisição de conteúdo, análise neste conteúdo e, posteriormente, na disponibilização do resultado desta análise através de um laudo pericial (Espindula, Jesus, & Geiser, 2011).

A aquisição de conteúdo consiste no procedimento de ter acesso aos dados de determinado dispositivo de armazenamento computacional e armazená-los de forma a manter a sua integridade após um processo de análise neste material. A análise de conteúdo consiste no processo de extrair artefatos do volume de dados adquirido. Estes artefatos podem ser, por exemplo, arquivos de documentos e planilhas armazenadas e/ou apagadas, bases de dados, histórico de conversas em programas de mensagens instantâneas, um conjunto de mensagens de e-mail e de participação em redes sociais (Eleutério & Machado, 2011).

O volume de dados de um dispositivo de armazenamento computacional está cada vez maior (Walter, 2005). Além disto, os dispositivos computacionais estão se popularizando em um escala onde diversos destes equipamentos podem ser encontrados em um único

alvo a ser investigado. Em muitos casos policiais existe a necessidade de analisar dispositivos de armazenamento computacional de diversos alvos correlacionados aumentando ainda mais o volume de dados a analisar. Estes tipos de análises periciais têm resultado em diversos gigabytes ou mesmo terabytes de dados forenses de usuários a serem investigados (Bezzera, 2011).

Para poder realizar esta análise correlacionada de dados forenses de diferentes alvos é necessário um espaço de armazenamento computacional adequado, além de capacidade de processamento para realizar o procedimento de indexação. A indexação é necessária para que uma posterior fase de pesquisas por palavras-chave seja realizada a fim de analisar, de forma correlacionada, os dados destas diversas mídias de armazenamento computacional.

A atividade de indexação torna-se um gargalo no procedimento de análise pericial neste volume de dados forenses, devido ao custo computacional de processamento envolvido. Além disto, é necessário um espaço de armazenamento adequado para abrigar este volume de dados com escalabilidade, disponibilidade e tolerância a falhas. Escalabilidade para poder dar suporte ao crescimento do volume de dados forenses, possibilitando que a arquitetura seja facilmente adaptada, através da aquisição de novos equipamentos, para atender a demanda de armazenamento. Disponibilidade para possibilitar que, no momento da indexação e pesquisa de informações, o ambiente de armazenamento possa estar pronto para atender estas demandas provendo um ambiente que torne célere o processo de análise pericial ou investigativa. E tolerância a falhas para poder manter a integridade do ambiente de armazenamento mesmo em situações de falhas de processamento ou de hardware que são tão comuns em um ambiente com grande volume de máquinas comuns e heterogêneas (Manning, Raghavan, & Shutze, 2008), (Hadoop, 2011) e (Lucene, 2011).

## 1.1. PROBLEMA DE PESQUISA

O problema a ser pesquisado no presente trabalho é a melhoria no desempenho do processo de indexação e armazenamento de dados forenses em um determinado volume de dados, composto de múltiplas fontes de informação.

## 1.2. HIPÓTESE DE PESQUISA

As hipóteses para esta pesquisa são as seguintes:

- A utilização mais eficiente de recursos computacionais disponíveis, coordenados através de algoritmos de computação distribuída, acelera o processo de indexação dos dados forenses.
- A utilização de um sistema de arquivos distribuído possibilita um meio com escalabilidade, disponibilidade e tolerância a falhas para armazenar o volume de dados forenses a ser analisado de forma correlacionada.

## 1.3. JUSTIFICATIVA DA HIPÓTESE

A utilização de computação distribuída pode acelerar o processo de indexação pelo fato de que esta arquitetura possibilita:

- Acesso a um maior poder de processamento, devido à utilização mais eficaz dos recursos computacionais disponíveis e interconectados;
- Processamento distribuído para a criação dos índices. As atividades de leitura dos arquivos, extração de dados relevantes e criação de arquivo de busca podem ser realizadas de forma paralela;
- Tolerância a falhas, onde o processamento de uma máquina que falhou pode ser redirecionado para outra, de forma automática;
- Facilidade para melhorar a escalabilidade da solução. Novas máquinas adicionadas ao ambiente podem melhorar o desempenho do processo como um todo, uma vez que oferecem mais recursos computacionais.

Testes indicam que um sistema distribuído de recuperação da informação, em geral, possui um desempenho superior à abordagem de recuperação da informação centralizada (Paltoglou, Salampasis, & Satratzemi, 2008).

Um sistema de arquivos distribuído possui as características de escalabilidade, disponibilidade e tolerância a falhas. Estas características são desejáveis para um adequando espaço de armazenamento para o volume de dados forenses a analisar em uma atividade pericial envolvendo diversos dispositivos de armazenamento computacional.

## 1.4. OBJETIVO GERAL

Otimizar o processo de indexação e armazenamento de dados forenses pelo uso de ferramentas e algoritmos de computação distribuída.

## 1.5. OBJETIVOS ESPECÍFICOS

São objetivos específicos do presente trabalho:

- Determinar e utilizar o algoritmo e as ferramentas de computação distribuída para realizar a indexação dos dados forenses, através do controle de processos concorrentes, balanceamento de carga, tolerância a falhas e gerenciamento do processo;
- Determinar qual a ferramenta de indexação já existente é a mais adequada a se incorporar na arquitetura do sistema distribuído proposto;
- Determinar e utilizar o sistema de arquivos distribuído que é o mais adequado para se incorporar na arquitetura do sistema distribuído proposto;
- Verificar o comportamento do algoritmo e das ferramentas selecionadas e realizar as modificações/adaptações necessárias para realizar de forma eficiente o processo de indexação e armazenamento de dados forenses;
- Determinar o desempenho, limitações, flexibilidade e escalabilidade do sistema distribuído proposto;
- Realizar um *benchmarking* entre algumas soluções para a comparação com referenciais de excelência.

## 1.6. DEFINIÇÕES DE ESCOPO

A seguir seguem definições de escopo para a determinação, utilização e customização do sistema distribuído:

- Os tipos de arquivos tratados serão aqueles já definidos na ferramenta de indexação selecionada;
- Os arquivos a serem indexados serão apenas os arquivos ativos em um determinado sistema de arquivos e não em áreas não alocadas neste sistema;

- O sistema distribuído será controlado por um nó central sem redundância que gerenciará o ambiente distribuído;
- A interface de pesquisa será apenas para testar a funcionalidade dos índices criados.

## 1.7. RESULTADOS ESPERADOS

São resultados esperados em consequência da utilização prática do trabalho apresentado:

- Possibilidade de análise correlacionada de dados forenses pela autoridade investigativa em um caso resultante do processamento de diversos dispositivos de armazenamento computacional;
- Maior qualidade e celeridade no processo de apuração de uma infração penal.

## 1.8. MÉTODO DE TRABALHO

Em princípio, foi realizada uma análise e revisão na literatura mais atualizada e disponível das áreas relacionadas à:

- Computação Distribuída;
- Recuperação de Informação;
- Ferramentas de Indexação;
- Sistemas de Arquivos Distribuídos;
- Segurança Computacional.

Foram realizadas diversas reuniões com os orientadores durante todas as fases de desenvolvimento do trabalho. Estas reuniões visaram à definição e acompanhamento do cronograma, esclarecimentos de dúvidas, direcionamentos das atividades e revisão da dissertação.

Com base no estudo das modernas técnicas de indexação, computação e sistemas de arquivos distribuídos foi realizado o levantamento dos requisitos gerais do sistema de indexação distribuída, incluindo também os testes nas ferramentas de indexação que mais se adequassem à solução. Foi dada preferência para a utilização de ferramentas de código-fonte aberto.

Estas atividades foram base para a análise, projeto e desenvolvimento da prova de conceito na linguagem Java (Java, 2011). Esta prova de conceito foi construída utilizando as ferramentas selecionadas e realizando as customizações que foram necessárias.

Posteriormente foram elaborados e utilizados alguns cenários de testes, para avaliar as características de implantação e os resultados obtidos.

Durante a aplicação da prova de conceito foram realizados ajustes nos passos anteriores, com o intuito de corrigi-los ou melhorá-los, avaliando-se os impactos das mudanças nos resultados observados.

A elaboração da dissertação inclui a avaliação final dos resultados, bem como os produtos das atividades anteriores, limitações, dificuldades observadas e sugestões de estudos complementares.

## 1.9. ESTRUTURAÇÃO DO TRABALHO

O enfoque deste trabalho está na utilização da computação distribuída como forma de melhorar o desempenho do processo de indexação de dados forenses, além de prover um espaço de armazenamento com recursos de escalabilidade, disponibilidade e tolerância a falhas para estes dados.

As ferramentas para realizar procedimento de indexação, de leituras de arquivos em diversos formatos, de gerenciamento de computação distribuída e de sistema de arquivos distribuídos são analisadas e selecionadas. Estas ferramentas serão adaptadas, quando necessário, para montar o sistema distribuído que possa contemplar o objetivo do presente trabalho.

Os cenários de testes elaborados são utilizados com a finalidade de avaliar o funcionamento, o desempenho, as limitações e os benefícios da arquitetura de sistema distribuído proposto.

O Capítulo 2 apresenta uma revisão bibliográfica dos principais conceitos utilizados neste trabalho. Inicialmente serão analisados os conceitos envolvidos com as áreas da

recuperação da informação, da computação distribuída e da programação paralela. Serão apresentados os conceitos básicos de segurança computacional que são relevantes a sistemas distribuídos. Posteriormente serão abordados os conceitos básicos de informática forense e os principais procedimentos realizados em uma atividade de análise forense computacional. Por fim, serão apresentados alguns trabalhos correlatos com a pesquisa do presente trabalho, onde será realizada uma descrição de cada trabalho e uma análise correlacionando com o objeto de pesquisa desta dissertação.

O Capítulo 3 apresenta o algoritmo e as ferramentas selecionadas para a elaboração do sistema distribuído apresentado neste trabalho. É realizada uma análise comparativa do algoritmo e das ferramentas selecionadas com outras disponíveis. Esta análise comparativa visa mostrar quais características de cada recurso selecionado melhor se adequa ao presente trabalho.

O Capítulo 4 apresenta a arquitetura da prova de conceito proposta, com as customizações que foram realizadas em cada ferramenta para que contemplar a finalidade do presente trabalho. Este capítulo também apresenta considerações de segurança na arquitetura da prova de conceito proposta.

O Capítulo 5 apresenta os cenários elaborados e utilizados para a avaliação do desempenho da prova de conceito criada. São apresentados os critérios de elaboração dos cenários; analisados os requisitos, características e componentes, bem como os resultados obtidos através da utilização da prova de conceito em cada um deles.

O Capítulo 6 apresenta as conclusões obtidas com a realização do presente trabalho, sua aplicação para a perícia criminal e indica possibilidades de evolução.

O Anexo A contém o formato do índice utilizado na prova de conceito elaborada.

O Anexo B contém os procedimentos de instalação e utilização da prova de conceito.

# 2. REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta uma revisão bibliográfica do tema deste trabalho. Inicialmente serão analisados os conceitos envolvidos com as áreas da recuperação da informação, da computação distribuída e da programação paralela. Serão apresentados os conceitos básicos de segurança computacional que são relevantes a sistemas distribuídos. Posteriormente serão abordados os conceitos básicos de informática forense e os principais procedimentos realizados em uma atividade de análise forense computacional. Por fim, serão apresentados alguns trabalhos correlatos com a pesquisa do presente trabalho, onde é realizada uma descrição de cada trabalho e uma análise correlacionando com o objeto de pesquisa desta dissertação.

## 2.1. RECUPERAÇÃO DA INFORMAÇÃO

Recuperação da Informação (IR – *Information Retrieval*) pode ser conceituada como uma área da ciência da computação que estuda métodos para pesquisas em materiais (geralmente documentos), organizados de maneira não estruturada (geralmente textos) que satisfaz uma determinada necessidade de informação em grandes coleções (Manning, Raghavan, & Shutze, 2008).

Por conteúdo não estruturado entendem-se aqueles dados que não tem uma clara semântica para que possam ser facilmente organizados computacionalmente. Alguns tipos de dados como documentos científicos, documentos jurídicos, documentos pessoais, páginas da web e o conteúdo de e-mails não possuem uma estrutura pré-determinada, se enquadrando no conceito de material não estruturado que deve ser pesquisado através de técnicas de IR.

### 2.1.1. Tipos de Sistemas de Recuperação da Informação

O método tradicional de armazenamento de dados é através da estruturação da informação em entidades, campos e relacionamentos. Esta estrutura geralmente é armazenada em arquivos ou Sistemas Gerenciadores de Banco de Dados Relacionais – SGDBR, para poder propiciar a pesquisa, integridade e disponibilidade destas informações (Date, 2004).

Ferramentas que implementam técnicas de IR estão se popularizando devido a crescente produção de dados não estruturados e a consequente necessidade de pesquisa nestas informações. Um exemplo concreto é a ferramenta de pesquisa na Web do Google (Google, 2011). Esta ferramenta é um exemplo popular da implementação de conceitos de IR que torna possível uma rápida localização de vastos conteúdos dispersos na web.

O campo de abrangência da IR contempla técnicas para permitir usuários pesquisando e filtrando grandes coleções de documentos. *Clustering* de documentos é o processo de agrupar grandes coleções de documentos baseado em seu conteúdo semântico (Manning, Raghavan, & Shutze, 2008). Este conceito é diferente do *cluster* de computadores que será visto na próxima seção. Com modernas técnicas de *clustering* de documentos é possível categorizar uma vasta coleção de dados não estruturados.

Os sistemas de IR podem ser categorizados de acordo com a escala na qual eles operam. Os tipos de sistemas de IR são: os IR pessoais, os IR de pesquisa na Web e os IR coorporativos ou de domínio específico (Manning, Raghavan, & Shutze, 2008).

Os sistemas de IR pessoais nos últimos anos estão integrados com os sistemas operacionais populares, como o *Windows Desktop Search - Microsoft* ou o *Max OS X Spotlight - Apple* (Windows Search, 2011), (Apple Spotlight, 2011). Também estão presentes em outros programas utilitários como os clientes de e-mail: *Microsoft Outlook* ou *Mozilla Thunderbird* (Microsoft Outlook, 2011), (Mozilla Thunderbird, 2011). Estes exemplos de IR pessoais permitem uma rápida pesquisa no conteúdo de arquivos, com opções de filtragem e clusterização. As características típicas dos IR pessoais são que tratam de um extenso conjunto de tipos de documentos em um computador pessoal e que possuem um leve processo de manutenção do mecanismo de pesquisa.

No outro extremo estão os sistemas de IR para pesquisa na web que devem pesquisar bilhões de documentos que estão dispersos em milhões de computadores. As características típicas deste tipo de sistema é que requerem métodos com bom desempenho para a criação dos índices de pesquisa, mecanismos para tratar com hipertexto e evitar que marcadores específicos de algumas páginas fraudem os métodos de pontuação da relevância do conteúdo.

Como meio termo estão os IR coorporativos ou de domínio específico, onde os mecanismos de pesquisa devem prover meios para recuperar documentos coorporativos ou de um domínio específico de determinada atividade. Neste caso os documentos estarão tipicamente armazenados em um servidor de arquivos centralizado e uma ou diversas máquinas irão propiciar meios para realizar a pesquisa neste conteúdo.

### 2.1.2. O Processo de Indexação

Uma forma de realizar a pesquisa em um volume de dados não estruturado é através da busca linear ou *grepping*. Neste método computacional o conteúdo de todos os documentos são analisados linearmente em busca dos termos pesquisados. Com a velocidade dos computadores atuais e através da utilização de expressões regulares este método funciona bem em alguns casos (Friedl, 2006).

Contudo, em um grande volume de dados, na ordem de milhões ou bilhões de palavras, este processo de busca linear se torna oneroso e pouco eficiente. Além de não possibilitar formas avançadas de pesquisas e classificação por pontuação de relevância dos resultados recuperados.

Uma forma de evitar que seja realizada uma busca linear em todo o conteúdo de uma massa de dados é através da criação prévia de índices nestes dados. O processo de criação computacional destes índices é chamado de indexação (Manning, Raghavan, & Shutze, 2008).

Um típico processo de indexação consiste nas seguintes fases: aquisição do conteúdo, construção de documentos, análise do conteúdo destes documentos e na construção da estrutura de dados do índice (McCandless, Hatcher, & Gospodnetic, 2010).

A aquisição de conteúdo consiste na disponibilização do conteúdo que necessita ser indexado. Pode ser um simples procedimento de copiar um conjunto de arquivos de um diretório de um disco rígido para outro ou mesmo obter informações de um banco de dados bem organizado, até procedimentos mais complexos como obter conteúdos dispersos em bilhões de sítios da Internet ou em sistemas corporativos de gerenciamento de conteúdo.

A fase de construção de documentos consiste no procedimento de criação de uma estruturação básica no conjunto de dados já disponíveis e não estruturados. Esta estruturação consiste na criação de documentos com campos de pesquisa, como: título, autor, resumo, conteúdo, metadados e URL. A criação de documentos depende das necessidades específicas de pesquisa e pode variar muito para cada tipo de necessidade. Um e-mail pode ser um documento, um documento do *Microsoft Word* também pode ser outro documento. Mas em outras aplicações um e-mail pode ser um documento composto por seus diversos anexos que podem ser outros e-mails ou documentos do *Microsoft Word*. Assim que o leiaute do documento está estabelecido é necessário interpretar corretamente os diversos tipos de conteúdos para extrair suas informações textuais. Atualmente diversos tipos de conteúdo têm natureza binária, como arquivos do *Open Office*, PDF e *Adobe Flash*. E mesmo alguns conteúdos do tipo texto precisam de um tratamento especial como arquivos HTML, XML e RTF.

Após a definição da estrutura básica dos documentos e na interpretação do conteúdo que necessita ser indexado é necessário realizar uma análise nas informações contidas em cada campo dos documentos. Para possibilitar uma eficiente pesquisa os textos não podem ser indexados diretamente, antes disto é necessário dividir os textos em um conjunto de termos ou *tokens*. Estes termos podem ser entendidos como palavras, mas dependem muito da linguagem utilizada e do tipo de pesquisa que se deseja realizar. Alguns procedimentos comuns nesta fase são: exclusão das preposições, tratamento especial para palavras compostas, números de telefone, números de identificação (CPF), endereços de e-mail e correções gramaticais. Na divisão dos termos do texto também podem ser acrescentadas diversas outras informações em cada termo, como: sua localização no documento e sinônimos.

Com os termos de cada documento analisado pode-se criar uma estrutura de dados que será o índice. Diversas são as técnicas de criação de índices e neste trabalho será analisada a mais utilizada que é a criação de índices invertidos. Mas antes disto será necessário analisar como é o processo de pesquisa nos índices.

### 2.1.3. O Processo de Pesquisa

A pesquisa em um conteúdo indexado consiste no processo de procurar os termos da pesquisa no índice e verificar em quais documentos estes termos aparecem (McCandless, Hatcher, & Gospodnetic, 2010). A qualidade de uma pesquisa pode ser mensurada por duas métricas básicas: precisão e revogação (*recall*). A precisão mede se os resultados retornados são relevantes para a necessidade de informação solicitada na pesquisa. A revogação mede qual parte dos resultados que são relevantes para a necessidade de informação foram retornados pela pesquisa (Manning, Raghavan, & Shutze, 2008).

Alguns outros fatores também são considerados para uma pesquisa de qualidade como a velocidade de retorno dos resultados, a possibilidade de pesquisar em um grande volume de documentos, o suporte a pesquisa com termos simples, múltiplos ou mesmo frases, a possibilidade de utilizar expressões regulares, se os resultados são retornados de forma ordenada e com controle de relevância, além da possibilidade de disponibilizar onde os termos pesquisados foram encontrados nos documentos retornados.

Um típico mecanismo de pesquisa em uma ferramenta de IR consiste nos seguintes componentes: uma interface com o usuário, um processo para a construção lógica da pesquisa, um método de realizar a consulta no índice criado e uma interface para disponibilizar os resultados retornados (McCandless, Hatcher, & Gospodnetic, 2010).

A interface com o usuário é um importante componente de uma ferramenta de IR, pois é o que o usuário realmente vê e que possibilita que forneça os termos que deseja pesquisar. A interface deve ser mantida com o máximo de simplicidade possível e com boa qualidade na apresentação dos resultados. Nesta interface podem ser utilizados métodos de correção gramatical dos termos pesquisados, sugestões de termos comumente pesquisados e seleção dos termos pesquisados nos resultados encontrados.

O usuário pode fornecer diversos termos com operadores lógicos para realizar a pesquisa e a ferramenta de IR deve possuir um mecanismo para interpretar estes termos e utilizá-los para a realização da pesquisa. Na construção lógica da estrutura de dados, que conterá os termos que realizarão a pesquisa, podem ser acrescentados critérios de relevância e ordenação de cada termo para influir nos resultados que serão retornados.

Após a construção lógica dos termos da pesquisa é necessário consultar o índice e retornar os resultados que contemplem a pesquisa. Este mecanismo deve ser capaz de ler a estrutura interna do índice de forma a recuperar os documentos que contêm os termos pesquisados.

Considere os três modelos básicos de pesquisa comumente empregados:

- Modelo de Pesquisa Booleana: neste modelo os documentos contêm ou não os termos fornecidos na consulta, não existe pontuação de relevância de documentos e os resultados não são ordenados;

- Modelo de Pesquisa Vetorial: neste modelo tanto a consulta quanto os documentos são modelados em vetores em um espaço dimensional, onde cada termo é uma dimensão. A similaridade entre a consulta e o documento é calculada por uma medida da distância vetorial entre estes vetores;

- Modelo de Pesquisa Probabilístico: neste modelo é calculada a probabilidade que o documento tem uma boa correspondência com a consulta utilizando uma abordagem probabilística.

Uma vez que os documentos que contemplam a pesquisa são retornados, de acordo com os critérios do modelo de pesquisa utilizado, eles devem ser disponibilizados para o usuário. É papel da interface com o usuário fornecer meios para disponibilizar o resultado da pesquisa. Este resultado pode ser ordenado pelos critérios de relevância fornecidos ou refinados com novas consultas. Os critérios de relevância, por sua vez, são implementados por algoritmos com as mais variadas abordagens, sempre visando apresentar como resposta aquelas que sejam mais significativas para o usuário final.

### 2.1.3. O Processo de Criação de um Índice Invertido

Um índice invertido é uma estrutura de dados responsável por armazenar o mapeamento de um conteúdo (termos) para sua localização em um documento, conjunto de documentos ou mesmo em um banco de dados (Büttcher, Clarke, & Cormack, 2010). O índice invertido também é chamado de arquivo invertido ou arquivo de postagens.

Este nome “índice invertido” é utilizado, pois ele pode listar, para dado termo, os documentos que ele está contido. Isto é diferente do índice tradicional que onde os documentos é que listam os termos que ele contém e não o contrário.

A figura 2.1 mostra um simples exemplo de criação de um índice invertido básico.

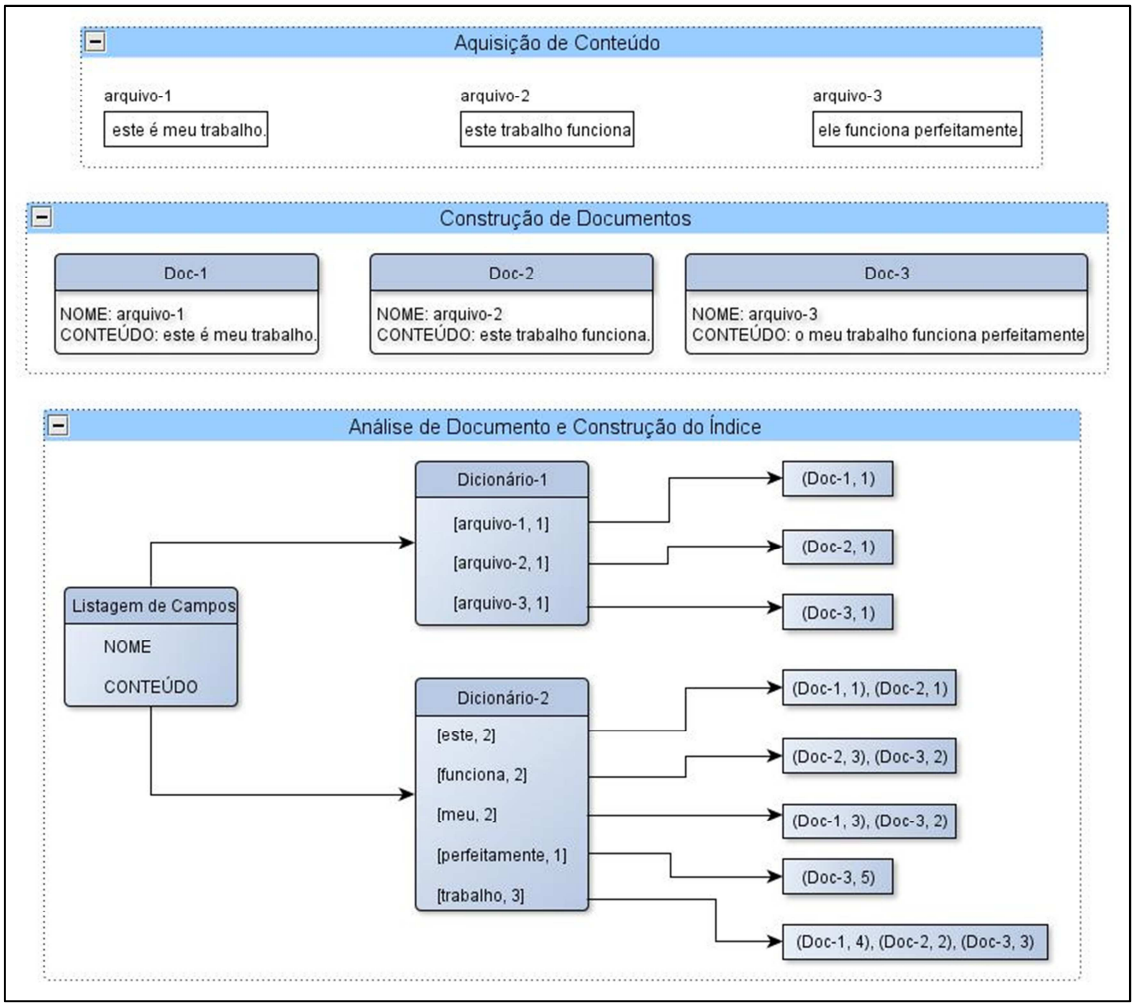



Figura 2.1 – Exemplo de Criação de Índice Invertido

A figura 2.1 contém o agrupamento das fases de indexação. Na fase de aquisição de conteúdo são fornecidos três arquivos texto para serem indexados. Estes arquivos possuem um conteúdo não estruturado. Na fase de construção de documentos é fornecida uma estrutura básica para estes arquivos, com dois campos: “nome” e “conteúdo”. O campo “nome” armazena o nome do arquivo e o campo “conteúdo” armazena todo o conteúdo contido no arquivo. Cada arquivo foi considerado um documento, sendo criados três documentos. Cada documento possui um identificador (“Doc-1”, “Doc-2” e “Doc-3”). A fase de análise do documento realiza um processamento em cada campo do documento

separando o conteúdo do campo em termos, de acordo com determinada linguagem. Termos como preposições e artigos são descartados. Antes de criar a estrutura do índice estes termos são ordenados.

Para a criação da estrutura do índice invertido são criadas as seguintes entidades: a listagem de campos, o dicionário e a listagem de postagens. A listagem de campos é uma estrutura para armazenar todos os campos de todos os documentos criados. O dicionário é uma estrutura que armazena todos os termos de todos os documentos criados. O conjunto de termos de todos os dicionários é chamado de vocabulário. A postagem é uma estrutura que contém uma referência aos documentos em que cada termo aparece, podendo ter campos adicionais como a posição no texto do documento em que o texto está inserido. O conjunto de postagens é chamado de listagem de postagens.

Nesta estrutura cada campo aponta para um dicionário que contém a listagem de todos os termos que existem naquele campo, ou seja, o vocabulário do campo. O termo presente no dicionário pode ter campos adicionais, como a quantidade de documentos em que ele aparece. Cada termo aponta para suas postagens, sendo que cada postagem contém a referência de qual documento o termo aparece, além da localização do termo no documento.

Analisando o exemplo da figura 2.1, verificam-se algumas particularidades. O campo “conteúdo” possui o seguinte vocabulário: “este”, “funciona”, “meu”, “perfeitamente” e “trabalho”. Os artigos e preposições contidos no arquivo original foram descartados e os termos aparecem organizados alfabeticamente. O termo “funciona” está contido em dois documentos. Este termo está contido no Doc-1 na terceira posição e no Doc-3 na segunda posição. O termo “trabalho” está contido em todos os três documentos, na quarta, segunda e terceira posições, respectivamente.

Geralmente a listagem de campos e o vocabulário estão disponíveis na memória principal e a listagem de postagens armazenados em disco, por serem maiores. Diversas são as estruturas de dados que podem armazenar o índice invertido, podendo ser desde simples arquivos texto até complexos bancos de dados relacionais.

### 2.1.4. Campos de Pesquisa em IR

Os pesquisadores em IR realizam pesquisas em áreas das diversas fases do processo de indexação e pesquisa em documentos. São exemplos destas áreas de pesquisas:

- algoritmos de pré-processamento de documentos (Kenneth & Karttunen, 2003);

- métodos para acrescentar funcionalidades e velocidade em índices invertidos (Paolo & Venturini, 2007);

- estruturas de dados para realizar pesquisas em dicionários (Knuth, 1997), (Paolo & Venturini, 2007);

- correções fonéticas em pesquisas e localizações aproximadas nos dicionários baseados em aproximações fonéticas e gramaticais (Toutanova, Kristina, & Moore, 2002);

- algoritmos para compressão de dicionários de dados e índices invertidos, com o conseqüente aumento do tempo de resposta em pesquisas e diminuição do espaço necessário para o armazenamento dos índices (Silvestri, 2007) e (Raffaele, Salvatore, & Silvestri, 2004);

- algoritmos para cálculos de relevância do resultado de pesquisas (Zobel & Alistair, 2006);

- algoritmos para classificação de documentos semanticamente similares em clusters (Treeratpituk & Callan, 2006);

- algoritmos para pesquisas na web (Boldi, Paolo, & Vigna, 2004);

- algoritmos para criação de índices invertidos com foco na escalabilidade, através da utilização da computação distribuída em grande volume de dados (Grossman & Frieder, 2004) e (Dean & Ghemawat, 2004).

No presente trabalho será apresentado um método de criação de índices utilizando a computação distribuída. O próximo capítulo contém mais detalhes dos algoritmos envolvidos neste processo.

## 2.2. COMPUTAÇÃO DISTRIBUÍDA

Um sistema distribuído é aquele onde os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas passando mensagens (Coulouris, Dollimore, & Kindberg, 2007).

O compartilhamento de recursos é o principal fator de motivação para a construção de sistemas distribuídos. O desenvolvimento na área da computação distribuída é constantemente incentivado devido à possibilidade de utilizar em conjunto e coordenadamente recursos de processamento, memória e armazenamento para tratar de problemas de alto custo computacional (Coulouris, Dollimore, & Kindberg, 2007).

Um exemplo de um popular e eficaz sistema distribuído é a Internet. A Internet pode ser considerada um sistema distribuído que possibilita o compartilhamento de diversos serviços, como serviços *web*, de transferência de arquivos e de e-mails. Outros sistemas distribuídos de menor escala podem ser construídos a partir de computadores móveis e outros dispositivos computacionais portáteis ligados através de uma rede sem fio (Bray & Sturman, 2002).

A construção de sistemas distribuídos deve tratar problemas como: a concorrência entre componentes, a falta de um relógio global, a heterogeneidade dos componentes, a tolerância a falhas, a segurança, o balanceamento de carga, a escalabilidade e a disponibilidade (Stanoevska & Wozniak, 2009).

O compartilhamento de recursos interconectados em uma rede de computadores possibilita que ocorram situações onde dois ou mais componentes busquem obter ou alterar recursos exclusivos ao mesmo tempo. Uma arquitetura de sistemas distribuídos deve tratar este problema de concorrência de componentes.

Cada computador normalmente tem seu relógio físico próprio o que pode causar a divergência na marcação do tempo com a existência de vários computadores. É necessária a existência de algoritmos de sincronização de relógios ou mecanismos de comunicação específicos para tratar este potencial problema.

Os sistemas distribuídos podem ser construídos a partir de uma variedade de redes, *hardware*, sistemas operacionais e linguagens de programação. O popular protocolo de comunicação em rede TCP/IP pode mascarar a diferença de tecnologias de rede, mas a arquitetura do sistema distribuído deve tratar os outros casos de heterogeneidade (Tanenbaum, 2003).

Em um ambiente com um grande volume de máquinas heterogêneas interconectadas é comum a ocorrência de falhas. O sistema distribuído deve possuir um mecanismo de tolerância a falhas capaz de conhecer as formas de falhas dos componentes envolvidos e tratar cada uma dessas falhas apropriadamente.

Existe uma necessidade crescente de manter a integridade, a privacidade e a disponibilidade dos recursos em sistemas distribuídos. Devem existir políticas e mecanismos de segurança para manter a integridade das mensagens transmitidas na rede, para controlar o acesso aos recursos compartilhados e manter operacional o ambiente distribuído (Prodan & Fahringer, 2006). A seção 2.3 abordará de forma mais detalhada estes aspectos de segurança computacional.

Para evitar que determinados componentes de um sistema distribuído sejam subutilizados ou sobrecarregados, deve existir um mecanismo que realize a distribuição da carga de trabalho. O balanceamento de carga é a atividade que o sistema distribuído pode realizar para encontrar uma utilização eficiente dos recursos, maximizando a vazão de trabalho, minimizando o tempo de resposta e evitando a sobrecarga no ambiente (Kopparapu, 2002).

A arquitetura de um sistema distribuído deve ser capaz de se adequar a crescentes demandas pelos recursos compartilhados. O potencial de expansão da capacidade de compartilhar recursos a um número maior de usuários de acordo com o acréscimo de novos componentes chama-se escalabilidade. Devem ser criados algoritmos específicos para melhorar o desempenho de acordo com o acréscimo de novos componentes (Bond, 2000).

A disponibilidade de um sistema é a medida da proporção de tempo em que ele está pronto para uso (Coulouris, Dollimore, & Kindberg, 2007). Por implementarem mecanismos de tolerância a falhas, os sistemas distribuídos fornecem um alto grau de disponibilidade em casos de falhas de hardware. Quando um dos componentes do sistema falha o ambiente distribuído pode prover meios para passar um determinado processamento a outro componente do ambiente de forma transparente ao usuário.

Um importante meio de prover tolerância a falhas e disponibilidade em sistemas distribuídos é através da replicação de recursos. A replicação pode ser de dados, quando o mesmo dado é armazenado em diferentes dispositivos de armazenamento, ou de

computação, quando a mesma atividade computacional é executada diversas vezes até obter o resultado esperado (Wiesmann & Pedone, 2000). O processo de replicação deve ser transparente para o cliente do recurso. A replicação pode ser tanto ativa como passiva. Na replicação passiva, ou *backup* primário, os clientes de comunicam com uma réplica distinta. Já na replicação ativa os clientes se comunicam com todas as réplicas por meio de *multicast* (Tanenbaum, 2003).

### 2.2.1. Arquitetura de Sistemas Distribuídos

Os sistemas distribuídos podem ser modelados de acordo com o posicionamento de suas partes e o relacionamento entre elas. Exemplos desta arquitetura incluem o modelo cliente-servidor e o modelo ponto-a-ponto – *peer-to-peer* (Prodan & Fahringer, 2006). O modelo cliente-servidor, que é o predominante, é baseado no protocolo de requisição-resposta entre um cliente e um servidor para compartilhar recursos como páginas web, arquivos e nomes de domínio. No modelo *peer-to-peer* todos os componentes realizam funções semelhantes na exploração dos recursos de diversos computadores (Maarten & Tanenbaum, 2007). Nestes sistemas os dados e recursos computacionais são provenientes de muitos computadores de maneira uniforme, podendo abranger milhões de computadores e usuários necessitando de recursos compartilhados. Um problema comum neste tipo de arquitetura é a distribuição e o acesso aos recursos em muitos computadores de forma que equilibre a carga de trabalho e garanta a disponibilidade sem adicionar sobrecargas indevidas.

Um problema comum a todos os modelos de arquitetura de sistemas distribuídos é a forma de comunicação entre os componentes. O mecanismo de troca de mensagens em uma rede de computadores é a forma comum de comunicação entre os componentes do sistema distribuído (Stanoevska & Wozniak, 2009). Esta troca de mensagens pode sofrer atrasos e, devido à inexistência de um relógio global, devem existir métodos para tratar este tipo de problema. Os sistemas de troca de mensagens podem ser síncronos ou assíncronos. Nos sistemas síncronos devem ser impostos limites conhecidos para o tempo de execução de processos, para o tempo de entrega de mensagens e para o tempo de desvio do relógio. Nos sistemas assíncronos não existe a necessidade de imposição de um limite de tempo de execução de um processo, de entrega de mensagens e de desvio no relógio (Stanoevska & Wozniak, 2009).

### 2.2.2. Classificação de Sistemas Distribuídos

O método mais utilizado de classificação dos sistemas distribuído é de acordo com o escopo dos recursos que ele compartilha (Stanoevska & Wozniak, 2009). De acordo com este critério os sistemas distribuídos podem ser classificados como sistemas em grade e sistemas em cluster.

Os sistemas distribuídos em grade são tipos de sistemas distribuídos que compartilham recursos como arquivos, computadores, programas, dados e sensores em uma escala muito grande (Maarten & Tanenbaum, 2007). O termo grade (*grid*) também é comumente utilizado para referenciar o *middleware* do sistema distribuído, ou seja, a camada que implementa todos os controles de um sistema distribuído, como: concorrência, tolerância a falhas, balanceamento de carga, comunicação entre processos, escalabilidade e disponibilidade. Normalmente, os recursos são compartilhados por grupos de usuários em diferentes organizações, que estão colaborando na resolução de problemas que exigem um grande número de computadores para resolvê-lo (Maarten & Tanenbaum, 2007).

Os sistemas distribuídos em cluster consistem em um conjunto de computadores heterogêneos, fracamente acoplados, geralmente localizados em uma rede local de alta velocidade. Eles são o resultado de uma convergência de um número de tendências como a disponibilidade de muitos microprocessadores com um custo reduzido, a existência de redes locais de alta velocidade e softwares robustos para o gerenciamento dos recursos de uma arquitetura de sistema distribuído. Eles geralmente são projetados para poder fornecer desempenho, escalabilidade e disponibilidade utilizando um conjunto de computadores com custo reduzido ao invés do investimento em caros supercomputadores (Prodan & Fahringer, 2006).

Um termo atualmente popular é o de computação nas nuvens (Stanoevska & Wozniak, 2009). Contudo este não é um novo tipo de classificação de sistemas distribuídos baseado no escopo de compartilhamento de seus recursos. Este termo é utilizado para designar um modelo de arquitetura de sistema baseado na Internet ou em uma Intranet coorporativa (Chen & Bairagi, 2010). O termo nuvens é uma metáfora que representa uma abstração de uma complexa infra-estrutura técnica de interconexão de computadores, como a Internet. Desta forma, o termo computação nas nuvens representa um paradigma de utilização de

um sistema distribuído. Na computação nas nuvens a informação é permanentemente armazenada em servidores e temporariamente armazenada na memória de clientes, como: computadores, notebooks, celulares e *tablets* (Knorr & Gruman, 2009). Computação nas nuvens é um conceito genérico que utiliza o software como um serviço que depende da rede (Internet ou Intranet) para prover suas necessidades (Chen & Bairagi, 2010).

### 2.2.3. Comunicação em Sistemas Distribuídos

Os sistemas distribuídos devem fornecer meios para a comunicação entre os diversos componentes do ambiente. Foi visto na seção 2.2.1 que a troca de mensagens é a forma comum de comunicação no ambiente distribuído.

Paradigmas de comunicação comumente utilizados em um ambiente distribuído é a comunicação cliente-servidor, onde as mensagens de requisição e resposta fornecem a base para a invocação de método remoto, e a comunicação em grupo onde uma mesma mensagem é enviada para vários processos.

O protocolo de rede padrão de boa parte dos sistemas distribuídos é o protocolo TCP/IP (Tanenbaum, 2003). Na pilha do protocolo TCP/IP existem os protocolos da camada de transporte que são o UDP – *User Datagram Protocol* e o TCP – *Transmission Control Protocol*. O UDP fornece um recurso simples de passagem de mensagem, que sofre de falhas por omissão, mas não tem penalidades de desempenho incorporadas. Já o protocolo TCP garante a entrega das mensagens, mas à custa de mensagens adicionais, latência e custos de armazenamento mais elevados.

No modelo de comunicação cliente-servidor em sistemas distribuídos o UDP é recomendado para a criação de um eficiente protocolo de comunicação de propósito específico. A mensagem de resposta constitui uma confirmação para a mensagem de requisição, evitando assim sobrecargas desnecessárias. Recursos adicionais podem ser acrescentados para tornar este protocolo mais confiável, como identificadores de mensagens e retransmissões (Maarten & Tanenbaum, 2007).

No modelo de comunicação em grupo pode ser utilizado o recurso de mensagem *multicast* do protocolo IP – *Internet Protocol*. O protocolo IP é um protocolo da camada de rede da

pilha TCP/IP que fornece meios de endereçamento e distribuição de pacotes (Tanenbaum, 2003). O *multicast* IP é um método de enviar um datagrama IP para um grupo específico de destinatários em uma única transmissão. Para isto são utilizados blocos específicos de endereçamento IP. O recurso de *multicast* possibilita a comunicação entre membros de um grupo de processos, podendo tanto ser utilizado em redes locais como na Internet. O *multicast* IP tem a mesma semântica de falhas que os datagramas UDP, sofrendo de falhas por omissão. Mas mesmo assim é uma ferramenta útil para muitas aplicações *multicast* como a comunicação em grupo nos sistemas distribuídos (Coulouris, Dollimore, & Kindberg, 2007).

Existem dois principais paradigmas de programação distribuída que são utilizados para a comunicação em um sistema distribuído, que são a invocação a método remoto e os sistemas baseados em eventos. Estes paradigmas consideram os objetos distribuídos como entidades independentes que podem se comunicar. Na invocação a método remoto, um método na interface remota de um determinado objeto é invocado de forma síncrona, com o invocador esperando por uma resposta. Nos sistemas baseados em eventos, a mensagem é enviada de forma assíncrona para vários assinantes, quando ocorre um evento em um objeto de interesse. Estes métodos de programação distribuída são comumente utilizados para implementar a comunicação síncrona e assíncrona em um ambiente distribuído (Prodan & Fahringer, 2006).

### 2.2.4. Sistema de Arquivos Distribuído

Um sistema de arquivos distribuído permite aos programas armazenarem e acessarem arquivos remotos exatamente como se fossem locais, permitindo que os usuários acessem os arquivos de qualquer computador em uma rede (Coulouris, Dollimore, & Kindberg, 2007). O desempenho e a segurança no acesso aos arquivos armazenados remotamente devem ser comparáveis ao armazenamento em um computador local.

Os computadores clientes não têm acesso direto aos blocos de armazenamento de dados, mas interagem através da rede utilizando um protocolo de comunicação. Isto possibilita que seja restringido o acesso ao sistema de arquivos de acordo com listas de controle de acesso tanto de clientes como de servidores, dependendo de como o protocolo é projetado (Tanenbaum, 2010).

Alguns sistemas de arquivos distribuídos simulam a interface de arquivos do UNIX com diferentes graus de escalabilidade, tolerância a falhas e variações na semântica. Exemplos de implementações simples de sistemas de arquivos distribuídos são o *Network File System* – NFS e o *Andrew File System* – AFS (Callaghan, 1999) e (Campbell, 1993).

O NFS é a tecnologia de sistemas de arquivos distribuído dominante contando com implementações otimizadas e suporte de hardware de alto desempenho (Coulouris, Dollimore, & Kindberg, 2007). O NFS é um padrão aberto descrito nas RCFs 1094, 1813, 3010 e 5661 (Request For Comments, 2011). Um Servidor NFS disponibiliza os dados remotos para os clientes. Neste servidor é possível fazer a administração de quais dados serão compartilhados e o controle de acesso aos mesmos. Aplicativos clientes NFS se comunicam com o servidor através de chamadas remotas a procedimentos. Uma vez que o cliente NFS se conectou com o servidor o usuário do sistema de arquivos local pode acessar os arquivos remotos como se fossem locais (Callaghan, 1999).

Os sistemas de arquivos distribuídos mais modernos exploram novos modos de organização dos dados no disco, ou entre vários servidores, para obter alto desempenho, tolerância a falhas, capacidade de mudar de escala e fazer balanceamento de carga. Alguns utilizam replicação de arquivos ou de bloco de dados, como: *Hadoop Distributed File System* – HDFS e o *Google File System* – GFS (HDFS, 2011) e (Ghemawat, Gobioff, & Leun, 2003). Outros são implementados através de comunicação *peer-to-peer* (*Pastry* e *Tapestry*) ou especificamente para armazenar informações multimídia, como o servidor de arquivos de vídeo *Tiger* (Rowstron & Druschel, 2001), (Zhao & Huang, 2004) e (Bolosky & Fitzgerald, 1996). Com a implementação de mecanismos de tolerância a falhas estes sistemas continuam operacionais e sem perda de dados quando algum componente sai de operação. O sistema de arquivos distribuído HDFS será descrito com mais detalhes no capítulo 3.

### 2.2.5. Modelos de Programação Paralela

O compartilhamento de recursos de processamento é um fator que motiva o uso de sistemas distribuídos em diversas áreas da ciência e engenharia. Exemplos de aplicações que necessitam de um grande poder de processamento são: análises na atmosfera terrestre, física nuclear, biotecnologia, genética, química molecular, engenharia mecânica,

engenharia elétrica e diversas necessidades de processamentos matemáticos e de ciência da computação (Ian, 1995) e (Kaminsky, 2010).

Os sistemas distribuídos possibilitam que recursos de processamento de diversos computadores possam ser compartilhados para resolução de um problema, através de um ambiente com controle à concorrência, tolerância a falhas, balanceamento de cargas, escalabilidade e disponibilidade. Contudo, o algoritmo de resolução de determinado problema deve ser construído de forma específica para utilizar os benefícios da computação distribuída.

Tradicionalmente os algoritmos são construídos para um processamento serial (Ian, 1995). Ou seja, são construídos para serem executados em uma única unidade central de processamento (CPU – *Central Processing Unit*). O problema é dividido em uma série discreta de instruções, as instruções são executadas uma após a outra e apenas uma instrução é executada em um dado momento.

De uma forma simplificada a programação paralela é uma forma de construir algoritmos para que ele possa fazer uso simultâneo de múltiplos recursos computacionais para resolver um problema (Kaminsky, 2010). O propósito da programação paralela é que o algoritmo possa fazer uso de múltiplas CPUs, que o problema possa ser dividido em partes que podem ser resolvidas de forma concorrente, que cada parte possa ser dividida em uma série de instruções e que as instruções de cada parte possam ser executadas simultaneamente em diversas CPUs.

Os recursos computacionais podem ser um único computador com múltiplos processadores, um número arbitrário de computadores conectados em rede ou uma combinação de ambos. A resolução do problema computacional deve ser modelada de forma a poder ser dividida em diversas partes que possam ser resolvidas simultaneamente, executar múltiplas instruções em qualquer momento e ter a capacidade de ser solucionada em menos tempo com a utilização de múltiplos recursos computacionais, ao invés de utilizar um único recurso computacional (Kaminsky, 2010).

Um modelo de programação paralela pode ser implementado de diversas formas, como: a utilização de bibliotecas específicas que são chamadas por linguagens de programação

seqüenciais, extensões em linguagens de programação ou mesmo um modelo de execução inteiramente novo. Os modelos de programação paralelos podem ser divididos de forma geral em duas áreas: modelos baseados na forma de interação entre os processos e modelos baseados na forma de decomposição do problema (Kaminsky, 2010).

Os modelos baseados na interação dos processos estão relacionados aos mecanismos pelos quais os processos paralelos se comunicam. Os tipos mais comuns de interação entre processos são o compartilhamento de memória e a transmissão de mensagens, sendo que eles podem ser implícitos também (Kaminsky, 2010). No modelo de memória compartilhada os processos paralelos compartilham um espaço de endereçamento comum nos quais eles podem ler e escrever de forma assíncrona. Os sistemas de memória compartilhada podem ser emulados através de sistemas distribuídos de compartilhamento de memória. Exemplos deste modelo são o *OpenMP* (*Open Multi-Processing*) para uma programação com memória compartilhada e o MPI (*Message Passing Interface*) para programação com memória distribuída (Chen & Bairagi, 2010). Já nos sistemas de transmissão de mensagens os processos paralelos trocam dados uns com os outros através do envio de mensagens. Esta comunicação pode ser síncrona ou assíncrona. Exemplos destes sistemas são os sistemas baseados em objetos distribuídos e em chamada remota a procedimentos como: Java RMI e SOAP (Java RMI, 2011) e (SOAP, 2011). No modelo de troca de interação implícito a comunicação entre os processos não está visível ao programador, ao invés disto o compilador ou o ambiente de execução é responsável por implementar esta comunicação. Linguagens de programação que implementam este tipo de paralelismo são a MATLAB M-code e NESL (Ferreira, 2009) e (Blelloch, Programming Parallel Algorithms, 1996).

Os modelos baseados na forma de decomposição do problema estão fundamentados no modo pelo qual o problema é dividido em partes que podem ser executadas simultaneamente. Os tipos de modelos baseados na decomposição do problema são o paralelismo de tarefas e o paralelismo de dados, sendo que os dois também podem ser implementados de forma implícita (Chen & Bairagi, 2010). O foco do modelo de tarefas paralelas está nos processos (ou nas linhas de execução - *Threads*). Cada processo deste modelo deverá ter um comportamento distinto para tornar possível sua execução paralela. Neste modelo existe uma ênfase na necessidade de comunicação e a passagem de mensagem é o modo padrão de implementar esta comunicação. Exemplos de linguagens de

tarefas paralelas são a Verilog e a VHDL (Bergeron, 2003). Já no modelo com o paralelismo de dados o foco está na execução dos processos em um conjunto de dados que geralmente está estruturado em forma de um vetor n-dimensional. Um conjunto de tarefas irá processar estes dados de forma independente e simultânea em distintas partições de dados. Em um sistema com memória compartilhada os dados estarão disponíveis a todos os processos, já em um sistema de memória distribuída os dados estarão distribuídos entre as memórias e processados localmente. Existem extensões de linguagens de programação que implementam este tipo de paralelismo, como a Ateji (Ateji PX for Java, 2011). Já em um modelo de decomposição de problema implícito o processo de divisão de tarefas e de dados é realizado pelo próprio ambiente de execução, exemplos de implementação deste modelo estão nas linguagens Axum e SISAL (Axum, 2011) e (SISAL, 2011).

Geralmente os modelos de programação paralelos são construídos baseados em extensões de linguagens de programação ou em bibliotecas de ambientes de execução (Chen & Bairagi, 2010). Alguns exemplos extensões de linguagens de programação são: *OpenMP*, *OpenCL – Open Computing Language* e UPC – *Unified Parallel C*. Exemplos de bibliotecas de implementação de programação distribuída são: TBB – *Thread Building Blocks*, *MapReduce* e *Hadoop*, MPI – *Message Passing Interface* e CUDA – *Compute Unified Device Architecture*.

No capítulo 3 será analisado do algoritmo *MapReduce* e a ferramenta *Apache Hadoop*. Será feita uma análise comparativa com outras ferramentas e bibliotecas de programação distribuída existentes.

## 2.3. SEGURANÇA COMPUTACIONAL

Esta seção contém conceitos básicos de segurança computacional. Estes conceitos servirão de base para uma melhor compreensão da avaliação dos aspectos de segurança da prova de conceito elaborada, a qual é apresentada no capítulo 4.

O compartilhamento de recursos através de um ambiente distribuído ocasiona uma necessidade de proteger a integridade e privacidade dos recursos disponibilizados. Os aspectos de segurança em sistemas distribuídos estão fundamentalmente relacionados com a proteção nos canais de comunicação do ambiente e com o acesso aos recursos

compartilhados (Coulouris, Dollimore, & Kindberg, 2007). Políticas de segurança devem ser especificadas para estabelecer os limites pelos quais os recursos compartilhados poderão ser utilizados. Estas políticas de segurança deverão ser implementadas através de mecanismos de segurança, ou seja, os mecanismos de segurança são os meios pelos quais as políticas de segurança são implementadas (Anderson, 2008).

Os mecanismos de segurança são baseados em criptografia de chave pública e de chave secreta. Os algoritmos de criptografia misturam as mensagens de uma maneira que não possam ser revertidas sem a chave de decifração, ou seja, eles cifram as mensagens para posteriormente decifrá-las. A criptografia de chave secreta é simétrica, onde a mesma chave que foi utilizada para cifrar também deve ser utilizada para decifrar as mensagens. Se duas entidades possuem uma chave secreta que só elas conhecem elas podem compartilhar recursos de forma segura e com a garantia de autenticidade. Importantes algoritmos de chave secreta são: DES e AES. Já a criptografia de chave pública é assimétrica, são utilizadas chaves diferentes para cifrar e decifrar a mensagem. Uma chave é tornada pública e uma entidade a utiliza para cifrar informações que só poderão ser decifradas com o portador da chave privada. A chave pública está relacionada com a chave privada, mas é computacionalmente inviável derivar uma chave da outra. O portador da chave privada pode assinar mensagens e certificados. Os certificados podem ser utilizados como credenciais para ter acesso a recursos protegidos. Importantes algoritmos de criptografia assimétrica são: ElGammal e RSA. Tanto com a criptografia de chave simétrica quanto na assimétrica quanto maior o tamanho da chave mais tempo levará para cifrar a informação e mais difícil será para decifrar a informação por métodos de ataques de força bruta e até mesmo ataques por dicionário (Stinson, 2006).

A criptografia é um mecanismo de segurança computacional que fornece a base para a autenticação de mensagens, assim como sua integridade e privacidade. Contudo é necessária a existência de protocolos de segurança especialmente projetados para utilizá-la de forma segura. A escolha dos algoritmos de criptografia e do método de gerenciamento das chaves é fundamental para a eficácia, desempenho e utilidade dos mecanismos de segurança. A criptografia de chave pública facilita a distribuição de chaves, mas seu desempenho não é o melhor para cifrar grandes volumes de dados. Já a criptografia de chave secreta possui um bom desempenho em um elevado volume de dados. Protocolos mistos, como o TLS – *Transport Layer Security*, negociam as chaves secretas em um canal

seguro estabelecido pela criptografia assimétrica, e utilizam as chaves secretas criadas para cifrar e decifrar as mensagens (Stallings, 2008).

Os recursos compartilhados através de um sistema distribuído podem ser protegidos através de mecanismos de controle de acesso. Os mecanismos de controle de acesso atribuem direitos aos portadores de credenciais para efetuarem operações nos recursos distribuídos. Os direitos podem ser mantidos em listas de controle de acessos associadas aos recursos compartilhados e às credenciais. É função do mecanismo de controle de acesso fornecer meios para a autenticação e a autorização. A autenticação serve para a confirmação se uma entidade é quem ela afirma ser, obtendo assim a credencial de acesso aos recursos. A autorização serve para disponibilizar o acesso apenas aos recursos que a credencial autenticada possibilita. O primeiro protocolo de autenticação amplamente utilizado foi o *Needham-Schroeder*, servindo de base para outros protocolos. Um importante protocolo de controle de acesso atual é o *Kerberos*, que efetua a autenticação de usuários e a autorização de acesso a serviços dentro de uma organização (Stallings, 2008).

Desta forma, através da definição de uma política de segurança e com a utilização de modernos mecanismos de segurança é possível proporcionar um bom nível de segurança tanto nos canais de comunicação de um ambiente distribuído como no controle de acesso aos recursos compartilhados.

## 2.4. INFORMÁTICA FORENSE

Os computadores tornaram-se parte da vida de boa parte da população atual, seja em atividades empresariais ou pessoais. Com isto surgem também novos meios para a prática de crimes. Ações delituosas cometidas com a utilização de computadores, redes de computadores ou demais recursos de Informática, são denominadas crimes por computador ou crimes informáticos (DPF, 2010).

As atividades ilícitas podem ser auxiliadas pelos computadores ou ter como alvo os sistemas computacionais propriamente ditos. Exemplos de atividades ilícitas que podem ter o auxílio tecnológico são as fraudes previdenciárias, as evasões de divisas, fraude em licitações, fraudes ao sistema financeiro e jogos ilegais. Alguns exemplos de atividades ilícitas que tem os sistemas computacionais como alvo são: furto e roubo de equipamento

ou de dados, divulgação e venda de material ilícito (pornografia infanto-juvenil, racismo e drogas) e ataques contra a integridade de serviços (vírus, negação de serviços, pichações de sítios da Internet) (Eleutério & Machado, 2011).

Estas atividades ilícitas praticadas com o auxílio de computadores ou contra os próprios recursos computacionais deixam vestígios. Os computadores armazenam e trocam informações através de meios físicos. Exemplos de locais onde se encontram evidências são: registros de autenticação de usuários, registros de conexões ou utilização de serviços na Internet/Intranet e dispositivos de armazenamento computacional (Eoghan, 2011).

Para a elucidação de determinada atividade ilícita deve-se conhecer os fatos ocorridos, preservar os vestígios encontrados, analisar os vestígios relevantes em busca de evidências probatórias do ilícito e apresentar provas convincentes. Estas provas convincentes podem elucidar a dinâmica do acontecimento do fato ilícito, provar a existência concreta da fraude e até mesmo identificar os autores (Saferstein, 2010). A criminalística é a ciência forense que utiliza os conhecimentos técnicos existentes em diversas áreas do conhecimento para provar cientificamente a existência, a dinâmica e os autores de determinada atividade ilícita. A perícia forense é o procedimento, que utiliza as ciências forenses, para a produção de provas científicas demonstrando a ocorrência de determinada atividade ilícita, bem como sua dinâmica e autoria (Espindula, Jesus, & Geiser, 2011).

A Informática Forense pode ser definida como o conjunto de procedimentos realizados sobre mídias de armazenamento e outros equipamentos computacionais em busca de evidências eletrônicas (Eleutério & Machado, 2011). Assim verifica-se que a Informática Forense é uma ciência forense que subsidia os processos de coleta, preservação, análise e apresentação de evidências eletrônicas que possam ser utilizadas e sustentadas em juízo (FBI, 2011).

A Informática Forense é uma das mais jovens disciplinas da Criminalística, surgida após a década de 1960 com a popularização dos computadores e das telecomunicações. Com a popularização da Internet e da telefonia móvel em meados da década de 1990, as Ciências Forenses também se modernizaram e a Informática Forense ganhou destaque no âmbito da Criminalística ao redor do mundo (Espindula, Jesus, & Geiser, 2011).

### 2.4.1. A Atividade de Análise Pericial

É necessária a realização de procedimentos bem definidos e documentados para a produção de provas que possam ter utilizadas e sustentadas em juízo. A atividade de busca e apreensão constitui o marco inicial de coleta das evidências que serão posteriormente analisadas. Este procedimento é realizado no contexto de atividades ilícitas criminais. Em atividades ilícitas civis e administrativas, podem ser utilizados procedimentos semelhantes em um processo similar de auditoria (Eleutério & Machado, 2011).

A figura 2.2 ilustra como o contexto da busca e apreensão para a análise pericial[1].

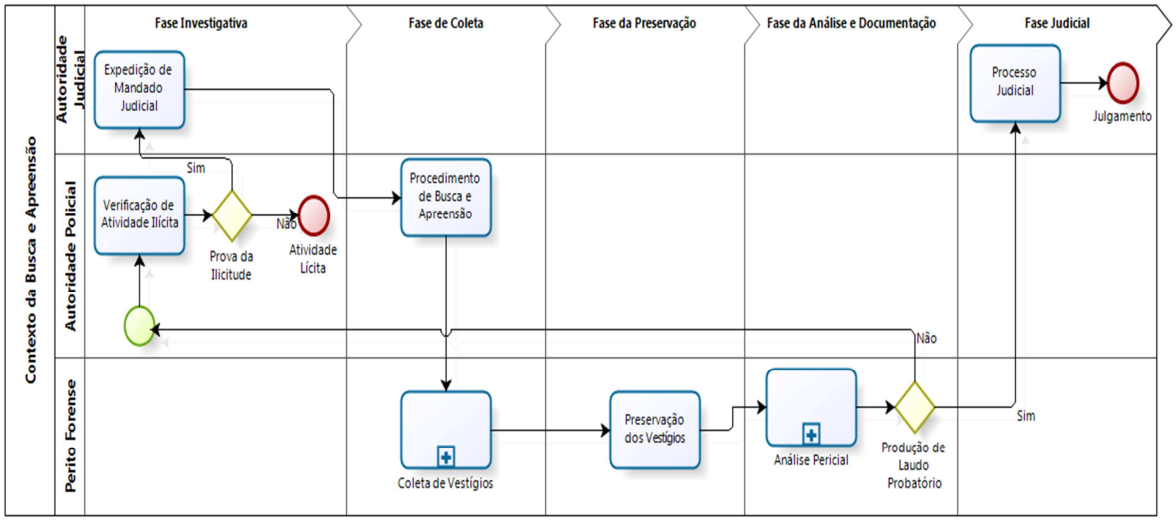



Figura 2.2 – Contexto da Busca e Apreensão na Análise Pericial

O processo de produção de provas periciais está contido em um conjunto de procedimentos de elucidação de atividades ilícitas. Inicialmente existe a fase investigativa que é realizada

[1] Esta figura foi simplificada para destacar a atividade de análise forense no contexto de uma busca e apreensão. Contudo, serão fornecidas algumas considerações a respeito do papel do investigado neste contexto. A perícia forense serve tanto para provar a culpa como a inocência de determinado investigado no processo. Nesse processo forense não se tem contato com o investigado, para possibilitar uma produção de prova técnica com isenção. A prova diretamente encaminhada pelo investigado raramente é analisada, pois é suspeita. A prova deve ser encontrada no local de origem da mesma para ter isenção. O inquérito policial é um processo inquisitório, sem ampla defesa. A defesa do investigado é facultativa e para esclarecimentos adicionais. É na fase judicial que a defesa do investigado é obrigatória. Nessa fase existe, inclusive, a figura do assistente técnico, que é um profissional que analisa as provas produzidas pelos peritos podendo questioná-las e pedir esclarecimentos de questões, o que não ocorre na fase do inquérito policial (Espindula, Jesus, & Geiser, 2011).

pelas autoridades policiais e judiciárias, ou seja, por policiais e juízes. A autoridade policial verifica a existência ou não de determinada atividade ilícita através de procedimentos investigativos. Quando é necessária a obtenção de mais dados, para posterior produção de provas, solicita à autoridade judicial um mandado de busca e apreensão para obter estas informações. Assim inicia-se a fase de coleta onde será realizada a busca e apreensão pela autoridade policial e a coleta de vestígios pelos peritos forenses. A atividade de busca e apreensão deverá arrecadar todos os materiais envolvidos com a atividade ilícita para posterior análise. Os peritos forenses devem coletar os vestígios de forma a manter a integridade dos mesmos, evitando adulterações. No laboratório, os peritos forenses realizam a atividade de preservação de vestígios para impedir que sejam realizadas alterações substanciais nos vestígios coletados. Com os vestígios preservados, é realizada a análise dos mesmos de forma a buscar elucidar a existência da atividade ilícita, sua dinâmica e autoria. Esta análise pode até mesmo provar a inexistência de atividade ilícita. Este procedimento é realizado pelo perito forense na fase de análise pericial. A atividade da análise pericial produz um documento que contém o resultado dos exames realizados. Se o resultado for conclusivo será encaminhado para a autoridade judicial através de um Laudo Pericial (Saferstein, 2010).

O Laudo Pericial é uma prova técnico-científica, que será utilizada no processo judicial até o julgamento. Se os dados coletados não forem conclusivos serão encaminhados através de um documento técnico para subsidiar novas investigações pela autoridade policial, ou para indicar que a investigação deve ser finalizada pela inexistência de atividade ilícita (Saferstein, 2010).

A informática forense define procedimentos específicos que devem ser realizados na fase de coleta, preservação e análise de vestígios eletrônicos. A fase de coleta deve ser realizada de forma a não perder informações voláteis e acondicionar os materiais de forma a evitar que os dados armazenados sejam danificados. Na fase de preservação os dados coletados devem ser duplicados de forma a não alterar a mídia original (Eleutério & Machado, 2011) e (Carrier, 2005).

Todo o procedimento de análise deve ser realizado nas cópias dos dados de forma a preservar os dados originais e possibilitar a reprodução dos resultados obtidos. Na fase de análise é extraído os dados relevantes das mídias apreendidas e analisado este material para

determinar a existência, dinâmica e autoria das atividades ilícitas. O Laudo Pericial deve possuir mecanismos que possam garantir sua integridade e autenticidade para impedir fraudes no decorrer do processo judicial (Eleutério & Machado, 2011) e (Carrier, 2005).

As atividades de coleta e análise pericial serão mais detalhadas para verificar o impacto das mesmas no processo de produção de provas. A figura 2.3 contém um detalhamento da fase de coleta.

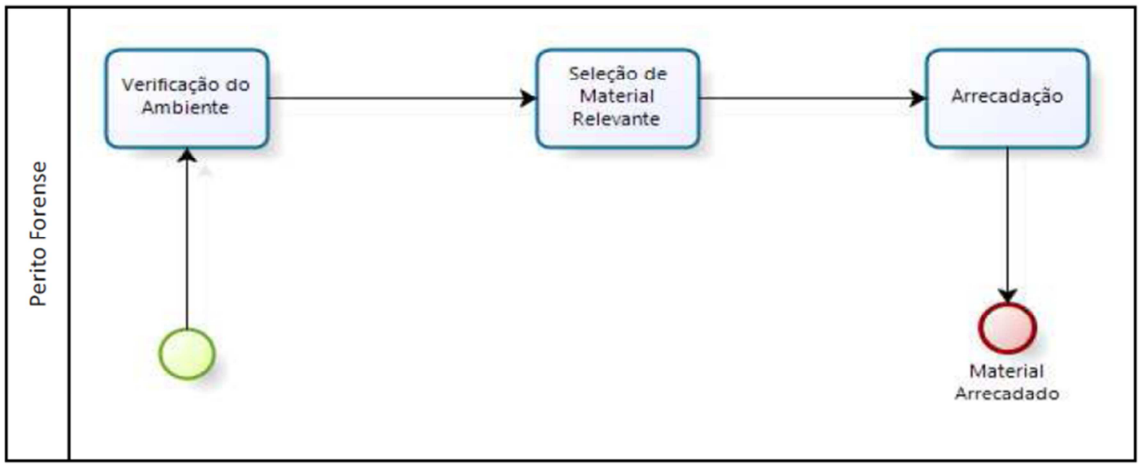



Figura 2.3 – Coleta de Vestígios Computacionais

Na atividade de coleta inicialmente deve-se verificar o ambiente onde será realizada a coleta. Devem ser adotados procedimentos específicos para coleta de vestígios de acordo com o que for encontrado no ambiente.

Vestígios computacionais podem ser encontrados em computadores pessoais e portáteis, servidores diversos, infra-estrutura de rede cabeada e móvel, além de outros dispositivos de armazenamento computacional. Mas deve ser arrecadado apenas o material relevante para determinado caso (DPF, 2010).

Com a seleção dos materiais relevantes os mesmos deverão ser arrecadados e acondicionados apropriadamente para posterior análise. A seleção do material relevante é chamada também de triagem de materiais (DPF, 2010).

A análise pericial é a atividade que irá realizar a produção de provas. A figura 2.4 contém uma visão geral das fases de uma análise pericial de informática.

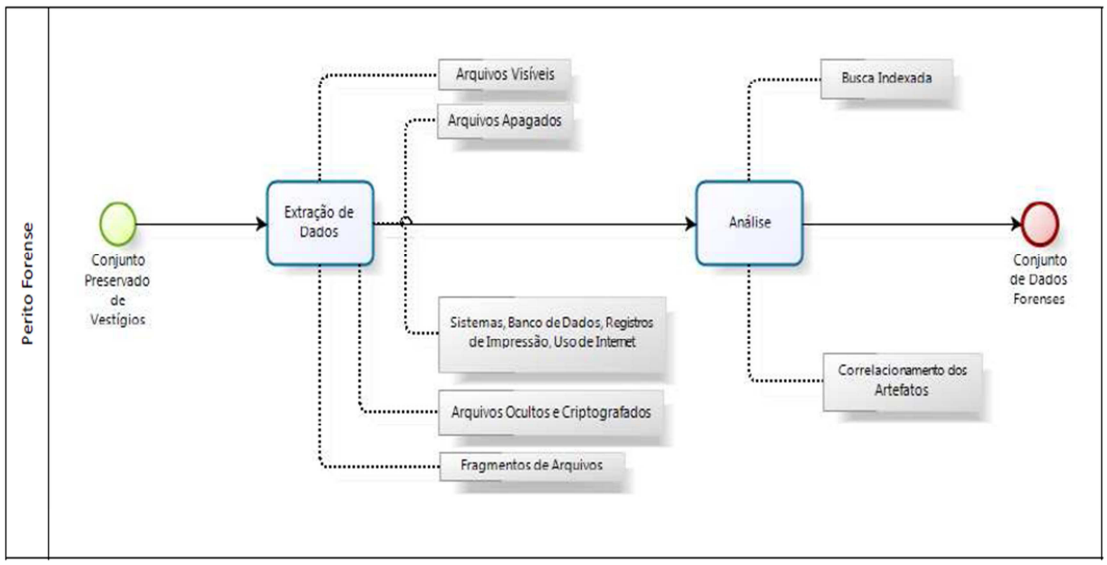



Figura 2.4 – Visão Geral da Análise de Vestígios Computacionais

A atividade de análise inicia-se com o fornecimento de um conjunto de vestígios já adequadamente preservados. É feita então uma extração de diversos artefatos presentes neste conjunto de vestígios. Exemplos de artefatos que podem ser extraídos são: arquivos visíveis e apagados, sistemas, banco de dados, registros de impressões, arquivos criptografados e fragmentados. Este conjunto de artefatos é analisado de acordo com cada tipo de atividade ilícita que pode estar sendo objeto de perícia. Uma atividade comum na análise pericial é a busca nos artefatos por palavras-chave. Esta busca dever ser indexada devido ao grande volume de artefatos que podem ser extraídos. Com a utilização da busca indexada é realizado um correlacionamento das informações contidas nos artefatos para encontrar vestígios das atividades ilícitas. O conjunto de artefatos encontrados que possuem vestígios de relevância são disponibilizados para posterior confecção do documento final. Os dados forenses são os dados resultantes de uma análise pericial comum que resulta na extração de uma grande quantidade de dados de usuário (Eoghan, 2011) e (FTK, 2011).

### 2.4.2. A Necessidade de uma Análise Correlacionada de Vestígios

Com o desenvolvimento tecnológico atual, a quantidade de dispositivos de armazenamento computacional presente no dia-a-dia das pessoas está cada vez maior. De acordo com a lei de *Kryder* (Walter, 2005), a capacidade dos dispositivos de armazenamento computacional tende a dobrar a cada ano, em uma velocidade até maior que o desenvolvimento dos

semicondutores dos processadores. Consequentemente, a quantidade de dados que uma análise forense computacional deve tratar é crescente.

Em 2010 os especialistas da Polícia Federal Brasileira elaboraram 9.050 laudos resultando em uma análise aproximada de 4.6 PB de dados sobre crimes cibernéticos (Bezerra, 2011).

Uma análise pericial comum muitas vezes apenas realiza a extração dos dados do usuário da máquina. Esta extração e disponibilização de dados geralmente é realizada individualmente para cada mídia de armazenamento computacional (Carrier, 2005). O processo de investigação consiste na pesquisa dessa massa de dados sobre evidências de crimes, correlacionamento de informações e apresentação de resultados (Galvão, 2006 Issue).

Muitas operações policiais nacionais brasileiras resultam na apreensão de centenas de computadores (Bezerra, 2011). Uma típica análise técnica pericial resulta em alguns gigabytes de dados para serem posteriormente analisados. Analisar de forma correlacionada este volume crescente de dados forenses exige a utilização de técnicas e recursos computacionais eficientes que nem sempre estão disponíveis, tornando muito onerosa esta atividade (Werneck, 2007).

Por análise correlacionada de dados entende-se a análise de dados de diversas mídias computacionais distintas, mas que pertencem ao mesmo caso analisado.

A figura 2.5 faz uma simplificação das fases descritas na figura 2.2 para detalhar o contexto de uma análise correlacionada de evidências. Tanto os trabalhos realizados nas atividades de investigação policial como no processo judicial podem demandar a coleta de vestígios adicionais. Em uma grande operação podem ser arrecadados dispositivos computacionais de diversos alvos, sendo que a figura ilustra apenas três. Os vestígios coletados são preservados. Após a preservação pode ser feita uma triagem nestes vestígios ou eles podem todos ser encaminhados diretamente para a extração de artefatos. A possibilidade da triagem é relevante já que diminui o volume de dados a ser analisado posteriormente. Um modo de realizar a triagem de dados é através da busca indexada, onde podem ser fornecidas palavras-chave específicas para cada caso e selecionar apenas os dispositivos computacionais que possuem registros destas palavras. A fase de análise

pericial apenas realiza a extração de todos os artefatos de interesse do caso analisado. Este conjunto de artefatos, ou seja, estes dados forenses são todos agrupados para a realização de uma análise correlacionada de evidências. Um método de realizar esta análise correlacionada é através de buscas indexadas, onde termos-chave podem ser pesquisados nos dados forenses encontrados em todos os alvos do caso. Pode ser feito um correlacionamento de informações presentes em diversos alvos para comprovar atividades ilícitas de grandes organizações. O resultado da análise correlacionada de evidências pode servir como subsídio para novas investigações ou mesmo como prova técnico-científica em processos judiciais (DPF, 2010) e (FTK, 2011).

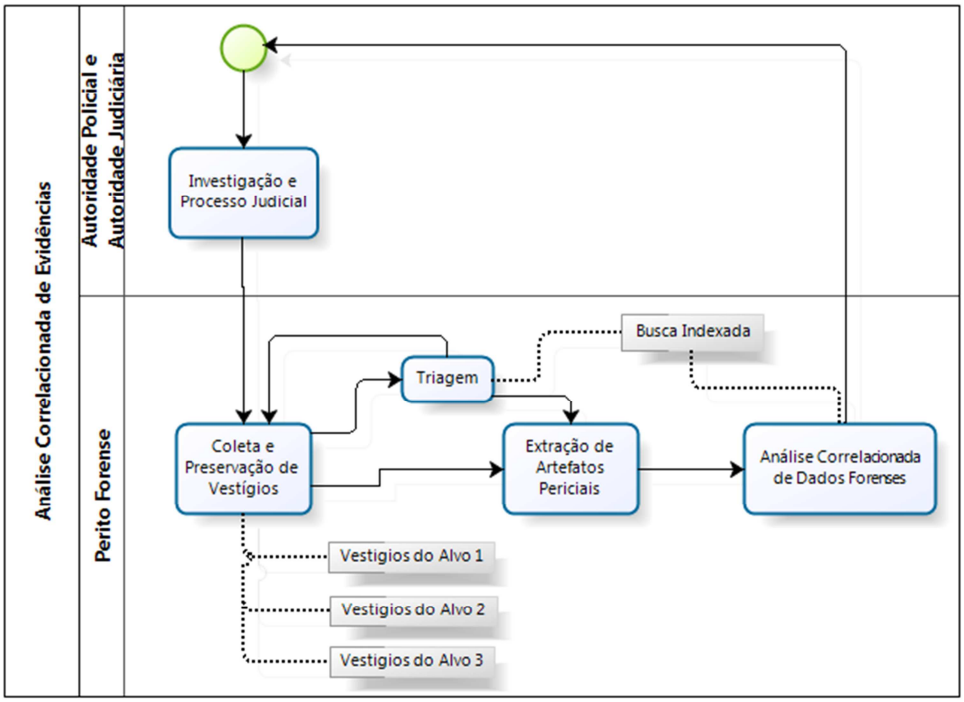



Figura 2.5 – Análise Correlacionada de Evidências

### 2.4.3. A Importância da Indexação na Análise de Dados Forenses

Uma operação fundamental na análise correlacionada de informações é a pesquisa no conjunto de dados. (Manning, Raghavan, & Shutze, 2008) explicam que realizar a pesquisa em um grande conjunto de dados não estruturados é um processo oneroso e pouco eficiente. Uma forma de evitar que seja realizada uma busca linear em todo o conteúdo de uma massa de dados é através da criação prévia de índices nestes dados.

O processo de criação computacional destes índices é chamado de indexação (Manning, Raghavan, & Shutze, 2008). A atividade de indexação em um grande volume de dados é

um processo oneroso, tendo em vista os diversos procedimentos que deve ser realizados. Deve ser realizada a leitura de cada diferente tipo de arquivo e construção de uma estrutura de dados que armazene de forma homogênea os dados relevantes contidos nestes diferentes tipos de arquivos. Posteriormente, é necessário realizar uma análise nesta estrutura de dados para separar os termos que serão indexados, incluindo metadados importantes como a posição e frequência de cada termo no arquivo. Por último é realizada a criação do índice propriamente dito, com a execução do algoritmo de criação do índice na estrutura de dados previamente criada (FTK, 2011).

Com um índice criado a pesquisa por palavras-chave em um grande volume de dados é realizada com maior eficiência, tendo em vista que não é necessário pesquisar o conteúdo de todos os dados forenses, sendo necessário apenas realizar a pesquisa na estrutura otimizada de dados do índice para localização do conteúdo (Manning, Raghavan, & Shutze, 2008).

Uma forma computacionalmente eficiente para realizar a indexação é necessária tendo em vista que muitas vezes o tempo necessário para se obter o resultado de uma investigação é reduzido (DPF, 2010). Este trabalho propõe a utilização da computação distribuída para realizar a indexação dos dados forenses.

### 2.4.4. O Armazenamento de Dados Forenses

Uma importante fase no processo de indexação consiste na aquisição dos dados forenses. Para armazenar um crescente volume de dados é necessário um considerável espaço de armazenamento computacional.

Uma possível alternativa é a aquisição de robustos servidores de arquivos para realizar o armazenamento destes dados de forma eficiente, com disponibilidade e tolerância falhas (Schulz, 2011). Outra alternativa, geralmente menos onerosa, é a utilização de um sistema distribuído para prover um meio de armazenamento computacional que armazene um grande volume de dados forenses. Este sistema pode possibilitar, através da utilização de computadores comuns ou modernos, a capacidade de armazenamento necessária a um grande volume de dados, provendo também recursos de tolerância a falhas, disponibilidade e escalabilidade (Coulouris, Dollimore, & Kindberg, 2007) e (HDFS, 2011). Esta é a

proposta feita no presente trabalho, utilizar um sistema de arquivos distribuído como mecanismo de armazenamento de um grande volume de dados.

## 2.5 TRABALHOS RELACIONADOS

Esta seção aborda alguns trabalhos relacionados com a pesquisa do presente trabalho, que é a utilização da computação distribuída para o armazenamento e indexação de dados forenses. É realizada uma descrição de cada trabalho e realizada uma análise correlacionando com o objeto de pesquisa desta dissertação.

### 2.5.1. Modelo de Indexação e Recuperação de Metadados Distribuídos

A pesquisa realizada por (Aires & Vaz, 2007) apresenta um sistema distribuído construído com o objetivo de melhorar o método de catalogação e recuperação de dados multimídia. Para isto é criada uma estrutura de metadados contendo detalhes das características dos objetos multimídia para que a busca indexada seja realizada nestes metadados.

A figura 2.6 ilustra a arquitetura do trabalho desenvolvido nesta pesquisa.

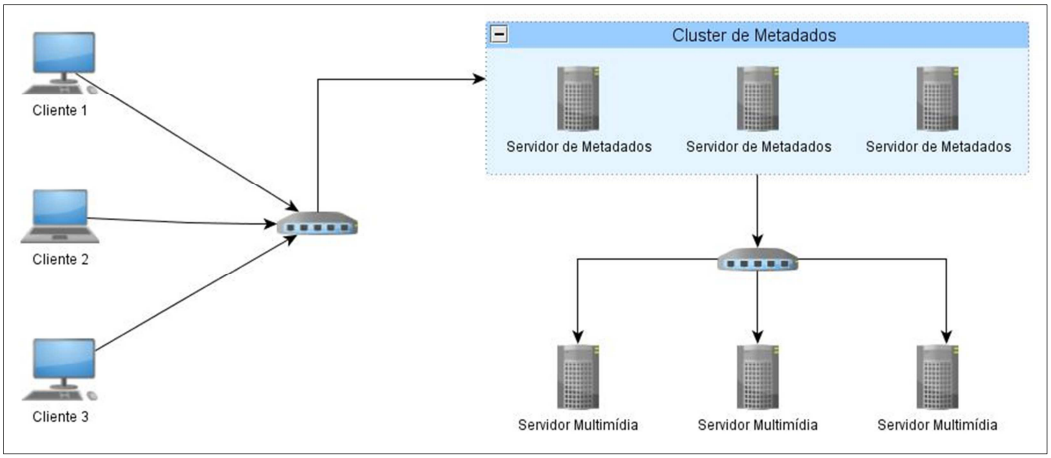



Figura 2.6 – Arquitetura do sistema distribuído (adaptado de Aires & Vaz, 2007)

Para obter o conteúdo multimídia os clientes acessam um cluster de servidores de metadados. A requisição será encaminhada para o servidor que estiver menos sobrecarregado no cluster. O servidor de metadados selecionado buscará a informação descrita na pesquisa em seus registros de metadados. Após localizar os metadados específicos ele localizará em que servidor multimídia o arquivo está armazenado e

encaminhará ao cliente. O cluster de metadados é estruturado utilizando banco de dados distribuídos. As informações dos metadados são replicadas em todos os servidores de metadados no cluster. Os arquivos multimídia não são replicados, sendo armazenados individualmente nos servidores multimídia (Aires & Vaz, 2007).

Este trabalho apresenta um método de realização de indexação distribuída através da indexação dos metadados de objetos multimídia no cluster de metadados. Ele implementa a tolerância a falhas através da replicação dos metadados entre os diversos servidores. Existe um mecanismo de balanceamento de carga no cluster de metadados onde as requisições de pesquisas e cadastros são encaminhadas aos servidores com uma quantidade menor de conexões. Esta solução também pode ser escalável através da adição de novos servidores ao ambiente (Aires & Vaz, 2007).

O foco deste trabalho está mais na distribuição de metadados do que em um mecanismo de indexação distribuída utilizando técnicas de IR. É utilizada a indexação de cada banco de dados particular para recuperar a informação dos metadados. Desta forma, o presente trabalho serve mais para a indexação distribuída de um conteúdo estruturado do que grandes volumes de dados não estruturados.

### 2.5.2. Um Sistema Distribuído de Indexação de Texto para o *Hadoop*

O artigo publicado por (Butler & Rutherford, 2008) mostra um projeto inicial de um sistema distribuído para realizar a indexação de informação textual, chamado *Distributed Lucene*. Este sistema foi construído utilizando-se o *Apache Hadoop,* o *Apache Lucene* e tendo recursos similares ao HDFS. O projeto do *Distributed Lucene* está baseado no funcionamento do HDFS mas não utiliza nenhum sistema de arquivos distribuído. Existem os *Datanodes* e o *Namenode*, sendo que o propósito dos *Datanodes* é armazenar pedaços de um índice criado e do *Namenode* de gerenciar a localização e a replicação dos índices criados. O objetivo desta arquitetura é possibilitar a criação paralela de índices com o *Apache Lucene* e também atualizações nos índices criados. A não utilização do HDFS foi devido ao mesmo não permitir múltiplos clientes atualizando os mesmos dados, a existência de muitos arquivos menores que o tamanho padrão do bloco de dados (64 MB) e possibilitar que os índices pudessem ser atualizados sem haver a necessidade de gerar novamente todo o índice, ou seja, a indexação incremental (Butler & Rutherford, 2008).

No momento da análise deste trabalho ele encontrava-se em uma versão inicial de desenvolvimento e sem nenhuma disponibilização de uma versão para testes. O artigo apresentado não abordou como é realizado o armazenamento dos dados que serão indexados e nem o desempenho desta arquitetura. A não utilização de um sistema de arquivos distribuído mostra que a localidade de dados não é utilizada no *Distributed Lucene*.

A prova de conceito abordou o mesmo problema de uma forma diferente. Foram criados *SequenceFiles* para agrupar diversos arquivos com tamanho menor que o tamanho do bloco HDFS no processo de cópia de dados. Isto permitiu uma melhoria no desempenho da indexação com maior possibilidade de existência da localidade de dados no momento do processamento. Também foi utilizado o HDFS com todos os benefícios de um sistema de arquivo distribuído para armazenar os dados forenses. Os índices foram armazenados como arquivos normais, podendo tanto ser armazenado no HDFS como em outro ambiente.

### 2.5.3. O Projeto *Katta*

O projeto *Katta* foi desenvolvido com o objetivo de prover um ambiente com escalabilidade, disponibilidade, balanceamento de carga e tolerância a falhas para realizar pesquisas indexadas a um grande número de usuários (Katta, 2011).

Para contemplar este objetivo, o projeto *Katta* utiliza o *Apache Hadoop* e o HDFS para os requisitos de um ambiente distribuído, o *Apache Lucene* para indexação, o *Apache ZooKeeper* para controle de redundância, além de outras ferramentas para operacionalização do ambiente. No momento da pesquisa realizada o *Katta* estava na versão 0.6.4, com o código-fonte e a documentação disponíveis em seu site original. Para melhorar o desempenho de buscas indexadas o *Katta* permite a gravação de grandes índices em seu sistema de arquivos distribuído, de forma que os índices são gravados em pequenos pedaços chamados *shards*. Os *shards* são distribuídos entre os diversos *datanodes* do HDFS, fornecendo tolerância a falhas e disponibilidade ao ambiente. Quando uma pesquisa é realizada a busca é feita apenas nos *datanodes* necessários. É implementado um mecanismo de cálculo de relevância distribuído para retornar a relevância de cada termo pesquisado de forma global no índice distribuído. Este método de

pesquisa permite que o *Katta* forneça um recurso de balanceamento de carga no momento da leitura de índices, provendo assim alta escalabilidade ao processo de pesquisa indexada (Katta, 2011).

No *Katta* toda a comunicação realizada com o *MasterNode* e os *DataNodes* são implementadas através do *Apache ZooKeeper*, com a finalidade de obter redundância no processo de comunicação. O *ZooKeeper* é um serviço centralizado para manter informações de configuração e de nomes em um ambiente distribuído através de suas funcionalidades de sincronização e grupos de serviços distribuídos. O *ZooKeeper* permite ler e escrever em um tipo de sistema de arquivos distribuído e virtual. A comunicação entre os processos clientes, o *JobTracker*, o *NameNode* e os *DataNodes* são implementadas através do *ZooKeeper*. Desta forma, o *Katta* possui um mecanismo de redundância no caso de falha no *JobTracker*, permitindo o redirecionamento das requisições para outra instância deste serviço que executa no ambiente (ZooKeeper, 2011) e (Katta, 2011).

Apesar do *Katta* possuir uma função básica de indexação distribuída, ele não tem o objetivo de ser um indexador distribuído e sim de ser uma ferramenta para melhorar o desempenho em buscas indexadas para um grande número de clientes. Por isto a funcionalidade de indexação distribuída do *Katta* não trata importantes requisitos de localidade de dados e de grande quantidade de arquivos menores que o tamanho do bloco HDFS. O presente trabalho busca tratar estes problemas para prover um ambiente de indexação distribuída para dados forenses.

### 2.5.4. O Projeto Apache Nutch

O projeto *Apache Nutch* é um subprojeto do *Apache Lucene* com o objetivo de realizar o processo de *crawlling* e de buscas indexadas com escalabilidade (Nutch, 2011).

O objetivo da ferramenta de *crawlling* é navegar pela web de forma metódica e automática realizando uma cópia das páginas e arquivos pesquisados. Esta cópia é indexada pela ferramenta de forma distribuída com o *Hadoop* e *Lucene*. Os índices criados são copiados para o ambiente distribuído do HDFS visando prover desempenho no processo de busca indexada (Nutch, 2011).

De forma geral o processo de indexação desta ferramenta funciona da seguinte forma (Nutch, 2011):

- uma tarefa *map* lê diversos tipos de dados, gerando como resultado um objeto com um formato padrão, chamado *NutchWritable*;

- uma tarefa *reduce* pega os objetos *NutchWritable* gerados, agrupados por URL, e gera documentos *Lucene* , chamados *LuceneDocumentWrapper*. Este documento, além do texto específico de cada página, contém metadados como relevância da página e URLs que apontam para a página;

- os objetos *LuceneDocumentWrapper* são indexados pelo *Lucene*;

- são criados vários índices *Lucene* parciais por cada tarefa *reduce*;

- o mecanismo de pesquisa indexada utiliza estes índices parciais para fazer a pesquisa,

O objetivo principal desta ferramenta é realizar o processo de *crawlling* e prover um ambiente com desempenho adequado para um grande volume de pesquisas indexadas. Apesar de não ter o propósito de ser um indexador distribuído, possui um método eficaz de realizar a indexação distribuída de arquivos. O trabalho objeto desta dissertação apresenta uma abordagem diferente para realizar a indexação distribuída, utilizando o agrupamento de arquivos em objetos *SequenceFiles* no processo de cópia e utilizando *pipeline* para a descompactação dos *SequenceFiles*, interpretação dos arquivos e indexação. Estas funcionalidades foram com o objetivo de prover maior localidade de dados e desempenho para o processo de indexação distribuída.

### 2.5.5. Um Laboratório de Análise Forense

Foi analisada uma ferramenta comercial que possui uma arquitetura de processamento distribuído específica para análises forenses em um grande volume de dados. A ferramenta *AccessData Lab – AD Lab* tem o objetivo de ser um laboratório que possibilita o processamento distribuído dos casos e um trabalho colaborativo na análise destes casos (AD Lab, 2011). A figura 2.7 contém uma ilustração com o básico da arquitetura desta solução (Ad Lab Architecture, 2011).

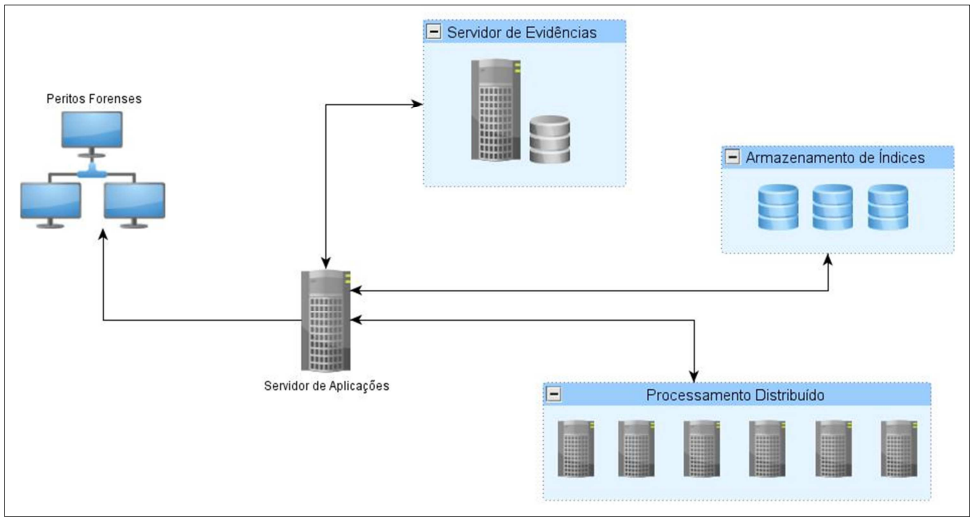



Figura 2.7 – Ilustra a arquitetura do AD Lab (adaptado de Ad Lab Architecture, 2011)

A arquitetura do *AD Lab* define um laboratório para processamento de evidências forenses com computação distribuída e um trabalho colaborativo para analisar as evidências. Os dados presentes nos dispositivos de armazenamento computacional que serão analisados são disponibilizados no servidor de evidências. Este servidor é responsável por armazenar todos os metadados extraídos das evidências disponibilizadas no ambiente em seu banco de dados, além de também armazenar os arquivos das evidências. O ambiente requer que esteja disponível um robusto servidor com boa capacidade de armazenamento e processamento. O processamento das evidências é realizado em um ambiente distribuído com unidades de processamento chamado *workers*. Este processamento corresponde na extração e análise automática de todos os artefatos encontrados na mídia analisada, como arquivos ativos, apagados e criptografados, fragmentos de arquivos, banco de dados e e-mails. Para realizar este processamento os *workers* necessitam indexar todos os dados encontrados nas mídias, utilizando a biblioteca *dtSearch* nesta indexação. Cada *worker* processa determinada parte de uma evidência para proporcionar desempenho e escalabilidade no tempo necessário para processar as evidências. O ambiente também requer um servidor de armazenamento com desempenho e capacidade adequados para o armazenamento dos índices. Estes índices serão utilizados tanto para determinados tipos de processamentos, como extração de arquivos ocultos no sistema, como no processo de análise pelos peritos forenses. O servidor de aplicações disponibiliza uma interface web que permite a visualização e análise dos artefatos encontrados nas evidências. Diversos peritos forenses acessam um servidor de aplicações com um banco de dados para poder definir critérios de acesso aos casos analisados e também para realizar a análise das

evidências. Desta forma, o ambiente do *AD Lab* disponibiliza uma arquitetura capaz de utilizar computação distribuída para o processamento de evidências e também distribuir as atividades operacionais de análises realizadas pelos peritos forenses (Ad Lab Architecture, 2011).

O *AD Lab* é uma ferramenta comercial que especifica uma arquitetura madura para o processamento de um grande volume de dados, também fornecendo os programas que implementam esta arquitetura. Contudo, no momento da pesquisa realizada não foi encontrada nenhuma publicação demonstrando o desempenho da ferramenta proposta em cenários reais. De acordo com a análise desta arquitetura verifica-se que ela não implementa o conceito de localidade de dados entre as unidades de armazenamento e processamento das evidências. Esta característica requer a existência de uma robusta infra-estrutura de rede, além dos robustos requisitos de equipamentos exigidos na arquitetura proposta. O objetivo do trabalho apresentado nesta dissertação é utilizar a computação distribuída para indexar dados forenses com a utilização de máquinas comuns e ferramentas com código-fonte aberto e gratuitas. Também será realizada uma análise de desempenho da prova de conceito proposta.

# 3. ANÁLISE DO ALGORITMO E FERRAMENTAS SELECIONADAS

Foi realizada uma pesquisa em ferramentas e algoritmos disponíveis para serem utilizados na implementação deste trabalho. Esta pesquisa visou à reutilização e aperfeiçoamento de métodos e técnicas já existentes, em busca de eficácia e eficiência.

Na análise de cada ferramenta buscou-se identificar aquelas que apresentavam as características necessárias para a combinação com outras ferramentas. Isto certamente facilitaria a composição de um ambiente único e integrado. Foram priorizadas as soluções com código-fonte aberto para possibilitar os aperfeiçoamentos e customizações necessários.

Neste capítulo é realizada uma análise comparativa do algoritmo e das ferramentas selecionadas com outras disponíveis no mercado com o intuito de mostrar quais características de cada recurso selecionado melhor se adequaram ao presente trabalho.

## 3.1. INDEXAÇÃO DE ARQUIVOS

Para realizar a indexação de arquivos foi selecionada a ferramenta *Apache Lucene*, versão 3.1.0 (Lucene, 2011). Esta ferramenta tem código-fonte aberto, utiliza modernas técnicas de recuperação da informação e possui uma comunidade de usuários ampla e ativa.

O *Lucene* é uma biblioteca de IR de alto desempenho e escalável (McCandless, Hatcher, & Gospodnetic, 2010). O *Lucene* possui recursos para realizar a indexação e busca indexada em documentos, no conteúdo e nos metadados de documentos. Esta biblioteca é um projeto de código-fonte aberto e implementado em Java, que já se encontra em um estado de maturidade que possibilita sua utilização em diversos lugares, como: no DVD da Enciclopédia Britânica, na revista *New Scientist*, nos sites *NetFlix*, *MySpace*, *LinkedIn* e MIT's *OpenCourceWare* (Lucene, 2011). O *Lucene* não é uma ferramenta de IR com recursos de interpretação de diferentes formatos de arquivos, com interface gráfica de pesquisa e visualização de arquivos. Esta biblioteca apenas implementa modernas técnicas de IR para realizar a indexação de conteúdos textuais de arquivos e possibilitar a pesquisa

no conteúdo indexado. Outras ferramentas podem ser utilizadas em conjunto com esta biblioteca para fornecer as funcionalidades de uma ferramenta de IR completa.

### 3.1.1. Importantes Características Técnicas do *Lucene*

A figura 3.1 contém as classes principais envolvidas do processo de indexação do *Lucene*. A análise destas classes será importante para esclarecer mais detalhes do método que esta biblioteca utiliza para indexar arquivos.

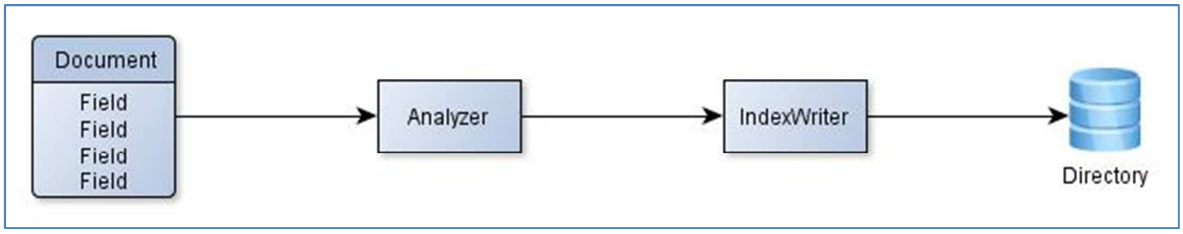



Figura 3.1 – Classes Utilizadas na Indexação de Dados do *Lucene*

A classe *Document* contém a estruturação inicial que o *Lucene* realiza nos dados não-estruturados que serão indexados. Para realizar esta estruturação é permitido que o documento seja composto de um conjunto de campos – *Fields*. Cada *Field* contém um nome e um valor, armazenando o conteúdo textual de determinado tipo de informação do documento, como: autor, título, conteúdo e URL. O tipo original do dado é irrelevante para o *Lucene*, ou seja, seja um arquivo do *Microsoft Word* ou campos de banco de dados, eles serão estruturados no formato de documentos. Cada aplicativo especifico que utiliza a biblioteca *Lucene* deve implementar seu formato de estruturação de documentos particular, utilizando a classe *Document* e os *Fields*. Antes que o texto do documento seja indexado é necessário que seu conteúdo seja analisado. A classe *Analyzer* realiza este processo de análise extraindo os termos relevantes e eliminando outros irrelevantes para indexação, como preposições e artigos. Existem classes *Analyzers* para diversos tipos de linguagens e necessidades de negócio, sendo que elas podem ser expandidas também. O mesmo *Analyzer* utilizado na indexação também deverá ser utilizado na pesquisa. A classe *IndexWriter* é o componente central do processo de indexação. Esta classe é responsável por toda a criação e manutenção da estrutura de dados de um índice. O *Lucene* utiliza o formato de índice invertido para implementar a estrutura de dados do índice. O formato detalhado do índice invertido criado pelo *Lucene* está contido no Apêndice A deste trabalho. A classe *Directory* representa o local onde o índice será armazenado. O índice do

*Lucene* pode ser armazenado em diversos meios de armazenamento computacional. Exemplos destes locais são: memória RAM, sistema de arquivos e banco de dados (McCandless, Hatcher, & Gospodnetic, 2010).

A figura 3.2 ilustra as classes do *Lucene* utilizadas para realizar a pesquisa em documentos através do índice previamente criado.

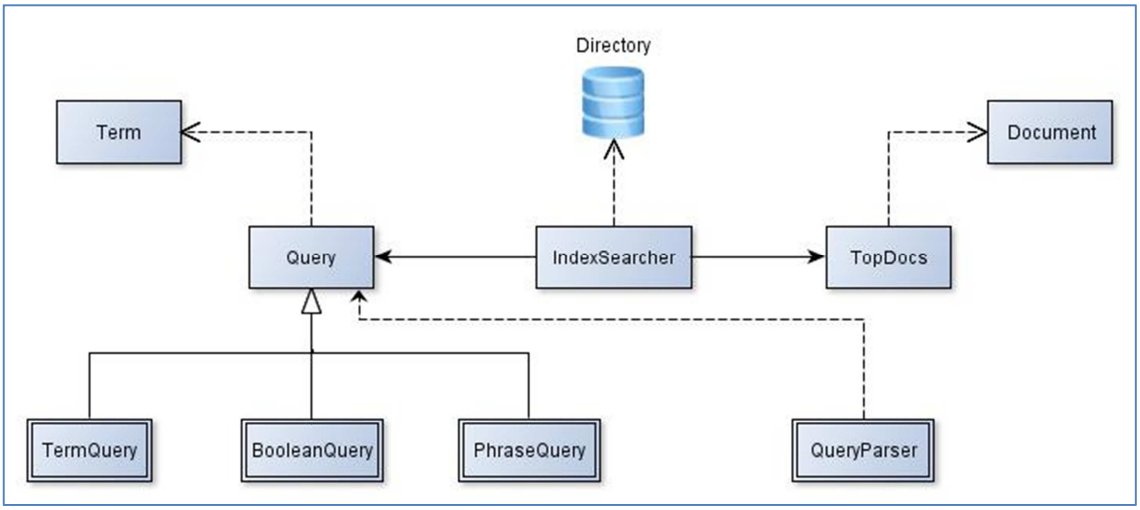



Figura 3.2 – Classes Utilizadas na Pesquisa Indexada do *Lucene*

Para realizar a pesquisa deve-se utilizar a classe *IndexSearcher*. Esta classe referencia a classe *Directory* que contém o método de extração do índice gravado em determinado meio de armazenamento. A classe *IndexSearcher* é o componente central do processo de pesquisa, contendo as principais funcionalidades de pesquisa no índice previamente criado. Para realizar a pesquisa é necessário o fornecimento de um objeto *Query* contendo os termos - *Terms* que estão sendo pesquisados. A classe *IndexSearch* retorna objetos da classe *TopDocs* contendo a relação dos documentos recuperados na pesquisa. A classe *TopDocs* contem poucos campos do documento recuperado, como o identificador do documento. Para recuperar um documento específico é necessário instanciar um objeto da classe *Document*, isto proporciona desempenho no processo de pesquisa. A classe *Query* que contém os termos que estão sendo pesquisados pode ser especializada de diversas maneiras. Alguns exemplos de especializações desta classe são as classes *TermQuery*, *BooleanQuery* e *PhraseQuery* que permitem a utilização de pesquisa em *Fields* específicos, uma combinação booleana dos termos pesquisados e uma automática recuperação dos termos relevantes em uma frase de pesquisa, respectivamente. Também existe a opção de utilização de uma classe especializada chamada de *QueryParser* que

transforma uma pesquisa textual livre em conjuntos de objetos *Query* especializados para a realização da busca indexada (McCandless, Hatcher, & Gospodnetic, 2010).

Tanto na fase de indexação como na de pesquisa o *Lucene* possui diversos recursos que implementam modernas técnicas de IR (McCandless, Hatcher, & Gospodnetic, Lucene in Action, 2010). Alguns exemplos destes recursos são:

- Mecanismo de pontuação de relevância dos termos encontrados em um documento;

- Possibilidade de gravação de vetores com metadados de cada termo;

- Recursos de indexação de dados numéricos e data e hora;

- Recursos de *merging* de índices;

- Possibilidade de correções gramaticais e busca fonética na pesquisa e indexação;

- Recursos de cálculo de relevância na recuperação de documentos;

- Mecanismos para transformar uma consulta feita textualmente em estruturas de dados de uma busca indexada;

- Possibilidade de pesquisas e filtros em múltiplos campos;

- Recursos para pesquisas em múltiplos índices.

Além disto, existem diversas ferramentas de terceiros que estendem funcionalidades e facilitam o gerenciamento dos índices. Alguns exemplos destas ferramentas são:

- Biblioteca para seleção do texto pesquisado no resultado da busca (Lucene Contrib, 2011);

- Biblioteca para análise da estrutura do índice criado (Luke, 2011);

- Biblioteca para extração de informação textual em diversos tipos de arquivos (Tika, 2011).

### 3.1.2. Outras Ferramentas de Indexação

As ferramentas de IR de uso pessoal foram descartadas por não atenderem ao propósito do presente trabalho, ou seja, de poder ser reutilizada e adaptada com as necessidades de indexação de dados forenses. Algumas ferramentas de IR de uso pessoal analisadas foram

o *rapid-i*, *copernic* e *auto-focus*. Foi verificado que a ferramenta *rapid-i* está fora do escopo do presente trabalho por ser mais focada em técnicas de *data mining* do que de recuperação da informação (rapid-i, 2011). O escopo do *Copernic* é indexar e visualizar arquivos ativos em um computador. Apesar de poder indexar e visualizar alguns tipos de arquivos não possui escalabilidade para um grande volume de dados (Copernic, 2011). O *AutoFocus* é um indexador e visualizador de arquivos que utiliza mapas *clusterizados* para visualizar os resultados. Os mapas *clusterizados* permitem que os dados indexados sejam agrupados semanticamente de forma visual, facilitando a forma de visualização e pesquisa. Esta ferramenta não se aplicou ao escopo do presente trabalho por estar mais focado na visualização do conteúdo do que no processo de indexação em si, além de não permitir o acréscimo de funcionalidades (Aduna Software, 2011).

As bibliotecas de indexação comerciais foram descartadas devido ao objetivo do presente trabalho buscar utilizar programas de código-fonte aberto para possibilitar a adaptações com a integração com outras ferramentas. Os softwares de análises forenses mais utilizados no Brasil e Estados Unidos são o FTK e o EnCase (DPF, 2010) e (FBI, 2011). A biblioteca de indexação utilizada nestes programas é o *dtSearch* (dtSearch, 2011), (FTK, 2011) e (EnCase, 2011).

A seleção da biblioteca de indexação utilizada neste trabalho recebeu forte influência dos estudos realizados por (Middleton & Baeza-Yates, 2007). Neste trabalho ele realiza uma análise comparativa de diversas bibliotecas de indexação de código-fonte abertas, entre elas: *MG4J, Lucene, Terrier, Swish-E, Swish++, ht://Dig, XMLSearch e Zettair*. As três primeiras são implementadas na linguagem Java e as demais em C++. Alguns critérios para as comparações realizadas foram o formato de armazenamento de índices, os recursos e desempenho na indexação e pesquisa, além da comunidade de usuários e desenvolvedores da ferramenta. A ferramenta *Apache Lucene* apresentou um resultado adequado nos testes e por possuir uma integração adequada com a ferramenta de processamento distribuído que foi selecionada se mostrou a mais apta para ser a utilizada neste trabalho.

## 3.2. INTERPRETAÇÃO DE ARQUIVOS

Para realizar a indexação de documentos através do *Apache Lucene* é necessário que seja extraído o conteúdo textual de cada tipo de arquivo analisado. Arquivos comumente encontrados em um conjunto de dados forenses são: arquivos do *Word*, *Excel*, PDF, *OpenOffice*, RTF, *Outlook*, além de outros formatos binários como TAR e ZIP (DPF, 2010). Para extrair o conteúdo textual de cada diferente tipo de arquivo é necessário um algoritmo que possa realizar a interpretação da estrutura destes arquivos.

Para realizar a interpretação dos diferentes tipos de arquivos presentes nos dados forenses coletados foi utilizada a ferramenta *Apache Tika*, versão 0.9 (Tika, 2011). 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* nasceu do projeto *Apache Lucene* e foi estruturado de forma a ser uma biblioteca expansível com recursos para a conexão de componentes que interpretam tipos particulares de arquivos (Tika, 2011). O *Tika* utiliza classes padrões para extrair textos e metadados de documentos de forma a abstrair os detalhes internos de cada tipo de arquivo que é tratado pelos componentes de interpretação. O *Tika* é apenas a estrutura que permite a interconexão de diversos componentes de interpretação de arquivos. Os componentes é que fazem a lógica de interpretação de cada arquivo particular. O *Tika* permite que novos componentes possam ser incorporados em sua estrutura sempre que necessário. Exemplos de tipos de arquivos interpretados pelo *Tika* são: HTML, XML, *Microsoft Office*, *OpenOffice*, PDF, EPUB, RTF, ZIP, TAR, BZIP, TXT, WAV, MP3, MP4, Imagens e MBox (Mattmann & Zitting, 2010).

Como o *Tika* nasceu do projeto *Lucene* ele possui classes específicas que se integram com o indexador do *Lucene*. Esta integração pode inclusive ser realizada através de *pipeline* para possibilitar o ganho de desempenho (McCandless, Hatcher, & Gospodnetic, 2010). Neste método, à medida que o arquivo vai sendo interpretado o texto extraído já vai sendo indexado. Isto possibilita que arquivos com grande conteúdo textual sejam indexados com um menor consumo de memória, tendo em vista que não será necessário armazenar todo o conteúdo extraído na memória antes da indexação.

Outras ferramentas de interpretação de arquivos foram analisadas. Foram verificadas a existência de robustas ferramentas de interpretação de arquivos como *Stellent's* e a *OutSide-In* (Huff, 2006) e (OutSide-In, 2011). Mas foram descartadas por serem ferramentas comerciais que dificultaria o processo de adaptação de código e integração com outras ferramentas selecionadas. Uma ferramenta analisada foi a *Aperture* (Aperture, 2011). Além de extrair textos e metadados de documentos esta ferramenta possui recursos de *crawling*, ou seja, pode-se conectar a sistemas de arquivos, servidores web e servidores de e-mail para obter informações automaticamente destas fontes de dados. Contudo por apresentar uma arquitetura flexível e expansível, além da integração com o *Lucene*, a biblioteca *Tika* se apresentou a mais vantajosa para o presente trabalho.

## 3.3. ALGORITMO DE COMPUTAÇÃO DISTRIBUÍDA

O algoritmo selecionado para realizar a computação distribuída foi o *MapReduce* (Dean & Ghemawat, 2004).

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* (Smith, 1969).

Essencialmente este modelo de programação permite que usuários escrevam componentes *map*/*reduce* em um código. 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 (Buyya, Broberg, & Goscinski, 2011).

### 3.3.1. Importantes Características Técnicas do *MapReduce*

A função *Map* é uma das principais funções do algoritmo *MapReduce*, tendo o objetivo de distribuir o processamento de determinado problema computacional. A função *Map* pega

um par chave/valor como entrada e produz uma lista com funções chave/valor como saída, conforma a função 3.1. 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.

$$map: \quad (chave_1, valor_1) \rightarrow lista(chave_2, valor_2) \tag{3.1}$$

A função *Reduce* é uma função importante do algoritmo *MapReduce*, apesar de ser facultativa. Seu objetivo é refinar o processamento distribuído de tarefas *Maps* anteriores que foram agrupadas por determinada chave. 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, conforme a função 3.2.

$$reduce: \quad (chave_2, lista(chave_2)) \rightarrow lista(valor_3) \tag{3.2}$$

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 (Dean & Ghemawat, 2004).

O ambiente de execução *MapReduce* realiza diversas atividades para que o programador deste método se foque mais na resolução do problema computacional do que no gerenciamento do ambiente distribuído. Exemplos destas atividades são: 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 figura 3.3 ilustra o funcionamento geral do *MapReduce*.

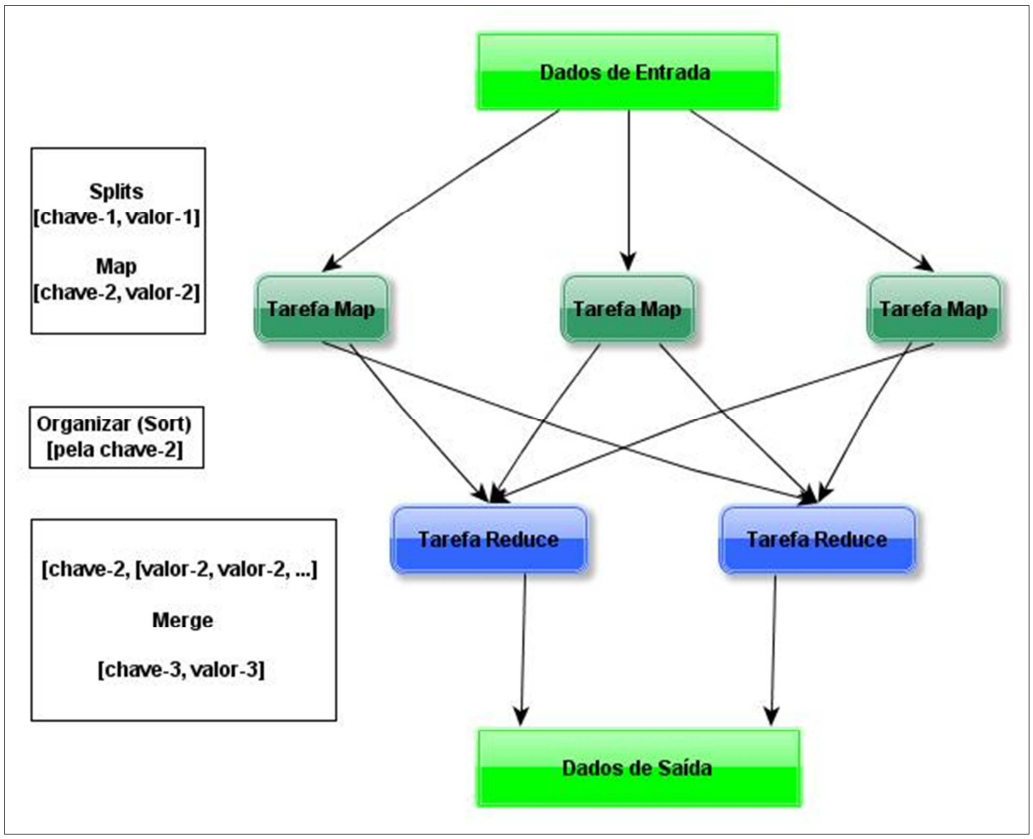



Figura 3.3 - Gráfico esquemático da funcionalidade do *MapReduce*

Neste gráfico 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. Um exemplo de divisão de dados de entrada que pode ser realizado é onde a chave é o *offset* de um arquivo e o valor é o conteúdo de uma linha deste arquivo. 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 fórmula 3.2. A fórmula padrão de particionamento está contida na equação 3.3:

$$particionamento: \quad hash(chave_2) \ mod \ R \tag{3.3}$$

Na fórmula 3.3, *R* é a quantidade de tarefas *Reduce* presentes no processamento e *hash* é uma função *hash SHA-1* ou *MD-5* (Stallings, 2008). Isto tende a resultar em partições bem balanceadas para distribuir para as tarefas *Reduce.* Esta fórmula também pode ser customizada 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 figura 3.1 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 figura 3.4, 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.

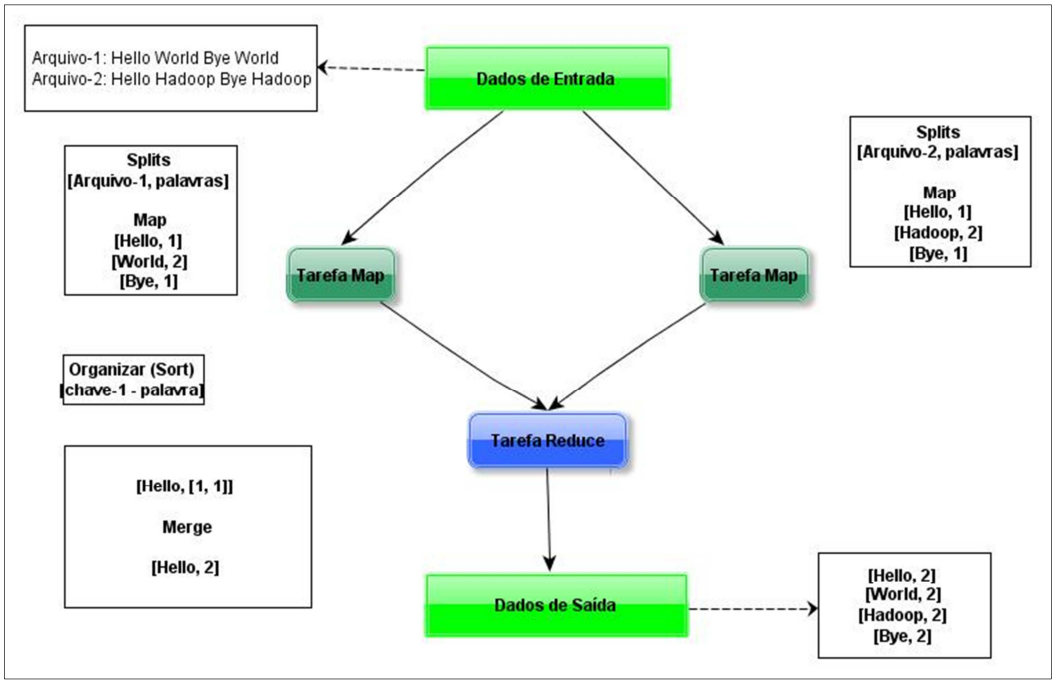


Figura 3.4 - Exemplo de funcionamento do *MapReduce*

No exemplo da figura 3.4 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 uma 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 figura 3.4 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*.

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 (Bernam, Fox, & Hey, 2005). 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* (Google, 2011), utiliza o sistema de arquivos distribuído *Google File System* – GFS (Ghemawat, Gobioff, & Leun, 2003). O GFS divide cada arquivo em blocos de 64 MB e armazena diversas cópias (três 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.

### 3.3.2. Outros Algoritmos de Distribuição do Processamento

Existem alguns sistemas que fornecem um modelo de programação que possibilita a criação de algoritmos de processamento paralelo (Blelloch, 1989), (Gorlatch, 1996), (Ladner & Fischer, 1980). O *MapReduce* pode ser considerado como uma simplificação destes modelos existentes, provendo mecanismos de tolerância a falhas e balanceamento de carga que escalam bem para milhares de processos (Dean & Ghemawat, 2004). Ao contrário dos outros algoritmos estudados, no *MapReduce* o foco do usuário é principalmente na resolução do problema computacional, não necessitando realizar controles típicos de ambientes distribuídos.

O modelo BSP (*Bulk Synchronous Programming*) (Valiant, 1997) e algumas primitivas de MPI (*Message-Passing Interface*) (Gropp, Lusk, & Skjellum, 1999) provêem abstrações de alto-nível que facilita a criação de programas paralelos. A diferença chave entre estes sistemas e o *MapReduce* é que neste é utilizado um modelo de programação específico para a criação de programas paralelos e o mecanismo de tolerância a falhas é transparente. Nos dois modelos anteriores são utilizados algoritmos com o objetivo de serem portáveis para diferentes sistemas distribuídos e diferentes problemas computacionais. Contudo, nestes algoritmos devem ser fornecidos parâmetros específicos da arquitetura do ambiente, como: número de processadores, tamanho da memória e custo de comunicação em rede. Além disto, não possuem recursos transparentes de tolerância a falhas.

O mecanismo de localidade de dados implementado no *MapReduce* é baseado em técnicas como a dos discos ativos (Huston, Sukthankar, & Wickremesinghe, 2004), (Riedel, Faloutsos, & Gibson, 2001), onde a computação é colocada em elementos de processamento que estão próximos dos discos locais para reduzir a quantidade de dados transmitidos em sistemas de E/S ou pela rede.

O sistema de gerenciamento de *cluster* do *MapReduce* é baseado no espírito do sistema *Condor* (Thain, Tannenbaum, & Livny, 2004), onde é realizada a distribuição do processamento e monitorado o ambiente de execução de tarefas de usuários em um grande volume de dados.

O sistema *River* (Arpaci-Dusseau, Anderson, & Treuh, 1999) provê um modelo de programação onde os processos se comunicam enviando dados através de filas distribuídas. Assim como no *MapReduce*, o ambiente *River* tenta prover uma boa performance mesmo na presença de máquinas com diferentes configurações ou em falhas de sistemas. Para implementar isto o *River* cuidadosamente aloca as transferências de disco e de rede para balancear a carga. A abordagem do *MapReduce* é diferente. Através da restrição do modelo de programação o ambiente *MapReduce* é capaz de dividir um problema em um grande número de tarefas. Estas tarefas são dinamicamente alocadas em máquinas disponíveis de uma forma que as mais rápidas podem processar mais tarefas.

## 3.4. CONTROLE DE PROCESSAMENTO DISTRIBUÍDO

Para realizar a atividade de computação distribuída foi selecionada a ferramenta *Apache Hadoop*, versão 0.20.203 (Hadoop, 2011). Esta ferramenta tem código-fonte aberto, utiliza modernas técnicas de computação distribuída, implementando o algoritmo *MapReduce*, e possui uma comunidade de usuários ampla e ativa.

O *Apache Hadoop* é um projeto originário do *Apache Nutch* (Nutch, 2011). Após a publicação dos artigos técnicos sobre o *GFS* (Ghemawat, Gobioff, & Leun, 2003) e sobre o *MapReduce* (Dean & Ghemawat, 2004) , o *Nutch* recebeu uma grande reformulação dando origem ao projeto que se chamou *Hadoop*.

Para realizar o gerenciamento do ambiente distribuído e fornecer a estrutura lógica para a criação de tarefas paralelas o *Hadoop* implementa o algoritmo *MapReduce* com as características detalhadas na seção 3.3. 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 seguir será descrito como o *Hadoop* implementa o *MapReduce* e suas peculiaridades.

### 3.4.1. Importantes Características Técnicas do *Hadoop*

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++. A título de exemplo 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;

- Existência de um Nó Mestre, chamado de *JobTracker*. Ele 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 customização 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 (White, 2009).

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 se chama *JobTracker*. Este processo implementa as atribuições do Nó Mestre do algoritmo *MapReduce* e, quando ativo, fica presente em apenas um nó do ambiente. O outro tipo de 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 figura 3.5, mostra de forma mais detalhada como o *Hadoop* implementa o processamento distribuído do *MapReduce*.

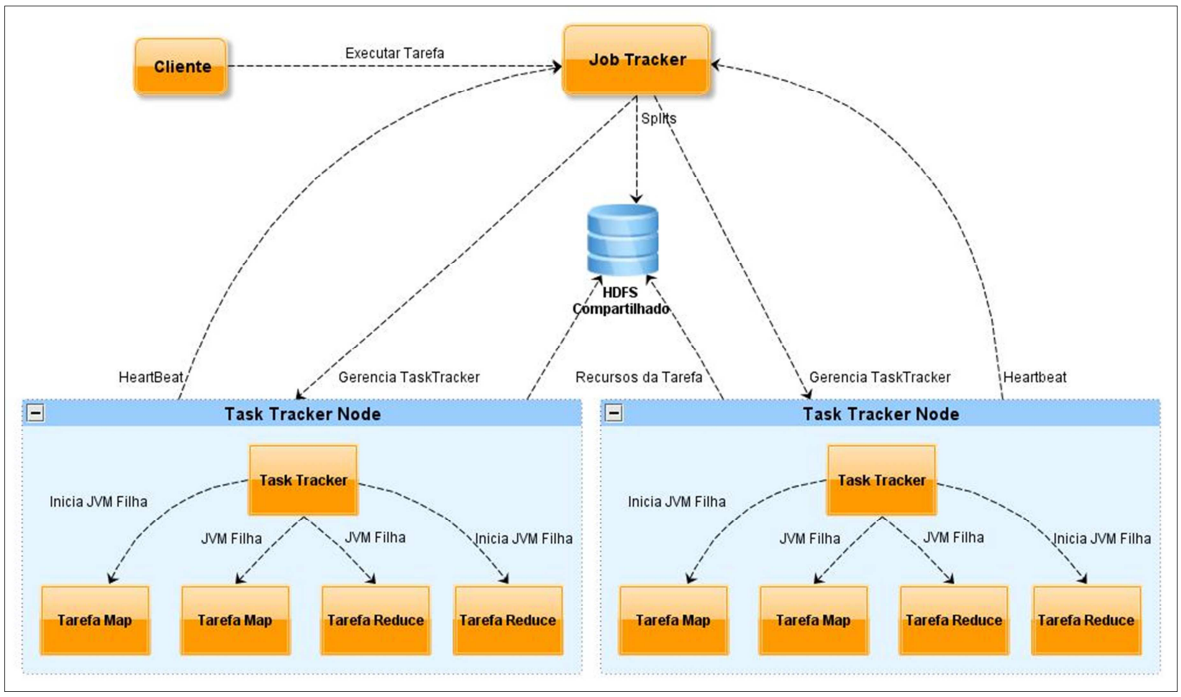



Figura 3.5 - Implementação do *MapReduce* pelo *Hadoop* (adaptada de White, 2009)

Conforme ilustrado na figura 3.5, existem os seguintes passos:

1. Primeiramente um processo cliente solicita a execução de um determinado processamento distribuído para o processo Nó Mestre, que se chama *JobTracker*;
2. Neste momento o processo cliente fornece a localização das classes que implementarão as atividades de processamento distribuído (tarefas *Map* e tarefas *Reduce*);

3. O processo *JobTracker* mantém o controle de todos os nós do ambiente distribuído e realiza a inicialização de cada nó para realizar o processamento:
    a. 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;
    b. Para o processamento das tarefas, o processo *TaskTracker* inicializa e gerencia processos filhos, que executarão as tarefas *Map* e *Reduce* que foram implementadas.
4. 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*:
    a. 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.
5. 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.

### 3.4.2. Outras Ferramentas de Computação Distribuída

O *Apache Hadoop* foi a ferramenta de computação distribuída selecionada por implementar o algoritmo *MapReduce*, por possuir o código-fonte aberto e por possuir uma comunidade de usuários ampla e ativa (Hadoop, 2011). Existem outras ferramentas que implementam o algoritmo *MapReduce*, como a *Phoenix* e a *Greenplum* (Phoenix, 2011) e (Greenplum, 2011). A *Greenplum* é um SGBDR comercial que incluiu recursos de execução do algoritmo *MapReduce*. Por ter outros recursos além do ambiente de execução do *MapReduce* a *Greenplum* consome mais recursos computacionais que o *Hadoop*, por não ser uma ferramenta apenas de implementação do *MapReduce* A *Phoenix* apesar de ser

código aberto no momento da realização do presente trabalho não possuía uma ampla adoção e uma comunidade de usuário ativa.

A opção de utilizar o *Hadoop* ao invés de um sistema gerenciador de banco de dados relacional – SGBDR é devido a alguns fatores. Os SGBDRs foram projetados para armazenar e processar dados estruturados e normalizados enquanto o *Hadoop* foi projetado para processar dados semi-estruturados e não estruturados. Um exemplo de dados semi-estruturados é uma planilha onde as células podem possuir dados de qualquer formato. Exemplos de dados não estruturados são informações textuais puras e imagens. Desta forma, para analisar um grande volume de dados não estruturados o *Hadoop* escala melhor do que os SGBDRs (White, 2009).

Algumas ferramentas de programação paralela, como as baseadas em MPI – *Message Passing Interface*, têm a característica geral de dividir o processamento através de um cluster de computadores com acesso a um sistema de arquivos compartilhado. Em atividades com predominância de processamento elas escalam bem, mas com um grande volume de dados elas tendem a obter um baixo desempenho devido ao consumo da largura de banda da rede. Além disto, estas ferramentas permitem customizações de acordo com o ambiente distribuído onde são executadas, o que torna seu uso mais dependente da arquitetura do ambiente distribuído. A localidade de dados, o mecanismo de balanceamento de carga e o mecanismo de tolerância a falhas, implementados pelo *Hadoop*, proporcionam boa escalabilidade em um grande volume de dados, além de evitar que os desenvolvedores se foquem em detalhes de gerenciamento do ambiente distribuído (White, 2009).

Existem sistemas distribuídos que são implementados através da computação voluntária. A computação voluntária se baseia na utilização da capacidade ociosa de processamento de computadores conectados na Internet, os voluntários, para analisar dados científicos (Nov, Anderson, & Arazy, 2011). Um exemplo de sistema baseado em computação voluntária é o BOINC (BOINC, 2011). O BOINC é uma plataforma, na forma de framework, que visa a facilitar a implementação de sistemas de computação voluntária, funcionando através de uma grade computacional de dimensões mundiais, através de computação distribuída (BOINC, 2011). A utilização da computação voluntária no presente trabalho foi descartada por motivos de desempenho e segurança. A computação voluntária utiliza recursos de computadores anônimos que estão distribuídos em uma rede de computadores, como a

Internet. Este fato prejudica a utilização destes computadores para o armazenamento dos dados forenses que possuem informações sensíveis. A computação voluntária é implementada através da tecnologia atual da Internet, e não possui bom desempenho para processar um grande volume de dados. O foco da computação voluntária é em problemas baseados em processamentos mais matemáticos do que de tratamento de dados.

Existem sistemas distribuídos de alto desempenho e configuráveis que estão disponíveis para serem alugados. O *Amazon Elastic Comput Cloud* – EC2 é um exemplo deste ambiente (Amazon EC2, 2011). O EC2 é um serviço da web que fornece capacidade computacional escalável na nuvem. É projetado para tornar a escalabilidade computacional de nível de web mais fácil para desenvolvedores. O próprio *Hadoop* pode ser instalado neste ambiente e as tarefas de processamento submetidas a ele (Amazon EC2, 2011). A utilização deste tipo de ferramenta foi descartada por motivos de manter uma segurança adequada ao armazenamento dos dados forenses e por ser comercial.

## 3.5. SISTEMA DE ARQUIVOS DISTRIBUÍDO

O sistema de arquivos distribuído selecionado foi o *Hadoop Distributed File System* – HDFS, versão 0.20.203 (HDFS, 2011). 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 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. (White, 2009).

### 3.5.1. Importantes Características Técnicas do HDFS

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 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 executado na máquina que já possui o bloco, diminuindo consideravelmente o volume de dados que trafega na rede e implementando a localidade de dados. 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 funcionalidades o HDFS possui dois tipos de processos principais, um se chama *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 executa o processo *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 podem implementar 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.*

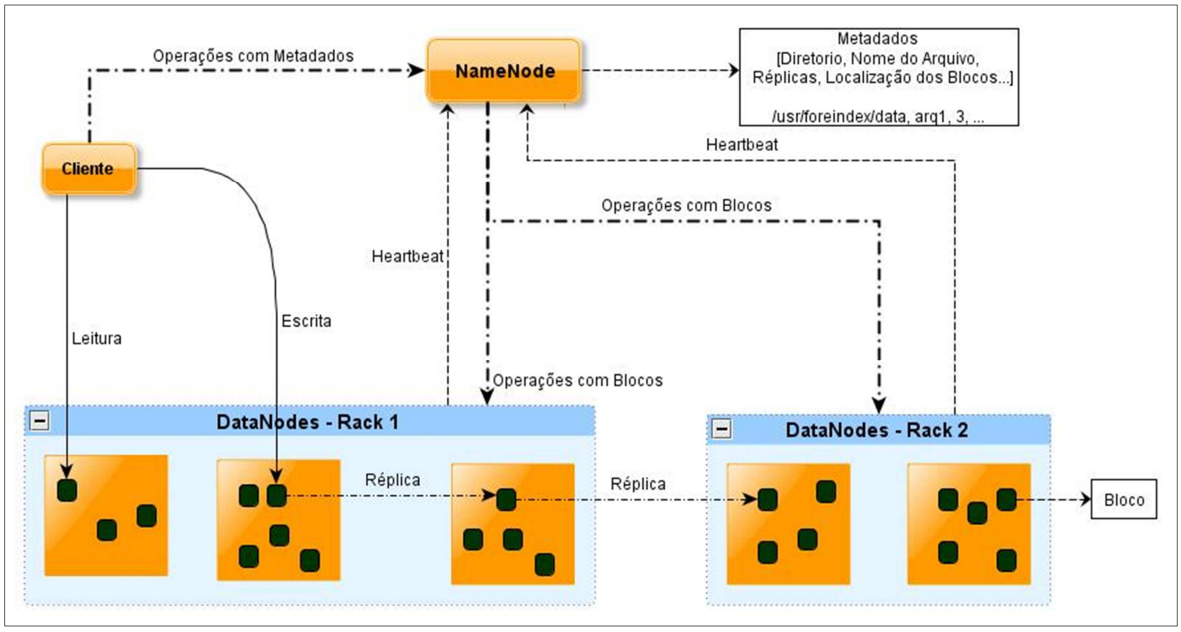



Figura 3.6 – Funcionamento Geral do HDFS (adaptada de White, 2009)

A figura 3.6 ilustra o funcionamento geral do HDFS. Neste diagrama é ilustrado que:

1. O cliente solicita a realização de operações de metadados como criação, exclusão e leitura de arquivos ao *NameNode*;
    a. 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.
2. 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;
3. Os *DataNodes* periodicamente encaminham ao *NameNode* mensagens indicando o status de armazenamento e a quantidade de blocos de dados que eles armazenam.
    a. Cada mensagem desta recebe o nome de *Heartbeat*.
4. O *NameNode* encaminha a localização dos *DataNodes* que possuem os blocos de dados que o cliente solicitou;
5. Os clientes se conectam diretamente com os *DataNodes* para realizar o processo de leitura e de escrita de dados.

a. 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*.

Além os recursos já citados o HDFS possui mecanismos de balanceamento do cluster, quando novas máquinas são excluídas ou acrescentadas este mecanismo pode ser acionado para que o ambiente do *NameNode* gerencie a tarefa de distribuição dos blocos de dados no cluster para manter o balanceamento de utilização do espaço de armazenamento. Este processo fortalece a escalabilidade, integridade e disponibilidade do sistema de arquivos distribuído.

### 3.5.2. Outros Sistemas de Arquivos Distribuídos

Atualmente existem dois grandes grupos de sistemas de arquivos distribuídos para processamento intensivo de dados, que são: os sistemas de arquivos para serviços de Internet e os sistemas de arquivos para computação do alto desempenho, cuja sigla é: HPC - *High Performance Computing* (Tantisiriroj, Patil, & Gibson, 2008). Exemplos de sistemas de arquivos para serviços de Internet são: GFS, HDFS e o S3 (Ghemawat, Gobioff, & Leun, 2003), (White, 2009) e (Amazon S3, 2011). Exemplos de sistema de arquivos distribuídos para sistemas HPC são: GPFS, LustreFS e o PVFS (IBM GPFS, 2011), (Sun LustreFS, 2011) e (PVFS, 2011).

Os sistemas de arquivos distribuídos para HPC necessitam de uma infra-estrutura especializada, suportam diferentes modos de acesso a arquivos e desta forma não são tão adequados para serem utilizados para suportar a demanda requerida em serviços para Internet. Os sistemas de arquivos GPFS e LustreFS assumem uma baixa latência de rede e a existência de um sistema distribuído com compartilhamento de memória. Estes requisitos inviabilizaram que estes sistemas pudessem ser utilizados no presente trabalho (Tantisiriroj, Patil, & Gibson, 2008).

O sistema de arquivos distribuído S3 é utilizado na *Amazon*, sendo o sistema de arquivos padrão do ambiente EC2. Contudo, tanto neste sistema como no PVFS, os nós que armazenam os dados não são os mesmos que realizam a computação. Apesar deste tipo de

arquitetura facilitar a manutenção do ambiente, ele inviabiliza a utilização da localidade de dados, exigindo uma rede de baixa latência e alto desempenho para manter uma perfomance adequada (Tantisiriroj, Patil, & Gibson, 2008).

O sistema HDFS foi baseado no GFS, o HDFS possui poucas diferenças do GFS, que são mais relacionadas à semântica de acesso a arquivos. Enquanto o GFS implementa recursos de acesso concorrente a arquivos o HDFS não implementa, baseado em sua arquitetura de uma única escrita e diversas leituras. Estas diferenças não são significativas para o presente trabalho, sendo que a preferência pelo HDFS ao invés do GFS foi devido ao HDFS ser uma ferramenta de código-fonte aberto e mais integrada com o ambiente *Hadoop* (Tantisiriroj, Patil, & Gibson, 2008).

Mesmo com a escolha do sistema de arquivos HDFS, qualquer outro sistema de arquivos pode ser utilizado no presente trabalho que a indexação distribuída funcionará. É recomendável a utilização de um sistema distribuído que possibilite a execução da localidade de dados para a obtenção de melhores taxas de desempenho.

## 3.5. MONITORAMENTO DO AMBIENTE DISTRIBUÍDO

Para coletar as métricas dos recursos computacionais utilizados nos cenários de testes elaborados foi selecionada a ferramenta *Ganglia Monitoring System*, versão 3.1.7, com a interface web versão 2.1.8 (Ganglia, 2011). Esta ferramenta tem código-fonte aberto, possui um flexível mecanismo de coleta de métricas em um ambiente distribuído e possui uma comunidade de usuários ampla e ativa.

O *Ganglia* é um sistema distribuído de monitoramento de sistemas para computação de alto desempenho, como ambientes em clusters ou grades. Ele é baseado em projeto hierárquico de clusters. Para isto ele faz uso tecnologias amplamente utilizadas como XML para representação de dados, XDR para um meio de transporte portável e compacto e *RRDTool* como meio de armazenamento e visualização de dados. Os algoritmos e estruturas de dados utilizados pelo *Ganglia* foram projetados para obter uma baixa sobrecarga na rede e alta concorrência. O *Ganglia* está em uso atualmente em mais de 500 clusters, podendo obter boa escalabilidade em clusters com mais de 2.000 nós (Ganglia, 2011).

A figura 3.7 ilustra os principais componentes da arquitetura do *Ganglia*.

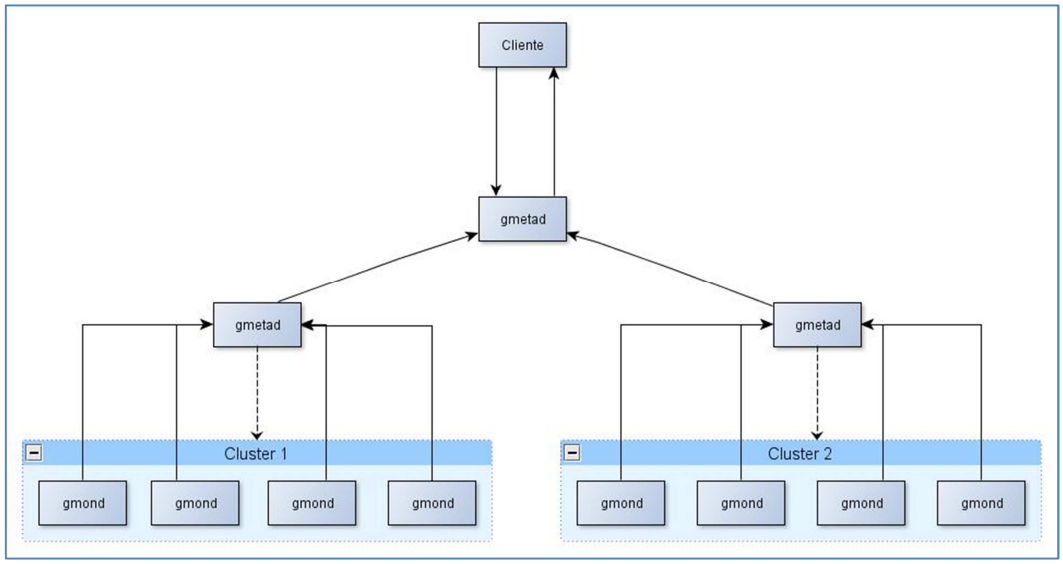



Figura 3.7 – Componentes da Arquitetura do *Ganglia* (adaptada de Ganglia, 2011)

Existem dois componentes principais no *Ganglia*, o *Ganglia Monitor Daemon – gmond* e o *Ganglia Meta Daemon – gmetad*. O *gmond* é um serviço leve que deve executar em todas as máquinas do cluster que necessita que suas métricas sejam coletadas. Ele coleta diversas métricas como a utilização do processador, memória, disco e rede. Periodicamente ele envia, por *multicast*, estas métricas para um processo chamado *gmetad*. O *gmetad* é um serviço leve que coleta dados dos processos *gmond* e de outros processos *gmetad*. Os dados coletados são armazenados em seu banco de dados indexado através do *Round Robin Database – RRDTool*. O *gmetad* também provê um mecanismo simples de consulta que permite a coleta de métricas específicas em cada máquina ou cluster. O *gmetad* possui um projeto hierárquico que permite a estruturação de clusters de forma a diminuir a sobrecarga de dados em ambientes com muitos nós. O processo *gmetad* também realiza o gerenciamento de quais máquinas estão ativas através do monitoramento das mensagens dos *gmonds* e outros *gmetads*. Um processo cliente se comunica com um processo *gmetad* principal para poder obter dados de todos os clusters gerenciados, podendo também obter dados de um cluster ou uma máquina específica no ambiente. O processo *gmetad* retorna os dados solicitados pelo cliente (Ganglia, 2011).

O ambiente do *Ganglia* também provê uma interface web que permite um monitoramento e gerenciamento remoto das métricas coletadas no sistema distribuído que ele monitora.

A figura 3.8 contém uma tela ilustrando esta interface de monitoramento do *Ganglia*.

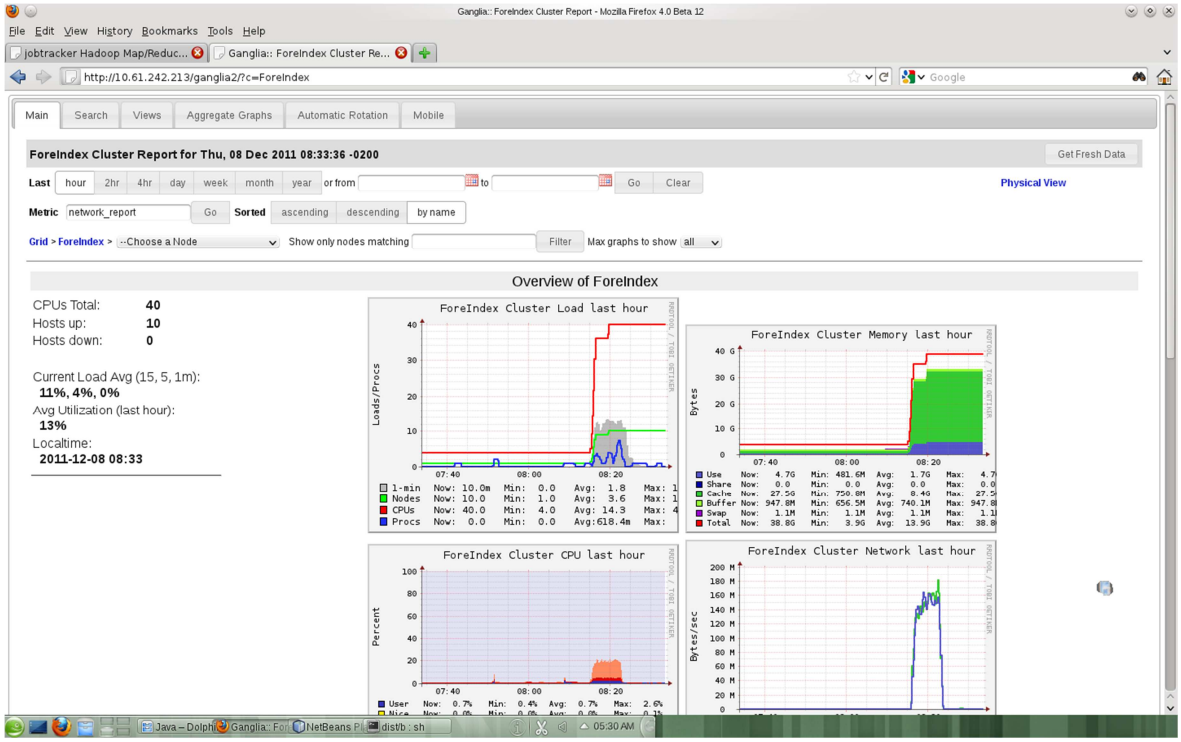



Figura 3.8 – Interface Web do *Ganglia*

Nesta interface é possível realizar funcionalidades como: a selecionar qual o cluster que será monitorado, em qual período e quais métricas tanto de máquinas individuais quanto agrupadas por cluster. No exemplo da figura 3.8 observa-se um cluster chamado *ForeIndex*, contendo 10 computadores e 40 processadores, mostrando que existem 4 processadores por máquina. Existem quatro gráficos mostrando métricas de tipos de processos iniciados, consumo de memória, utilização do processador e utilização da rede em todo o cluster. As métricas coletadas ilustram o momento em que um ambiente *Apache Hadoop* com HDFS foi inicializado. Observa-se assim que antes da inicializado do *Hadoop* e HDFS a sobrecarga no ambiente com o envio e recebimento de métricas foi reduzido.

Uma análise de desempenho do *Ganglia* em sistemas distribuídos foi realizada por (Massie, Chun, & Culler, 2004). Foram analisados quatro sistemas distribuídos compostos de centenas de máquinas, sendo que estes sistemas são amplamente utilizados em ambientes corporativos e científicos. Ficou demonstrado que a escalabilidade do *Ganglia* pode ser mensurada tanto pela quantidade de máquinas em um ambiente distribuído, como pela quantidade de clusters existentes. Ficou demonstrado que ele escala bem em um

ambiente com 2.000 máquinas ou com 42 clusters organizados de forma hierárquica, sendo que cada cluster pode possuir 2.000 máquinas que os impactos não serão relevantes.

Existem diversos sistemas acadêmicos e comerciais de monitoramento de um ambiente distribuído, mas nem todos têm o foco na escalabilidade de desempenho. O *Supermom* é um sistema de monitoramento de cluster hierárquico que utiliza um método estático de configuração de módulos acrescentados ao núcleo do sistema operacional de cada nó do ambiente (Sottile & Minnich, 2002). O CARD é um sistema de monitoramento de cluster hierárquico configurado para banco de dados relacionais para indexar dados no cluster (Anderson & Patterson, 1997). O PARMON é um sistema de monitoramento de cluster do tipo cliente/servidor que utilizam servidores para exportar algumas métricas e clientes que agrupam as métricas e interpretam os dados (Buyya, 2000).

Comparado com estes outros sistemas o *Ganglia* possui alguns importantes diferenciais: é utilizado um flexível e dinâmico mecanismo hierárquico de estruturação do ambiente distribuído, utiliza-se também um conjunto de ferramentas de código-fonte aberto e amplamente utilizadas, o que facilita a expansão de suas funcionalidades e tem demonstrado bom desempenho tanto em ambientes de teste como em ambientes de produção (Massie, Chun, & Culler, 2004). Além destes recursos o projeto Apache *Hadoop* também disponibilizou métricas específicas, tanto do ambiente *MapReduce* como do HDFS, que podem ser utilizadas com o *Ganglia*. Desta forma, o *Ganglia* se mostrou a ferramenta mais adequada para ser utilizada no presente trabalho.

# 4. ARQUITETURA DA PROVA DE CONCEITO

Este capítulo apresenta a arquitetura da prova de conceito proposta, com as customizações que foram necessárias em cada ferramenta para que pudesse contemplar a finalidade do presente trabalho. Também serão feitas considerações de segurança na arquitetura da prova de conceito elaborada.

## 4.1. FLUXO DE TRABALHO DA PROVA DE CONCEITO

Através do fluxo de trabalho da prova de conceito serão descritas as atividades que são realizadas em cada fase do processo proposto para análise de dados forenses utilizando computação distribuída (Ryan, Stephen, & Eng, 2009).

O escopo do presente trabalho é utilizar a computação distribuída para poder prover espaço de armazenamento para os dados e melhorar a 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 pode ser realizada pesquisas e análises diversas nos dados já indexados. A figura 4.1 ilustra o fluxo de trabalho da prova de conceito.

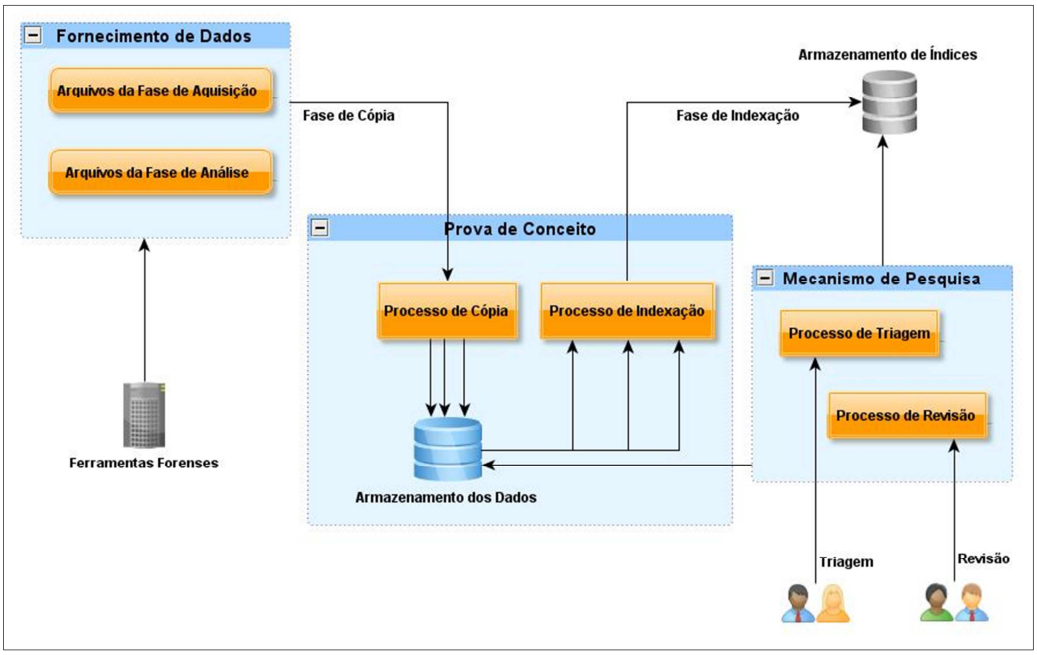



Figura 4.1 - Fluxo de Trabalho da Prova de Conceito

Conforme ilustrado no agrupamento central da figura 4.1, 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 como 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 recomendado de armazenamento do índice criado se encontra em um espaço externo ao ambiente distribuído.

Na esquerda do fluxo de trabalho da figura 4.1 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 como 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* (AccessData, 2011), (Guidance, 2011) e (SleuthKit, 2011).

O processo de indexação possibilita a realização posterior de um mecanismo de pesquisa. Este mecanismo está ilustrado no agrupamento da direita da figura 4.1. 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 da prova de conceito. O escopo principal da prova de conceito 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 presente trabalho foi desenvolvido um módulo para simular as fases de fornecimento e pesquisa dos dados, de forma a avaliar o trabalho realizado pela prova de conceito.

## 4.2. PROJETO DA PROVA DE CONCEITO

No projeto da prova de conceito (figura 4.2), os diversos componentes que integram este projeto são analisados, demonstrando o funcionamento geral da arquitetura.

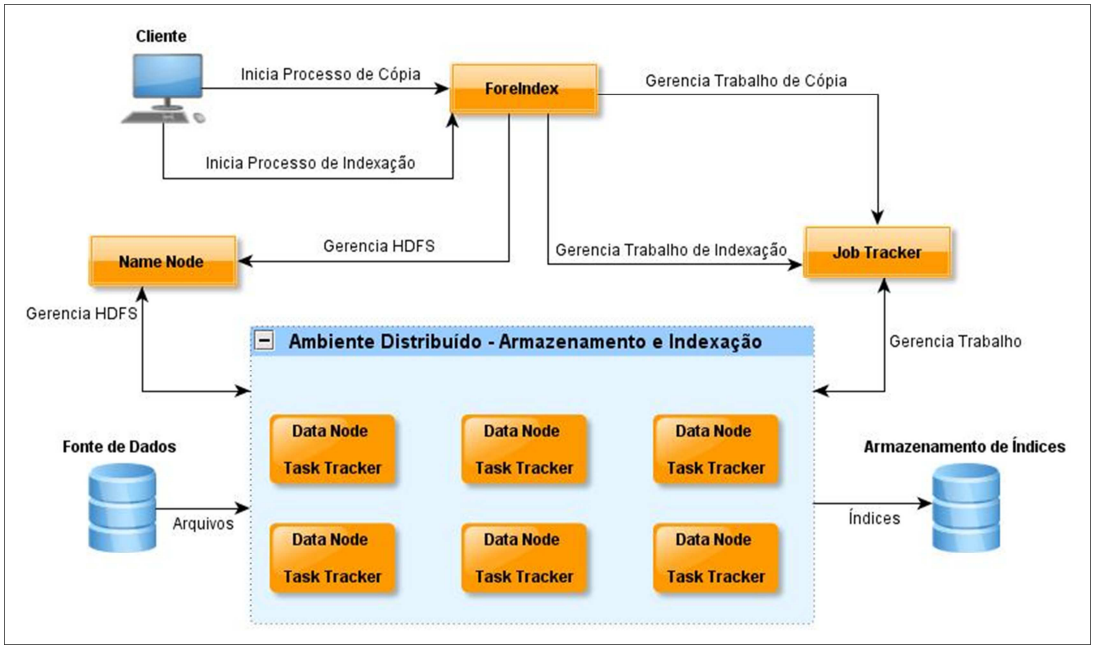



Figura 4.2 - Projeto da Prova de Conceito

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, que no caso é o *HDFS*. O *JobTracker* gerencia todo o processamento distribuído que será realizado de acordo com o algoritmo *MapReduce*. Tanto o *JobTracker* como o *NameNode* são componentes do *Apache Hadoop* e todas as funcionalidades desta solução estão disponíveis no ambiente da prova de conceito. O agrupamento central da prova de conceito 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 como 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. Este procedimento 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 (McCandless, Hatcher, & Gospodnetic, 2010).

## 4.3. PROCESSO DE CÓPIA DISTRIBUÍDA DE ARQUIVOS

Um conjunto de dados forenses geralmente possui algumas particularidades que requerem tratamento diferenciado no processamento distribuído gerenciado pelo *Apache Hadoop*. O volume de dados forenses a analisar está cada vez maior (Walter, 2005). Com isso, a quantidade e tipo de dados forenses a analisar têm 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 o desempenho do ambiente (Hadoop, 2011). 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 atualização no *Apache Hadoop* de forma a processar o arquivo completo em cada nó de processamento, ao invés 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 *Apache 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 (Hadoop, 2011).

Para solucionar estas limitações foi desenvolvida uma classe chamada *FileSetInputSplit*. Esta classe é responsável por customizar o processo de leitura do volume de dados a processar, realizando a divisão dos dados para os diversos nós do ambiente distribuído. Ao invés dos 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.

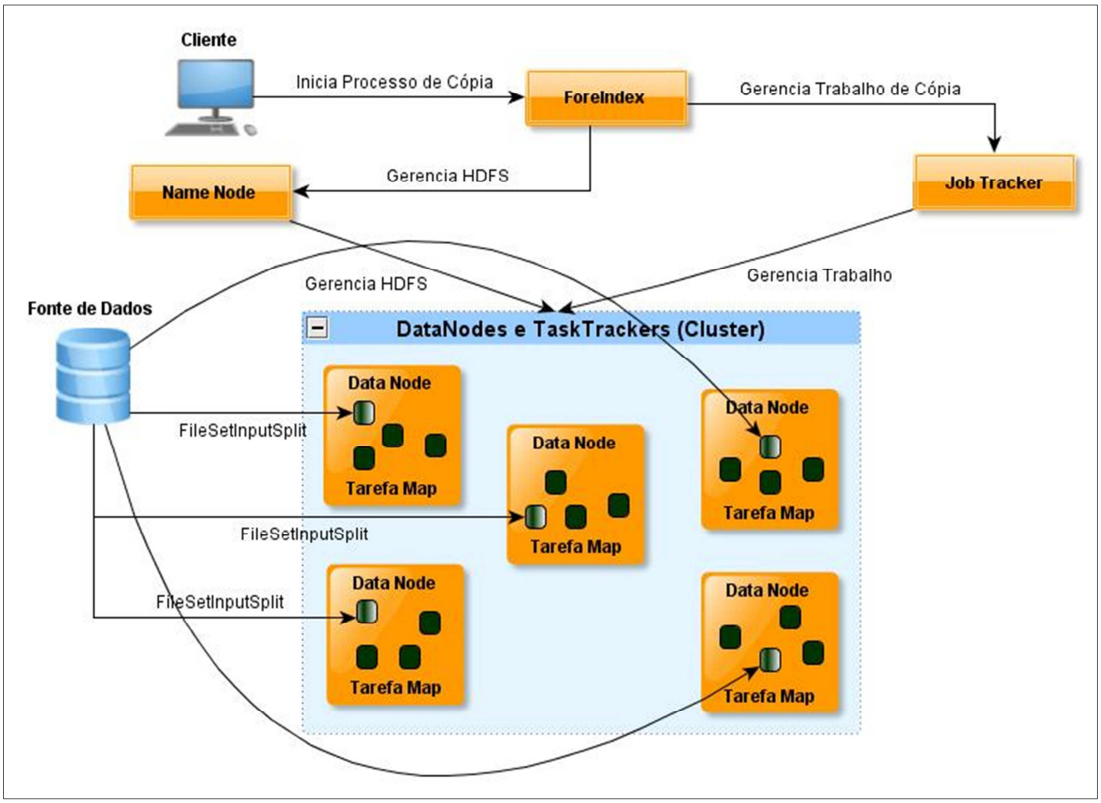



Figura 4.3 - Processo de Cópia Distribuída de Arquivos

O diagrama da figura 4.3 ilustra a fase 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 da prova de conceito. A fase de cópia de arquivos é responsável por copiar os dados de uma fonte de dados externa para o ambiente distribuído da prova de conceito. A classe *ForeIndex* controla o processo de cópia distribuída que é iniciado por um processo cliente. 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. Conforme descrito anteriormente, 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 monitora se os dados serão corretamente gravados e replicados nos *DataNodes*.

A figura 4.4 ilustra importantes componentes presentes nesta fase de cópia distribuída.

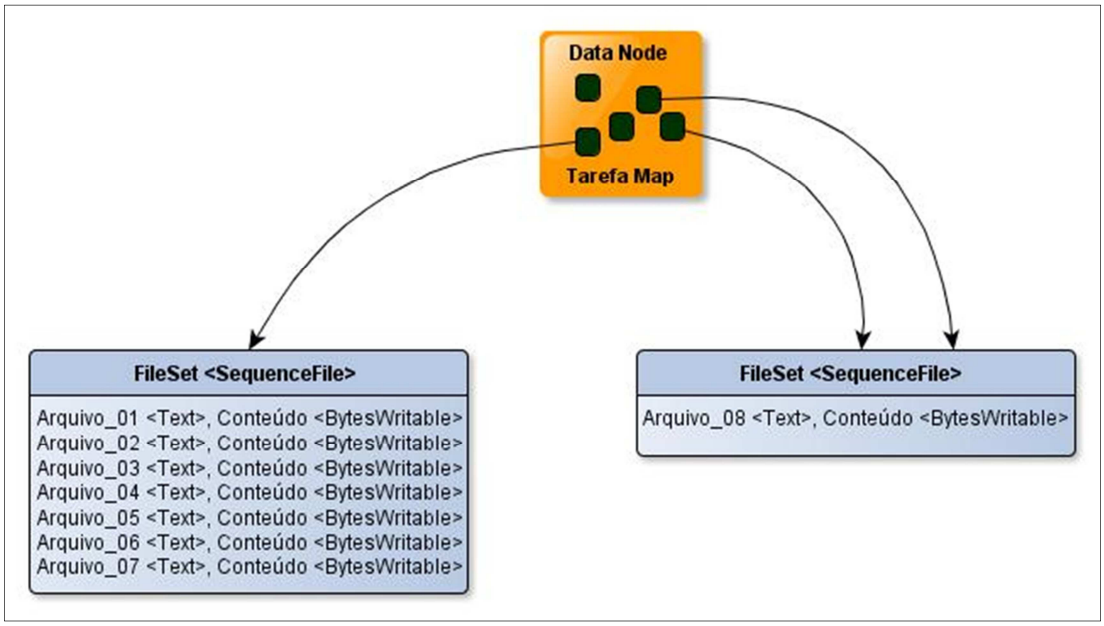



Figura 4.4 - Estrutura de Dados da Cópia Distribuída

A figura 4.4 ilustra um *DataNode* que 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. Também estão ilustrados os diversos blocos de dados que estão armazenados no *DataNode*. 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. Na figura 4.4 existem dois exemplos de objetos *FileSet*. O primeiro exemplo mostra que este objeto agrupou sete arquivos, sendo necessário o espaço de apenas um bloco de dados HDFS para armazená-lo. Já no segundo exemplo, o objeto *FileSet* agrupou apenas um arquivo, de nome Arquivo-8. Este arquivo possui um tamanho maior que o tamanho do bloco HDFS, sendo necessários dois blocos de dados HDFS para armazená-lo. Para cada arquivo que o objeto *FileSet* armazena ele grava o nome do arquivo e todos os bytes constituintes do conteúdo do arquivo.

As customizações realizadas na fase de cópia dos dados visaram a utilização do *Apache Hadoop* de forma a melhorar seu desempenho. Esta customização 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 refletirá tanto na fase de cópia de dados quanto na fase de indexação.

O tamanho dos blocos HDFS são grandes quando comparados com o tamanho dos blocos de discos rígidos e outros sistemas de arquivos. Este tamanho de bloco de 64 MB, além de reduzir a quantidade de metadados que serão armazenados no *NameNode*, aumentando conseqüentemente o volume de dados que ele pode gerenciar, também melhora o desempenho do processamento distribuído. Utilizar um bloco de dados com tamanho adequado faz com o que o tempo de transferência dos dados seja bem menor que o tempo de posicionamento da cabeça de leitura de um disco rígido. Com uma grande quantidade de blocos pequenos, o tempo de posicionamento da cabeça de leitura dos discos rígidos pode ser considerável. Além disto, o *Apache Hadoop* trabalha muito melhor blocos de dados com tamanhos maiores do que muitos blocos de tamanhos pequenos. Para realizar o processamento dos dados uma tarefa *Map* ou *Reduce* é utilizada, esta tarefa consome consideráveis recursos computacionais e existem poucas em cada nó do ambiente. Desta forma, quanto mais dados cada tarefa processar por vez, menores serão os acessos ao disco, os recursos inicializados no ambiente, e, conseqüentemente, melhor será o desempenho (Hadoop, 2011).

## 4.4. PROCESSO DE INDEXAÇÃO DISTRIBUÍDA DE ARQUIVOS

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 o desempenho do processo de indexação.

A figura 4.5 ilustra como a prova de conceito foi projetada para implementar o processo de indexação.

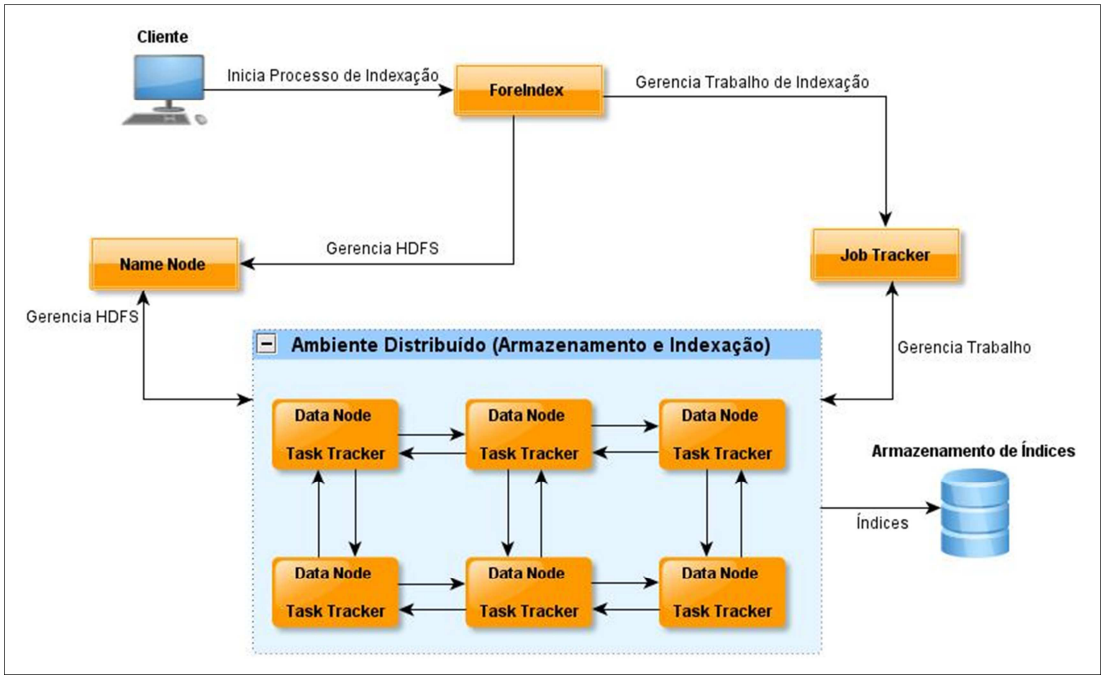



Figura 4.5 - Processo de Indexação Distribuída de Arquivos

A parte central da figura 4.5 contém um agrupamento que representa o ambiente distribuído da prova de conceito que realiza a indexação e o armazenamento dos dados forenses. 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, sendo

que estes dados 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.

A figura 4.6 ilustra como estão organizadas as tarefas *Map* e *Reduce* que implementarão a indexação distribuída.

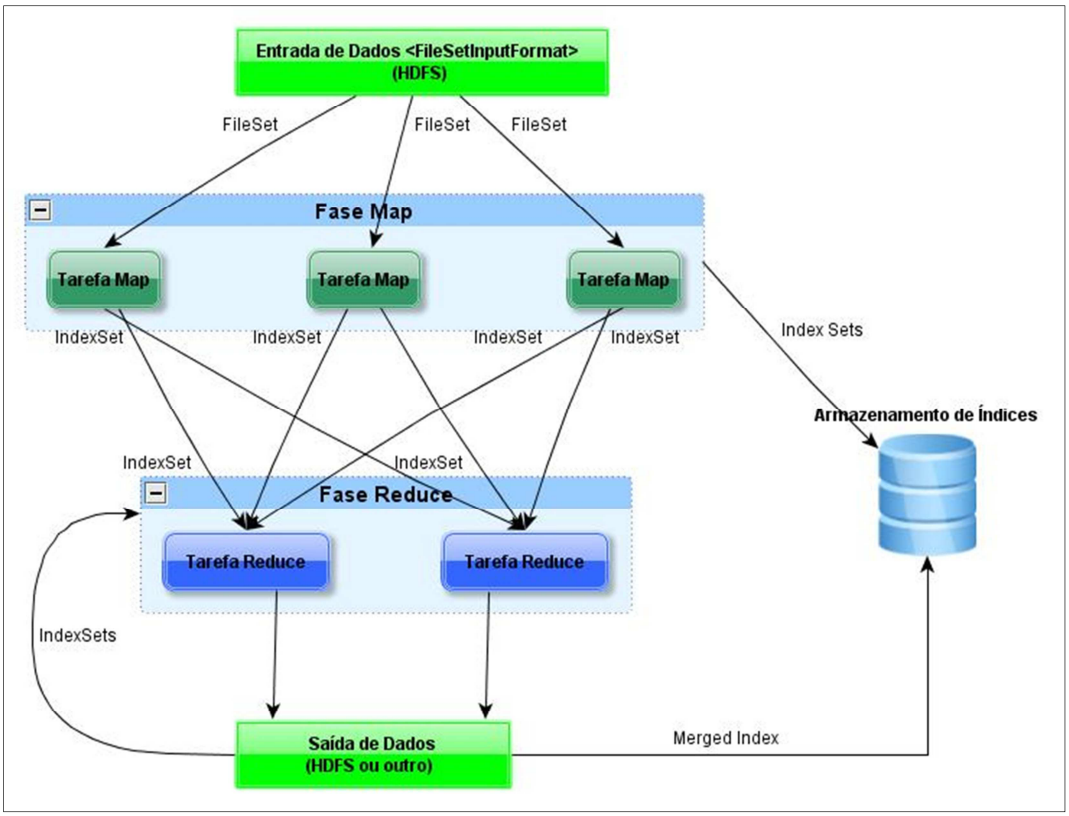



Figura 4.6 - Estrutura de Dados e Processos da Indexação Distribuída

Existem duas fases de indexação. Uma fase que será realizada por tarefas *Map* e outra fase que será executada por tarefas *Reduce*. Na fase que será executada pelas tarefas *Map* um pedaço (*Split*) do volume de dados a indexar será encaminhado para cada nó de processamento. De acordo com o descrito no processo de cópia, os *Splits* estarão no formato de objetos do tipo *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, a tarefa *Map* extrai todos os arquivos que estão agrupados no objeto *FileSet*, interpreta estes arquivos extraindo os textos relevantes de seu conteúdo e

cria a estrutura de dados que será o índice. 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 e escritas adicionais em disco. Cada índice criado pela tarefa *Map* é agrupado em um objeto chamado *IndexSet*, sendo que este objeto é gravado em uma unidade de armazenamento de índices. Quando as atividades *Map* finalizam a sua execução existem diversos objetos *IndexSets* 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 o desempenho na pesquisa, pode ser realizado um agrupamento de todos os *IndexSets*. Este processo de agrupamento se chama *Merge* e é executado por tarefas *Reduce*. As tarefas *Reduce* recebem uma quantidade de *IndexSets*, na medida em que tarefas *Maps* vão finalizando seu processamento e realiza o *merge* destes índices. O processamento das tarefas *Reduce* disponíveis pode ser executado quantas vezes for necessário para agrupar todos os *IndexSets* 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 customizações 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 o desempenho 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.

Nos cenários de teste que foram elaborados será demonstrado como cada uma destas características melhoraram o desempenho global da prova de conceito.

## 4.5. CONSIDERAÇÕES DE SEGURANÇA COMPUTACIONAL

Na prova de conceito elaborada foram observados alguns aspectos de segurança computacional visando harmonizar a necessidade de desempenho com a necessidade de segurança dos canais de comunicação utilizados no sistema distribuído e o controle de acesso aos recursos compartilhados.

A figura 4.7 ilustra a estrutura de processos da prova de conceito, a definição do ambiente controlado e as interfaces que necessitam de um maior controle de acesso.

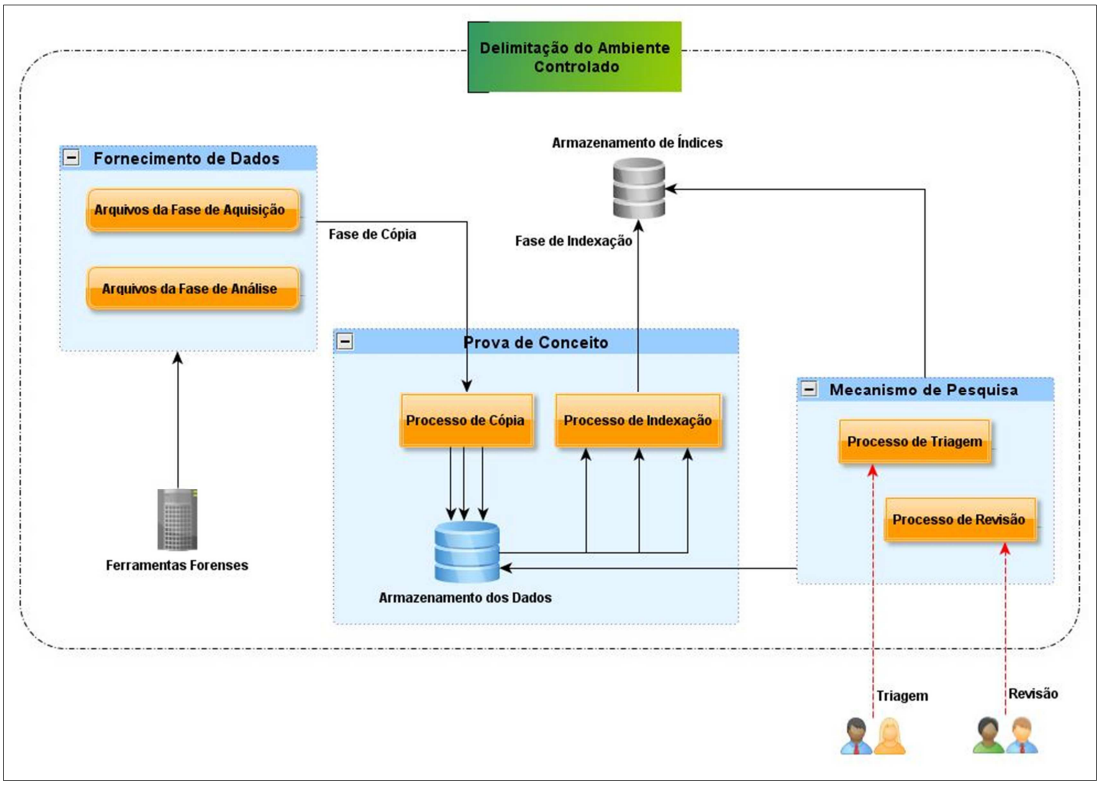



Figura 4.7 – Diagrama com Delimitações de Segurança na Prova de Conceito

A política de segurança adotada neste trabalho delimitou toda a área com os processos de fornecimento e pesquisa a dados, além da cópia e indexação distribuída como um ambiente controlado. Devido ao motivo da prova de conceito se tratar da indexação e pesquisa a dados forenses, ou seja, dados resultantes de apreensões policiais e sujeitos estarem inclusive sob segredo de justiça, a política de segurança buscou contemplar todos estes recursos. O ambiente controlado é composto de todos os processos construídos pela prova de conceito criada, com exceção da interface com o usuário que realiza a pesquisa e indexação dos dados. A política de segurança requer que o cluster construído seja

hospedado em um local com controle físico de acesso e sem conexão com redes externas, com exceção da conexão que os processos de triagem e revisão devem proporcionar.

Na prova de conceito elaborada, o cluster foi construído nas instalações do laboratório do Serviço de Perícias de Informática do Departamento de Polícia Federal. Estas instalações possuem controle físico de acesso para poder acessar os recursos computacionais ali disponíveis. No laboratório foi construído o cluster e o protótipo para simular o processo de fornecimento e pesquisa aos dados. O cluster foi construído como uma Intranet fechada, sendo que apenas as máquinas autorizadas poderiam se conectar com esta Intranet. Apenas uma máquina deste ambiente possuía conexão com o meio externo para possibilitar pesquisas remotas. Esta conexão externa que foi objeto de implementação de mecanismos específicos de segurança.

A figura 4.7 possui uma linha tracejada nas conexões dos processos responsáveis por realizar a pesquisa nos índices e dados do ambiente distribuído. Estas conexões necessitam de mecanismos de segurança para garantir a privacidade aos dados e a integridade dos canais de comunicação. Estes clientes acessam o sistema distribuído através de uma conexão TLS e utilizando listas de controle de acesso. Desta forma, proporciona segurança ao meio de comunicação e controle de acesso aos recursos compartilhados.

Além disto, o próprio *Apache Hadoop* possui alguns recursos adicionais de segurança que foram implementados no ambiente distribuído. Existe uma lista que controla quais computadores poderão se conectar com *JobTracker* e com o *NameNode*. Nesta lista fica registrado o endereço IP dos computadores autorizados e não autorizados a acessar o ambiente distribuído. Este recurso implementa um simples mecanismo de controle de acesso aos recursos compartilhados. Além disto, o fato do *Apache Hadoop* implementar requisitos da arquitetura de um sistema distribuído e utilizar um sistema de arquivos distribuído como o HDFS faz com que o ambiente tenha uma segurança maior na integridade dos dados, na tolerância a falhas e disponibilidade do ambiente. O HDFS implementa replicação de bloco de dados. Quando um bloco de dados é lido é checado se o bloco de dados está íntegro através da verificação de um código de detecção de erro. Se o bloco não está íntegro ele é descartado, é utilizado um bloco íntegro de outro *DataNode* e uma nova replicação do bloco íntegro é realizada. Este mecanismo proporciona uma maior segurança na integridade das informações armazenadas no ambiente distribuído. Se algum

computador do ambiente distribuído ficar indisponível tanto os dados armazenados nele como as tarefas que ele estava processando são disponibilizados por outros computadores do ambiente, este recurso aumenta o nível de segurança do ambiente distribuído proporcionando tolerância a falhas e disponibilidade ao ambiente.

## 4.6. INTERFACES DA PROVA DE CONCEITO

Foi construída uma interface gráfica para poder simular os processos de fornecimento e pesquisa aos dados armazenados. Nesta interface também é possível acionar os processo de cópia e indexação distribuída. Todas as funcionalidades disponíveis na interface gráfica também podem ser acessadas via linha de comando, utilizando a classe *ForeIndex*, conforme descrito no anexo B. As figuras 4.8, 4.9 e 4.10 contêm as telas da interface gráfica da prova de conceito[2].

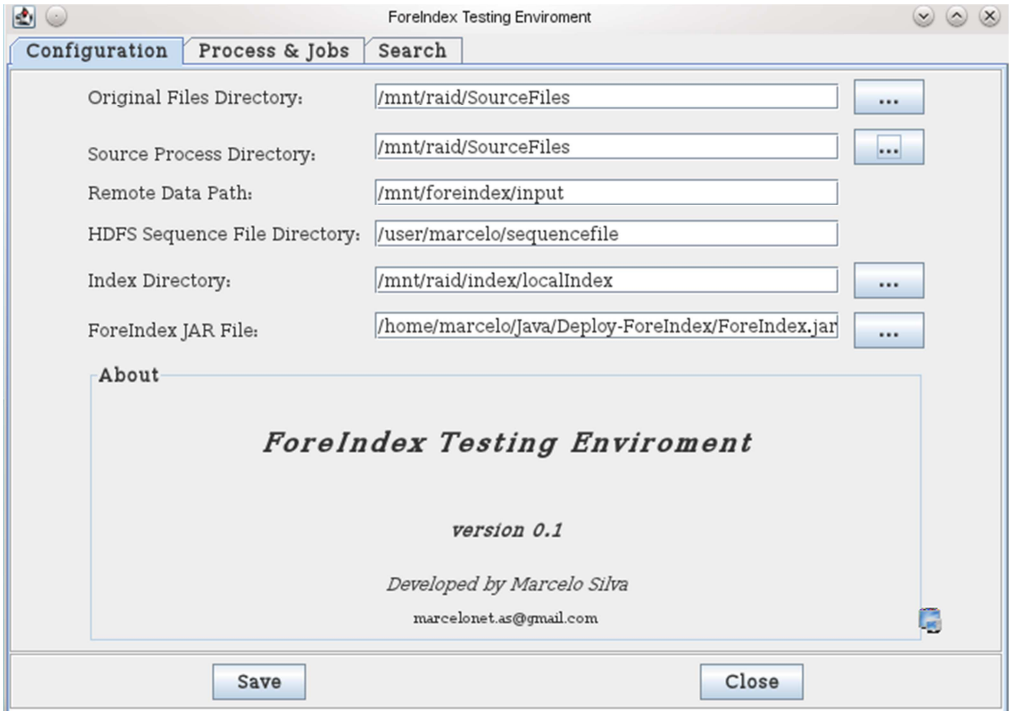



Figura 4.8 – Tela de Configuração da Interface Gráfica

A figura 4.8 ilustra a tela da interface gráfica responsável por realizar a configuração das funcionalidades da prova de conceito. Nesta tela é possível configurar a localização

[2] A interface gráfica está em inglês para utilizar um padrão internacional que possibilita que o presente trabalho seja utilizado em outros projetos de código-fonte aberto.

original dos dados forenses e a localização para a qual os dados serão copiados para simular o processo de fornecimento de dados. Esta configuração é definida nos campos *Original Files Directory* e *Source Process Directory*, respectivamente. É possível configurar o nome do compartilhamento NFS que possibilitará o acesso remoto aos dados que serão copiados de forma distribuída no processo de cópia distribuída. Esta configuração é realizada no campo *Remote Data Path*. O campo *HDFS Sequence File Directory* contém o diretório do sistema de arquivos distribuído HDFS onde os objetos *FileSets* serão gravados. O campo *Index Directory* contém o nome do compartilhamento NFS onde o índice resultante do processo de indexação distribuída será armazenado. Por fim, o campo *ForeIndex JAR File* contém o arquivo do tipo *Java Archive* – JAR, onde toda a implementação do processo distribuído do *Apache Hadoop* é realizada. Este arquivo será utilizado para iniciar as atividades de cópia e indexação distribuídas.

A figura 4.9 contém recursos para o acionamento dos processos distribuídos e outras atividades de manutenção no ambiente da prova de conceito.

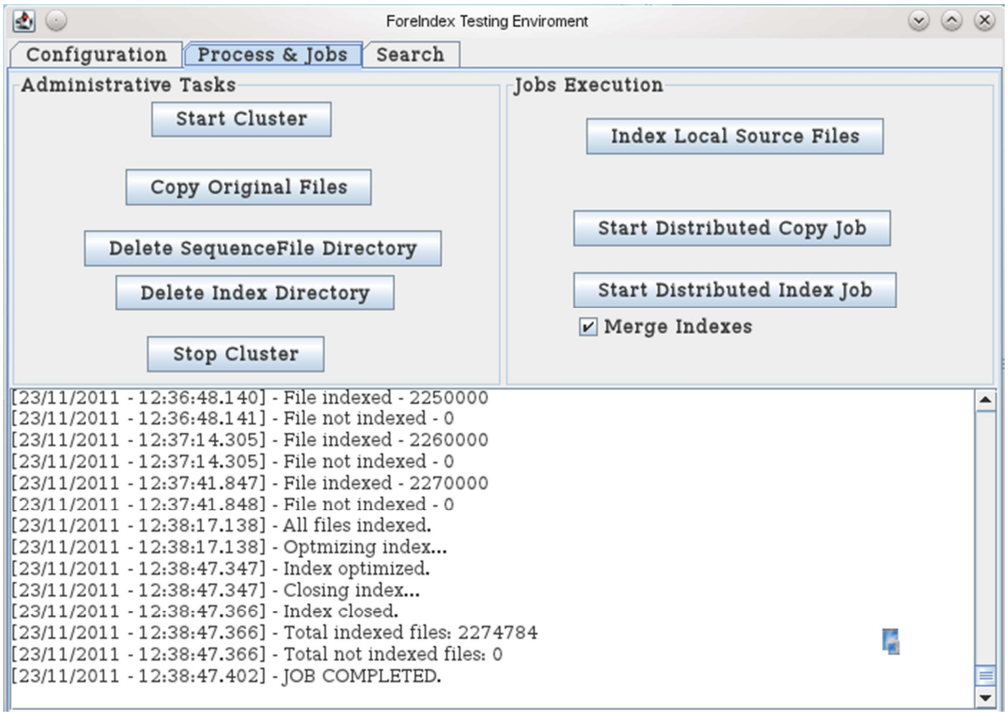



Figura 4.9 – Tela de Acionamento de Processos e Atividades da Prova de Conceito

A tela ilustrada na figura 4.9 possibilita algumas atividades administrativas da prova de conceito. As atividades disponíveis são:

- inicializar e finalizar o servidor *Apache Hadoop* e o HDFS, através dos botões *Start Cluster* e *Stop Cluster*;

- excluir os diretórios onde se encontram os objetos *FileSets* e os índices criados no processamento distribuído através dos botões *Delete SequenceFile Directory* e *Delete Index Directory*;

- realizar a simulação da fase de fornecimento de dados clicando no botão *Copy Original Files*.

Esta tela também permite o acionamento dos processos de cópia distribuída de arquivos, de indexação local de arquivos e de indexação distribuída de arquivos, sendo que a indexação distribuída pode ser realizada tanto com o *merge* dos índices resultantes como sem o *merge*. Estas funcionalidades estão disponíveis nos botões: *Index Local Source Files*, *Start Distributed Copy Job* e *Start Distributed Index Job*. Além disto, nesta tela também é possível visualizar o histórico de execução das atividades. No exemplo da figura 4.9 está sendo ilustrado o histórico das atividades de um processo de indexação local de arquivos.

A figura 4.10 ilustra a tela que possibilita a realização de consultas indexadas nos índices criados.

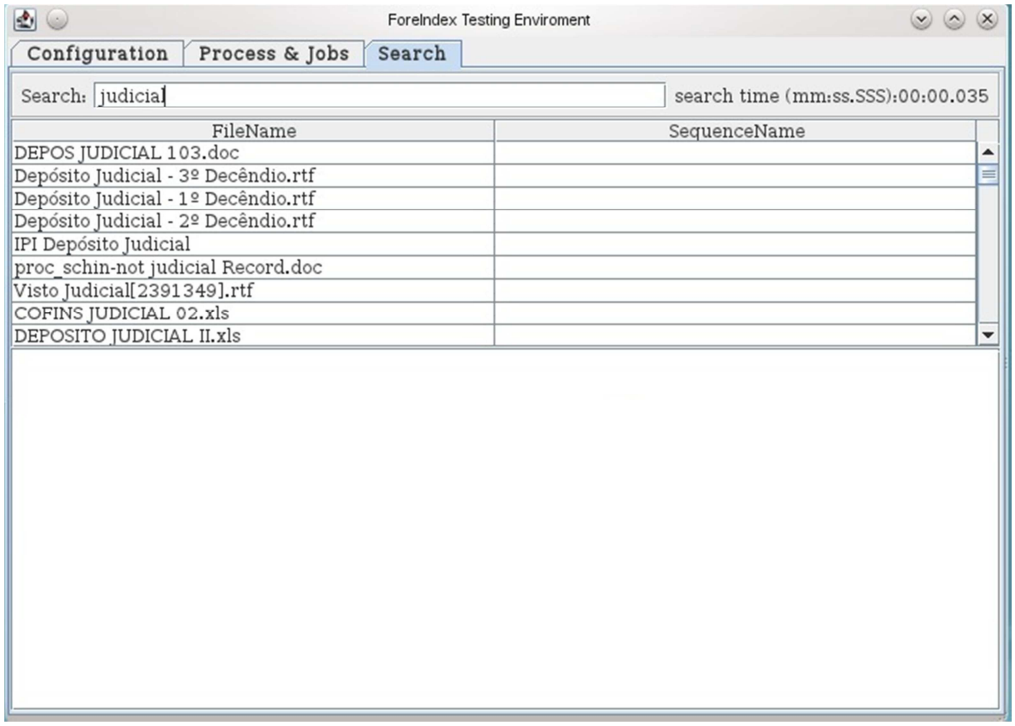



Figura 4.10 – Tela de Busca Indexada nos Índices Criados

Na tela ilustrada na figura 4.10 é possível fornecer termos de pesquisa para realizar a busca indexada nos dados. Esta pesquisa deve ser realizada após os processos de cópia e indexação distribuída dos dados forenses. Esta tela mostra todos os arquivos recuperados, o tempo de realização da pesquisa e o conteúdo de cada arquivo. O objetivo deste recurso é simular os processos de triagem e revisão dos dados, além de medir o desempenho do processo de pesquisa.

Quando uma pesquisa é realizada o histórico de atividades de cada detalhe da pesquisa também está listado nesta interface gráfica, conforme ilustra a figura 4.11.

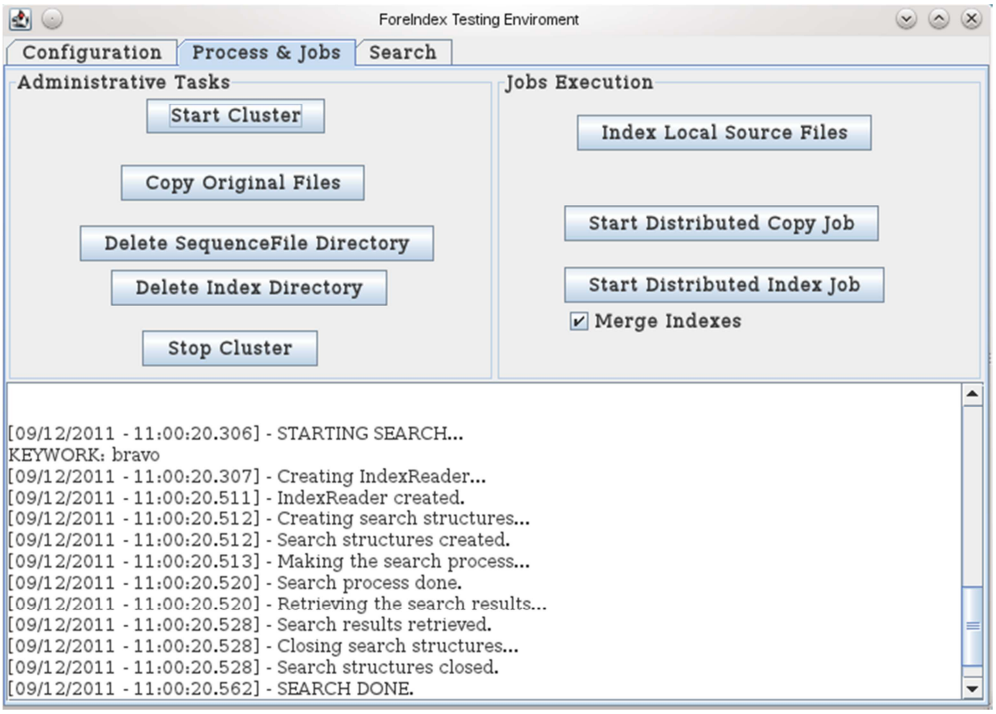



Figura 4.11 – Tela com o Histórico de Atividades Realizadas em uma Busca Indexada

Os procedimentos necessários tanto para a execução desta interface gráfica como para a instalação, configuração e execução de toda a prova de conceito elaborada estão descritos no anexo B deste trabalho.

# 5. CENÁRIOS DE TESTE DA PROVA DE CONCEITO

Este capítulo apresenta os cenários que foram elaborados e utilizados para poder realizar uma avaliação da prova de conceito criada. Serão apresentados os critérios de elaboração dos cenários; analisados os requisitos, características e componentes, bem como os resultados obtidos através da utilização da prova de conceito em cada um deles.

## 5.1. REQUISITOS PARA A AVALIAÇÃO DA PROVA DE CONCEITO

A prova de conceito elaborada possui duas principais funcionalidades que são os processos de cópia e indexação distribuídos. Um importante requisito para a avaliação destas funcionalidades é a realização de um comparativo do desempenho destes processos, tanto em um ambiente centralizado como em um ambiente distribuído. Através deste comparativo é possível analisar os impactos que a escalabilidade do ambiente distribuído pode causar no desempenho destes processos.

Os processos de cópia e indexação distribuídos foram elaborados com o objetivo de trabalhar com dados forenses. Um requisito para a elaboração dos cenários de testes é que sejam fornecidos dados forenses representativos do universo de dados comumente encontrados em processos periciais forenses.

Atualmente existem ferramentas que são comumente utilizadas para a realização de atividades periciais em dispositivos de armazenamento computacional. Conforme exposto no Capítulo 3, o FTK e o *EnCase* são as principais ferramentas utilizadas para a análise de dados forenses. A realização de um comparativo do desempenho da funcionalidade de indexação destas ferramentas com a funcionalidade de indexação distribuída da prova de conceito elaborada é um ponto importante para a avaliação geral de desempenho do presente trabalho, uma vez que realiza a comparação com as principais ferramentas periciais utilizadas no mercado.

Após a criação dos índices é necessária a realização de pesquisas nos índices criados para verificar o desempenho destas pesquisas, tanto para os índices criados de forma centralizada como para os índices criados de forma distribuída. Também é necessária a

verificação do desempenho das pesquisas utilizando os índices que foram criados sem e com a funcionalidade de *merge* distribuído.

## 5.2. CARACTERÍSTICAS GERAIS DOS CENÁRIOS ELABORADOS

Para a realização dos testes foi selecionada uma massa de dados resultante de uma análise pericial do Departamento de Polícia Federal. 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 5.1 demonstra as características dos dados forenses que compõem a massa de dados selecionada.

Tabela 5.1 - Massa de Dados do Cenário de Teste

| Parâmetro | Valor |
|---|---|
| Quantidade de Arquivos | 2.274.784 |
| Tamanho Total dos Arquivos | 279 GB |
| Formatos dos Arquivos Considerados | .txt, .rtf, .csv, .pdf, .xls(x), .doc(x) |

Esta massa de dados foi utilizada para testar os processos de cópia e indexação, sendo que estes processos foram testados de forma centralizada e distribuída.

Em todos os testes realizados foram utilizados computadores com a mesma configuração de componentes físicos. A configuração principal do tipo de computador utilizado nos testes está detalhada na tabela 5.2.

Tabela 5.2 – Configuração do Tipo de Computador Utilizado nos Testes

| Parâmetro | Valor |
|---|---|
| Processador | Intel Core-2 Quad, 2.66 GHz |
| Placa-Mãe | Asus, modelo P5WDGWS Professional |
| Memória RAM | 04 GB de RAM, DDR2 667 - Kingston |
| Disco Rígido do Sistema Operacional | 01 HD SATA-II, 320 GB, modelo ST3320620AS |

Para a realização dos testes no ambiente distribuído foi montado um cluster contendo doze computadores. Estes doze computadores são todos da mesma configuração do computador descrito na tabela 5.2. Esta similaridade não é obrigatória (o cluster poderia ser montado com máquinas de diversas configurações). Para a interconexão dos computadores do cluster foram utilizados dois *switches* da marca *Netgear*, modelo GS108. Com a utilização destes switches foi configurada uma rede *Gigabit Ethernet* para o tráfego de dados do cluster. A figura 5.1 contém um gráfico descrevendo a arquitetura dos componentes utilizados no cluster.

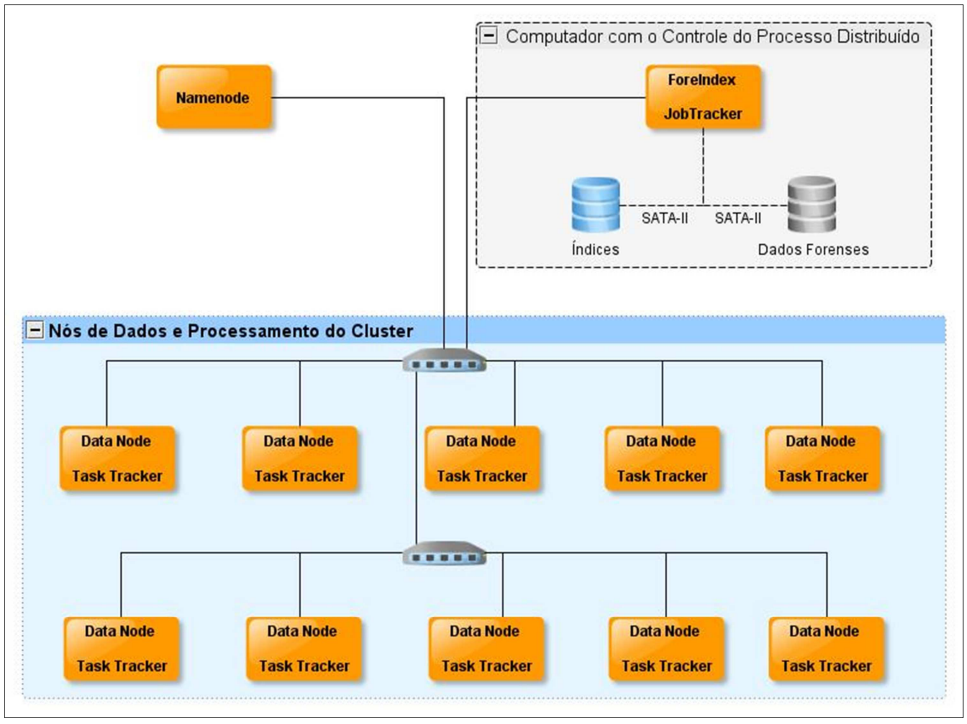



Figura 5.1 – Cluster Montado para os Cenários de Testes

Cada retângulo de bordas arredondadas da figura 5.1 representa um computador do cluster. Esta figura mostra os doze computadores componentes do cluster. No cluster montado foi instalado e configurado o *Apache Hadoop*. O processo *Namenode*, que é o processo responsável pelo gerenciamento do sistema de arquivos distribuído HDFS, ficou instalado em um único computador. O *JobTracker*, que é o processo responsável pela coordenação do processamento distribuído com *MapReduce*, ficou instalado em um computador. No mesmo computador do *JobTracker* também é executado o cliente *ForeIndex*, que é a classe que submete a execução dos processos de cópia e indexação distribuídos. Ainda no computador onde está o *JobTracker*, estão fisicamente instalados o disco rígido que contém os dados forenses que são copiados para o ambiente distribuído e o disco rígido

que contém os índices que são criados pelo processo de indexação distribuída. Estes discos rígidos estão fisicamente instalados no computador do *JobTracker* através da interface SATA-II. Os outros dez computadores contém tanto um processo *DataNode* como um processo *Task Tracker*. O processo *Datanode* contém uma parte dos dados do HDFS. O processo *Task Tracker* gerencia a execução de uma tarefa distribuída, seja uma tarefa *Map* ou uma tarefa *Reduce*. Estes dez computadores, chamados aqui de *workers*, estão interconectados por um *switch Gigabit Ethernet* formando uma rede local. Estes *switches* estão conectados entre si e com os computadores *Namenode* e *JobTracker*, de forma a permitir a interconexão de todo o cluster. Os *workers* possuem acesso direto aos discos rígidos conectados no computador *JobTracker* utilizando o *NFS – Network FileSystem* que é o sistema de arquivos remoto do *Linux*. Todos os doze computadores do cluster possuem instalados como sistema operacional a distribuição openSUSE 11.4 do Linux. Esta distribuição contém o *kernel* do Linux versão 2.6. A ferramenta utilizada para coletar as métricas de processamento, memória, rede e disco de todas as máquinas no cluster foi o *Ganglia*, que está descrito no capítulo 3. A tabela 5.3 resume as características dos componentes utilizados no cluster configurado.

Tabela 5.3 – Características do Cluster Configurado

| Parâmetro | Valor |
|---|---|
| Quantidade de Computadores | 12 |
| Quantidade de *JobTrackers* | 01 |
| Quantidade de *Namenodes* | 01 |
| Quantidade de *Workers* | 10 |
| Sistema Operacional | Linux 2.6 (distribuição OpenSUSE 11.4) |
| Versão do *Hadoop* | 0.20.203 |
| Tipo de Rede | Gigabit Ethernet |
| Ferramenta de Coleta de Métricas | Ganglia 3.1.7 (Interface Web 2.1.8) |

Serão apresentados os três cenários de teste, a saber: Cópia e Indexação Simples; Cópia e Indexação Composta, sem Merge; e Cópia e Indexação Composta, com Merge.

## 5.3. PRIMEIRO CENÁRIO DE TESTE - CÓPIA E INDEXAÇÃO SIMPLES

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 figura 5.2 ilustra a organização dos processos e componentes envolvidos neste cenário.

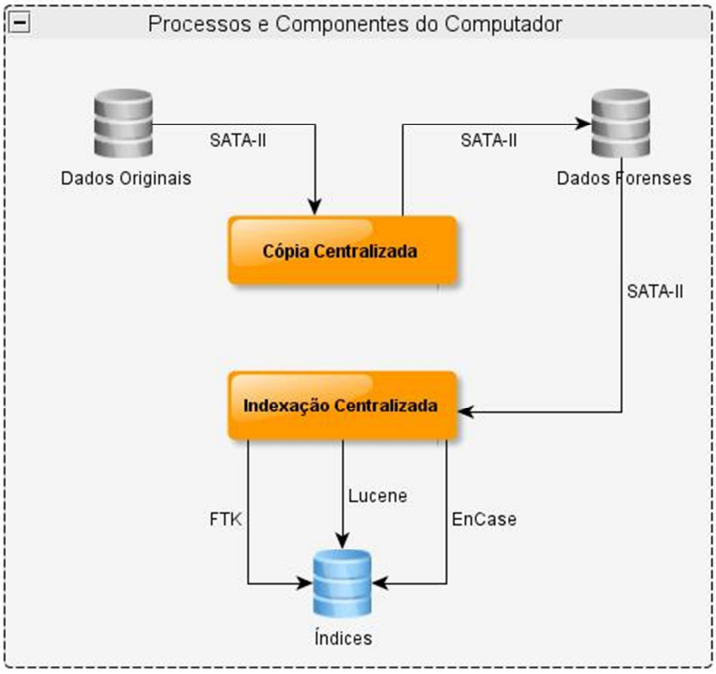



Figura 5.2 – Componentes do Primeiro Cenário de Testes

No computador utilizado neste cenário de testes existem três discos rígidos conectados através da interface SATA-II. O primeiro disco rígido contém os dados forenses originais e o segundo conterá os dados forenses após o processo de cópia local. O terceiro disco rígido conterá os índices que serão produzidos no processo de indexação. Foram instalados os sistemas operacionais Windows 7 e Linux 2.6, com a finalidade de medir o desempenho da cópia e indexação de arquivos nestes ambientes.

O processo de cópia centralizada consistirá simplesmente na utilização dos recursos do sistema operacional para realizar a cópia dos dados forenses do disco de origem para o disco de destino. Este processo visa a fornecer um parâmetro de comparação com o processo de cópia distribuída. O processo de indexação consistirá na utilização centralizada da ferramenta de indexação selecionada neste projeto que é o *Apache Lucene*. Para efeito

de comparação, com as principais ferramentas forenses comerciais, também será mensurado o tempo de indexação com o FTK e o *EnCase*. A tabela 5.4, contém o resumo dos resultados obtidos nos testes deste cenário.

Tabela 5.4 – Resultados do Primeiro Cenário de Testes

| Configuração do Teste | Tempo de Cópia *(hh:mm:ss)* | Tempo de Indexação *(hh:mm:ss)* | Tamanho do Índice |
|---|---|---|---|
| Linux e Lucene | 08:01:28 | 04:21:17 | 504 MB |
| Windows e FTK | 17:28:35 | 21:50:55 | 1,2 GB |
| Windows e EnCase | 17:28:35 | 25:55:34 | 1,4 GB |

A figura 5.3 ilustra algumas métricas do teste de indexação centralizada.

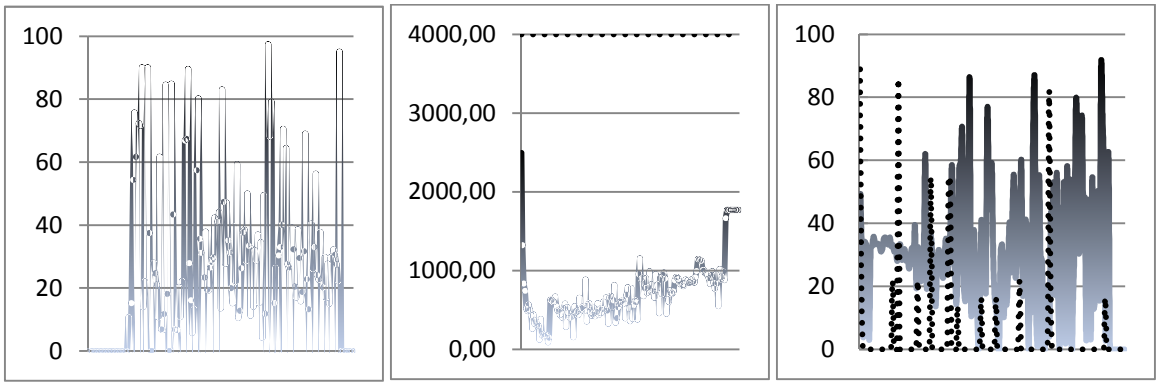



Figura 5.3 – Métricas de Processador, Memória e Disco da Indexação Centralizada

Foi realizada uma junção das métricas obtidas com o processo de indexação tanto do *Lucene*, como do FTK e do *EnCase* por apresentarem resultados similares. Será realizada a análise da junção destas métricas que estão expostas na figura 5.3.

No processo de indexação centralizada nas ferramentas utilizadas foi verificado um uso médio de 1,2 GB de memória RAM dos 4,0 GB disponíveis. O uso de memória foi crescente no decorrer da atividade de indexação, mas moderado. O processador obteve uma taxa media de 60% de uso durante o período de processamento. A taxa de leitura do disco (linha contínua) ficou uma média de 50 MB/s e de escrita (linha tracejada) em 20

MB/s. Foi verificado que os principais fatores de desempenho do processamento foi o tempo de acesso ao disco e a capacidade de processamento.

## 5.4. SEGUNDO CENÁRIO DE TESTE – CÓPIA E INDEXAÇÃO COMPOSTA, SEM MERGE

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 doze computadores, com processamento distribuído e sem a realização do merge dos índices. Serão utilizados todos os recursos do cluster configurado, desta forma, a relação de processos e componentes será a mesma ilustrada na figura 5.1 e descrita na tabela 5.3.

Os processos de cópia e indexação distribuídos foram executados de acordo com o descrito na seção 4.3 e 4.4 do capítulo 4 – Arquitetura da Prova de Conceito.

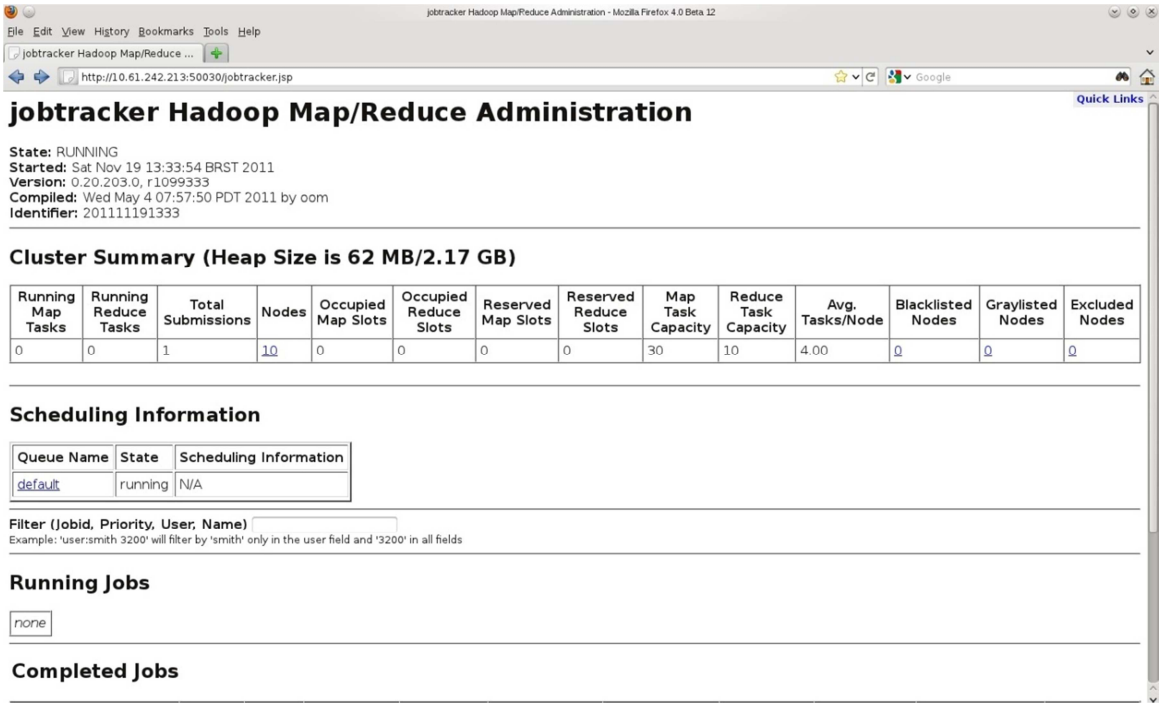



Figura 5.4 – Tela de Gerenciamento do *JobTracker*

As figuras 5.4 e 5.5 ilustram telas de gerenciamento do *JobTracker* após a realização do processamento de cópia distribuída de arquivos. A tela da figura 5.4 observa-se dez nós que correspondem aos dez *workers* descritos na figura 5.1 anterior. Cada *worker* tem um

processador com quatro núcleos e o ambiente aloca 40 *slots* de tarefas que podem ser distribuídas simultaneamente no cluster, sendo 30 para tarefas *Map* e 10 para tarefas *Reduce*. A figura 5.5 ilustra o trabalho de cópia distribuída que foi submetido ao ambiente de processamento *MapReduce* do *Hadoop*, através da submissão do *job* nº 201111191333_0001 ao *JobTracker*. Dentre os dados do trabalho de cópia distribuída que foi submetido ao *Hadoop*, a tabela desta figura mostra os seguintes dados: o nome do trabalho, o status de execução, o horário inicial e final da execução do trabalho, bem como a porcentagem de processamento das tarefas *Map* e *Reduce*.

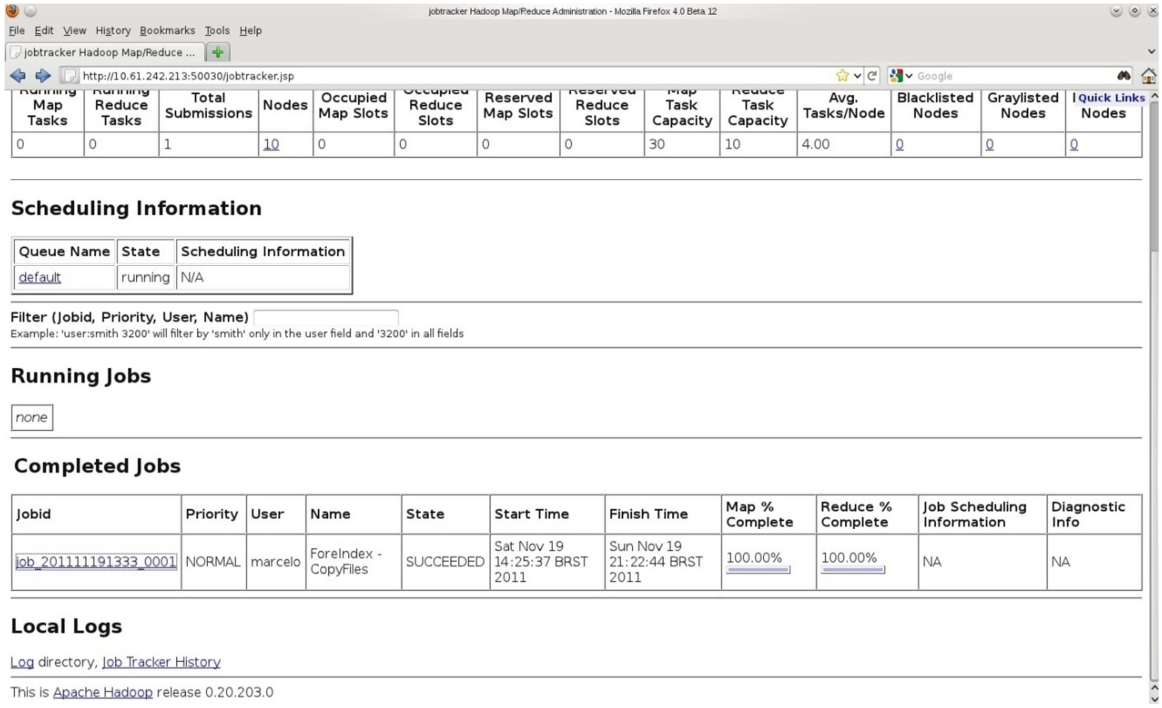
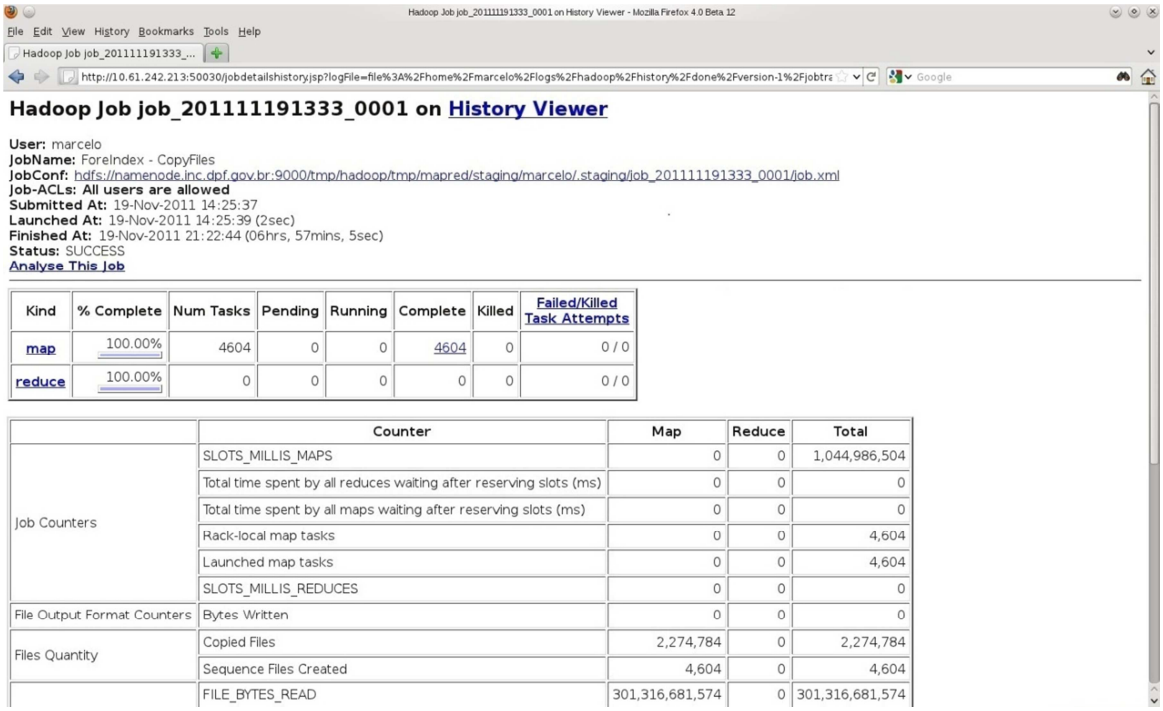



Figura 5.5 – Tela de Gerenciamento do *JobTracker* após a Cópia Distribuída

Estas telas de gerenciamento do *JobTracker* permitem um monitoramento remoto da execução dos trabalhos submetidos ao *Hadoop*. A figura 5.6 ilustra outra tela do ambiente *Hadoop* que detalha os resultados obtidos com o processamento do trabalho de cópia distribuída de arquivos.



Hadoop Job job_201111191333_0001 on History Viewer

**User:** marcelo
**JobName:** ForeIndex - CopyFiles
**JobConf:** hdfs://namenode.inc.dpf.gov.br:9000/tmp/hadoop/tmp/mapred/staging/marcelo/.staging/job_201111191333_0001/job.xml
**Job-ACLs:** All users are allowed
**Submitted At:** 19-Nov-2011 14:25:37
**Launched At:** 19-Nov-2011 14:25:39 (2sec)
**Finished At:** 19-Nov-2011 21:22:44 (06hrs, 57mins, 5sec)
**Status:** SUCCESS
Analyse This Job

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 100.00% | 4604 | 0 | 0 | 4604 | 0 | 0 / 0 |
| reduce | 100.00% | 0 | 0 | 0 | 0 | 0 | 0 / 0 |

| | Counter | Map | Reduce | Total |
|---|---|---|---|---|
| Job Counters | SLOTS_MILLIS_MAPS | 0 | 0 | 1,044,986,504 |
| | Total time spent by all reduces waiting after reserving slots (ms) | 0 | 0 | 0 |
| | Total time spent by all maps waiting after reserving slots (ms) | 0 | 0 | 0 |
| | Rack-local map tasks | 0 | 0 | 4,604 |
| | Launched map tasks | 0 | 0 | 4,604 |
| | SLOTS_MILLIS_REDUCES | 0 | 0 | 0 |
| File Output Format Counters | Bytes Written | 0 | 0 | 0 |
| Files Quantity | Copied Files | 2,274,784 | 0 | 2,274,784 |
| | Sequence Files Created | 4,604 | 0 | 4,604 |
| | FILE_BYTES_READ | 301,316,681,574 | 0 | 301,316,681,574 |

Figura 5.6 – Tela de Gerenciamento do *JobTracker* detalhando a Cópia Distribuída

A tela ilustrada na figura 5.6 mostra o tempo total de processamento do processo de cópia distribuída, que foi de 06:57:05 (hh:mm:ss). Pela verificação dos dados contidos na figura 5.6 verifica-se que para realizar este processamento foram utilizadas 4.604 tarefas *Map*, que foram distribuídas para os 10 *workers* de processamento do cluster no decorrer do processamento. O processo de cópia de 2.274.784 arquivos de dados forenses resultou na geração de 4.604 arquivos do tipo *FileSet*. Durante a realização do processamento da cópia distribuída foram coletadas algumas métricas, como a utilização do processador, memória e rede. As figuras 5.7 e 5.8 ilustram estas métricas.

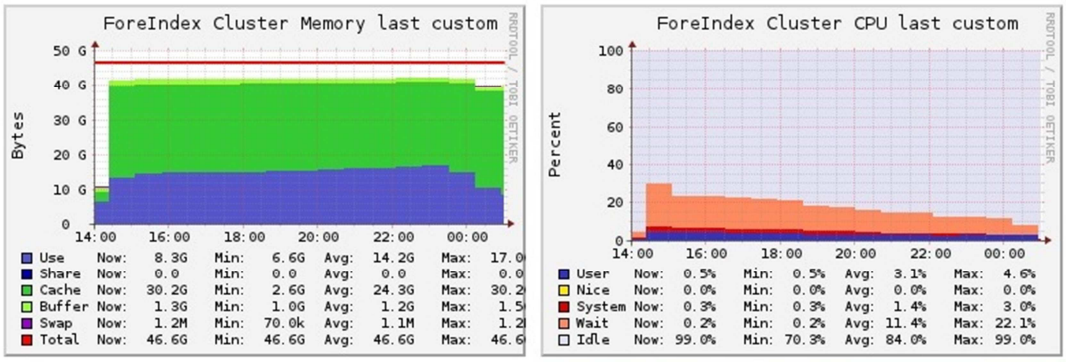



Figura 5.7 – Métricas de Memória e Processador do Processo de Cópia


Figura 5.8 – Métricas de Rede e Memória do Processo de Cópia

A figura 5.8 ilustra as métricas de rede de utilização de todo o cluster e o detalhamento das métricas de utilização de memória de cada nó do cluster, ilustrando que os *workers* tiveram uma utilização moderada e equivalente de memória, o *JobTracker* teve uma maior utilização de memória para gerenciar todo o processo distribuído e o *NameNode* teve pouca utilização de memória para gerenciar o sistema de arquivos distribuído.

Pela análise das métricas ilustradas nas figuras 5.7 e 5.8 verifica-se que dos 48 GB de RAM disponíveis em todo o cluster houve um consumo médio de 14,2 GB de RAM com processos de cópia distribuída, o que corresponde a 29,58% da capacidade total do ambiente. Os processadores do cluster ficam 84,9% do tempo desocupados, 11,4% do tempo bloqueados e apenas 3,1% do tempo em processamento efetivo. A utilização da rede foi maior nas duas primeiras horas de processamento com redução gradual do tráfego no restante do tempo. Estes dados indicam que a característica deste processo é predominantemente de Entrada/Saída, com pouca utilização de CPU e memória. A rede configurada atendeu bem a demanda de transferência de dados, mostrando que o principal fator determinante do tempo do processo de cópia distribuída foi o tempo de leitura dos dados nos disco rígido de origem dos dados forenses.

A tabela 5.5 contém a consolidação dos resultados do processamento de cópia distribuída de arquivos realizados neste cenário de testes.

Tabela 5.5 - Resultados do Processamento de Cópia Distribuída de Arquivos

| Parâmetro | Valor |
|---|---|
| Tempo de cópia | 06:57:05 (hh:mm:ss) |
| Quantidade de Arquivos Copiados | 2.274.784 |
| Quantidade de Arquivos *FileSet* Criados | 4.604 |

Neste segundo cenário de testes também foi realizada a indexação dos 4.604 arquivos *FileSet* criados. O processo de indexação realizado neste cenário de testes foi sem a realização do *merge* dos índices criados de forma distribuída. A figura 5.9 mostra a tela do detalhamento da atividade de indexação durante a execução do processo distribuído.

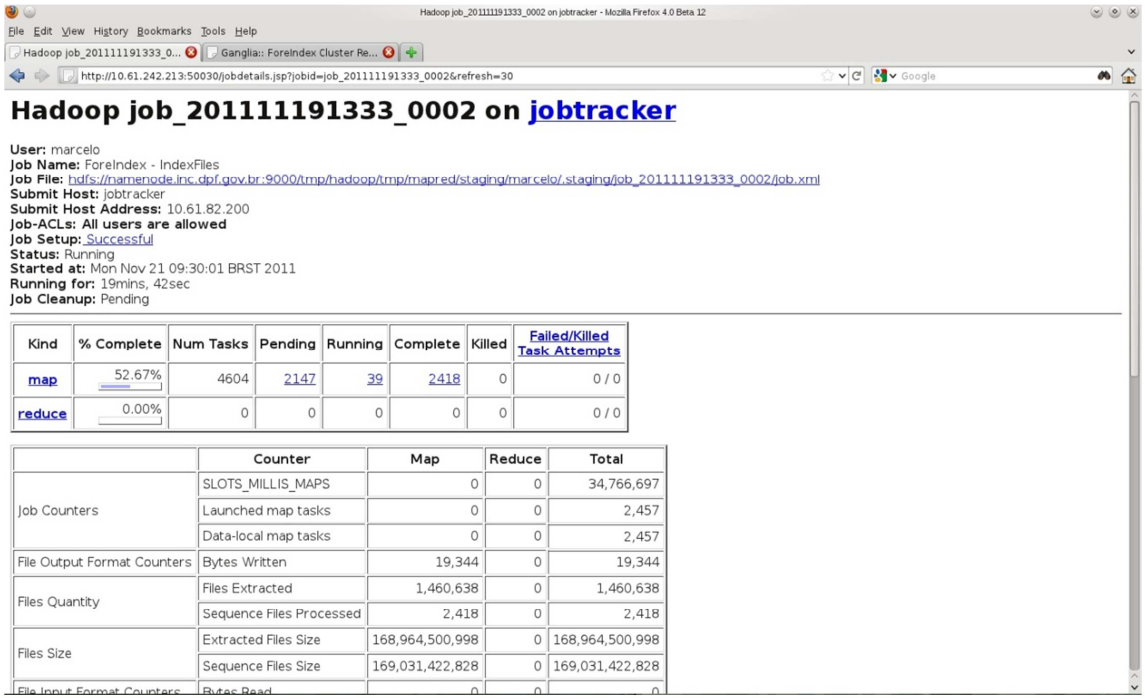
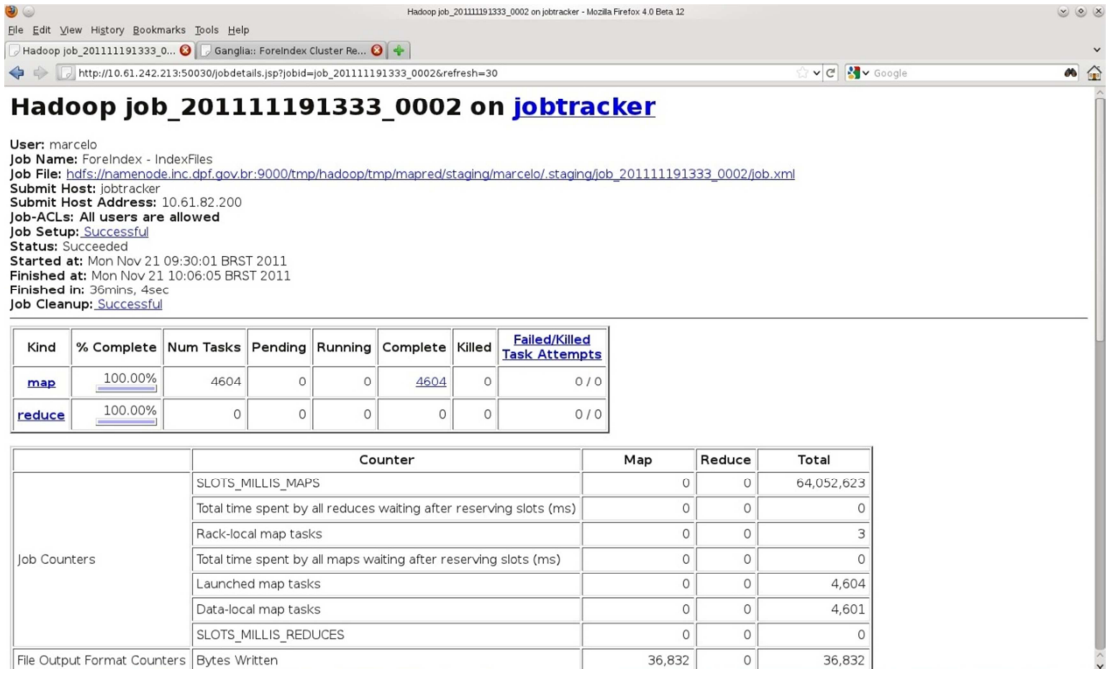



Figura 5.9 – Tela de Gerenciamento do *JobTracker* Detalhando a Indexação Distribuída

Nesta tela observa-se os seguintes dados: o número do trabalho submetido ao *JobTracker* é 201111191333_0002, que das 4.604 tarefas *Map*, já foram processadas 2418 tarefas (52,76% do trabalho), que estão sendo processadas 39 tarefas *Map* e que faltam 2.147

tarefas a serem processadas. As figuras 5.10 e 5.11 contêm detalhes do resultado do processamento da indexação distribuída sem merge de índices.



Hadoop job_201111191333_0002 on jobtracker - Mozilla Firefox 4.0 Beta 12

File Edit View History Bookmarks Tools Help

Hadoop job_201111191333_0... | Ganglia:: ForeIndex Cluster Re...

http://10.61.242.213:50030/jobdetails.jsp?jobid=job_201111191333_0002&refresh=30

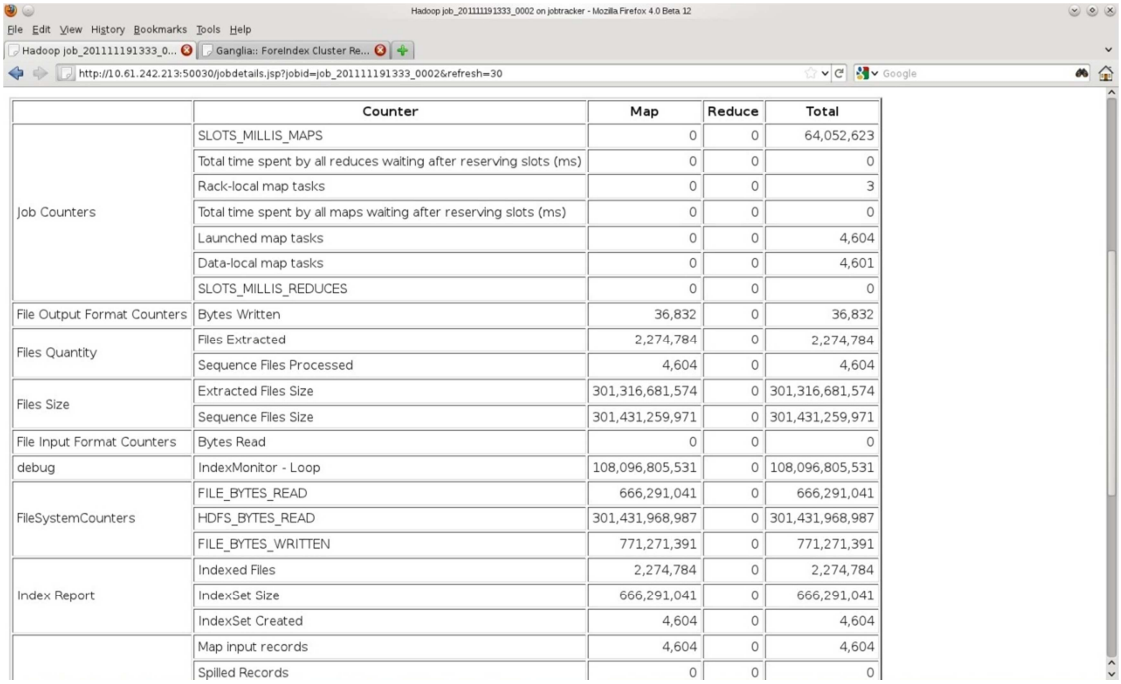
## Hadoop job_201111191333_0002 on jobtracker

**User:** marcelo
**Job Name:** ForeIndex - IndexFiles
**Job File:** hdfs://namenode.inc.dpf.gov.br:9000/tmp/hadoop/tmp/mapred/staging/marcelo/.staging/job_201111191333_0002/job.xml
**Submit Host:** jobtracker
**Submit Host Address:** 10.61.82.200
**Job-ACLs: All users are allowed**
**Job Setup:** Successful
**Status:** Succeeded
**Started at:** Mon Nov 21 09:30:01 BRST 2011
**Finished at:** Mon Nov 21 10:06:05 BRST 2011
**Finished in:** 36mins, 4sec
**Job Cleanup:** Successful

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 100.00% | 4604 | 0 | 0 | 4604 | 0 | 0 / 0 |
| reduce | 100.00% | 0 | 0 | 0 | 0 | 0 | 0 / 0 |

| | Counter | Map | Reduce | Total |
|---|---|---|---|---|
| Job Counters | SLOTS_MILLIS_MAPS | 0 | 0 | 64,052,623 |
| | Total time spent by all reduces waiting after reserving slots (ms) | 0 | 0 | 0 |
| | Rack-local map tasks | 0 | 0 | 3 |
| | Total time spent by all maps waiting after reserving slots (ms) | 0 | 0 | 0 |
| | Launched map tasks | 0 | 0 | 4,604 |
| | Data-local map tasks | 0 | 0 | 4,601 |
| | SLOTS_MILLIS_REDUCES | 0 | 0 | 0 |
| File Output Format Counters | Bytes Written | 36,832 | 0 | 36,832 |

Figura 5.10 – Tela de Gerenciamento do *JobTracker* Detalhando a Indexação Distribuída



Hadoop job_201111191333_0002 on jobtracker - Mozilla Firefox 4.0 Beta 12

File Edit View History Bookmarks Tools Help

Hadoop job_201111191333_0... | Ganglia:: ForeIndex Cluster Re...

http://10.61.242.213:50030/jobdetails.jsp?jobid=job_201111191333_0002&refresh=30

| | Counter | Map | Reduce | Total |
|---|---|---|---|---|
| Job Counters | SLOTS_MILLIS_MAPS | 0 | 0 | 64,052,623 |
| | Total time spent by all reduces waiting after reserving slots (ms) | 0 | 0 | 0 |
| | Rack-local map tasks | 0 | 0 | 3 |
| | Total time spent by all maps waiting after reserving slots (ms) | 0 | 0 | 0 |
| | Launched map tasks | 0 | 0 | 4,604 |
| | Data-local map tasks | 0 | 0 | 4,601 |
| | SLOTS_MILLIS_REDUCES | 0 | 0 | 0 |
| File Output Format Counters | Bytes Written | 36,832 | 0 | 36,832 |
| Files Quantity | Files Extracted | 2,274,784 | 0 | 2,274,784 |
| | Sequence Files Processed | 4,604 | 0 | 4,604 |
| Files Size | Extracted Files Size | 301,316,681,574 | 0 | 301,316,681,574 |
| | Sequence Files Size | 301,431,259,971 | 0 | 301,431,259,971 |
| File Input Format Counters | Bytes Read | 0 | 0 | 0 |
| debug | IndexMonitor - Loop | 108,096,805,531 | 0 | 108,096,805,531 |
| FileSystemCounters | FILE_BYTES_READ | 666,291,041 | 0 | 666,291,041 |
| | HDFS_BYTES_READ | 301,431,968,987 | 0 | 301,431,968,987 |
| | FILE_BYTES_WRITTEN | 771,271,391 | 0 | 771,271,391 |
| Index Report | Indexed Files | 2,274,784 | 0 | 2,274,784 |
| | IndexSet Size | 666,291,041 | 0 | 666,291,041 |
| | IndexSet Created | 4,604 | 0 | 4,604 |
| | Map input records | 4,604 | 0 | 4,604 |
| | Spilled Records | 0 | 0 | 0 |

Figura 5.11 – Tela de Gerenciamento do *JobTracker* Detalhando a Indexação Distribuída

Nos dados contidos nas figuras 5.10 e 5.11 verifica-se que o processo de indexação distribuída sem o merge de índices levou 00:36:04 (hh:mm:ss). Para realizar este processamento foram utilizadas 4.604 tarefas *Map*, que foram distribuídas para os 10 *workers* de processamento do cluster. O processo de indexação distribuído realizou a leitura dos 4.604 *FileSets* previamente criados no processo de cópia distribuída. Estes arquivos estavam disponíveis no HDFS. Das 4.604 tarefas *Map* 4.601 possuíam dados locais evidenciando a utilização da localidade de dados. Foram indexados 2.274.784 arquivos de dados forenses que estavam contidos nos 4.604 *FileSets*. Cada tarefa *map* criou um índice funcional dos arquivos indexados, resultando em um total de 4.604 arquivos do tipo *IndexSets* criados.

Durante a realização do processamento da indexação distribuída sem merge de índices foram coletadas algumas métricas, como a utilização do processador, memória, rede e disco. A figura 5.12 ilustra o agrupamento destas métricas.

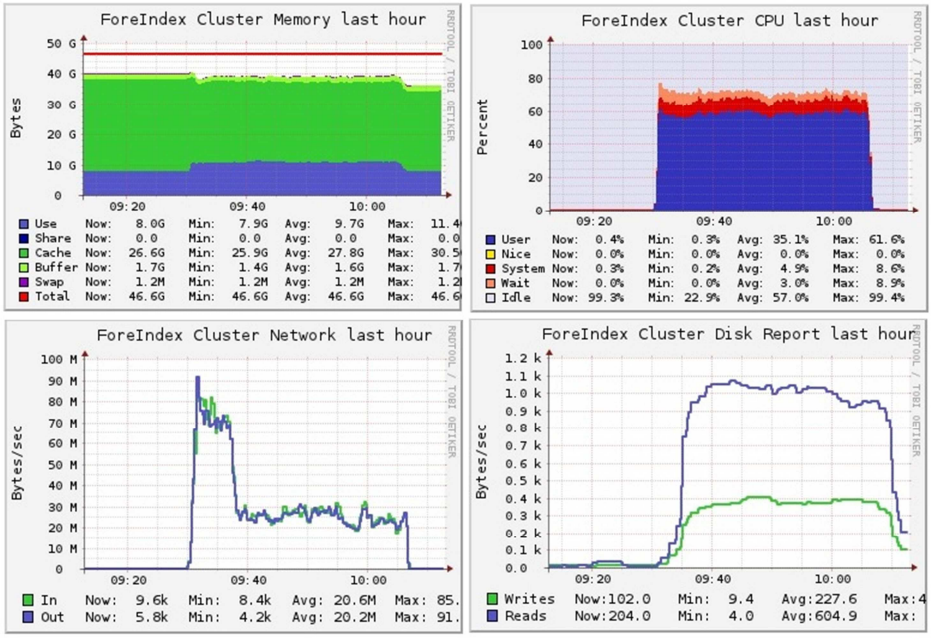



Figura 5.12 – Métricas de Memória, Processamento, Rede e Disco do Processo de Indexação Distribuída sem Merge de Índices

Da análise das métricas da figura 5.12 verifica-se que dos 48 GB de RAM disponíveis em todo o cluster houve um consumo médio de 10 GB de RAM com processos da indexação distribuída, o que corresponde a 20,83% da capacidade total do ambiente. Os processadores do cluster ficaram 100% do tempo com uma taxa de utilização de 60%. A

utilização da rede foi maior nos 10 primeiros minutos do processamento e constante no tempo restante, indicando pouco consumo de rede com uma taxa máxima de 90MB/s e uma média de 20MB/s de dados transmitidos e recebidos. A taxa de leitura dos dados foi aproximadamente o triplo da taxa de escrita, indicando a necessidade de um espaço de armazenamento de aproximadamente 33% do tamanho dos dados forenses para a criação dos índices. Estes dados indicam que a característica deste processo é predominantemente de processamento, com pouca utilização de memória e rede. A rede configurada atendeu bem a demanda de transferência de dados, que já foi reduzida devido à utilização da localidade de dados. Como o tempo gasto no processamento da tarefa de indexação distribuída é predominantemente determinado pela capacidade de processamento do ambiente, a utilização do cluster diminuiu consideravelmente o tempo gasto no processamento dos dados forenses.

As figuras 5.13, 5.14 e 5.15 mostram como foi a utilização de rede, processador e memória das máquinas do cluster durante o processamento da indexação distribuída sem o merge dos índices.

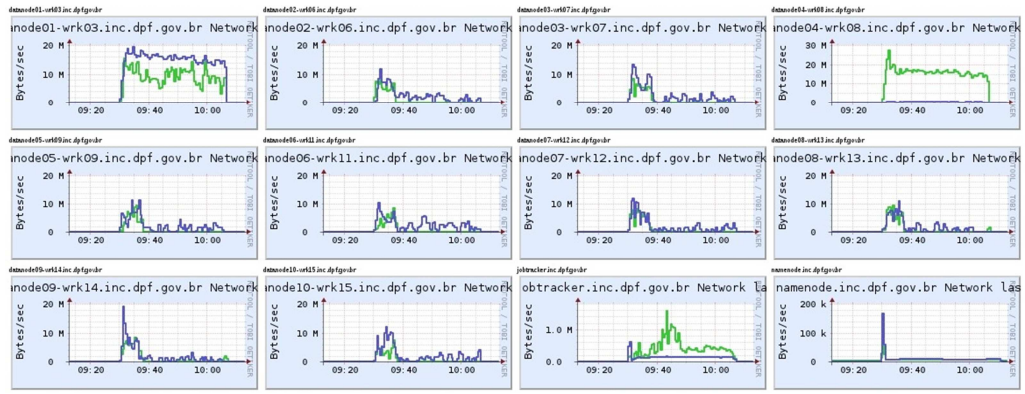



Figura 5.13 – Métrica de Utilização de Rede nas Máquinas do Cluster

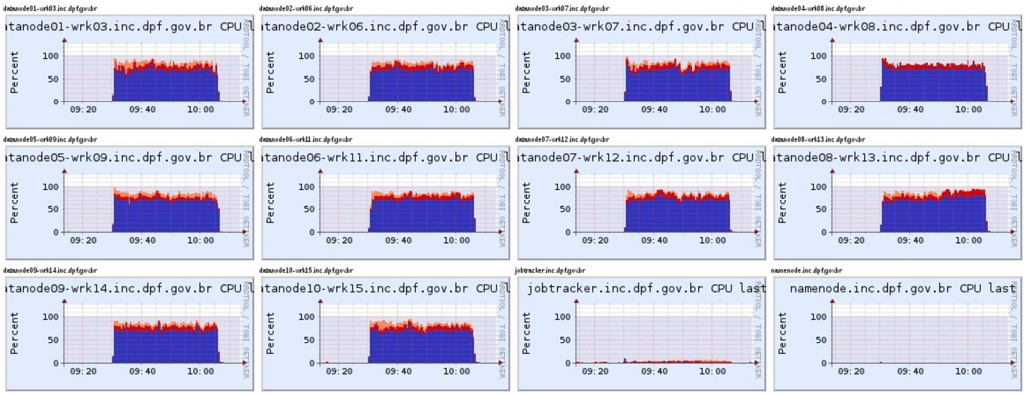



Figura 5.14 – Métrica de Utilização do Processador nas Máquinas do Cluster

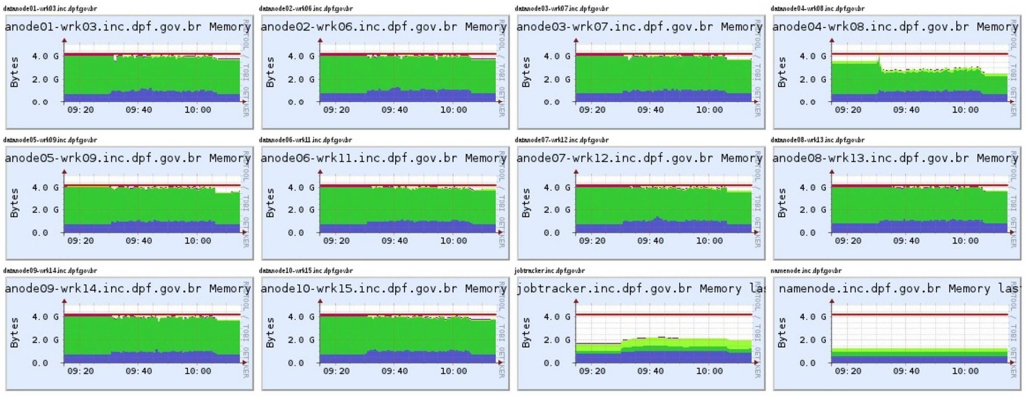



Figura 5.15 – Métrica de Utilização de Memória nas Máquinas do Cluster

Da análise geral das métricas de rede, processador e memória das máquinas do cluster verifica-se que os *workers* tiveram uma utilização similar destes recursos. O *JobTracker* e o *NameNode* tiveram pouca utilização de capacidade de rede e processamento, e um uso moderado de memória. Estes dados mostram que a capacidade de processamento, memória e rede do cluster são adequadas para os testes realizados.

A tabela 5.6 contém a consolidação dos resultados do processamento de indexação distribuída sem merge de índices realizado neste cenário de testes.

Tabela 5.6 - Resultados do Processamento de Indexação Distribuída sem Merge de Índices

| Parâmetro | Valor |
|---|---|
| Tempo de Processamento | 00:36:04 (hh:mm:ss) |
| Quantidade de *FileSets* Processados | 4.604 |
| Quantidade de Arquivos Indexados | 2.274.784 |
| Quantidade de Arquivos *IndexSet* Criados | 4.604 |
| Tamanho Total dos Índices Criados | 759 MB |

## 5.5. TERCEIRO CENÁRIO DE TESTE – CÓPIA E INDEXAÇÃO COMPOSTA, COM MERGE

Nesta seção será apresentado o cenário de teste elaborado para realizar o processo de indexação da massa de dados forenses selecionada utilizando-se 12 computadores, com processamento distribuído e realização do merge dos índices criados. Neste cenário não

será realizado o processo de cópia distribuída e o processo de indexação utilizará os arquivos gerados no processo de cópia distribuída ilustrado no primeiro cenário. O processo de indexação distribuída realizará o merge dos índices de acordo com o procedimento descrito na seção 4.4 do capítulo 4 – Arquitetura da Prova de Conceito.

A figura 5.16 ilustra a tela de gerenciamento do *JobTracker* durante a execução do trabalho de indexação distribuída com merge de índices. A tela presente na figura 5.16 mostra um sumário da configuração do cluster e a relação dos trabalhos que estão em execução neste ambiente. Com 10 *workers* com 04 processadores em cada, verifica-se que o cluster tem a capacidade de processar 40 tarefas simultaneamente. Destas 40 tarefas, 30 são tarefas *Map* e 10 são tarefas *Reduce*. No momento estão em execução 26 tarefas. Na relação de trabalhos em execução consta o trabalho número 201111220817_0001, que é o trabalho que realiza a indexação distribuída com merge de índices. Este processamento possui um total de 4.604 tarefas *Map* e 1 tarefa *Reduce*, sendo que 38 tarefas *Map* já foram executadas equivalendo a 1,12% do processamento total.

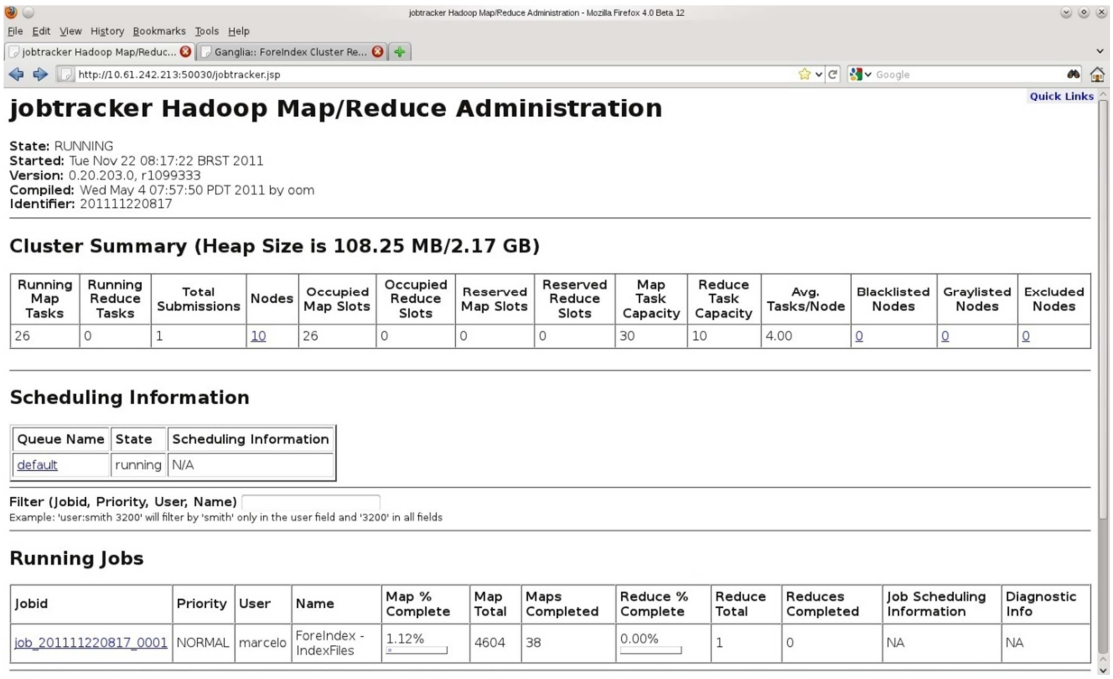



Figura 5.16 – Tela de Gerenciamento do *JobTracker* Durante a Indexação Distribuída com Merge de Índices

A figura 5.17 contém mais detalhes de outro momento da execução do processo de indexação.

Hadoop job_201111220817_0001 on jobtracker

**User:** marcelo
**Job Name:** ForeIndex - IndexFiles
**Job File:** hdfs://namenode.inc.dpf.gov.br:9000/tmp/hadoop/tmp/mapred/staging/marcelo/.staging/job_201111220817_0001/job.xml
**Submit Host:** jobtracker
**Submit Host Address:** 10.61.82.200
**Job-ACLs:** All users are allowed
**Job Setup:** Successful
**Status:** Running
**Started at:** Tue Nov 22 08:19:29 BRST 2011
**Running for:** 18mins, 52sec
**Job Cleanup:** Pending

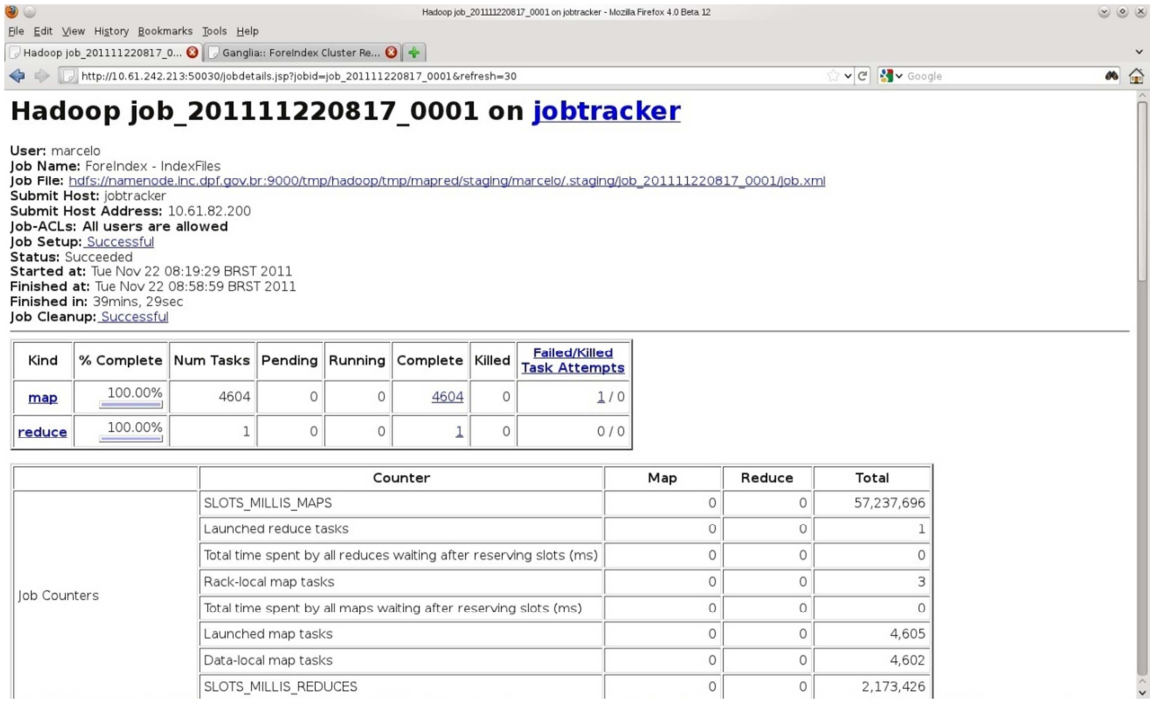
| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 50.12% | 4604 | 2284 | 30 | 2290 | 0 | 0 / 0 |
| reduce | 16.46% | 1 | 0 | 1 | 0 | 0 | 0 / 0 |

| | Counter | Map | Reduce | Total |
|---|---|---|---|---|
| Job Counters | SLOTS_MILLIS_MAPS | 0 | 0 | 29,440,357 |
| | Launched reduce tasks | 0 | 0 | 1 |
| | Launched map tasks | 0 | 0 | 2,320 |
| | Data-local map tasks | 0 | 0 | 2,320 |
| Files Quantity | Files Extracted | 1,427,205 | 0 | 1,427,205 |
| | Sequence Files Processed | 2,290 | 0 | 2,290 |
| Files Size | Extracted Files Size | 161,173,290,564 | 0 | 161,173,290,564 |
| | Sequence Files Size | 161,226,057,281 | 0 | 161,226,057,281 |

Figura 5.17 – Resultados Parciais da Indexação Distribuída com Merge de Índices

A figura 5.17 mostra a execução da atividade de indexação distribuída de arquivos. Nesta fase do processamento já foram processadas 2.290 tarefas *Maps*, das 4.604 a serem processadas, sendo que 30 estão em processamento. Isto equivale a 50,12% do total de tarefas *Maps* a serem processadas. Em paralelo ao processamento das tarefas *Maps*, a tarefa *Reduce* também está em execução, já tendo executado 16,46% de seu processamento. A tarefa *Reduce* é responsável por realizar o merge dos índices que vão sendo criados. Das tarefas *Maps* processadas e em processamento verifica-se que 2.320 possuem dados que foram encontrados no próprio *DataNode* onde o *TaskTracker* executa, confirmando assim a localidade de dados. No momento deste processamento (00:18:52 de execução), 2.290 *FileSets* já foram processados e 1.427.205 arquivos indexados.

A execução de todo o processo de indexação distribuída com o merge dos índices demorou 00:39:29 (hh:mm:ss). Os detalhes do resultado do processamento deste cenário de teste estão disponíveis nas telas de gerenciamento do *JobTracker* que estão ilustradas nas figuras 5.18, 5.19 e 5.20.



Hadoop job_201111220817_0001 on jobtracker - Mozilla Firefox 4.0 Beta 12

File Edit View History Bookmarks Tools Help

Hadoop job_201111220817_0... | Ganglia:: ForeIndex Cluster Re...

http://10.61.242.213:50030/jobdetails.jsp?jobid=job_201111220817_0001&refresh=30

**Hadoop job_201111220817_0001 on jobtracker**

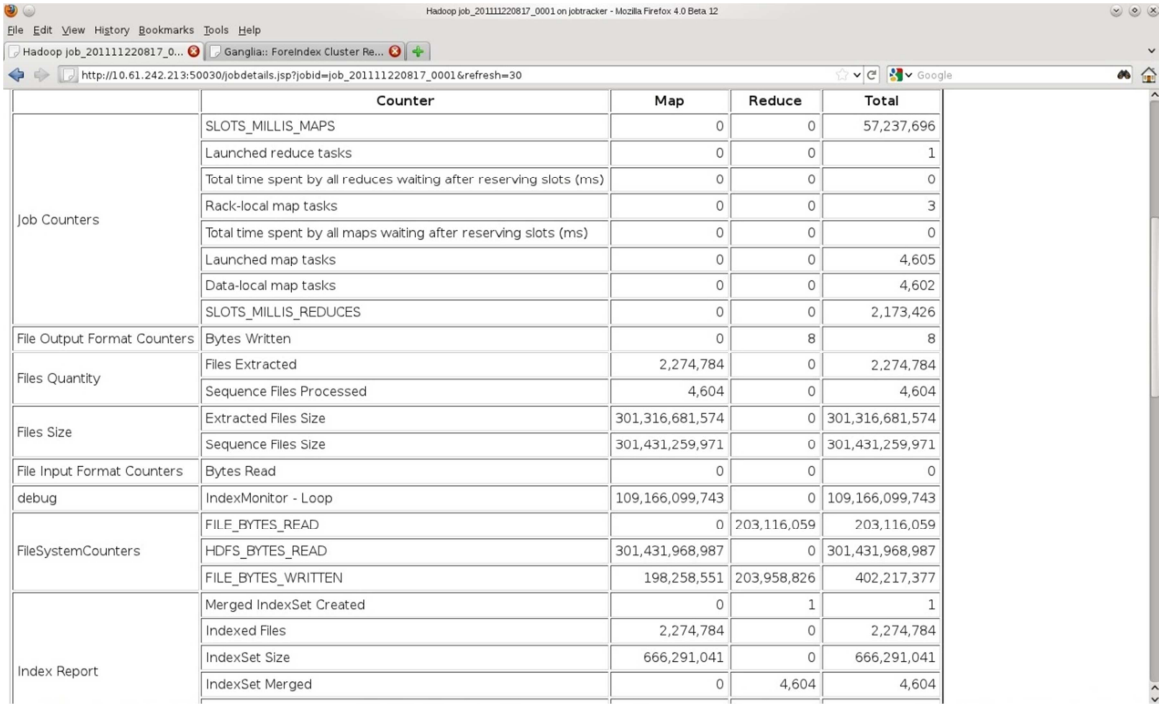
**User:** marcelo
**Job Name:** ForeIndex - IndexFiles
**Job File:** hdfs://namenode.lnc.dpf.gov.br:9000/tmp/hadoop/tmp/mapred/staging/marcelo/.staging/job_201111220817_0001/job.xml
**Submit Host:** jobtracker
**Submit Host Address:** 10.61.82.200
**Job-ACLs: All users are allowed**
**Job Setup:** Successful
**Status:** Succeeded
**Started at:** Tue Nov 22 08:19:29 BRST 2011
**Finished at:** Tue Nov 22 08:58:59 BRST 2011
**Finished in:** 39mins, 29sec
**Job Cleanup:** Successful

| Kind | % Complete | Num Tasks | Pending | Running | Complete | Killed | Failed/Killed Task Attempts |
|---|---|---|---|---|---|---|---|
| map | 100.00% | 4604 | 0 | 0 | 4604 | 0 | 1 / 0 |
| reduce | 100.00% | 1 | 0 | 0 | 1 | 0 | 0 / 0 |

| | Counter | Map | Reduce | Total |
|---|---|---|---|---|
| Job Counters | SLOTS_MILLIS_MAPS | 0 | 0 | 57,237,696 |
| | Launched reduce tasks | 0 | 0 | 1 |
| | Total time spent by all reduces waiting after reserving slots (ms) | 0 | 0 | 0 |
| | Rack-local map tasks | 0 | 0 | 3 |
| | Total time spent by all maps waiting after reserving slots (ms) | 0 | 0 | 0 |
| | Launched map tasks | 0 | 0 | 4,605 |
| | Data-local map tasks | 0 | 0 | 4,602 |
| | SLOTS_MILLIS_REDUCES | 0 | 0 | 2,173,426 |

Figura 5.18 – Tela de Gerenciamento do *JobTracker* Após a Indexação Distribuída com Merge de Índices

A figura 5.18 mostra que foram processadas 4.604 tarefas *Maps* corretamente. Destas tarefas *Maps* 4.602 possuíam dados locais o que demonstra a existência da localidade de dados na execução deste processo.



Hadoop job_201111220817_0001 on jobtracker - Mozilla Firefox 4.0 Beta 12

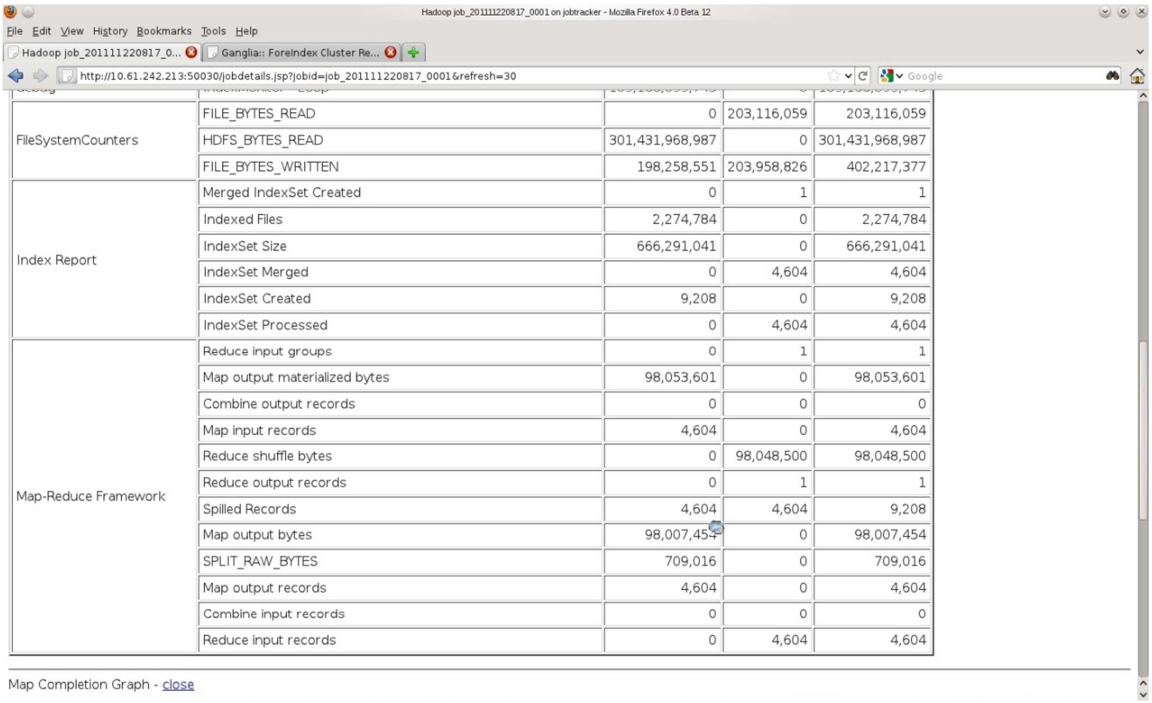
File Edit View History Bookmarks Tools Help

Hadoop job_201111220817_0... | Ganglia:: ForeIndex Cluster Re...

http://10.61.242.213:50030/jobdetails.jsp?jobid=job_201111220817_0001&refresh=30

| | Counter | Map | Reduce | Total |
|---|---|---|---|---|
| Job Counters | SLOTS_MILLIS_MAPS | 0 | 0 | 57,237,696 |
| | Launched reduce tasks | 0 | 0 | 1 |
| | Total time spent by all reduces waiting after reserving slots (ms) | 0 | 0 | 0 |
| | Rack-local map tasks | 0 | 0 | 3 |
| | Total time spent by all maps waiting after reserving slots (ms) | 0 | 0 | 0 |
| | Launched map tasks | 0 | 0 | 4,605 |
| | Data-local map tasks | 0 | 0 | 4,602 |
| | SLOTS_MILLIS_REDUCES | 0 | 0 | 2,173,426 |
| File Output Format Counters | Bytes Written | 0 | 8 | 8 |
| Files Quantity | Files Extracted | 2,274,784 | 0 | 2,274,784 |
| | Sequence Files Processed | 4,604 | 0 | 4,604 |
| Files Size | Extracted Files Size | 301,316,681,574 | 0 | 301,316,681,574 |
| | Sequence Files Size | 301,431,259,971 | 0 | 301,431,259,971 |
| File Input Format Counters | Bytes Read | 0 | 0 | 0 |
| debug | IndexMonitor - Loop | 109,166,099,743 | 0 | 109,166,099,743 |
| FileSystemCounters | FILE_BYTES_READ | 0 | 203,116,059 | 203,116,059 |
| | HDFS_BYTES_READ | 301,431,968,987 | 0 | 301,431,968,987 |
| | FILE_BYTES_WRITTEN | 198,258,551 | 203,958,826 | 402,217,377 |
| Index Report | Merged IndexSet Created | 0 | 1 | 1 |
| | Indexed Files | 2,274,784 | 0 | 2,274,784 |
| | IndexSet Size | 666,291,041 | 0 | 666,291,041 |
| | IndexSet Merged | 0 | 4,604 | 4,604 |

Figura 5.19 – Tela de Gerenciamento do *JobTracker* Após a Indexação Distribuída com Merge de Índices




| | | | | |
|---|---|---|---|---|
| FileSystemCounters | FILE_BYTES_READ | 0 | 203,116,059 | 203,116,059 |
| | HDFS_BYTES_READ | 301,431,968,987 | 0 | 301,431,968,987 |
| | FILE_BYTES_WRITTEN | 198,258,551 | 203,958,826 | 402,217,377 |
| Index Report | Merged IndexSet Created | 0 | 1 | 1 |
| | Indexed Files | 2,274,784 | 0 | 2,274,784 |
| | IndexSet Size | 666,291,041 | 0 | 666,291,041 |
| | IndexSet Merged | 0 | 4,604 | 4,604 |
| | IndexSet Created | 9,208 | 0 | 9,208 |
| | IndexSet Processed | 0 | 4,604 | 4,604 |
| Map-Reduce Framework | Reduce input groups | 0 | 1 | 1 |
| | Map output materialized bytes | 98,053,601 | 0 | 98,053,601 |
| | Combine output records | 0 | 0 | 0 |
| | Map input records | 4,604 | 0 | 4,604 |
| | Reduce shuffle bytes | 0 | 98,048,500 | 98,048,500 |
| | Reduce output records | 0 | 1 | 1 |
| | Spilled Records | 4,604 | 4,604 | 9,208 |
| | Map output bytes | 98,007,454 | 0 | 98,007,454 |
| | SPLIT_RAW_BYTES | 709,016 | 0 | 709,016 |
| | Map output records | 4,604 | 0 | 4,604 |
| | Combine input records | 0 | 0 | 0 |
| | Reduce input records | 0 | 4,604 | 4,604 |

Figura 5.20 – Tela de Gerenciamento do *JobTracker* Após a Indexação Distribuída com Merge de Índices

Através dos dados contidos nas figuras 5.18, 5.19 e 5.20 verifica-se que para realizar este processamento foram utilizadas 4.604 tarefas *Map*, que foram distribuídas para os 10 *workers* de processamento do cluster e uma tarefa *Reduce*. O processo de indexação distribuído realizou a leitura dos 4.604 *FileSets* previamente criados no processo de cópia distribuída. Estes arquivos estavam disponíveis no HDFS. Das 4.604 tarefas *Map*, 4.602 possuíam dados locais evidenciando a utilização da localidade de dados. Foram indexados 2.274.784 arquivos de dados forenses que estavam contidos nos 4.604 *FileSets*. Cada tarefa *Map* criou um índice funcional dos arquivos indexados, resultando em um total de 4.604 *IndexSets* criados. Na medida em que os *IndexSets* foram criados também foi realizado o *merge* dos mesmos na tarefa *Reduce*, resultando em um único *IndexSet* no final do processamento.

Durante a realização do processamento da indexação distribuída com merge de índices foram coletadas algumas métricas, como a utilização do processador, memória, rede e disco. A figura 5.21 ilustra o agrupamento destas métricas.

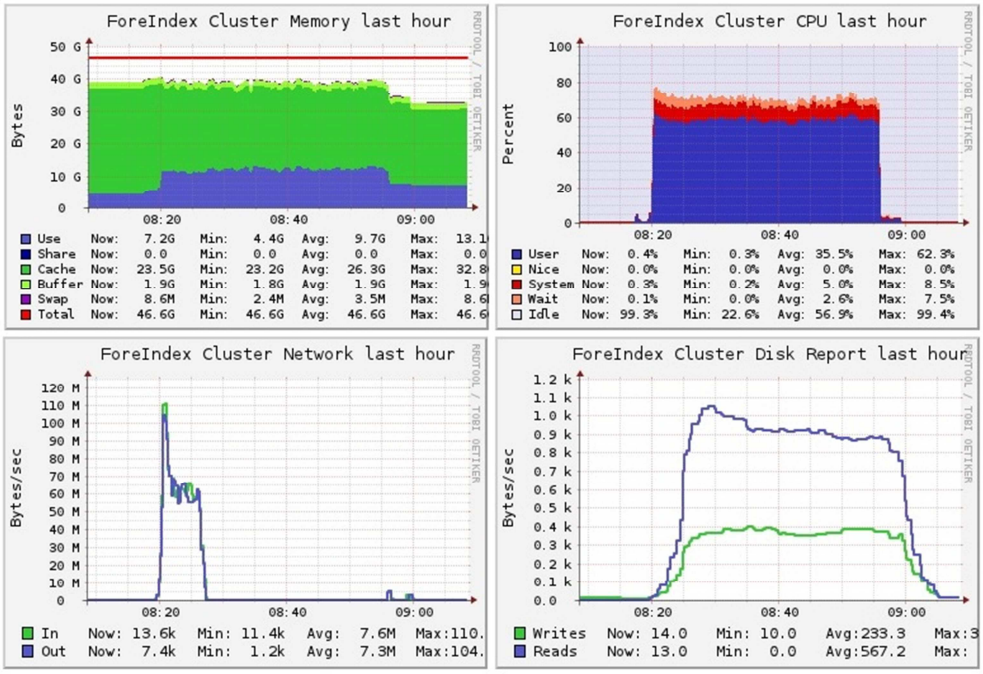



Figura 5.21 – Métricas de Memória, Processamento, Rede e Disco do Processo de Indexação Distribuída com a Realização de Merge de Índices

Da análise das métricas da figura 5.21 verifica-se que dos 48 GB de RAM disponíveis em todo o cluster houve um consumo médio de 10 GB de RAM com processos do usuário, o que corresponde a 20,83% da capacidade total do ambiente. Os processadores do cluster ficaram 100% do tempo com uma taxa de utilização de 65%. A utilização da rede foi maior nos 10 primeiros minutos do processamento e bem baixa no tempo restante, indicando pouco consumo de rede com uma taxa máxima de 118MB/s e uma média de 8MB/s de dados transmitidos e recebidos. A taxa de leitura dos dados foi aproximadamente o dobro da taxa de escrita. Estes dados indicam que a característica deste processo é predominantemente de processamento, com pouca utilização de memória e rede. A rede configurada atendeu bem a demanda de transferência de dados, que já foi reduzida devido à utilização da localidade de dados. Como o tempo gasto no processamento da tarefa de indexação distribuída é predominantemente determinado pela capacidade de processamento do ambiente, a utilização do cluster diminui consideravelmente o tempo gasto no processamento dos dados forenses. Como foi realizado o *merge* dos índices, na medida em que os mesmos foram sendo criados, a transmissão de dados na rede diminuiu consideravelmente, quando comparado com o processo de indexação sem merge, tendo em vista que apenas um único *IndexSet* foi transmitido no final do processo de indexação.

As figuras 5.22, 5.23 e 5.24 mostram como foi a utilização da memória, processador e rede das máquinas do cluster durante o processamento da indexação distribuída com o merge dos índices.

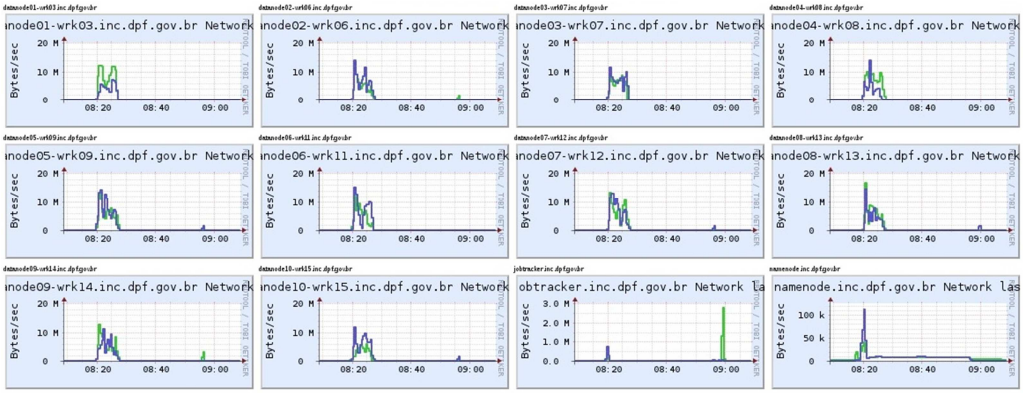



Figura 5.22 – Métrica de Utilização de Rede nas Máquinas do Cluster

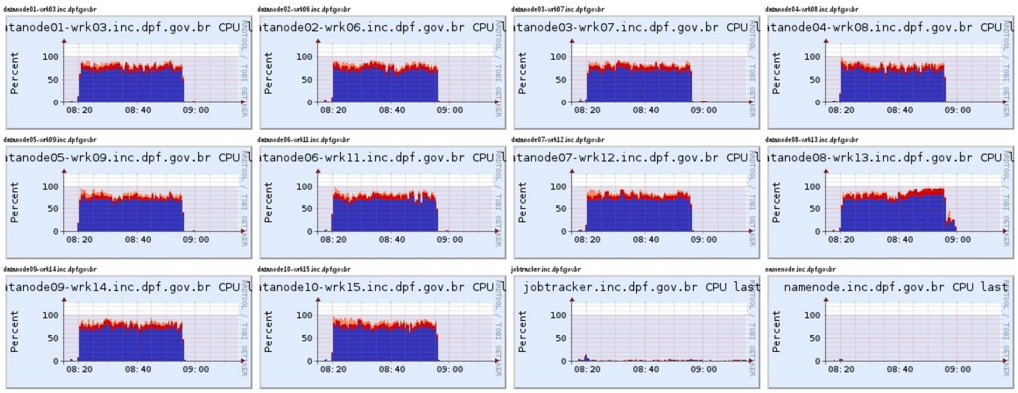



Figura 5.23 – Métrica de Utilização do Processador nas Máquinas do Cluster

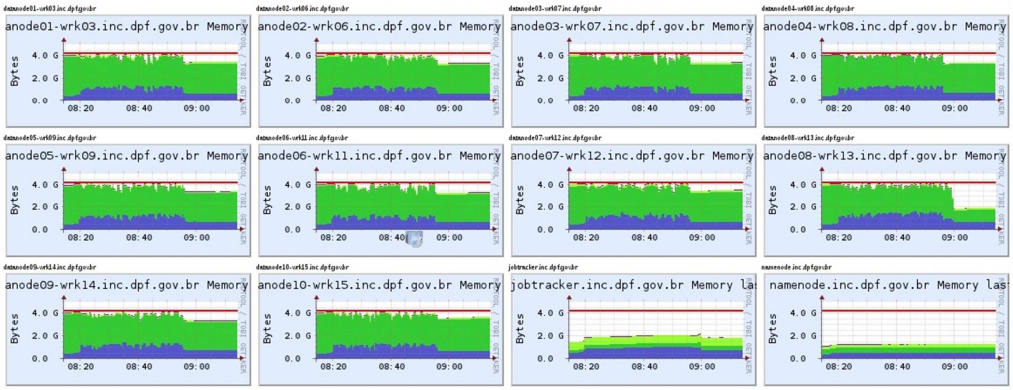



Figura 5.24 – Métrica de Utilização de Memória nas Máquinas do Cluster

Da análise geral das métricas de rede, processador e memória das máquinas do cluster verifica-se que os *workers* tiveram uma utilização similar destes recursos. O *JobTracker* e *NameNode* tiveram pouca utilização de capacidade de rede e processamento, e um uso

moderado de memória. Estes dados mostram que a capacidade de processamento, memória e rede do cluster são adequadas para os testes realizados.

A tabela 5.7 contém a consolidação dos resultados do processamento de indexação distribuída com merge de índices realizado neste cenário de testes.

Tabela 5.7 - Resultados do Processamento de Indexação Distribuída com Merge de Índices

| Parâmetro | Valor |
|---|---|
| Tempo de Processamento | 00:39:29 (hh:mm:ss) |
| Quantidade de *FileSets* Processados | 4.604 |
| Quantidade de Arquivos Indexados | 2.274.784 |
| Quantidade de Arquivos *IndexSet* Criados | 01 |
| Tamanho Total dos Índices Criados | 760 MB |

## 5.6. ANÁLISE COMPARATIVA DOS RESULTADOS OBTIDOS

Nesta seção será apresentada uma consolidação dos resultados obtidos em cada cenário de teste elaborado. Também será analisado o desempenho da pesquisa nos índices criados nos diferentes cenários de teste.

### 5.6.1. Desempenho da Pesquisa nos Índices Criados em Cada Cenário de Testes

Para a realização da pesquisa nos índices previamente criados foi utilizada a interface gráfica do protótipo desenvolvido para testar as funcionalidades da prova de conceito. As figuras 5.25 e 5.26 mostram telas da interface gráfica de pesquisa do protótipo.

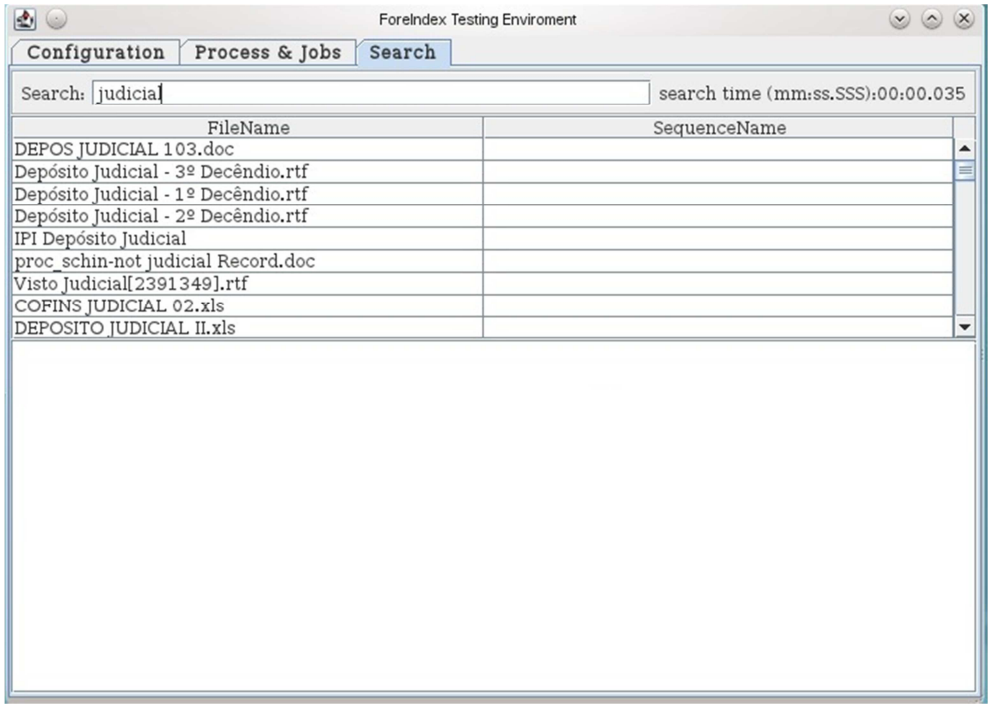



Figura 5.25 – Tela de Pesquisa do Protótipo

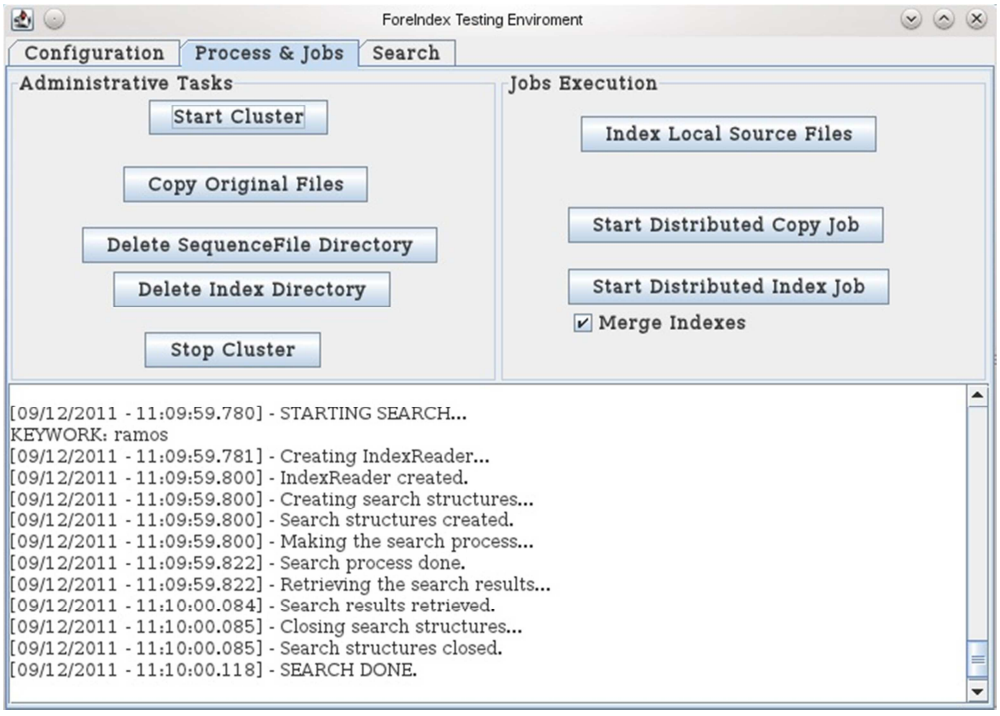



Figura 5.26 – Tela de Pesquisa do Protótipo

A figura 5.25 ilustra um simples exemplo de pesquisa, onde foi digitada a palavra "judicial" e foi mostrada uma tabela contendo a listagem de todos os arquivos recuperados. A pesquisa é realizada tanto no nome como no conteúdo e metadados dos arquivos, ou

seja, em todo conteúdo indexado. Nesta tela também conta o tempo gasto para realizar a pesquisa no índice, que no exemplo foi de 35 milissegundos. A figura 5.26 contém outra tela do protótipo da prova de conceito onde é possível visualizar o tempo gasto nas diversas fases utilizadas para realizar a pesquisa no índice em cada palavra pesquisada. Para medir o tempo gasto na pesquisa das palavras utilizadas neste cenário de teste será calculado o tempo gasto na inicialização das estruturas de dados de pesquisa e o tempo da busca no índice.

Neste cenário de testes foram realizados testes com cinco palavras e foram realizadas pesquisas em três índices. O primeiro índice, chamado de “Lucene_Local”, foi o índice criado no primeiro cenário de teste, com a indexação centralizada de dados com o indexador *Lucene*. O segundo índice, chamado de “Lucene_Sem_Merge”, foi o índice criado no segundo cenário de teste através da indexação distribuída sem o merge dos índices. O terceiro índice, chamado de “Lucene_Com_Merge”, é o índice criado no terceiro cenário de testes através da indexação distribuída de arquivos com o merge dos índices.

A tabela 5.8 contém o agrupamento do tempo gasto em cada pesquisa.

Tabela 5.8 – Resultado do Tempo Gasto em Pesquisas nos Índices Criados

| Índice | Palavra-Chave | Inicialização (ms) | Pesquisa (ms) |
|---|---|---|---|
| Lucene_Local | bravo | 66 | 21 |
| Lucene_Local | ramos | 24 | 11 |
| Lucene_Local | judicial | 17 | 21 |
| Lucene_Local | contas | 16 | 28 |
| Lucene_Local | geral | 18 | 16 |
| Lucene_Sem_Merge | bravo | 3038 | 101 |
| Lucene_Sem_Merge | ramos | 3451 | 135 |
| Lucene_Sem_Merge | judicial | 3761 | 123 |
| Lucene_Sem_Merge | contas | 3513 | 178 |
| Lucene_Sem_Merge | geral | 3561 | 248 |
| Lucene_Com_Merge | bravo | 283 | 52 |
| Lucene_Com_Merge | ramos | 151 | 15 |

| | | | |
|---|---|---|---|
| Lucene_Com_Merge | judicial | 176 | 12 |
| Lucene_Com_Merge | contas | 146 | 6 |
| Lucene_Com_Merge | geral | 160 | 5 |

Na tabela 5.8 a coluna “índice” contém o índice que foi utilizado para a realização da pesquisa, a coluna “palavra-chave” contém a palavra que foi pesquisada, a coluna “inicialização” contém o tempo gasto, em milissegundos, na inicialização das estruturas de dados responsáveis por realizar a pesquisa através da biblioteca *Lucene* e a coluna “pesquisa” contém o tempo que a biblioteca *Lucene* gasta para pesquisar os dados. O tempo de inicialização é calculado para cada pesquisa, contudo em uma aplicação de pesquisa real as estruturas de dados inicializadas podem ser reutilizadas para a realização de pesquisas enquanto o índice não for atualizado.

A figura 5.27 contém um gráfico representando os valores obtidos no tempo gasto com a inicialização das estruturas de dados dos três índices pesquisados. Observa-se que o tempo gasto na inicialização dos dados com o índice criado sem o merge é fortemente superior aos demais tempos. Isto é devido ao fato de existirem 4.604 sub-índices que foram criados e no momento da inicialização estes sub-índices são combinados em memória e pesquisados seqüencialmente.

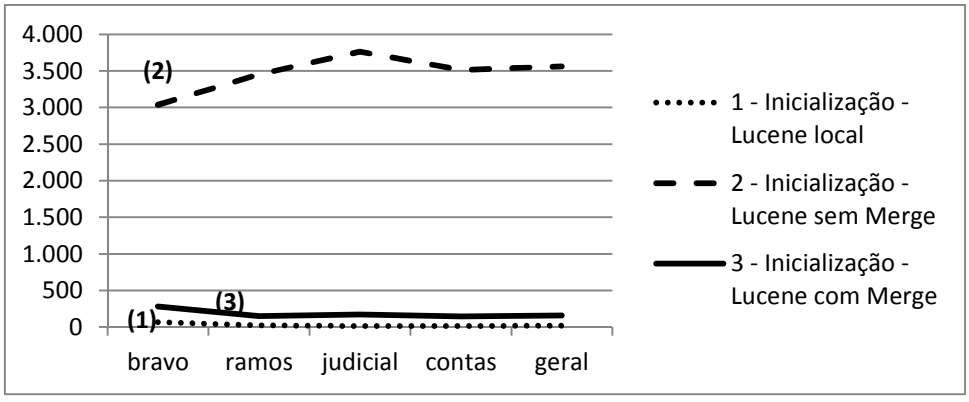



Figura 5.27 – Tempo de Inicialização das Estruturas de Pesquisa dos Índices

A figura 5.28 contém um gráfico representando os valores obtidos no tempo gasto na pesquisa das palavras-chave nos índices criados.

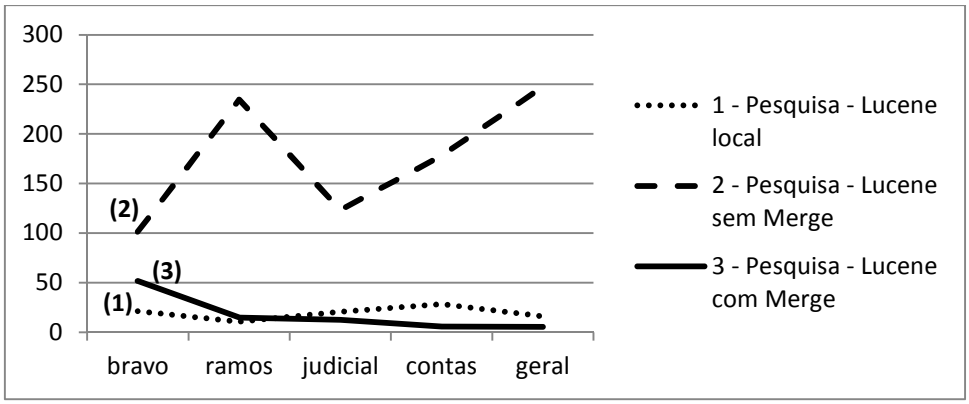



Figura 5.28 – Tempo de Pesquisa nos Índices Criados

A figura 5.28 mostra que o tempo gasto na pesquisa no índice criado de forma distribuída e sem o merge dos dados é fortemente superior ao tempo de pesquisa nos outros índices. Isto ocorre devido à busca sequencial que se faz necessária nos 4.604 sub-índices em cada palavra-chave pesquisada. A pesquisa com a utilização do índice local e o índice criado de forma distribuída com o merge apresentou resultados similares. Este resultado evidencia que o índice criado de forma distribuída com merge é funcional e apresenta bom desempenho.

### 5.6.2. Análise Comparativa do Tempo de Execução dos Cenários de Testes

A tabela 5.9 contém a relação do tempo de cópia e indexação dos processamentos realizados no primeiro, segundo e terceiro cenários de testes.

Tabela 5.9 – Tempos de Cópia e Indexação Obtidos nos Cenários de Teste

| Cenário de Teste | Tempo de Cópia (hh:mm:ss) | Tempo de Indexação (hh:mm:ss) |
|---|---|---|
| Primeiro (01 computador, Windows-FTK) | 17:28:35 | 21:50:55 |
| Primeiro (01 computador, Windows-EnCase) | 17:28:35 | 25:55:34 |
| Primeiro (01 computador, Linux-Lucene) | 08:01:28 | 04:21:17 |
| Segundo (12 computadores, sem Merge) | 06:57:05 | 00:36:04 |
| Terceiro (12 computadores, com Merge) | 06:57:05 | 00:39:29 |

A figura 5.29 contém um gráfico comparativo com o tempo de cópia no Windows e Linux utilizado a abordagem centralizada (primeiro cenário de testes) e o tempo de cópia no Linux utilizando a abordagem distribuída (segundo e terceiro cenários de testes).

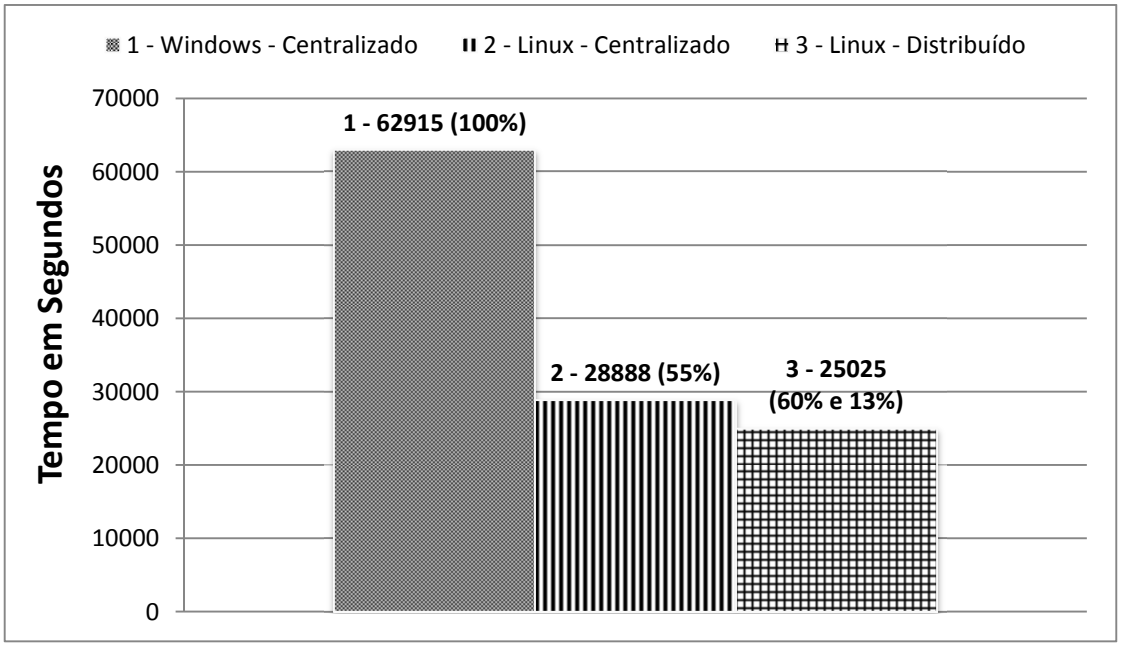



Figura 5.29 – Tempo de Cópia dos Dados Forenses

O gráfico ilustrado na figura 5.29 ilustra os tempos de cópia dos dados forenses obtidos nos cenários de testes realizados. O tempo de cópia foi representado em segundos para facilitar a comparação. O processo de cópia de 2.274.784 arquivos é uma atividade por si onerosa tendo em vista que estes arquivos estão dispersos em diversas áreas do disco rígido, sendo necessário milhares de operações de posicionamento do braço de leitura dos discos rígidos.

Para a cópia no Windows e Linux foram utilizados os recursos de cópias de arquivos disponíveis no sistema operacional. Mesmo assim o tempo de cópia dos arquivos no Linux foi 54,08% inferior ao tempo de cópia centralizada no Windows.

O tempo de cópia obtido no processo de cópia distribuída da prova de conceito foi 60,22% inferior ao tempo de cópia obtido na abordagem centralizada do Windows e 13,37% inferior ao tempo de cópia da abordagem centralizada do Linux. Conforme descrito no capítulo 4, o processo de cópia distribuída realiza diversas outras atividades de processamento, além da cópia de dados em si. Exemplos destas atividades são: verificação

do tamanho dos arquivos, agrupamento dos arquivos em arquivos *FileSets*, cópia dos arquivos no sistema de arquivos distribuído HDFS e replicação de cada bloco de dados para outras duas máquinas (de acordo com o fator de replicação definido).

Mesmo com todas estas atividades adicionais o paralelismo propiciou uma considerável diminuição do tempo. A diminuição do tempo de cópia só não foi maior pois ficou dependente da velocidade de leitura do disco de origem dos dados, conforme demonstrado no segundo cenário de testes.

A figura 5.30 contém um gráfico comparativo com o tempo de indexação centralizada no FTK, *EnCase* e *Lucene* (primeiro cenário de testes), além do tempo de indexação distribuída com e sem o merge dos índices (segundo e terceiro cenário de testes).

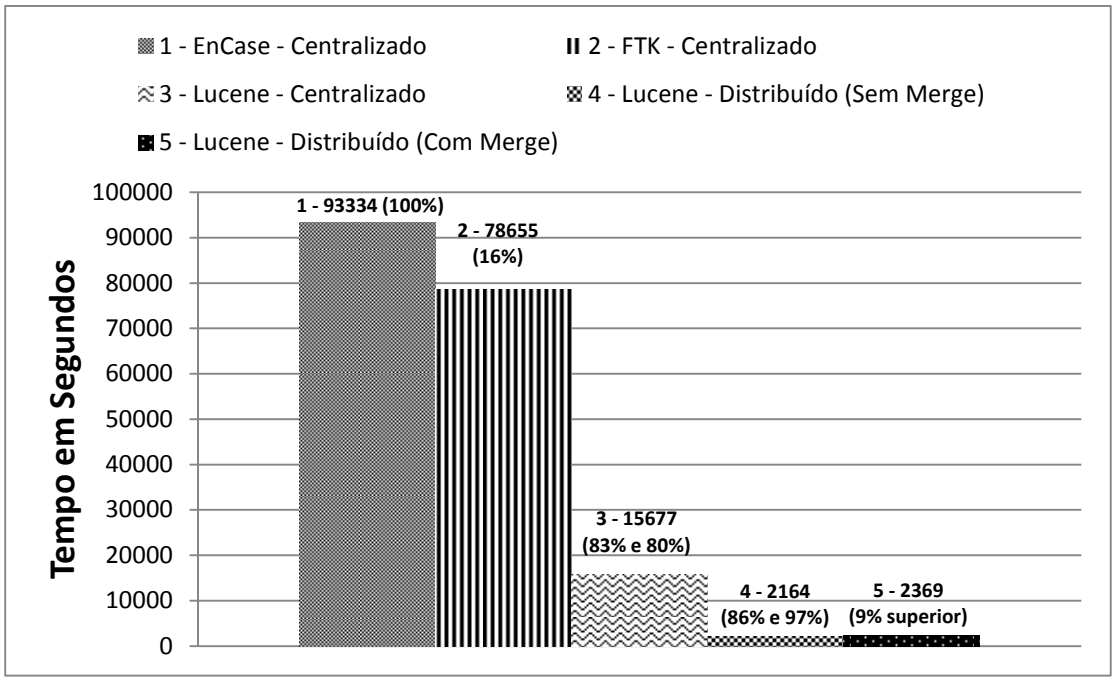



Figura 5.30 – Tempo de Indexação dos Dados Forenses

No gráfico da figura 5.30 o tempo de indexação foi exposto em segundos para facilitar a comparação dos resultados. Conforme demonstra o gráfico o tempo de indexação do *Lucene* é consideravelmente inferior ao tempo de indexação do FTK e *EnCase* na abordagem centralizada. O tempo de indexação centralizada no *Lucene* foi 83.20% inferior ao tempo de indexação no *EnCase* e 80,26% inferior ao tempo de indexação no FTK. De acordo com o observado nas métricas do segundo e terceiro cenários de testes constou que

a atividade de indexação tem um forte ganho de desempenho com o acréscimo do poder de processamento.

Desta forma, a abordagem distribuída proporcionou consideráveis ganhos de desempenho mesmo quando comparada com a indexação centralizada do *Lucene*. O tempo de indexação distribuído sem o merge dos índices foi 86,19% inferior ao tempo de indexação centralizada do *Lucene*, e com todo o processamento do merge dos índices foi 84,88% inferior à abordagem centralizada do *Lucene*. Se comparar a abordagem distribuída do *Lucene* em relação à abordagem centralizada do FTK e *EnCase* a diminuição ainda é maior, obtendo um tempo 96,99% inferior ao FTK e 97,46% inferior ao *EnCase*, no caso da indexação distribuída com o merge de índices.

Conforme exposto na subseção anterior, a abordagem da indexação distribuída com o merge de índices é desejável ao invés de sem o merge de índices, devido ao aumento considerável do desempenho na pesquisa e o pequeno acréscimo no tempo de indexação que corresponde a apenas 8,65% a mais do tempo da abordagem sem o merge de índices.

A tabela 5.10 contém a relação do tempo de cópia somado ao tempo de indexação dos processamentos realizados no primeiro, segundo e terceiro cenários de testes.

Tabela 5.10 – Tempo de Cópia Somado ao de Indexação dos Cenários de Teste

| Cenário de Teste | Tempo de Cópia e Indexação (hh:mm:ss) |
|---|---|
| Primeiro (01 computador, Windows-FTK) | 39:19:40 |
| Primeiro (01 computador, Windows-EnCase) | 43:34:09 |
| Primeiro (01 computador, Linux-Lucene) | 12:22:45 |
| Segundo (12 computadores, sem Merge) | 07:43:09 |
| Terceiro (12 computadores, com Merge) | 07:46:34 |

A figura 5.31 contém um gráfico comparativo com o tempo total de cópia e indexação (em segundos) de todos os testes realizados para a verificação de desempenho de todo o processo.

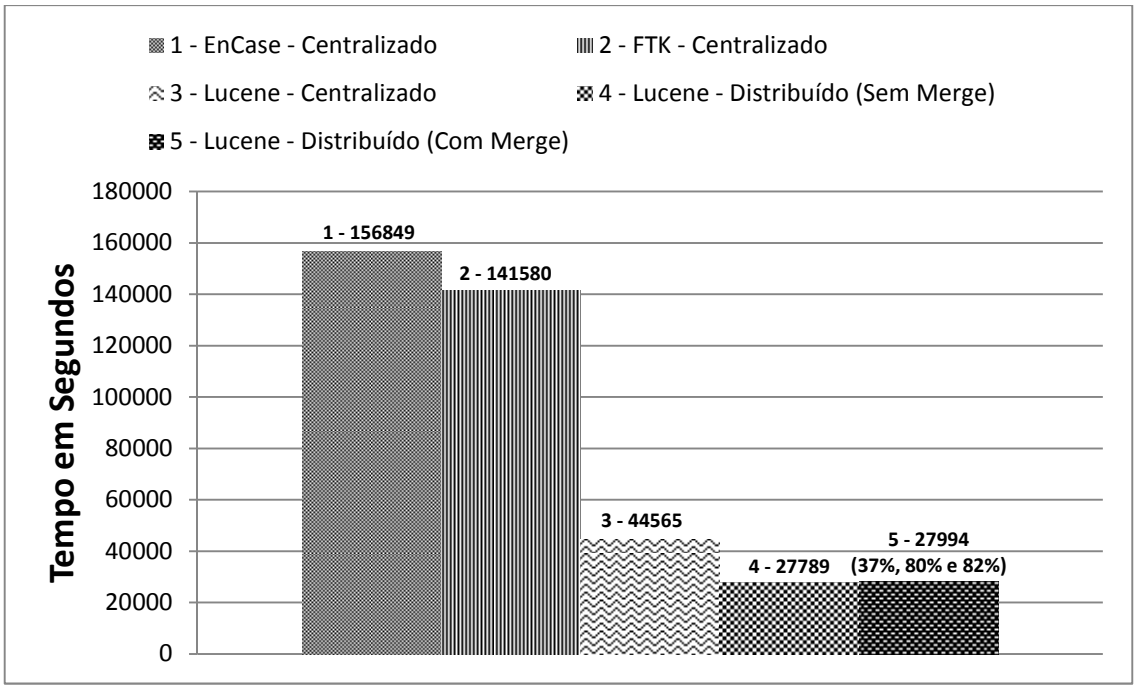



Figura 5.31 – Tempo de Cópia e Indexação dos Dados Forenses

Pelas vantagens de desempenho já expostas o processo de indexação distribuído com merge é o mais adequado levando em consideração o desempenho nos tempos de indexação e pesquisa. De acordo com o exposto na figura 5.31 verifica-se que o tempo total de cópia e indexação da abordagem distribuída com o merge de índices foi 37,18% inferior à abordagem centralizada do *Lucene*, 80,22% inferior à abordagem centralizada do FTK e 82,15% inferior à abordagem centralizada do *EnCase*. Isto evidencia que as abordagens utilizadas pela prova de conceito proporcionaram um considerável ganho de desempenho no processo como um todo.

# 6. CONCLUSÕES

O presente trabalho utilizou a computação distribuída na tentativa de melhorar o desempenho do processo de indexação e possibilitar um recurso com escalabilidade, disponibilidade e tolerância a falhas para armazenar os dados forenses. Obteve-se um resultado satisfatório com um importante ganho no desempenho da indexação e no fornecimento de um meio de armazenamento adequado para os dados forenses.

Foram elaborados cenários para poder realizar testes de desempenho com a prova de conceito criada. Nestes cenários foram analisados o desempenho nos processos de cópia e indexação distribuídos. Nesta análise também foi realizado um comparativo com outras ferramentas que são referenciais de excelência em análise forense. Dentre os resultados obtidos com os testes realizados, verificou-se que a prova de conceito apresentou um desempenho 37,18% superior à mesma abordagem centralizada e até 82,15% em relação a abordagem centralizada de ferramenta de referência em análise forense. Também foram encontrados resultados relevantes nas buscas indexadas com um tempo médio de 25 milissegundos de busca no índice criado através do processo de indexação distribuída com *merge*.

Estes resultados são relevantes pois demonstram que através da utilização de algoritmos e ferramentas que implementam recursos de recuperação da informação, programação paralela, computação distribuída e informática forense é possível criar um ambiente com desempenho adequado para processar e armazenar um grande volume de dados forenses. A utilização apenas de ferramentas com código-fonte aberto e que não necessita de recursos computacionais tão robustos diminui o custo desta solução. O compartilhamento dos recursos computacionais, com a utilização de modernos algoritmos de programação paralela e ferramentas de computação distribuída, forneceram um ambiente com tolerância a falhas, balanceamento de carga, escalabilidade e disponibilidade, além do bom desempenho no processo de indexação.

A arquitetura e os resultados preliminares deste projeto de pesquisa foram apresentados oralmente no evento *The Sleuth Kit and Open Source Digital Forensics Conference*, realizado em Virgínia/Estados Unidos em 14/06/2011 e publicado nos anais da conferência sob o título “*ForeIndex: A Framework for Analysis and Triage of Data Forensics”*. Foi

realizada uma apresentação oral deste trabalho no 3º Colóquio de Arquitetura da Informação, realizado pelo CPAI/UnB/Brasília/Brasil em 15/07/2011, sob o título “Um Sistema Distribuído para o Armazenamento e Indexação de Dados Forenses”. Este trabalho também foi apresentado oralmente na 6ª Conferência Internacional em Ciência Forense Computacional , realizado em Florianópolis/Brasil em 06/10/2011, sendo publicado no *The International Journal of Forensic Computer Science,* Volume 6, Number 1, 2011, ISBN: 978-65069-05-2, com o título “Utilização da Computação Distribuída para o Armazenamento e Indexação de Dados Forenses”.

Alguns possíveis cenários de testes não puderam ser realizados no presente trabalho e ficam como sugestões para trabalhos futuros. Estes cenários são: realização de testes com indexação distribuída em um cluster com maior volume de máquinas onde os processos distribuídos serão testados com diferentes números de máquinas visando a obtenção de parâmetros de escalabilidade; testes de pesquisas com os índices estando armazenados no sistema HDFS para realizar uma análise comparativa com os índices armazenados em um sistema de arquivos não-distribuído; a realização do processamento distribuído com diferentes sistemas de arquivos distribuídos para analisar as particulares de cada sistema e a realização de analises comparativas utilizando uma ferramenta comercial para processamento distribuído de evidencias, como o FTK Lab. Um outro possível trabalho futuro é a criação de um mecanismo de redundância para o *JobTracker*, podendo ser através de um recurso similar ao utilizado pela ferramenta *Apache Katta* que usa o *Apache ZooKeeper* para implementar esta funcionalidade.

O trabalho elaborado possibilita que, posteriormente, possam ser desenvolvidas 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, através da utilização do sistema distribuído apresentado neste trabalho. Os recursos apresentados neste trabalho possibilitam uma análise correlacionada de dados forenses pela autoridade investigativa em um caso resultante do processamento de diversos dispositivos de armazenamento computacional. O desempenho do processo investigativo e de produção de provas, adicionado à análise correlacionada de evidências, resulta em maior qualidade e celeridade no processo de apuração de uma infração penal, trazendo benefícios para a sociedade.

# REFERÊNCIAS BIBLIOGRÁFICAS


AccessData. (2011). *Site Oficial do FTK*. Acesso em 19 de 07 de 2011, disponível em http://accessdata.com/products/computer-forensics/ftk

*AD Lab*. (2011). *Site Oficial do Projeto AccessData Lab*. Acesso em 19 de 07 de 2011, disponível em http://accessdata.com/products/computer-forensics/lab

*Ad Lab Architecture.* (2011). *Relatório Técnico com a Arquitetura do AD Lab*. Acesso em 19 de 07 de 2011, disponível em http://accessdata.com/downloads/media/DivideandConquer-OvercomingForensicsBacklog.pdf

*Aduna Software*. (2011). *Site Oficial do Projeto Auto-Focus*. Acesso em 19 de 07 de 2011, disponível em http://www.aduna-software.com/

AEB. (2011). *Agência Espacial Brasileira*. Acesso em 10 de 07 de 2011, disponível em Site Oficial da AEB: http://www.aeb.gov.br/

Aires, J. P., & Vaz, M. S. (2007). Modelo de Indexação e Recuperação de Dados Distribuídos. *Escola Regional de Informática - Paraná [ERIPR - XIV]*. Guarapuava, Paraná: SBC.

*Amazon EC2*. (2011). *Site Oficial do Projeto Amazon EC2*. Acesso em 19 de 07 de 2011, disponível em http://aws.amazon.com/pt/ec2//185-9523022-9896517/

*Amazon S3*. (2011). *Site Oficial do Sistema de Arquivos S3*. Acesso em 19 de 07 de 2011, disponível em http://aws.amazon.com/pt/s3/

Anderson, E., & Patterson, D. A. (1997). Extensible, Scalable Monitoring for Clusters of Computers. *Proceedings of the 11th Systems Administration Conference*, (pp. 9-16). California, EUA.

Anderson, R. (2008). *Security Engineering - A Guide to Building Dependable Distributed Systems.* Indianopolis, EUA: Wiley.

*Aperture*. (2011). Fonte: Site Oficial da Ferramenta Aperture: http://aperture.sourceforge.net/

*Apple Spotlight.* (2011). *Site Oficial da Apple*. Acesso em 12 de 11 de 2011, disponível em http://www.apple.com/br/xsan/features/spotlight.html

Arpaci-Dusseau, R. H., Anderson, E., & Treuh, N. (1999). Cluster I/O with River: Making the Fast Case Common. *Sixth Workshop on Input/Output in Parallel and Distributed Systems (IOPADS '99)*, (pp. 10-22). Atlanta, Georgia, Estados Unidos.

*Ateji PX for Java.* (2011). *Site Oficial do Projeto Ateji PX*. Acesso em 19 de 07 de 2011, disponível em: http://www.ateji.com/px/whitepapers/Ateji PX for Java v1.0.pdf

*Axum.* (2011). *Site Oficial do Axum*. Acesso em 19 de 09 de 2011, disponível em http://www.microsoft.com/download/en/details.aspx?id=21024

Bergeron, J. (2003). *Writing Testbenches: Functional Verification of HDL Models.* Califórnia: Springer.

Bernam, F., Fox, G. C., & Hey, A. J. (2005). *Grid Computing - Making the Global Infrastructure a Reality.* Nova Jersey, Estados Unidos: Wiley.

Bezerra, C. C. (2011). *Relatório Estatístico de Atividades do Sistema Nacional de Criminalística.* Brasília/DF: Diretoria Técnico-Científica, Polícia Federal Brasileira.

Blelloch, G. E. (1996). *Programming Parallel Algorithms.* Pensilvânia, EUA: ACM Press.

Blelloch, G. E. (1989). *Scans as Primitive Parallel Operations.* IEEE Transactions on Computers.

*BOINC*. (2011). *Site Oficial do Projeto BOINC*. Acesso em 22 de 08 de 2011, disponível em http://boinc.berkeley.edu/

Boldi, Paolo, & Vigna, S. (2004). Codes for the World-Wide Web. *Internet Mathematics* , 405-407.

Bolosky, W., & Fitzgerald, R. (1996). The Tiger video fileserver. *6th NOSSDAV Conference.* Zushi, Japão.

Bond, A. B. (2000). Characteristics of scalability and their impact on performance. *Proceedings of the 2nd international workshop on Software and performance*, (pp. 195-203). Ontario, Canadá: ACM Press.

Bray, J., & Sturman, C. F. (2002). *Bluetooth: Connect Without Cables.* Prentice Hall.

Butler, M. H., & Rutherford, J. (2008). Distributed Lucene: A distributed free text index for Hadoop. *HP Laboratories.*

Büttcher, S., Clarke, C. L., & Cormack, G. V. (2010). *Information Retrieval - Implementing and Evaluating Search Engines.* Waterloo: MIT Press.

Buyya, R. (2000). Parmon: a portable and scalable monitoring system for clusters. *Software - Practica and Experience, nº 30* , pp. 1-17.

Buyya, R., Broberg, J., & Goscinski, A. M. (2011). *Cloud Computing: Principles and Paradigms.* Nova Jersey, Estados Unidos: Wiley.

Callaghan, B. (1999). *NFS Illustrated.* Mariland, EUA: Addison-Wesley.

Campbell. (1993). *Managinf AFS: The Andrew File System.* Califórnia, EUA: Prentice-Hall.

Carrier, B. (2005). *File System Forensic Analysis.* Boston, Estados Unidos: Addison Wesley.

Chen, L. T., & Bairagi, D. (2010). *Developing Parallel Programs - A Discussion of Popular Models.* Califórnia, EUA: Oracle White Paper.

*Copernic*. (2011). *Site Oficial do Projeto Copernic*. Acesso em 10 de 07 de 2011, disponível em http://www.copernic.com/

Coulouris, G., Dollimore, J., & Kindberg, T. (2007). *Sistemas Distribuídos - Conceitos e Projeto.* São Paulo/SP: Bookamn.

Date, C. J. (2004). *Introdução a Sistemas de Banco de Dados.* São Paulo: Campus.

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

DPF, S. (2010). *Apostila de Informática Forense.* Brasília: Academina Nacional de Polícia. *dtSearch*. (10 de 10 de 2011). Fonte: Site Oficial da Ferramenta dtSearch: http://www.dtsearch.com/

Eleutério, P. M., & Machado, M. P. (2011). *Desvendando a Computação Forense.* São Paulo: Novatec.

*EnCase*. (2011). *Site Oficial da Ferramenta EnCase*. Acesso em 12 de 10 de 2011, disponível em http://www.guidancesoftware.com/forensic.htm

Eoghan, C. (2011). *Digital Evidence and Computer Crime: Ciência Forense, Computadores e a Internet.* Reino Unido-GB: Elsevier Academic Press.

Espindula, A., Jesus, A. V., & Geiser, G. C. (2011). *Ciências Forenses - Uma Introdução Às Principais Áreas da Criminalística Moderna.* São Paulo: Millennium.

FBI. (2011). *Handbook of Forensic Services.* Fonte: Site Oficial do FBI. Acesso em 10 de 10 de 2011, disponível em: http://www.fbi.gov/about-us/lab/forensic-science-communications/fsc/oct2000/handbook.htm/

Ferreira, A. J. (2009). *MATLAB Codes for Finite Element Analysis.* Califórnia, EUA: Springer.

Friedl, J. (2006). *Mastering Regular Expressions.* Califórnia: O'Reilly Media.

FTK. (2011). *Forensics Toolkit. User Guide.* Oren, Utah, EUA: AccessData.

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

*Ganglia*. (2011). *Site Oficial da Ferramenta Ganglia*. Acesso em 10 de 10 de 2011, disponível em http://ganglia.sourceforge.net/

Geetha, E., Naidu, K., & Kuma, S. (2008). Simulative Performance Evaluation of Information Retrieval Systems. *Second Asia International Conference on Modelling Simulation* (pp. 297-302). Bangalore, Índia: IEEE Computer Society.

Ghemawat, S., Gobioff, H., & Leun, S.-T. (2003). The Google File System. *9th Symposium on Operating Principles*, (pp. 29-43). Nova York, Estados Unidos.

Google. (2011). *Site Oficial do Google Brasil*. Acesso em 12 de 07 de 2011, disponível em http://www.google.com.br

Gorlatch. (1996). Systematic Eficient Parallelization of Scan. *Euro-Par'96 Parallel Processing* (pp. 401-408). Lion, França: Lecture Notes in Computer Science.

*Greenplum*. (2011). *Site Oficial da Greenplum*. Acesso em 12 de 10 de 2011, disponível em http://www.greenplum.com/technology/mapreduce

Gropp, W., Lusk, E., & Skjellum, A. (1999). *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* Cambridge, MA, Estados Unidos: MIT Pres.

Grossman, D. A., & Frieder, O. (2004). *Information Retrieval: Algorithms and Heuristics.* Califórnia: Springer.

Guidance. (2011). *Site Oficial do EnCase Forensic*. Acesso em 19 de 07 de 2011, disponível em http://www.guidancesoftware.com/forensic.htm

Hadoop. (2011). *Apache Hadoop*. Acesso em 08 de 07 de 2011, disponível em Sítio do Projeto Apache Hadoop: http://hadoop.apache.org/

HDFS. (2011). *HDFS*. Acesso em 08 de 07 de 2011, disponível em Sítio do Projeto HDFS: http://hadoop.apache.org/hdfs/

Huff, B. (2006). *The Definitive Guide to Stellent Content Server Development.* Califórnia, EUA: APress.

Huston, L., Sukthankar, R., & Wickremesinghe, R. (2004). Diamond: A Storage Architecture for Early Discard in Interactive Search. *USENIX File and Storage Technologies FAST Conference.* Califórnia, Estados Unidos.

Ian, F. (1995). *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering.* Acesso em 10 de 12 de 2011, disponível em Página pessoal de Ian Foster: http://www.mcs.anl.gov/~itf/dbpp/text/book.html

*IBM GPFS*. (2011). *Site Oficial do Sistema de Arquivos GPFS*. Acesso em 19 de 10 de 2011, disponível em http://www-03.ibm.com/systems/software/gpfs/
*Java RMI*. (2011). *Site Oficial da Oracle*. Acesso em 12 de 10 de 2011, disponível em: http://docs.oracle.com/javase/6/docs/technotes/guides/rmi/index.html

Java. (2011). *Site Oficial da Linguagem Java*. Acesso em 10 de 06 de 2011, disponível em http://www.java.com/

Kaminsky, A. (2010). *Building Parallel Programs: SMPs, Clusters & Java.* Boston, EUA: Cengage Learning.

Katta. (2011). *Site Oficial do Projeto Katta*. Acesso em 10 de 06 de 2011, disponível em http://katta.sourceforge.net/

Kenneth, R. B., & Karttunen, L. (2003). Finite State Morphology. *CSLI Publications* .

Knorr, E., & Gruman, G. (2009). What cloud computing really means. *InfoWorld* .

Knuth, D. E. (1997). *The Art of Computer Programming, Volume 3: Sorting and Searching.* Califórnia: Addison Wesley.

Kopparapu, C. (2002). *Load Balancing Servers, Firewalls, and Caches.* Califórnia, EUA: Wiley.

Ladner, R. E., & Fischer, M. J. (1980). *Parallel Prefix Computation.* Jornal da ACM.

Lima, R. B. (2011). *Manual de Processo Penal - Volume I.* São Paulo: Impetus.

Lucene. (2011). *Apache Lucene*. Acesso em 08 de 07 de 2011, disponível em Sítio do Projeto Apache Lucene: http://lucene.apache.org/

*Lucene Contrib*. (2011). *Site Oficial de Contribuições ao Projeto Apache Lucene*. Acesso em 12 de 10 de 2011, disponível em http://lucene.apache.org/java/2_9_4/lucene-contrib/index.html

*Luke*. (2011). *Site Oficial do Projeto Luke*. Acesso em 12 de 10 de 2011, disponível em http://code.google.com/p/luke/

Maarten, V. S., & Tanenbaum, A. S. (2007). *Sistemas Distribuídos - Princípios e Paradigmas.* São Paulo: Prentice Hall - Br.

Manning, C., Raghavan, P., & Shutze, H. (2008). *Introduction to Information Retrieval.* Cambridge, Inglaterra: Cambridge University Press.

Massie, M. L., Chun, B. N., & Culler, D. E. (2004). The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing, Vol. 30* .

Mattmann, C. A., & Zitting, J. L. (2010). *Tika in Action.* Stamford, CT, Estados Unidos: Manning Publications Co.

McCandless, M., Hatcher, E., & Gospodnetic, O. (2010). *Lucene in Action.* Greenwich, CT, Estados Unidos: Manning Publications Co.

McCandless, M., Hatcher, E., & Gospodnetic, O. (2010). *Lucene in Action.* Stamford, CT, Estados Unidos: Manning Publications Co.

*Microsoft Outlook.* (2011). *Site Oficial da Microsoft*. Acesso em 19 de 07 de 2011, disponível em http://office.microsoft.com/pt-br/outlook/

Middleton, C., & Baeza-Yates, R. (2007). *A Comparison of Open Source Search Engines.* Acesso em 07 de 10 de 2011, disponível em http://wrg.upf.edu/WRG/dctos/Middleton-Baeza.pdf

*Mozilla Thunderbird.* (2011). *Site Oficial da Mozilla no Brasil*. Acesso em 07 de 10 de 2011, disponível em http://br.mozdev.org/thunderbird/

Nov, O., Anderson, D., & Arazy, O. (2011). Volunteer Computing: A Model of the Factors Determining Contribution to Community-based Scientific Research. *Proceedings of the 19th international conference on World wide web* (pp. 741-750). Nova York, EUA: ACM Press.

*Nutch*. (2011). *Site Oficial do Projeto Apache Nutch*. Acesso em 07 de 10 de 2011, disponível em http://nutch.apache.org/

Nutch. (2011). *Site Oficial do Projeto Nutch*. Acesso em 13 de 07 de 2011, disponível em http://nutch.apache.org/

*OutSide-In*. (2011). *Site Oficial do Outside-in*. Acesso em 07 de 10 de 2011, disponível em http://www.oracle.com/us/technologies/embedded/025613.htm

Paltoglou, G., Salampasis, M., & Satratzemi, M. (2008). A Comparison of Centralized and Distributed Information Retrieval Approaches. *2008 Panhellenic Conference on Informatics*, (pp. 21-25). Ilha Samos, Grécia.

Paolo, F., & Venturini, R. (2007). Compressed Permuterm Indexes. *ACM Press* , Proc. SIGIR.

*Phoenix*. (2011). *Site Oficial do Projeto Phoenix MapReduce*. Acesso em 07 de 10 de 2011, disponível em http://mapreduce.stanford.edu/

Prodan, R., & Fahringer, T. (2006). *Grid Computing.* Innsbruck, Austria: Springer. *PVFS*. (10 de 10 de 2011). Fonte: Site Oficial do Sistema de Arquivos PVFS: http://www.pvfs.org/

Raffaele, P., Salvatore, O., & Silvestri, F. (2004). Assigning document identifiers to enhance compressibility of web search engines indexes. *ACM Symposium on Applied Computing* , 600-605.

*rapid-i*. (2011). Site Oficial do Projeto Rapid-i. Acesso em 07 de 10 de 2011, disponível em http://www.rapid-i.com/

*Request For Comments*. (2011). *Site Oficial das RFCs*. Acesso em 15 de 10 de 2011, disponível em http://www.ietf.org

Riedel, E., Faloutsos, C., & Gibson, G. A. (Junho de 2001). Active Disks for Large-Scale Data Processing. *IEEE Computer* , pp. 68-71.

Rowstron, A., & Druschel, P. (2001). *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.* Heidelberg, Alemanha: ACM Press.

Ryan, K. L., Stephen, S. G., & Eng, W. L. (2009). Business Process Management (BPM) Standards: A Survey. *Emerald Group Publishing Limited* , 774-791.

Saferstein, R. (2010). *Criminalistics: An Introduction to Forensic Science.* Cafilórnia, EUA: Prentice Hall.

Schulz, G. (2011). *Cloud and Virtual Data Storage Networking.* Flórida, EUA: CRC Press.

Silvestri, F. (2007). Sorting out the document identifier assignment problem. *Proc. ECIR* , 101-112.

*SISAL*. (10 de 12 de 2011). Fonte: Tutorial sobre o SISAL: http://www2.cmp.uea.ac.uk/~jrwg/Sisal/00.Contents.html

SleuthKit. (2011). *Site Oficial do Sleuth Kit*. Acesso em 19 de 07 de 2011, disponível em http://www.sleuthkit.org/

Smith, D. C. (1969). *MLISP User's Manual.* Califórnia, Estados Unidos: Departamento de Ciência da Computação - Universidade de Stanford.

*SOAP*. (2011). *Site com a Especificação SOAP*. Acesso em 07 de 08 de 2011, disponível em http://www.w3.org/TR/soap/

Sottile, M., & Minnich, R. (2002). Supermon: a high-speed cluster monitoring system. *IEEE International Conference* (pp. 39-46). Novo México, EUA: IEEE.

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

Stallings, W. (2008). *Criptografia e Segurança de Redes - Princípios e Práticas.* São Paulo: Prentice Hall.

Stanoevska, K., & Wozniak, T. (2009). *Grid and Cloud Computing.* Nova York, EUA: Springer.

Stinson, D. R. (2006). *Cryptography - Theory and Practice.* Flórida, EUA: Chapman & Hall.

*Sun LustreFS*. (2011). Sitio Oficial do Sistema de Arquivos LustreFS. Acesso em 07 de 09 de 2011, disponível em http://wiki.lustre.org/index.php/Main_Page

Tanenbaum, A. S. (2003). *Redes de Computadores.* São Paulo: Campus.

Tanenbaum, A. S. (2010). *Sistemas Operacionais Modernos.* São Paulo: Pearson.

Tantisiriroj, W., Patil, S., & Gibson, G. (2008). *Data-intensive File Systems for Internet*

*Services.* Pensilvânia, EUA: Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-08-114.

Thain, D., Tannenbaum, T., & Livny, M. (2004). Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience* .

Tika. (2011). *Apache Tika*. Acesso em 08 de 07 de 2011, disponível em http://tika.apache.org/

Toutanova, Kristina, & Moore, R. C. (2002). Pronunciation Modeling for Improved Spelling Correction. *Proc. ACL* , 144–151.

Treeratpituk, P., & Callan, J. (2006). An experimental study on automatically labeling hierarchical clusters using statistical features. *ACM Press* , 707-708.

Valiant, L. G. (1997). *A Bridging Model for Parallel Computation.* Communications of the ACM.

Walter, C. (2005). Kryder's Law. *Scientific American Magazine* .

Wang, X., & Ju, S. (2009). A Set-Covering-Based Approach for Overlapping Resource Selection. *World Congress on Computer Science and Information Engineering [WRI]* (pp. 272-276). Los Angeles, California, Estados Unidos: IEEE Computer Society.

Werneck, B. P. (2007). Um sistema Multiagente para Exames Periciais em Sistemas de Informática. *ICoFCS – The Second Internacional Conference of Forensic Computer Science, Volume 2* (p. 104). Brasília: ABEAT.

White, T. (2009). *Hadoop, The Definitive Guide.* Georgia, Estados Unidos: O'Reilly Media.

Wiesmann, M., & Pedone, M. (2000). Understanding replication in databases and distributed systems. *Proceeding 20th Internation Conference on Ditributed Computing Systems (ICDCS'2000).* República da China: IEEE.

*Windows Search.* (2011). *Site Oficial da Microsoft Brasil*. Acesso em 07 de 11 de 2011, disponível em: http://windows.microsoft.com/pt-BR/windows7/products/features/windows-search

Zhao, B. Y., & Huang, L. (2004). Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *Jounal on Selected Areas In Communications* (pp. 41-53). IEEE.

Zobel, J., & Alistair, M. (2006). Inverted files for text search engines. *ACM Computing Surveys* , 38.

ZooKeeper. (2011). *Site Oficial do Projeto Apache ZooKeeper*. Acesso em 10 de 06 de 2011, disponível em http://zookeeper.apache.org/

# ANEXOS

# A. FORMATO DO ÍNDICE UTILIZADO

Neste anexo será apresentada uma visão geral da estrutura lógica e física do índice utilizado pela prova de conceito elaborada no presente trabalho. Será também ilustrado como foi a estruturação realizada nos dados forenses para possibilitar a indexação e busca indexada nestes dados.

A biblioteca de IR utilizada para realizar a indexação e pesquisa dos dados forenses foi a *Apache Lucene*, versão 3.1.0, conforme exposto no capítulo 3 do presente trabalho. A estrutura do índice utilizada pelo *Lucene* é no formato de índice invertido e foi projetada para maximizar o desempenho e minimizar o consumo de recursos. Em índices invertidos o termo é a estrutura principal. Os termos listam os documentos que os referenciam, ao invés de um índice comum onde os documentos é que listam os termos que ele possui (McCandless, Hatcher, & Gospodnetic, 2010) e (Lucene, 2011).

Os conceitos fundamentais da estrutura lógica do índice criado pelo *Lucene* são documento, campo e termo. Um índice é composto por uma série de documentos. O documento é uma estrutura que possui diversos campos. Os campos são grupos nomeados de termos, sendo que os termos são compostos por conteúdo textual. A figura A.1 ilustra a estrutura lógica do índice do *Lucene* utilizado na prova de conceito.

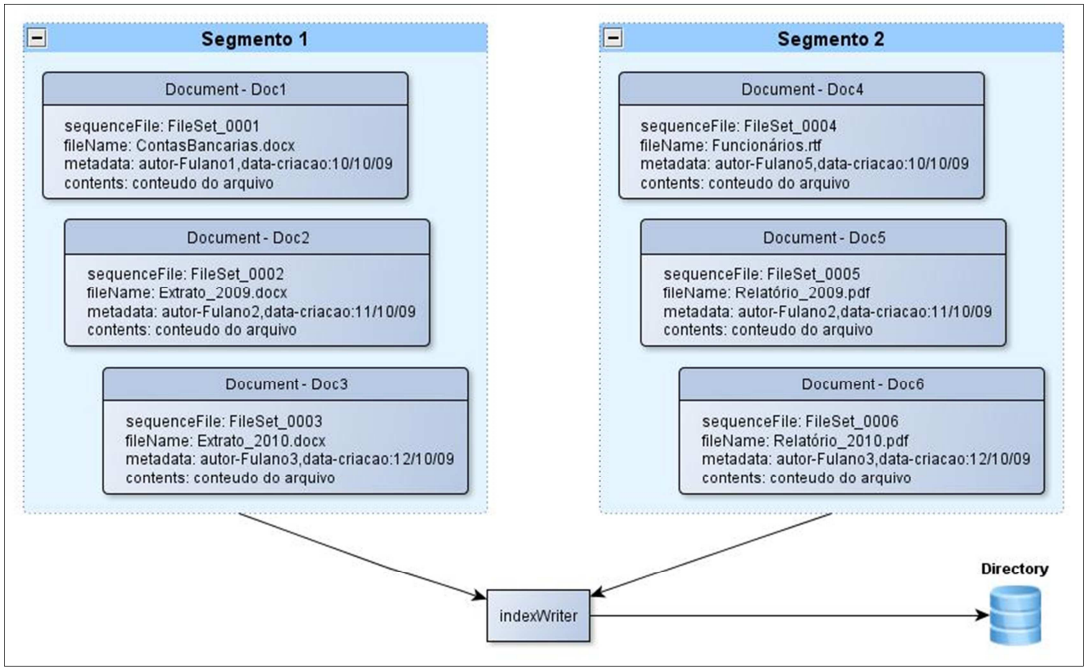



Figura A.2 – Estrutura Lógica do Índice do *Lucene*

A figura A.1 mostra diversos documentos, sendo que cada documento tem um identificador único, como: Doc-1, Doc-2 e Doc-3. Cada documento possui um conjunto de campos. Os campos ilustrados são *sequenceFile*[3], *fileName*, *metadata* e *contents*. Estes campos formam a estrutura do documento que armazena um arquivo de dado forense, sendo a estruturação realizada de forma a facilitar futuras consultas indexadas. O campo *sequenceFile* contém o nome do arquivo de agrupamento *FileSet* que armazena o documento encontrado, servindo para localizar este documento no HDFS. O campo *fileName* contém o nome do arquivo que o documento *Lucene* representa, este nome servirá para localizar o arquivo dentro do objeto *FileSet*. O campo *metadata* contém todos os metadados extraídos do arquivo no momento da interpretação do mesmo, exemplos de metadados são o autor e a data de criação do arquivo. A existência deste campo no índice possibilita que as buscas indexadas localizem arquivos que possuam determinados metadados. O campo *contents* armazena o conteúdo textual do arquivo que foi indexado, possibilitando assim buscas indexadas no conteúdo dos dados forenses. O conteúdo textual armazenado no documento é armazenado no formato de termos. Os termos são os elementos essenciais do índice invertido que possibilitam a busca indexada nos dados forenses.

Os campos podem ser armazenados no índice ou não. Quando os campos são armazenados no índice eles são gravados literalmente, ou seja, de uma maneira não invertida. Os campos que são gravados de uma maneira invertida são chamados de indexados. O armazenamento e a indexação de cada campo são facultativos. O texto contido em um campo pode ser divido em termos ou não. A maioria dos campos de texto devem ser divididos em termos, mas podem ter alguns que exijam uma pesquisa no conteúdo completo do campo, sem a exclusão de termos não relevantes. Os campos *sequenceFile*, *fileName* e *metadata* são armazenados e indexados, mas não são divididos em termos para possibilitar consulta em seu conteúdo original. O campo *contents* não é armazenado mas é indexado e dividido em termo para possibilitar a busca indexada por termo em todo o conteúdo do documento indexado. Esta configuração diminui o espaço necessário de armazenamento do índice pois o conteúdo textual dos documentos não serão armazenados no índice, apenas os termos extraídos. Os campos *sequenceFile* e *fileName* permitem a localização do documento no HDFS caso seja necessário a visualização do conteúdo do documento recuperado.

[1] Os campos do índice estão em inglês por seguir um padrão internacional que possibilita que o presente trabalho seja utilizado em outros projetos de código-fonte aberto.

Outro elemento importante do índice do *Lucene* é o segmento. Um índice pode ter vários segmentos, sendo que cada segmento tem um conjunto de documentos. Um segmento é similar a um sub-índice e é uma estrutura que armazena um índice totalmente funcional. O objetivo do segmento é possibilitar a indexação incremental, ou seja, possibilitar que novos documentos sejam adicionados ao índice sem ser necessária uma nova indexação. A biblioteca *Lucene* possui recursos para fazer o *merge* dos segmentos presentes no índice, fazendo com que o índice fique com apenas um segmento.

Para poder armazenar fisicamente esta estrutura lógica dos índices são utilizados os objetos *IndexWriter* e *Directory*. O objeto *IndexWriter* cria toda a estrutura lógica dos objetos do índice do *Lucene*, sendo que o objeto *Directory* armazena os índices em um determinado meio físico. Vários são os meios físicos que podem armazenar o índice invertido, exemplos são: a memória RAM, o sistema de arquivos ou um banco de dados.

Em todos os tipos de meios físicos de armazenamento do índice *Lucene* existe uma estrutura padrão de armazenamento. Para analisar esta estrutura será visto o armazenamento do índice no sistema de arquivos, que é o que foi utilizado na prova de conceito do presente trabalho. Um índice *Lucene* é composto de vários segmentos. Cada segmento é feito de diversos arquivos de índices. Os arquivos de índice que pertencem ao mesmo segmento compartilham um prefixo comum e se diferem pelo sufixo. A figura A.2 contém uma listagem de arquivos de índice do *Lucene*.

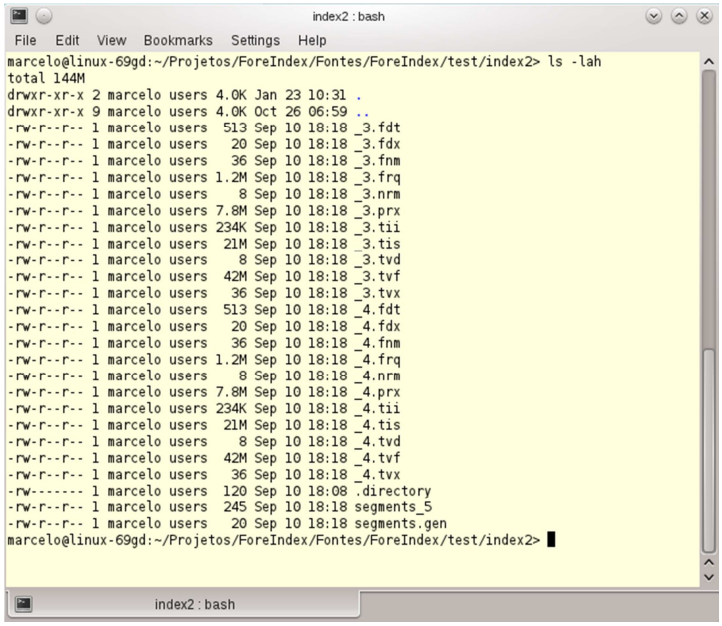



Figura A.2 – Arquivos de um Índice *Lucene*

Na listagem de arquivos de um índice *Lucene*, ilustrada na figura A.2, verifica-se que existem dois segmentos, um iniciando com o prefixo _3 e outro iniciando com o prefixo _4. O arquivo de nome segments_5 armazena o nome e outros detalhes de todos os segmentos presentes no índice. O número exato de arquivos presente em um índice e em cada segmento de índice do *Lucene* varia de índice para índice e depende de como os campos foram indexados. Mas em todo índice *Lucene* existe ao menos um arquivo de segmento e um arquivo *segments.gen* que contém o prefixo de geração do segmento atual, sendo um modo redundante para o *Lucene* determinar o segmento atual.

A figura A.3 ilustra a estrutura dos principais arquivos do índice *Lucene*.

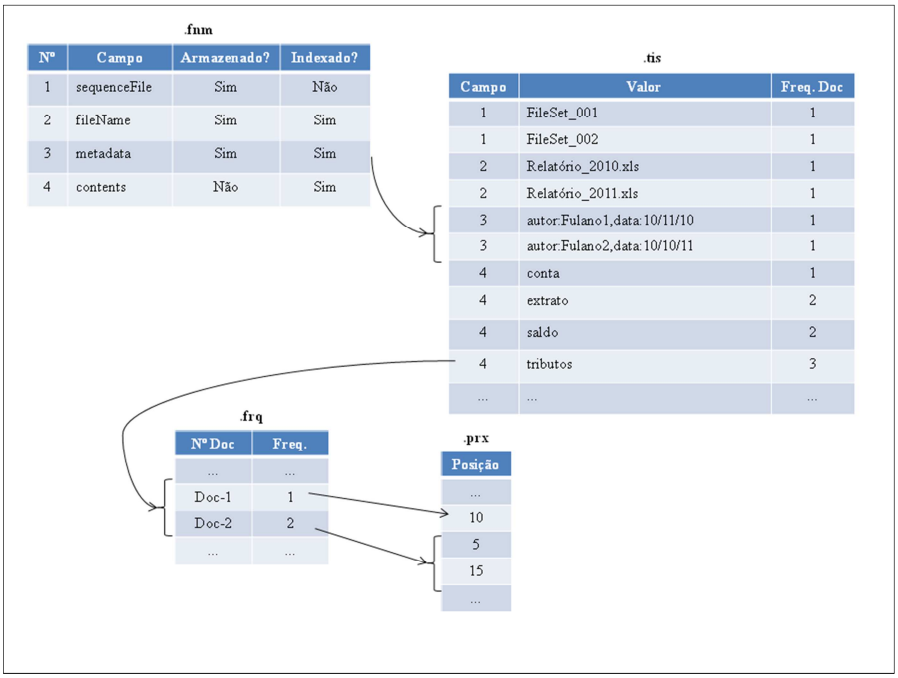



Figura A.3 – Estrutura Física do Índice do *Lucene*

O arquivo *.fnm* (*Fields Names*) contém todos os nomes de campos de todos os documentos de determinado segmento. Cada campo contém atributos adicionais como seu identificador, se foi armazenado e se foi indexado. A ordem dos campos é determinada durante a indexação e não necessita ser alfabética e o número do campo é que será utilizado para referenciar o campo para economizar espaço. Todos os termos de um segmento são armazenados no arquivo *.tis* (*Term Infos*). Os termos são ordenados alfabeticamente pelo número do campo e posteriormente pelo texto do termo. Cada termo contém sua freqüência em documentos, que é o número de documentos que contém o

termo dentro do segmento. O arquivo *.tii* (*Term Info Index*) é um arquivo de indexação que fica armazenado na memória RAM permitindo um acesso aleatório ao arquivo *.tis*. Para cada termo no arquivo *.tis* o arquivo *.frq* (*Frequencies*) contém os registros de cada documento que contém o termo. Nestes registros também estão armazenados a quantidade de vezes que o tempo aparece no documento. Para cada documento listado no arquivo *.frq* o arquivo *.prx* (*Positions*) contém registros da localização de cada ocorrência do termo dentro do documento.

De acordo com o ilustrado na figura A.3 observa-se existem os campos *sequenceFile*, *fileName*, *metadata* e *contents*. No arquivo *.tis* consta que o campo *contents* possui um termo com o termo “tributos” que tem o valor 03 para sua freqüência em documentos, mostrando que aparece três vezes em todos os documentos do segmento. O arquivo .frq registra que o termo “tributos” aparece nos documentos com os identificadores: Doc-1 e Doc-2. O arquivo *.prx* registra a localização do termo no documento Doc-1 (posição 10) e também registra as posições deste termo no documento Doc-1 (posições 5 e 15).

Além destes arquivos principais os arquivos de índice do *Lucene* também podem conter os seguintes arquivos:

- *.fdx* (*Field Index*) e *.fdt* (*Field Data*): o primeiro contém ponteiros para os dados de cada campo e o segundo contém o armazenamento dos campos do documento;

- *.tvf* (*Term Vector Fields*) , *.tvd* (*Term Vector Documents*) e *.tvx* (*Term Vector Index*): estes campos armazenam dados vetoriais armazenados em campos e documentos, bem como ponteiros apontando para os dados destes campos;

- *.del* (*Deleted*): informações sobre os arquivos que foram excluídos;

- *.cfs* (*Compound File*): um arquivo opcional que armazena todos os outros arquivos do índice. Este arquivo pode ser utilizado para impedir a criação de diversos arquivos de índice, agrupando internamente os dados dos outros arquivos. Este recurso é utilizado para diminuir a quantidade de arquivos existentes, mas diminui o desempenho do processo de indexação;

- *.write.lock*: para prevenir que múltiplos indexadores realizem atualizações no índice ao mesmo tempo.

Conforme ilustrado a estrutura dos índices da biblioteca *Lucene* visa maximizar o desempenho e minimizar os recursos utilizados. Exemplos práticos existentes no projeto da estrutura do índice são:

- se um campo não tiver indexado ele pode ser excluído das pesquisas que são realizadas, através de uma rápida consulta ao arquivo .fnm;

- o dicionário de termos pode ser rapidamente acessado com o arquivo .tii armazenado na memória RAM e,

- se as informações de posicionamento dos termos não estiverem presentes as consultas não precisam inicializar recursos para recuperar esta informação. Minimizar a quantidade de arquivos a pesquisar melhora consideravelmente o tempo de pesquisa.

# B. UTILIZAÇÃO DA PROVA DE CONCEITO

Neste anexo será apresentada uma visão geral do processo de utilização da prova de conceito elaborada no presente trabalho.

A prova de conceito elaborada é um sistema que realiza o armazenamento e a indexação de dados utilizando computação distribuída. Para utilizar este sistema devem ser atendidos os seguintes requisitos:

- Existência de um conjunto de dados forenses a indexar;

- Existência de um ambiente distribuído capaz de executar o *Apache Hadoop* e o HDFS. A recomendação é um cluster, com no mínimo cinco computadores, interconectados por uma rede *Gigabit Ethernet* ou superior. O sistema operacional recomendado é o Linux 2.6 ou superior.

O código-fonte e executável da prova de conceito podem ser obtidos no site: `http://sourceforge.net/projects/foreindex/`

Para a instalação do *Apache Hadoop* e do HDFS basta seguir as instruções de seu site oficial: `http://hadoop.apache.org/common/docs/current/cluster_setup.html`

Após a instalação do *Apache Hadoop* cada nó do cluster deve ser configurado. A recomendação é que um computador seja o *JobTracker*, outro computador diferente seja o *NameNode* e os outros sejam os *DataNodes* e *TaskTrackers*. Esta configuração facilita a expansão e gerenciamento do ambiente.

Os arquivos de configuração do *Apache Hadoop* ficam no sub-diretório `conf` do diretório raiz da instalação. Os principais arquivos que devem ser configurados são: `core-site.xml`, `hdfs-site.xml` e `mapred-site.xml`. Em seguida serão detalhados os principais parâmetros de configuração que devem ser estabelecidos em cada um destes arquivos no *JobTracker*, *NameNode*, *TaskTracker* e *DataNode*.

A figura B.1 contém os dados básicos do arquivo `core-site.xml`.

```
<configuration>
        <property>
                <name>fs.default.name</name>
                <value>hdfs://10.61.82.201:9000</value>
        </property>
        <property>
                <name>hadoop.tmp.dir</name>
                <value>/home/{user.name}/tmp/hadoop</value>
        </property>
</configuration>
```

Figura B.3 – Elementos de Configuração do Arquivo `core-site.xml`

A propriedade `fs.default.name` deve possuir o endereço IP e porta do *NameNode*, ou seja, do sistema de arquivos distribuído que será utilizado. Opcionalmente também deve ser configurado um diretório de armazenamento de dados temporários. O conteúdo deste arquivo é o mesmo para o *JobTracker*, *NameNode*, *DataNode* e *TaskTracker*.

A figura B.2 contém o conteúdo do arquivo `hdfs-site.xml`.

```
<configuration>
     <property>
         <name>dfs.replication</name>
         <value>3</value>
     </property>
     <property>
         <name>dfs.name.dir</name>
          <value>/home/marcelo/hdfs/name</value>
    </property>
    <property>
          <name>dfs.data.dir</name>
          <value>/home/marcelo/hdfs/data</value>
    </property>
</configuration>
```

Figura B.2 – Elementos de Configuração do Arquivo `hdfs-site.xml`

As duas primeiras propriedades devem estar presentes apenas no *NameNode*. A propriedade `dfs.replication` define o fator de replicação do HDFS, ou seja, a quantidade de cópias que cada bloco de dados deverá possuir no sistema de arquivos distribuído. A propriedade `dfs.name.dir` define o diretório onde serão localizados os metadados do *NameNode*. Neste diretório serão armazenados todos os arquivos de configuração do sistema de arquivos distribuído. A última propriedade, `dfs.data.dir`, deve estar presente apenas nos computadores que serão *DataNodes* e define o diretório onde serão armazenados os blocos de dados do sistema de arquivo distribuído. O *JobTracker* não deve possuir nenhuma propriedade configurada neste arquivo.

A figura B.3 contém o conteúdo do arquivo `mapred-site.xml`.

```
<configuration>
        <property>
                <name>mapred.job.tracker</name>
                <value>10.61.82.200:9001</value>
        </property>
        <property>
                <name>mapred.local.dir</name>
                <value>/home/marcelo/mapred/local</value>
        </property>
        <property>
                <name>mapred.tasktracker.map.tasks.maximum</name>
                <value>2</value>
        </property>
        <property>
                <name>mapred.tasktracker.reduce.tasks.maximum</name>
                <value>2</value>
        </property>
        <property>
                <name>mapred.child.java.opts</name>
                <value>-Xmx250m</value>
        </property>
</configuration>
```

Figura B.3 – Elementos de Configuração do Arquivo `mapred-site.xml`

A propriedade de nome `mapred.job.tracker` contém a localização do JobTracker. A propriedade `mapred.local.dir` contém o diretório temporário do processamento realizado pelo *TaskTracker*. As propriedades `mapred.tasktracker.map.tasks.maximum` e `.tasktracker.reduce.tasks.maximum` definem o número máximo de tarefas *Maps* e *Reduces* que executarão em cada *TaskTracker*. A recomendação é que o total de tarefas seja igual ao número de processadores da máquina. A propriedade `mapred.child.java.opts` contém a quantidade máxima de memória RAM que cada tarefa do TaskTracker utilizará. Todas estas propriedades devem ser configuradas apenas no *TaskTracker* os outros processos não necessitam destas configurações.

As propriedades de segurança `dfs.hosts` e `mapred.hosts` que devem ser configuradas tanto no *JobTracker* quanto no *NameNode* para definir quais computadores do cluster terão permissão para ser um *TaskTracker* ou *DataNode*. Adicionalmente existe o arquivo *slaves* que contém o endereço de todos os computadores que são *TaskTracker* e *DataNodes*. Este arquivo deve existir tanto no *JobTracker* e como no *NameNode*. Este arquivo serve apenas para ser utilizado nos scripts de inicialização e gerenciamento do ambiente e não tem a função de segurança.

Para utilizar as funcionalidades da prova de conceito desenvolvida basta utilizar a classe `br.gov.dpf.unb.foreindex.ForeIndex`. Esta classe permite a inicialização da interface gráfica do aplicativo e também o acionamento das funcionalidades por linha de comando. Para acionar esta classe as bibliotecas do *Apache Hadoop*, *Apache Lucene* e *Apache Tika* devem estar no *classpath*.