ULBRA CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Recredenciado pela Portaria Ministerial nº 3.607 - D.O.U. nº 202 de 20/10/2005

Wisley Cristiano de Souza Milhomem

INDEXAÇÃO DE TERMOS PARA UM SISTEMA DE RECUPERAÇÃO DA INFORMAÇÃO UTILIZANDO COMPUTAÇÃO DISTRIBUÍDA

Palmas - TO

2013

Wisley Cristiano de Souza Milhomem

INDEXAÇÃO DE TERMOS PARA UM SISTEMA DE RECUPERAÇÃO DA INFORMAÇÃO UTILIZANDO COMPUTAÇÃO DISTRIBUÍDA

Trabalho de Conclusão de Curso (TCC) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.Sc. Fernando Luiz de Oliveira

Palmas – TO

2013

Wisley Cristiano de Souza Milhomem

INDEXAÇÃO DE TERMOS PARA UM SISTEMA DE RECUPERAÇÃO DA INFORMAÇÃO UTILIZANDO COMPUTAÇÃO DISTRIBUÍDA

Trabalho de Conclusão de Curso (TCC) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.Sc. Fernando Luiz de Oliveira

Aprovada em: Junho de 2013.

BANCA EXAMINADORA

_____

Prof. M.Sc. Fernando Luiz de Oliveira

Centro Universitário Luterano de Palmas

_____

Prof. M.Sc. Fabiano Fagundes

Centro Universitário Luterano de Palmas

_____

Prof. M.Sc. Madianita Bogo Marioti

Centro Universitário Luterano de Palmas

Palmas - TO

2013

AGRADECIMENTOS

Agradeço a Deus por ter me concedido a oportunidade de concluir essa graduação e ter me dado forças para superar todas as dificuldades. Agradeço muito aos meus pais (João e Regina) e ao meu irmão (Júnior), por terem me apoiado no que foi preciso e me incentivado a não desistir de lutar e finalizar esse curso.

Agradeço também aos amigos que conquistei nesses anos de faculdade, vocês fizeram as coisas serem mais legais nesse tempo.

Deixo aqui também, o meu agradecimento ao meu orientador Fernando, pela paciência em me orientar em todos os meus trabalhos científicos. Não poderia deixar de agradecer também a todos os professores do curso, que sempre estiveram de prontidão para ajudar.

RESUMO

Em decorrência da utilização de sistemas de informação, as bases de dados crescem de forma exponencial. Como consequência, indexar grandes bases de dados para que seja executado o processo de recuperação de informação para o usuário demanda uma maior quantidade de recursos no que tange a estrutura de *hardware*. Com a utilização de sistemas de recuperação de informação em ambientes distribuídos pode ser resolvido o problema de indexar grandes quantidades de dados, conseguindo desta forma, alto desempenho e escalabilidade, dividindo o processamento e os índices em vários nós. Este trabalho tem como objetivo apresentar os conceitos do processo de indexação na recuperação da informação e computação distribuída e desenvolver uma aplicação de indexação utilizando os *frameworks* Hadoop, Katta e Lucene. São utilizados dois cenários de execução do sistema (normal e distribuído) e ao final são apresentados os resultados, a média do uso de memória RAM e processador e tempo gasto nos processos de indexação executados.

PALAVRAS-CHAVE: indexação, sistemas distribuídos.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

SSH – Secure Shell;

HDFS – Hadoop Distributed File System;

SUMÁRIO

1 INTRODUÇÃO

A grande quantidade informação disponível na *World Wide Web* (*WWW),* além das diversas formas de disseminação de conteúdo e a facilidade de publicá-las e distribuí-las, contribui para que a *web* se transforme em um grande repositório de conhecimento. Como consequência, ocorre um crescimento exponencial da quantidade de informações produzidas e disponibilizadas aos usuários, nos mais distintos tipos de aplicação, o que ocasiona um grande volume de informações de diferentes origens e formatos (VALIATI, 2008, p. 14).

No entanto, Oliveira (2005, p. 11) afirma que as facilidades obtidas através da utilização da *web* no processo de distribuição da informação contrastam com sua dificuldade na tarefa de fornecer os mecanismos necessários para promover as devidas formas de se obter as informações específicas que satisfaçam as necessidades de seus usuários. A grande quantidade de dados produzidos dificulta ainda mais esse processo de recuperação de informações relevantes.

Para satisfazer a necessidade de seleção e recuperação de informações que sejam úteis, utilizam-se os Sistemas de Recuperação da Informação (SRI), que são responsáveis pelos processos de representação, armazenamento, organização e acesso à informação (BAEZA-YATES & RIBEIRO-NETO, 1999, p. 1). Os SRI são sistemas automáticos que tratam das operações de busca, classificação e indexação de documentos para atender a necessidades das buscas realizadas.

A utilização de SRI em grandes bases de dados é um processo demorado e complexo, principalmente nos processos de busca e indexação. Diferentes tipos de estruturas de dados podem ser usados no processo de indexação, no entanto, a estrutura mais difundida e utilizada é o arquivo invertido (AUGUSTO, 2010, p. 2). Esse tipo de estrutura de dados precisa de espaço relativo de 30 a 50% do tamanho da coleção a ser indexada para armazenar o índice. Os arquivos invertidos acompanham o crescimento no tamanho das coleções, e assim tem seu desempenho prejudicado quando crescem muito. Dessa forma, a escalabilidade do sistema é comprometida, sobretudo no que tange a espaço em disco, velocidade de acesso aos dados e memória RAM disponível.

A utilização de SRIs em ambientes distribuídos visa resolver esses problemas, permitindo a divisão de tarefas em múltiplos discos, memórias e processadores para alcançar melhores desempenhos nos processos de recuperação da informação e garantir escalabilidade. Neste processo de indexação em sistemas distribuídos, cada máquina possui um índice e consequentemente um arquivo invertido independente, para melhorar o desempenho do processo de indexação.

Este trabalho tem como objetivo desenvolver um sistema de indexação de termos para um sistema de recuperação da informação utilizando computação distribuída, apresentando os *frameworks* e bibliotecas utilizadas. Para exemplificar, foram criados dois cenários, um ambiente de indexação normal e um ambiente de indexação distribuída, os quais foram comparados conforme os seguintes critérios: processamento, uso de memória RAM e tempo gasto para indexação. Por fim, são apresentadas as considerações finais, com o intuito de expor as conclusões em relação ao trabalho como um todo.

2 REFERENCIAL TEÓRICO

Esta seção tem o intuito de apresentar os conceitos necessários para fornecer *background* teórico para produção do estudo de caso proposto neste trabalho. Na seção 1 são descritos os processos de recuperação de informação. Na seção 2.4, são abordadas as características da computação distribuída bem como sua estrutura e funcionamento.

2.1. Recuperação da Informação

Segundo Mooers (1951, *apud* OLIVEIRA, 2005 p.22), a Recuperação da Informação (RI) define o processo segundo o qual um usuário consegue converter seus termos de busca em uma lista real com referências para documentos armazenados contendo informações úteis e relevantes ao seu interesse. A RI determina métodos e técnicas para representação, armazenamento, organização e busca da informação e, ainda, sistemas, técnicas e máquinas usadas para executar tais operações.

A chamada era da informação iniciada no século XX teve como principal acontecimento a explosão documental dos anos 40. Consequentemente, surgiu a necessidade de uma ciência para estudar a informação e seus processos de construção, nascendo assim a ciência da informação (LE COADIC, 2004 *apud* BRANDT, 2009, p.16). Borko (1968, *apud* TEIXEIRA, 2010 p.20) define a ciência da informação como: “uma ciência interdisciplinar que estuda as propriedades e comportamentos da informação, o fluxo da informação, e os meios de processamento da informação para obter melhor acessibilidade e utilidade”.

Saracevic (1995 *apud* BRANDT, 2009, p. 16) afirma que a criação da ciência da informação deve-se a explosão documental e que a crescente quantidade de informação necessitava ser mais acessível. No entanto, antes mesmo do nascimento do conceito de ciência da informação, Paul Otlet já abordava formas de recuperação de informação diante do crescimento da quantidade de documentos na época (BRANDIT, 2009, p 16).

Com o surgimento da *web* no início dos anos 90, a RI teve uma nova abordagem com relação às informações advindas da *web*. Baeza-Yates e Ribeiro-Neto (1999 *apud* BRANDT, 2009, p.20) reforçam este fato afirmando que, até então, a recuperação da informação era limitada à área da ciência da informação e biblioteconomia. Este conceito prevaleceu por

muitos anos, apesar de alguns usuários avançados de computadores já utilizarem conceitos e ferramentas de RI em sistemas de multimídia e hipertexto. No entanto, essa situação mudou com o início da *World Wide Web*.

Como Oliveira (2005, p. 22) afirma, a informação passou a ter mais importância estratégica, motivando estudos e pesquisa para aperfeiçoar a forma como a informação deveria ser armazenada e recuperada. Como consequência disto, a quantidade de informação produzida cresceu demasiadamente, ocasionando problemas no que tange recuperação de informação de nichos específicos.

O processo de organização da informação precisa ser bem preparado, principalmente na descrição do seu conteúdo, pois é este atributo que será usado para recuperação dos documentos relativos aos termos de busca utilizados. Gomes e Campos (2007, *apud* BRANDT, 2009, p. 20) apontam basicamente três fatores para que o processo de recuperação da informação seja perfeito:

- deve existir vocabulário padrão;
- atribuir termos de acordo com critérios já definidos, para garantir a perfeição no tratamento dos dados;
- adaptar o sistema de recuperação de informação com as características das informações trabalhadas.

Essa grande quantidade de informações transforma a *web* em um grande repositório de dados. No entanto, a aplicação de técnicas de Recuperação da Informação para que esses dados sejam apresentados de forma relevante para o usuário depende da utilização de Sistemas de Recuperação da Informação (SRI). Este conceito, bem como suas características, será o assunto da próxima seção.

2.2. Sistemas de Recuperação da Informação

Devido à grande quantidade de documentos advindos, principalmente, da *web* e da heterogeneidade dos conteúdos existentes nesse meio, tornou-se necessária a utilização de Sistemas de Recuperação da Informação (SRI) para que a recuperação rápida e precisa de documentos seja relevantes às necessidades do usuário. Neste contexto, os SRI visam resolver o problema da RI descrito por Huibers & Lamas & Rijsbergen (1996, *apud* Oliveira, 2005, p. 25) referente à "como separar informação relevante de informação irrelevante".

Os SRI são sistemas automatizados que tratam, basicamente, da indexação, busca e classificação de documentos com o objetivo de atender necessidades informacionais expressas

através das buscas efetuadas (GONZÁLES E LIMA, 2003, p. 18). Estes sistemas têm o objetivo de armazenar e recuperar as informações de maneira que o esforço humano na busca das informações seja mínimo. Para que a recuperação da informação aconteça de forma apropriada e bem sucedida, é imprescindível que o tratamento dos documentos inseridos no SRI seja executado de forma adequada e com qualidade, pois a forma como o SRI processa e armazena as informações interfere diretamente nas chances prováveis do usuário obter documentos úteis e relevantes. A Figura 1 apresenta graficamente os processos executados nos (SRI).

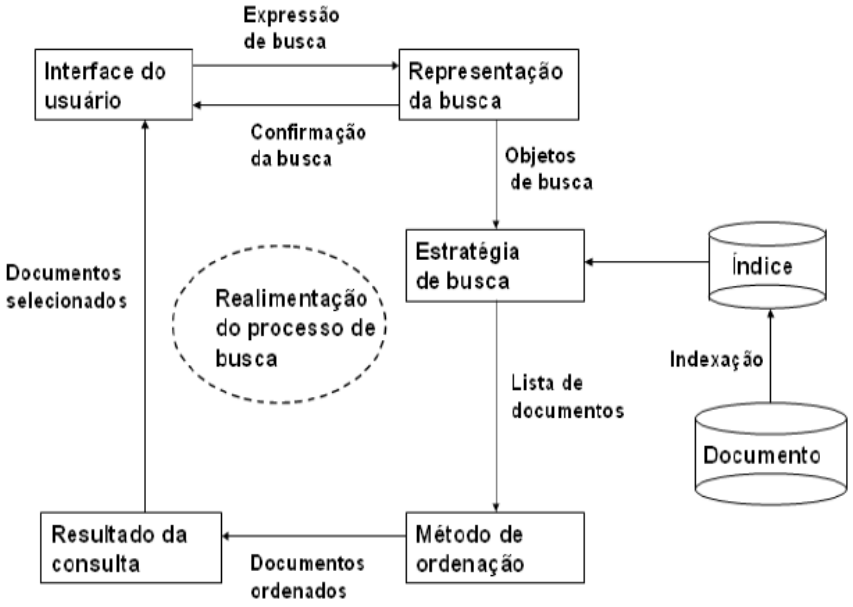

Figura 1- Representação do processo de recuperação da informação (LEITE, 2009, p. 9).

No processo básico de RI, o usuário utiliza a interface do sistema para buscar o que deseja, o sistema captura os dados inseridos pelo usuário e transforma em uma expressão de busca entendível pela máquina de acordo com as estratégias de busca. Essas expressões são utilizadas para consultar o arquivo de índices e ai recuperar os documentos que satisfaçam a consulta inicial feita pelo usuário. A lista de resultados passa por um processo de ordenação para assim ser apresentada ao usuário.

A Figura 1 ilustra as ações executadas em um SRI demonstrando a forma como os processos de indexação, busca e classificação de documentos funcionam para realizar a recuperação da informação para o usuário. Na seção 2.3, são apresentados os processos pertinentes ao funcionamento de um SRI.

2.3. Processo de Recuperação da Informação

O processo de recuperação é responsável por executar as ações para que as informações referentes ao termo buscado sejam retornadas de forma relevante. As etapas e os subprocessos referentes ao processo de funcionamento de um SRI são apresentados nas seções a seguir.

2.3.1. Documento

Segundo Baeza-Yates e Ribeiro-Neto (1999, p. 5), documentos são como uma visão lógica da fonte de dados. Essas fontes podem ser representadas por conjuntos de palavras-chaves extraídas do próprio texto da fonte de dados. Em linhas gerais, pode ser um parágrafo, uma seção, um capítulo, uma página web, um registro do banco de dados ou um livro inteiro. O importante é que deve representar um conteúdo pesquisado.

Seria essencial trabalhar com documentos construídos utilizando todas as palavras da fonte de dados, pois eles possuem quantidade maior de informações que podem auxiliar na recuperação da informação. No entanto, manusear documentos de texto completo (*full text)* necessita de alto custo computacional para indexar termos com muitas informações, principalmente quando se trata de grandes coleções de documentos.

O processo de tratamento do documento é ilustrado na Figura 2. As etapas são realizadas de acordo com o domínio do SRI definido na estrutura de reconhecimento. O tratamento do documento é necessário para definir termos que façam jus ao contexto da informação. Após esse processo, os termos do documento são utilizados nos índices para o processo de indexação.

Para reduzir a quantidade de palavras contidas nos termos, os documentos são submetidos ao processo de tratamento para remoção de *stopwords* e *stemming*. O grupo de palavras que não são considerados na busca é eliminado. Essas palavras são chamadas de *stopwords*, que podem ser palavras muito comuns do idioma ou do domínio do SRI e sem valor semântico para documento, bem como preposições, conjunções, advérbios e *etc*. O processo de *stemming* é referente à ação de reduzir as palavras ao seu radical, diminuindo palavras que estão em um mesmo contexto à sua raiz. Por exemplo, os termos “computador“ e “computação” seriam transformados no termo “comp”, fazendo referência a ambos. Na Figura 2, é apresentada a síntese do processo de tratamento do documento.

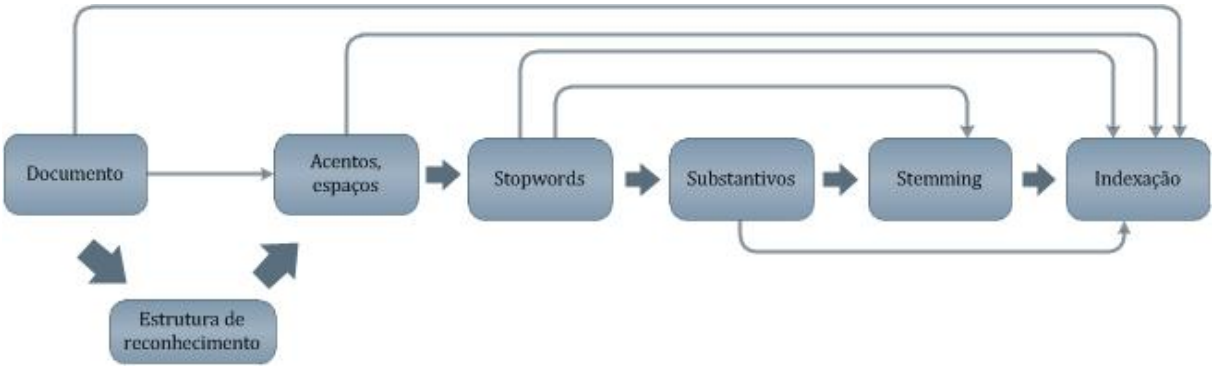

Figura 2 - Processo de tratamento do documento. Adaptado de (BAEZA-YATES e RIBEIRO-NETO, 1999, p. 24).

Na próxima seção, são abordadas as características dos índices e seu funcionamento.

2.3.2. Indexação

Ao ser realizada uma consulta básica, a forma mais simples de busca utilizada é a busca *online*, ou seja, percorrer o documento de forma sequencial procurando ocorrências dos termos informados para consulta (BAEZA-YATES e RIBEIRO-NETO, 1999, p.191). Essa técnica *online* é aplicável quando utilizada para busca em coleções de documentos de tamanho razoavelmente pequeno, além de ser a alternativa mais apropriada para coleções de documentos voláteis com constantes alterações em seu conteúdo. O processo de indexação é a parte de um SRI que fica responsável por criar índices para os documentos de uma coleção.

Para realizar buscas em grandes coleções de documentos, torna-se necessária a utilização de sistemas de indexação para pré-processar os documentos e construir índices para esses documentos, para que as consultas sejam processadas com maior eficiência e rapidez. Esse procedimento de indexação é apropriado para coleções de documentos com conteúdo semi-estático, ou seja, documentos que possuem baixa frequência de alterações, tais como, páginas da *web*, bancos de dados, obras literárias etc..

A indexação de termos consiste em associar os termos de um documento tratado a sua localização, para que a busca seja efetuada com maior precisão e rapidez. A Figura 3 destaca o funcionamento do processo de indexação em um SRI.

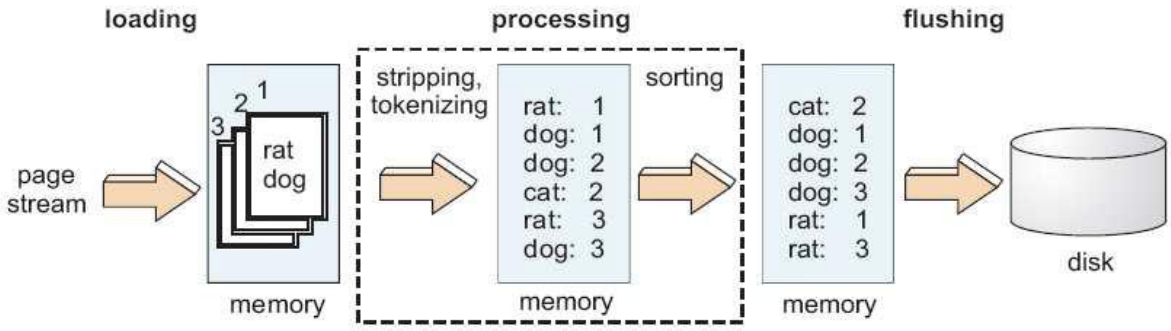

Figura 3 - Processo de indexação (ROCHA *et al.* 2006, p.5*).*

O processo de indexação destacado na Figura 3 é basicamente responsável pelo pleno funcionamento de um SRI bem como seu desempenho. No primeiro passo, os documentos são obtidos a partir de um fluxo de entrada. À medida que um documento passa pela primeira fase, ele é encaminhado para a fase seguinte, sendo inicialmente, removendo-se marcadores e outras informações não relacionadas ao seu conteúdo. Então, são identificados e extraídos cada um dos termos que o compõem. Novamente, é utilizada uma estrutura de dados para armazenar os termos e possibilitar o acesso ordenado (lexicograficamente)[1] a esses termos. Quando essa estrutura atinge o tamanho máximo de memória disponível, ela é encaminhada a fase de armazenamento, que grava os termos que estão na memória em um índice parcial no disco.

Para o processo de indexação ocorrer efetivamente, o sistema cria índices através de estruturas de dados. O funcionamento do processo de indexação e a estrutura de dados utilizada são descritos nas seções seguintes.

2.3.2.1. Índices

O conceito é definido por Baeza-Yates e Ribeiro-Neto (1999, p. 20) como termos pré-selecionados e armazenados em estruturas que fazem referência a documentos, possibilitando, desta forma, acesso aos dados com maior rapidez. Os índices são uma estrutura que permite ao SRI selecionar documentos que satisfaçam uma busca. Para isto, o mesmo é desenvolvido de forma a conter as informações importantes das coleções de documentos, como, vocabulário do sistema e relações de concorrência e similaridade das palavras.

Um índice bem formado contribui para o sucesso do funcionamento de um SRI, a construção desses índices é um processo trabalhoso. Tal processo envolve as tarefas de ranqueamento e indexação. Diferentes estruturas de dados podem ser utilizadas para os

[1] De modo lexicográfico, ordenado em forma de dicionário.

índices, contudo, esse trabalho irá detalhar o arquivo invertido, estrutura de dados mais utilizada e estudada nos artigos pesquisados para desenvolvimento desse trabalho. Na seção seguinte são ilustrados os conceitos de arquivo invertido.

2.3.2.2. Arquivo invertido

Várias estratégias de indexação podem ser utilizadas para criação de índices do processo de indexação, como, por exemplo, árvores e vetores de sufixo e suas variações e arquivos de assinatura. Contudo, a técnica de indexação mais utilizada e apropriada para utilização em Sistemas de Recuperação da Informação é a do arquivo invertido (BAEZA-YATES e RIBEIRO-NETO, 1999, p. 10).

O arquivo invertido é uma estrutura orientada a palavras, formada por um vocabulário e um conjunto de listas invertidas, em que é destinada uma lista invertida para cada termo diferente da coleção de documentos (BOTELHO, 2008, *online*).

O vocabulário é constituído pelo grupo de palavras exclusivas do documento organizadas em ordem alfabética onde cada palavra (termo) aponta para sua lista invertida correspondente. A lista invertida de cada termo pode conter diversas informações, tais como, ocorrência do termo no documento, identificação dos documentos que contêm o termo e a frequência do termo no documento. A Figura 4 exibe a representação visual de um arquivo invertido.

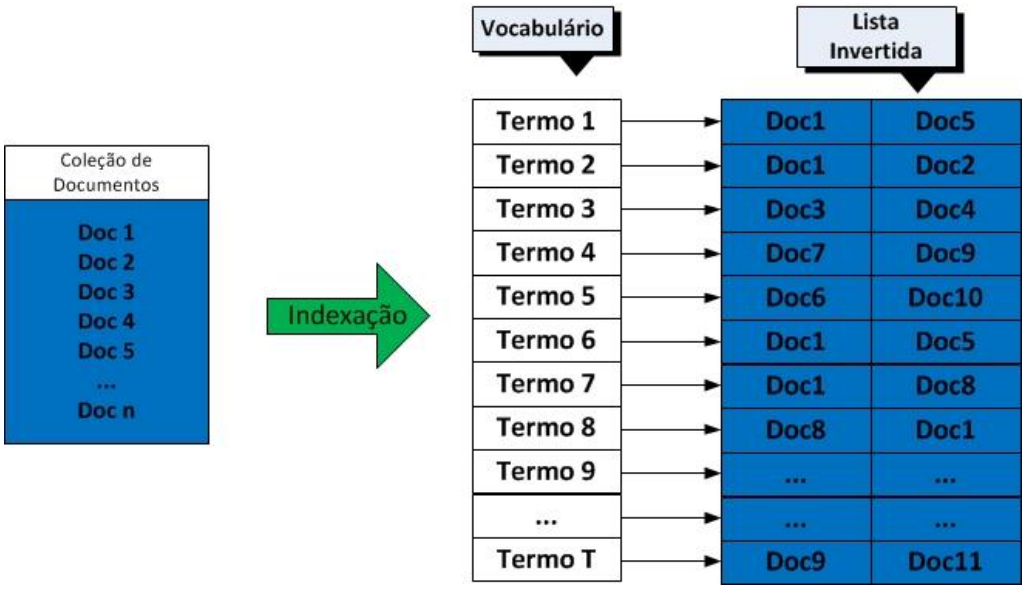

Figura 4 - Representação de arquivo invertido, de (BOTELHO, 2008, *online*).

Na Figura 4 é apresentada a estrutura de um arquivo invertido simples. Para cada documento é atribuído um conjunto de termos para compor o vocabulário e, a partir dos termos previamente definidos no vocabulário, é construída uma lista invertida para cada termo. Deste modo, no momento em que o SRI requisita uma informação, a busca é feita no arquivo invertido que é executado em memória RAM. A partir da informação encontrada, é para o SRI a localização do documento no disco, aumentando assim a eficiência na busca dos documentos.

O custo para dispor dessa eficiência da utilização do arquivo invertido é a necessidade de armazenar essa estrutura, que varia entre 30% e 60% em relação ao tamanho da coleção de documentos indexados (AUGUSTO, 2010, p. 14).

É importante ressaltar que um documento possui vários termos em seu contexto. Já a lista invertida, para cada termo, há uma lista de documentos onde este termo ocorre, fazendo jus ao nome da estrutura, arquivo invertido.

A ação responsável por executar o processo para retornar o dado da lista invertida fica a cargo do processo de busca, que será o foco da próxima seção.

2.3.3. Busca

A busca é o processo executado a partir de uma interface gráfica, onde o usuário entra com os termos desejados para a pesquisa no sistema, a partir dai, o sistema processa os dados e executa as operações de consulta sobre os índices, gerando um ranqueamento (classificação) dos resultados retornados.

O procedimento de busca em índices invertidos é composto por duas tarefas: definir a quantidade e quais os documentos que satisfazem uma consulta, e ordenar esses documentos de acordo com o domínio aplicado ao sistema de RI.

O SRI trabalha percorrendo o vocabulário procurando por lista invertidas referentes a cada um dos termos da busca. Para buscas por um único termo, a lista invertida referente ao termo buscado satisfaz o resultado da pesquisa. No entanto, quando o usuário utiliza vários termos na busca, o sistema deve considerar as operações booleanas envolvidas entre os termos. Nesse caso, quando a operação booleana *AND* é utilizada, é feita a interseção das listas invertidas e quando a operação *OR* é utilizada, faz-se a união das listas invertidas. Após essa operação, o sistema efetua a classificação (ranqueamento) dos resultados, utilizando os critérios anteriormente definidos na aplicação para poder uma lista de documentos relevantes para o usuário.

2.4. Computação distribuída

A necessidade de recursos computacionais de alto desempenho teve um grande crescimento a partir do momento em que a criação de conteúdo se deu de forma colaborativa, através da *web* 2.0, pois a quantidade de dados a serem armazenados e processados começaram a crescer exponencialmente (CONTI, 2009, p. 2). Essa demanda por recursos computacionais também é presente em outras áreas, tais como, meteorologia, astronomia, e genética, áreas que utilizam aplicações que executam cálculos de grande complexidade e repetição.

No entanto, o avanço tecnológico em alguns casos não consegue acompanhar essa demanda de alto poder de processamento e a aquisição de computadores avançados pode se tornar inviável financeiramente. Como solução para essa problema, passou-se a utilizar a soma de recursos computacionais pré-existentes de forma equilibrada e melhor aproveitada, desse modo possibilitando a execução de aplicações em ambiente distribuído, utilizando recursos de diversos computadores em paralelo.

Tanenbaum (2007, p. 2) afirma que, um sistema distribuído é uma coleção de computadores independentes que aparece para seus usuários como um sistema único e coerente. Essa definição possui dois aspectos importantes; primeiro, um sistema distribuído é composto de componentes (computadores) independentes e, segundo, os usuários (sejam eles pessoas ou programas) devem pensar que estão utilizando um único sistema.

Ainda sobre a definição de sistemas distribuídos, Colouris (2007, p.15) afirma que são conjuntos de computadores autônomos que possuem *softwares* para permitir o compartilhamento dos recursos do sistema: *hardware*, *software* e dados.

Em um sistema de computação distribuída, as diferenças entre os vários computadores e as formas em que eles se comunicam, bem como sua organização interna devem ser na sua transparentes para os usuários.

A escalabilidade do sistema distribuído é uma propriedade imprescindível para o que o sistema esteja continuamente disponível. A manutenção dessa escalabilidade também deve ser oculta para o usuário e aplicações, pois os mesmos não devem perceber que peças e/ou componentes do conjunto distribuído estão sendo substituídos ou em reparo, ou ainda que novas unidades estão sendo adicionadas para atender a mais usuários ou aplicativos.

Nos conceitos de computação distribuída destacam-se: cluster, grade computacional e computação nas nuvens. No entanto, na seção seguinte serão apresentados os tipos mais comuns de cluster.

2.4.1. Clusters

Segundo Buyya (1999 *apud* CONTI, 2009 p.2), "cluster é um tipo de sistema para processamento paralelo e distribuído que consiste de uma coleção de computadores interconectados e trabalhando juntos como um único recurso computacional integrado. Com esse conjunto de computadores é possível realizar processamentos que até então apenas computadores de alto desempenho conseguiam executar". A Figura 5 apresenta uma estrutura simples de cluster de computação distribuída.

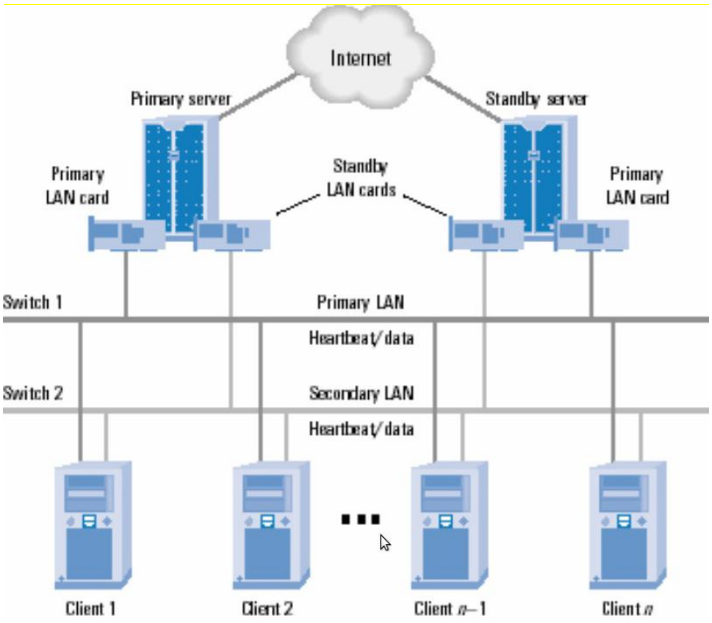

Figura 5 – Arquitetura de um Cluster (CONTI, 2009, p.3).

A Figura 5 ilustra a estrutura de um cluster, que basicamente é formado por vários computadores interligados através de uma interface de rede. Em um cluster, cada computador corresponde a um nó, sendo necessário manter a homogeneidade no padrão de configuração dos nós, além de uma padronização do sistema operacional utilizado, para que se obtenha uma menor complexidade na implantação, configuração e manutenção de um cluster. Fazem parte ainda da estrutura do cluster: *nobreaks*, cabos, *hubs* e *switches*. Na Figura 5, a interligação entre os nós é feita através de uma rede LAN sem topologia obrigatória, no entanto, diferentes tipos de redes podem ser utilizados para realizar a conexão entre clusters. Existem diversos tipos de cluster, três deles serão melhores detalhados a seguir.

2.4.1.1. Tipos de Cluster

Rodrigues (2007, p.10) afirma que a palavra cluster era relacionada a um sistema de alta performance, no entanto, atualmente existem tipos de clusters que não buscam essa característica. A seguir serão apresentados três tipos básicos de cluster.

2.4.1.2. Cluster para alta disponibilidade

O Cluster de alta disponibilidade tem o objetivo de evitar falhas de um determinado serviço ou a disponibilidade de dados que estejam em uma rede com trafego alto de informações. Também chamado de *failover*, esse tipo de cluster utiliza a replicação dos dados e ou serviços em seus nós (RODRIGUES, 2007, p.11). Essa redundância é intencional para que os serviços sempre estejam disponíveis, pois assim em uma eventual falha de um nó do cluster, automaticamente outro nó assume o serviço sem perda de dados e de forma transparente ao usuário.

Um exemplo para o cluster de alta disponibilidade é o de *storage,* um conjunto de servidores de armazenamento de dados. A Figura 6 apresenta o fluxo de um *failover*.

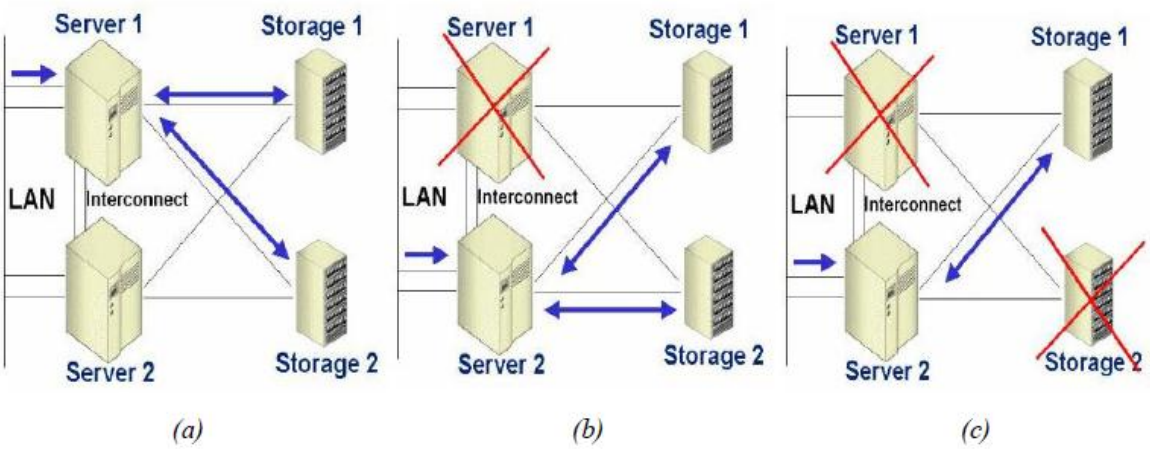

Figura 6 – Exemplo de um cluster de alta disponibilidade (RODRIGUES, 2007, p.12).

Na Figura 6 é ilustrado o fluxo da replicação de um *storage* em 3 etapas. Na situação (a) o sistema está normal, o *server1* recebe o fluxo da rede e faz a replicação das informações nos *storage1* e *storage2.* Em (b) ocorre uma falha no *server1* e nesse momento o *server2* assume o serviço e o armazenamento é replicado normalmente. Na etapa (c), ocorre uma falha mais crítica, no qual ocorrem falhas no *server1* e na mídia de armazenamento no *storage2*,

nesse caso o *server2* passar a responder as requisições e utilizar os dados replicados no *storage1*.

2.4.1.3. Cluster para balanceamento de carga

Esse tipo de cluster é utilizado para controlar a distribuição de carga entre os nós existentes do sistema. Através de um balanceador, as requisições são divididas entre os nós do cluster visando o uso balanceado dos recursos computacionais do cluster (RODRIGUES, 2007, p.12). Dessa forma, todos os nós recebem uma carga de serviço equivalente. A Figura 7 ilustra o esquema de um cluster de balanceamento da carga.

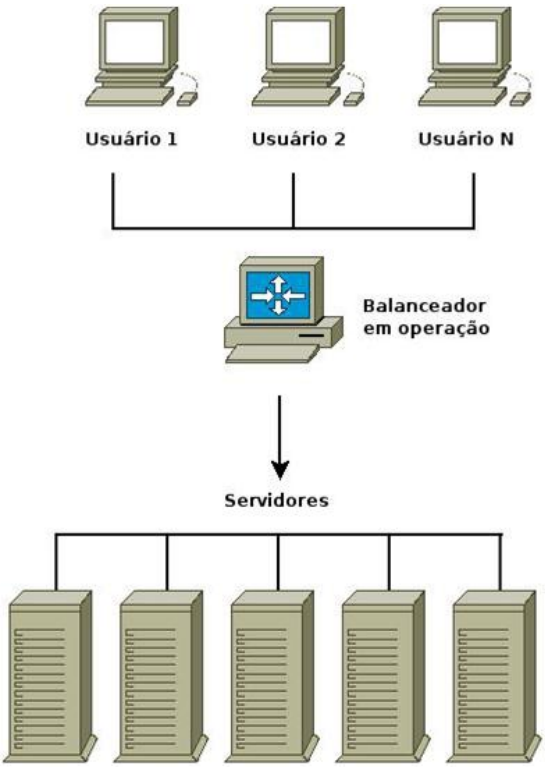

Figura 7 - Cluster Balanceamento de Carga (SILVA, 2011, online).

No esquema ilustrado na Figura 7, uma quantidade N de usuários acessam os serviços em um cluster com balanceamento de carga. O balanceador em operação atua monitorando os recursos computacionais dos servidores (nós) e distribui as requisições entre eles uniformemente.

2.4.1.4. Cluster para Alta Performance de Computação

Este tipo é mais comum e utilizado entre as comunidades científicas ou para tarefas que exigem alto poder de processamento, sendo que o cluster de alta performance é composto por

um nó (mestre) e vários nós escravos, sendo o primeiro o responsável por dividir o problema em vário pedaços para serem processados pelos nós escravos, que são exclusivos para realizar o processamento (RODRIGUES, 2007, p.10). Quando cada nó escravo consegue a solução, ele devolve seu fragmento do problema resolvido para o nó mestre montar a solução do problema. Esse processo é apresentado na Figura 8.

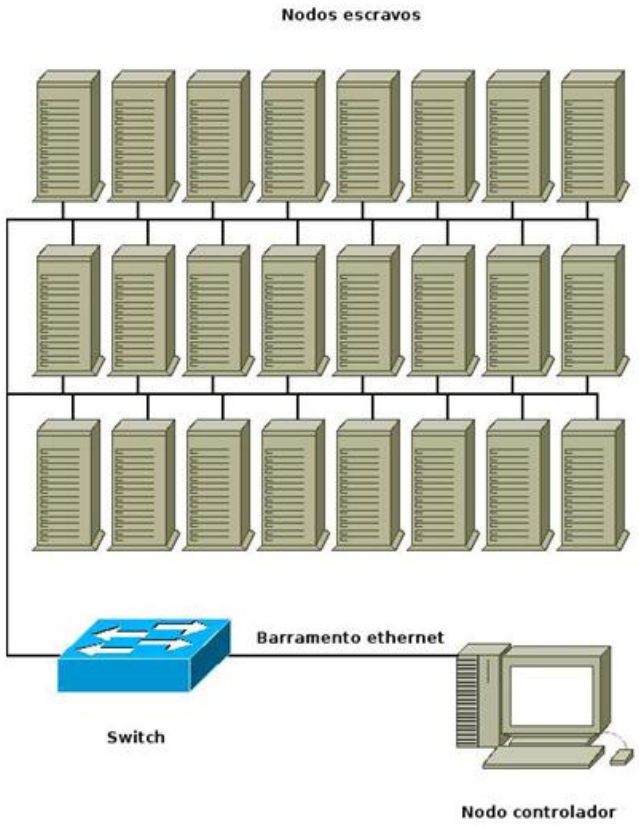

Figura 8 - Cluster para Alta Performance (SILVA, 2011, online).

A Figura 8 ilustra um cluster de alta performance com 24 nós exclusivos de processamento e um nó mestre. O mestre fica responsável por receber as requisições, distribui-las para os outros nós e devolver a resposta. Utilizando esse exemplo, um cálculo científico que necessite de grande poder computacional seria executado 24 vezes mais rápido nesse cluster do que em um único computador.

Para que seja possível executar aplicações em vários computadores paralelamente é necessário o uso de um *middleware*, pois assim é efetivamente aplicado o conceito de computação distribuída. No entanto, aplicações *P2P (peer-to-peer)* puras não necessitam de um *middleware* para compor um sistema de computação distribuída, pois funciona como uma

rede virtual, que é executada sobre a infra-estrutura de uma rede física. A próxima seção ilustra as propriedades de um *middleware* de computação distribuída.

2.4.2. Middleware

O *Middleware* é a camada responsável por fornecer as ferramentas que possibilitam aos vários computadores pertencentes a um cluster formarem um ambiente único, que seja configurável de forma simples e adaptável (CONTI, 2009, p.7). Esta camada permite que os usuários sejam eles pessoas ou sistemas possam interagir com os recursos computacionais de um sistema distribuído, ocultando a heterogeneidade e complexidade da infraestrutura, provendo uma interface padronizada de acesso a esses recursos.

Esta camada que intermedia o funcionamento do hardware e software em um cluster, ainda tem alguns funções importantes, tais como; monitoramento dos recursos, informação sobre desempenho do conjunto e ainda sobre a utilização dos recursos disponíveis, configurações de segurança e esquema de tolerância a falhas, alocação de processos do usuário aos recursos, submissão e terminação dos processos.

Fica a cargo do *middleware* gerenciar e distribuir os recursos do conjunto de nós contidos em um sistema distribuído, ou seja, o *middleware* pode ser responsável pela autenticação e autorização de acesso aos recursos, entrada e saída de dados e aplicações, execução de programas, localização de recursos e balanceamento de carga. A Figura 9 apresenta a forma como um *middleware* opera em um ambiente distribuído com quatro nós e três aplicações.

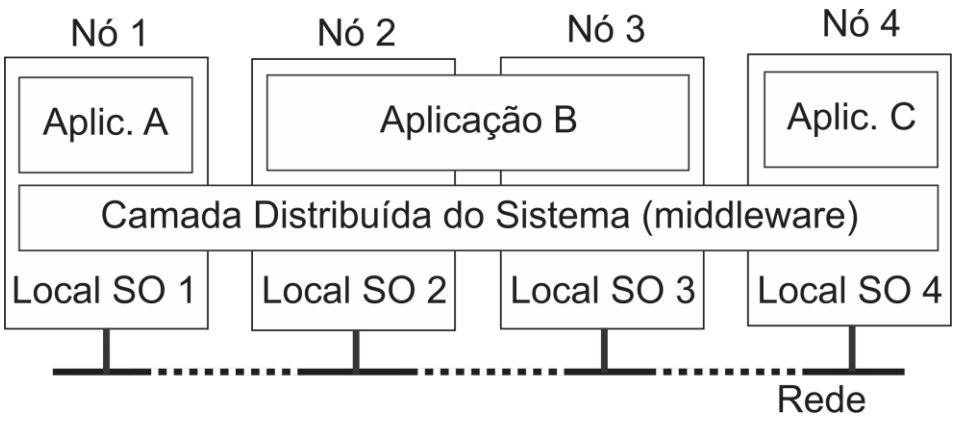

Figura 9 – Estrutura de um *middleware* – adaptado de (TANENBAUM e STEEN, 2007 p.3).

A Figura 9 exibe um cluster com quatro nós, onde são executadas três aplicações. A aplicação A no momento utiliza recursos apenas do nó 1, do mesmo modo a aplicação C

utiliza recursos apenas do nó 4, já a aplicação B necessitou distribuir sua execução entre os recursos dos nós 2 e 3.

O *middleware* permite que o sistema distribuído forneça meios para que os componentes de uma aplicação distribuída sejam executados em um ambiente com configurações distintas de hardware, pois oculta as diferenças de hardware e sistemas operacionais para cada aplicação.

A utilização de computação distribuída para indexação de dados se torna viável quando aplicada em grandes bases de dados. Para que essa estrutura de indexação distribuída funcione de forma efetiva, é necessária a utilização de índices distribuídos, assunto que será abordado na próxima seção.

2.4.3. Índices Distribuídos

A principal característica para uso de índices distribuídos para indexação de termos em um Sistema de Recuperação de Informação é a escalabilidade que o ambiente distribuído oferece. Entende-se por escalabilidade o poder em que o sistema tem de se adaptar a situações em que podem prejudicar o desempenho do sistema no que tange capacidade de processamento de hardware, ou seja, a capacidade que a estrutura tem de acompanhar o crescimento do uso da aplicação (GOMES, D., 2010. *online*).

Outro fator que incide grande importância na utilização de índices distribuídos consiste na utilização de SRIs em grandes coleções de dados, já que a soluções de índices normais em sistemas rodando em *single-servers* utilizam o arquivo invertido como estrutura de dados para indexação dos termos dos documentos.

Utilizar o arquivo invertido como estrutura de dados para indexar grandes coleções de documentos, pode afetar o desempenho do sistema de R.I, pois esse tipo de estrutura adquire tamanho em cerca de 50% (cinquenta por cento) menor que tamanho da coleção indexada. Assim, considerando uma coleção de documentos com 1 *terabyte* de tamanho, seu índice invertido teria o tamanho em cerca de 500 *gigabytes.* Essa grande quantidade de dados necessita de uma quantidade considerável de recurso computacional, principalmente no que tange quantidade de memória RAM e espaço em disco. Esse fator prejudica diretamente a capacidade de escalabilidade que o conjunto possa ter, pois um servidor para executar uma aplicação desse porte teria que ter uma configuração muito avançada.

Ao utilizar sistemas distribuídos para indexar grandes quantidades de dados, aumenta-se a escalabilidade do conjunto, pois no momento em que uma aplicação necessita de mais poder computacional, pode ser resolvido adicionando-se mais nós no cluster.

Uma característica importante é a forma como é tratada a separação entre máquinas que recebem solicitações e máquinas que de fato processam tais solicitações. As máquinas que recebem requisições são chamadas de *broker*, pois possuem as informações sobre a estrutura e organização do sistema distribuído, atuando também como *front-end*, responsável por redirecionar as requisições para as outras máquinas. O *broker* não armazena o índice, embora em algumas soluções armazene o vocabulário da coleção. As máquinas que recebem as requisições do *broker* estão localizadas no *back-end*, são os *index servers*, as máquinas que realmente processam as consultas realizadas (AUGUSTO, 2010, p.15).

A utilização de índices distribuídos visa sanar o problema em que o índice gerado para determinada coleção de documentos indexada se torne grande demais para caber na memória principal de uma única máquina. Isso é possível, pois cada nó processa o seu arquivo invertido independentemente dos outros nós e retorna o resultado para que o *broker* junte as informações e entregue a informação buscada para o usuário.

Na seção seguinte, são apresentados os quatro tipos mais comuns de índices distribuídos;

2.4.3.1. Índice Compartilhado

O índice compartilhado tem como base os sistemas de bancos de dados, no qual os índices se encontram apenas em uma máquina que recebe as requisições. Nesse caso, a escalabilidade continua prejudicada, pois apenas uma máquina continua sendo responsável pelo índice da coleção de dados (AUGUSTO, 2010, p.16).

A Figura 10 mostra um esquema utilizando índices compartilhados.

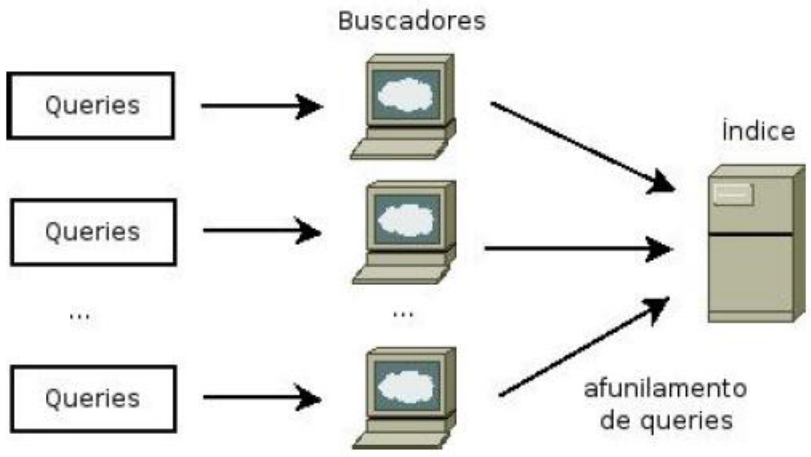

Figura 10 - Índice Compartilhado (AUGUSTO, 2010, p.16).

Funcionando como um banco de índices, a máquina responsável por prover os índices recebe requisições de diferentes tipos de buscadores (*brokers).* Essas requisições podem vir de um serviço na web ou da rede local. Como é visto na Figura 10, ocorre o afunilamento de *queries*, podendo acarretar em sobrecarga no servidor de índices.

2.4.3.2. Índice Replicado

No índice replicado, os índices são replicados em todos os *index-servers* do sistema. Dessa forma, as consultas podem ser igualmente divididas entre todos os nós. O *broker* fica responsável por dividir as consultas, podendo utilizar um algoritmo de *round-robin* ou por meio de pesos, por exemplo, uma busca tem 10 termos pode se dividir para 10 nós (AUGUSTO, 2010, p.17). Na Figura 11 é apresentado o modelo de índice replicado.

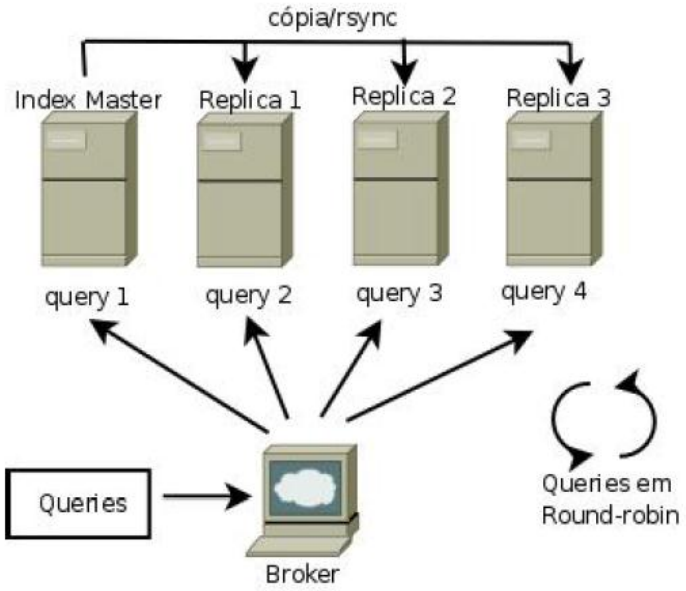

Figura 11 - Índice Replicado (AUGUSTO, 2010, p.17).

Na Figura 11 é possível ver o *index master* - nó indexador principal do sistema, após receber o índice de novos documentos, ele é replicado para os outros nós do sistema. O *broker* é o responsável por consultar as *queries* nos servidores replicados de índices e devolver os resultados da consulta.

2.4.3.3. Índices Locais

Nos índices locais, cada nó do sistema é responsável por indexar apenas parte da coleção de documentos, criando vários índices pequenos. No entanto, o balanceamento e adição de novos nós necessitam de uma reconstrução completa em todo índices do sistema, para que a coleção

de documentos seja redividida entre os nós e o índice seja recalculado. Esse processo deve ser realizado com o sistema parado, o que inviabiliza seu uso em sistema de produção de alto fluxo de dados (AUGUSTO, 2010, p.18). A estrutura de índices locais é apresentada na Figura 12.

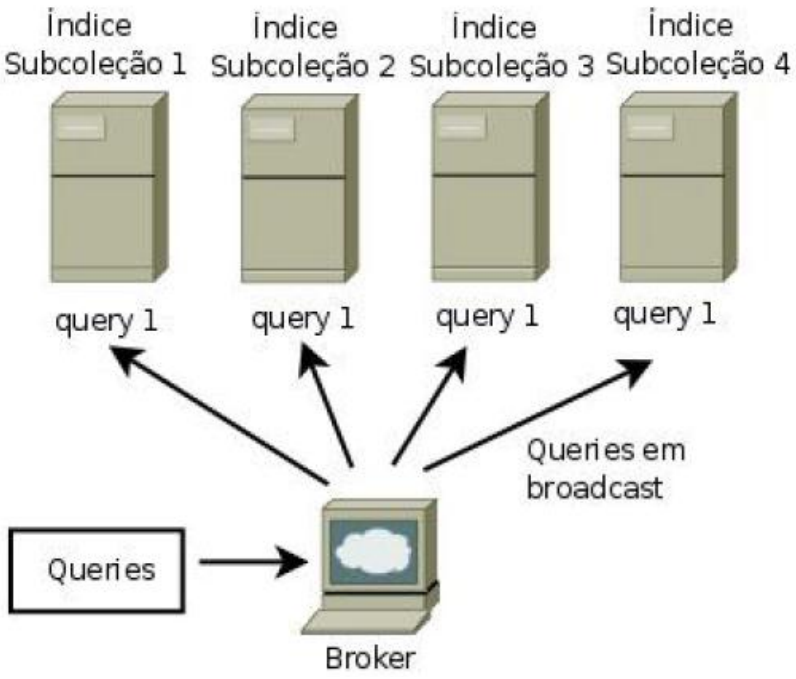

Figura 12 - Arquitetura de índices locais (AUGUSTO, 2010, p.19).

A Figura 12 apresenta o modo de funcionamento dos índices locais. Nesse esquema, cada nó conhece somente a parte que recebe da coleção de documentos. Dessa forma, cada busca deve ser retransmitida em *broadcast* para todos os nós do cluster. Cada nó tem a tarefa de indexar apenas a parte de texto recebido, o que incide na criação de índices pequenos e, consequentemente, listas invertidas também menores (ROCHA *et al.* 2006, p4**)**. Essa estrutura tem condições para suportar grandes aumentos na quantidade de coleções de texto, pois esses dados são divididos entre os nós do cluster de forma muito mais consistente, já que esse fator de crescimento é dividido por entre as máquinas do cluster.

2.4.3.4. Índice Global

O índice global tem sua estrutura baseada nos índices locais, sendo diferente na parte em que cada nó é responsável por um intervalo de termos do vocabulário. Dessa forma, as requisições não necessitam mais ser enviadas para todos os nós *index-servers* do sistema, mas apenas para aqueles que contenha termos presentes na consulta requisitada. Isso possibilita que os nós que

estejam ociosos possam ser utilizados em outras consultas em paralelo (ROCHA *et al.* 2006, p.4). Como nos índices locais, para manter o índice equilibrado, no índice global é necessário fazer periodicamente uma reindexação completa da coleção de documentos. Na Figura 13 é ilustrada a estrutura do índice global.

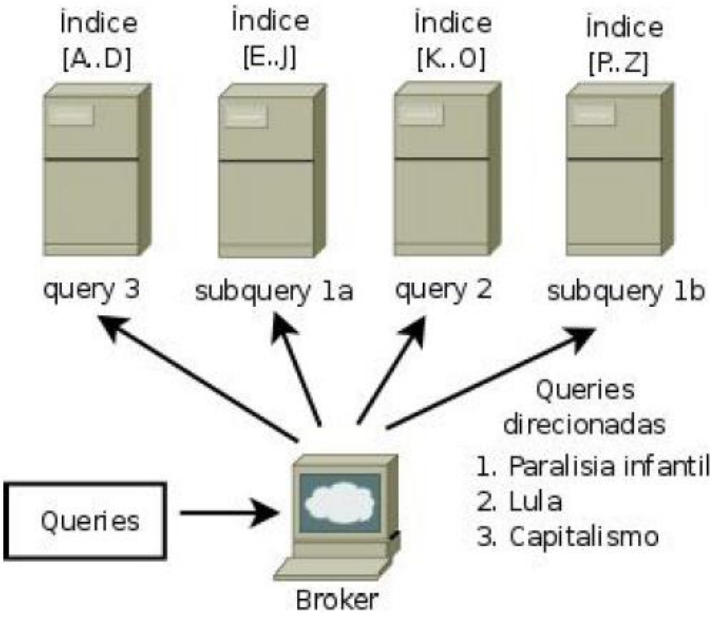

Figura 13 - Índice Global (AUGUSTO, 2010, p.20).

No índice local, cada nó do sistema cria um índice local da coleção de documentos que é responsável. O *broker* recolhe de cada estação e cria uma lista com todos os termos indexados, criando um vocabulário global. Cada nó que contém o índice recebe uma lista invertida com o intervalo de termos pelo qual será responsável. Assim, o *broker* redirecionará a requisição para o nó indexador de acordo com os termos contidos no vocabulário.

A seção seguinte apresenta o Apache Lucene, uma biblioteca desenvolvida em Java para indexação e busca, que em conjunto com um *middleware* de computação distribuída implementa os índices distribuídos.

2.5. Apache Lucene

O apache Lucene é uma biblioteca *open-source* para indexação e busca, criada em Java por Doug Cuttingol no ano 2000, posteriormente foi aprimorada e depois incorporada como subprojeto pela Apache Foundation[2]. Atualmente, o Lucene é um projeto próprio da Apache Foundation (LUCENE, 2012, *online*).

[2] http://www.apache.org/

O Lucene pode ser integrado em qualquer aplicação que necessite de um sistema rápido, altamente escalável e robusto de buscas, pois possui implementações de *ports* para outras linguagens de programação. O *ports* é um subprojeto do Lucene que disponibiliza APIs compatíveis para indexação e busca a partir de outras linguagens de programação.

Ao utilizar as funcionalidades do Lucene, o programador não necessita de implementar algoritmos de busca e classificação, pois a biblioteca dispõe de mecanismos calcular a pontuação de cada documento que corresponda à uma consulta e retornar documentos relevantes de acordo com essas pontuações, possui suporte a consultas como, *PhraseQuery*, *WildcardQuery*, *RangeQuery*, *FuzzyQuery*, *BooleanQuery*, além de permitir busca e indexação simultaneamente (GOSPODNETIC, 2005, *online*).

Para o cliente que faz a requisição de uma busca, o processo de pesquisa é efetuado apenas em uma base de índices indexada pelo Lucene, que faz uma lista de resultados relevantes ao contexto da busca. A Figura 14 apresenta essa estrutura que o Lucene utiliza.

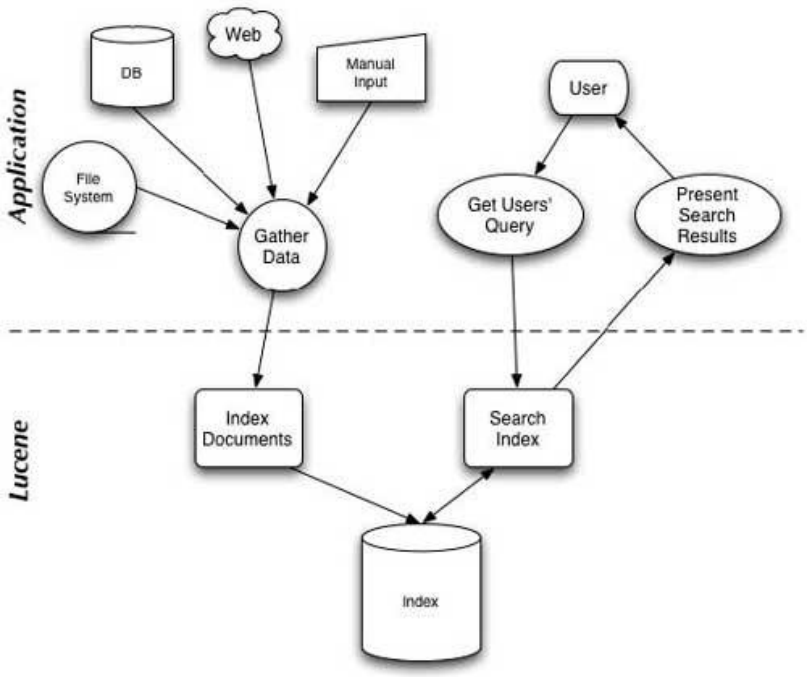

Figura 14 - Estrutura de utilização do Lucene (GOSPODNETIC, 2005, *online)*.

No Lucene, o índice é formado por duas estruturas lógicas: documentos e campos (*fields*). Cada posição no índice corresponde a um documento (org.apache. lucene.document.Document), e cada documento tem um conjunto de campos (**org.apache. lucene.document.Field**). Desse modo, fazendo uma comparação com um banco de dados

relacional, os documentos são como as linhas de uma tabela, e os campos são como as colunas, nesse contexto, os termos correspondem aos campos de um documento.

O Lucene consegue indexar todos os tipos de dados que podem ser convertidos em texto, como por exemplo, PDF, HTML, DOC. Assim para iniciar o processo de indexação, deve ser feito a extração do texto e enviado as informações para o Lucene processe e grave o arquivo de índices. No entanto, esse processo de extração textual deve ser realizado de externamente ao Lucene, pois a biblioteca não possui estrutura para realizar esse procedimento. A Figura 15 ilustra a arquitetura de indexação do Lucene.

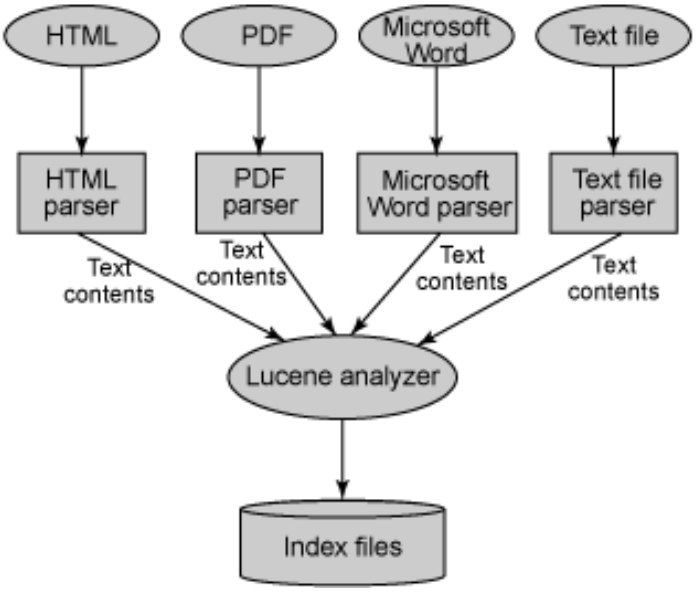

Figura 15 - Componente Analyzer do Lucene (ZHOU, 2006, *online*).

A Figura 15 mostra como é executado o processo de extração textual para indexação do Lucene. São usados diferentes analisadores diferentes tipos de documentos. Por exemplo, em um documento HTML o analisador faz o pré-processamento, como filtragem das tags HTML, bem como em um documento em formato PDF pode executar o processo de remoção de *stopwords*. A classe responsável por esse processo é a *apache.lucene.analysis.Analyzer*, essa implementação é feita por *default* no Lucene, porém, é possível utilizar a classe *StandardAnalyzer* para implementar analisadores em idiomas diferentes e de forma adaptável ao SRI.

Após a indexar o índices, o Lucene armazena-os no disco, esse processo será executado utilizando o sistema de arquivos HDFS do Apache Hadoop utilizando o processo *map/reduce* em um sistema distribuído. O Apache Hadoop é abordado na próxima seção.

2.6. Apache Hadoop

O Apache Hadoop é uma plataforma de computação distribuída que possibilita processamento e armazenamento de grandes quantidades de dados, com suporte para clusters com milhares de nós. Foi inspirado inicialmente no sistema de arquivos do Google, o GFS (*Google File System*) e em seu processo *map/reduce* (HADOOP, 2012, *online*).

O GFS (*Google File System*) é um sistema de arquivos escalável desenvolvido para aplicações distribuídas de tráfego intenso de dados. O armazenamento dos dados é feito em blocos, definidos como *chunks*.

Os clusters GFS são formados por um único nó mestre, chamado de *master*, que é responsável por controlar o cluster e informar a localização de um *chunk* em um *chunkserver.* Os *chunkservers* são os nós do cluster responsáveis por armazenar os chunks. Um cluster GFS pode ter uma quantidade qualquer de *chunkservers*, que podem ser acessados simultaneamente por vários clientes.

O *master* é replicado para outras máquinas, pois caso falhe, uma das réplicas passa a responder por *master*. Isso acontece, pois é o *master* que contém os *namespaces* com os arquivos armazenados no GFS e em qual *chunk* está guardado.

Dessa forma, quando um dado é requisitado, essa requisição é feita ao *master*, que então passa a localização do *chunkserver* em que se encontra o arquivo. No caso em que o arquivo esteja replicado em vários *chunkservers*, o *master* decide qual deles ira repassar para o cliente, acontecendo o balanceamento de carga do sistema.

Os blocos de armazenamento de arquivos do GFS, os *chunks*, são implementados com base no sistema de arquivos do Linux e possuem tamanho de 64MB e endereçamento de 64 bits (AUGUSTO, 20120, p.29). O tamanho desse *chunk* contribui para que o tamanho do metadado do GFS seja menor, além de diminuir a quantidade de conexões TCP para consulta ao *chunkserver.*

O Hadoop utiliza o sistema de arquivos HDFS (*Hadoop Distributed File System*), implementado com base no GFS, garantindo segurança, disponibilidade e integridade dos dados distribuídos, otimizando o desempenho para processamento de grandes quantidades de dados, além de replicação e balanceamento de carga.

A Tabela 1 apresenta a comparação da terminologia entre o Google GFS e o Hadoop.

Tabela 1 - Equivalência entre componentes do Google GFS e Hadoop.

Google	Hadoop
Master	***JobTracker***
Worker	***TaskTracker***
GFS *Master*	**HDFS *NameNode***
GFS *Chunkserver*	**HDFS *DataNode***

Um cluster Hadoop possui um *NameNode* e diversos *DataNodes*, (o *NameNode* é equivalente ao mestre do GFS e o *DataNode* equivalente ao *chunkserver*). Os arquivos são armazenados em *blocks* nos *DataNodes.* O *NameNode* possui um *namespace* para associar ao nomes de arquivos, é também tarefa do *NameNode* decidir e redirecionar as replicas dos *blocks* para as requisições. A Figura 16 ilustra o funcionamento interno do sistema de arquivos do Apache Hadoop.

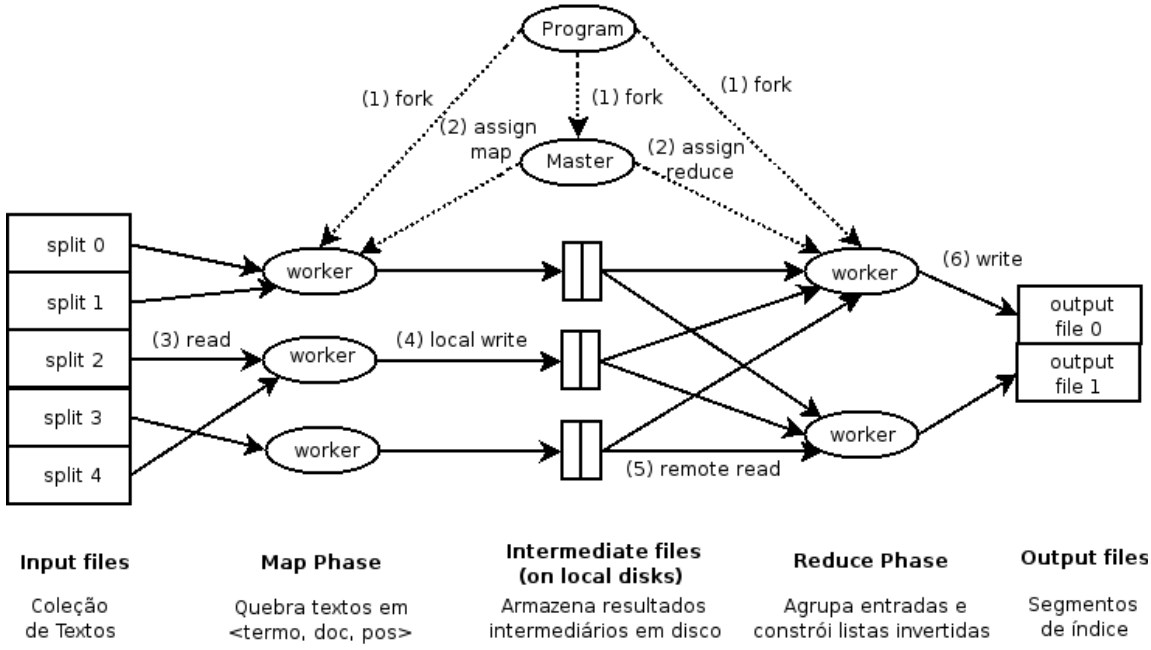

Figura 16 - Processo map/reduce (AUGUSTO, 2010, p.25).

Na Figura 16 é apresentada a execução do processo *map/reduce*, que faz o sistema de arquivos do Hadoop efetivamente funcionar. O *map/reduce*, desenvolvido pelo Google, é um modelo de programação que permite implementar uma camada de abstração para a processamento de alto desempenho para executar tarefas com grandes quantidades de dados, principalmente para sistemas de recuperação de informação (AUGUSTO, 2010, p.25).

O *map/reduce* implementa os mecanismos de comunicação, paralelismo, escalabilidade, troca de dados, e de onde e como cada código deve ser executado, sendo necessário apenas a definição das funções, *map* e *reduce*.

A função *map* tem o objetivo de percorrer os dados de entrada e selecionar informações que sejam relevantes para o sistema. A função *reduce* recebe os dados da saída da função *map* e tem o papel de resumir as (incluir, filtrar ou agrupar) informações e devolver para o sistema. O processo *map/reduce* é o ponto em que efetivamente o ambiente se torna distribuído, pois é quando ocorre a distribuição e paralelização do código, onde os dados de entrada são divididos em N estações que realizarão a função *map* que posteriormente serão reutilizados em M estações possivelmente diferentes das estações anteriores para realizar o processo *reduce.*

O Hadoop trabalha de forma diferente quanto à forma de se distribuir a aplicação. Aplicações habituais em ambiente distribuído movem o dado para ser processado em vários nós. O Hadoop trabalha ao contrário: os dados são segmentados em N partições, e então é movido a computação para cada um dos nós do cluster. Isso acontece através do uso do sistema distribuído de arquivos próprio do Hadoop (HDFS) que gera um *jar* contendo a implementação da função *map/reduce* necessário para processar o dado segmento anteriormente.

A função *map/reduce* se encaixa perfeitamente para soluções de indexação de grandes quantidades documentos, onde os índices seriam divididos por meio da função *map*, e quando requisitados seriam reunidos pela função *reduce.* O *framework* Katta disponibiliza uma integração entre o sistema de arquivos distribuído do Hadoop e a biblioteca de buscas do Lucene para indexação de termos em ambiente distribuído. Na seção seguinte serão abordadas as definições do Katta.

2.7. Projeto Katta

O Katta é um *framework open-source* de computação distribuída, que possibilita a implementação da biblioteca do Apache Lucene em ambiente distribuído. É o *middleware* que efetivamente possibilita a utilização de recursos computacionais de vários nós de um cluster. A partir de seu uso, é possível utilizar o poder do *framework* Lucene para busca e indexação, juntamente com o sistema de arquivos do *framework* Hadoop para indexação de documentos em ambiente distribuído.

Mantido pela empresa alemã 101Tec[3] inc, o *framework* Katta é uma solução escalável e tolerante a falhas para armazenamento distribuído de dados para acesso em tempo real. O *framework* possui estrutura para replicação de dados, fragmentação de índices e suporte para manipular coleções de documentos de grande porte. A implementação atual do Katta é compatível com o sistema de índices do Lucene e sistema de armazenamento *MapFile* do Hadoop.

Para manter a escalabilidade do conjunto, o *framework* faz a replicação dos fragmentos dos índices em todos os nós (clusters) do sistema, para melhorar o desempenho e tolerância a falhas. Possui ainda um mecanismo de *fail-over*, processo em que quando um nó falha em realizar seus serviços, outro nó assume os serviços e os processa. A Figura 17 apresenta a visão do funcionamento do sistema distribuído implementado pelo Katta.

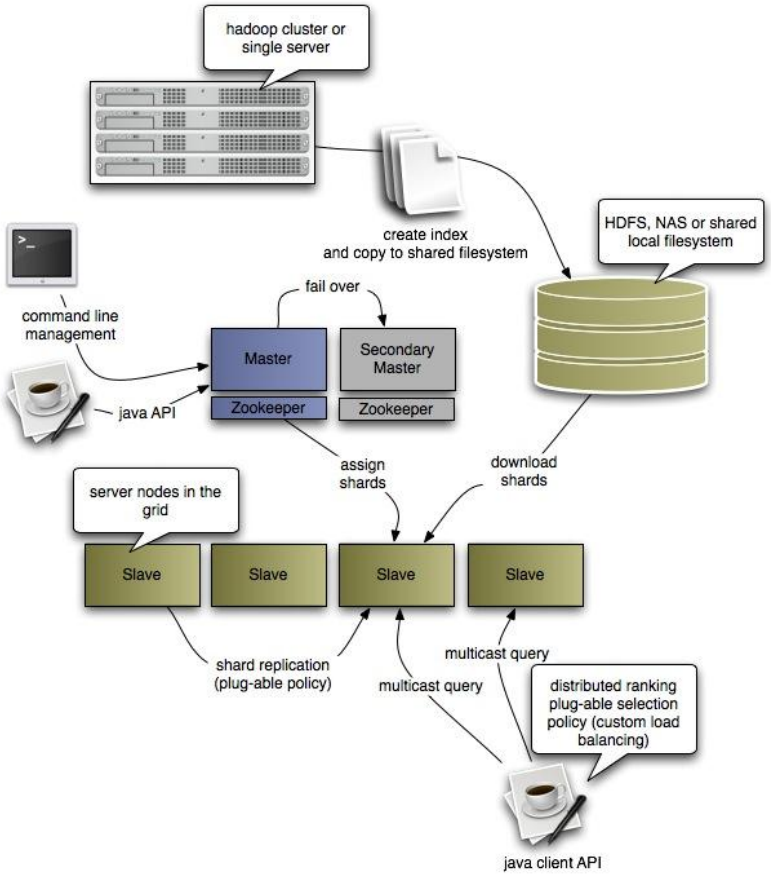

Figura 17 - Estrutura de funcionamento do *framework* (KATTA, 2009, *online*).

Na estrutura do Katta, ilustrada pela Figura 17, são apresentados os processos para o funcionamento pleno do sistema distribuído. A partir de uma interface de administração via

[3] http://101tec.com

linha de comando ou utilização de uma API em Java, são enviadas as requisições de busca para um servidor *master* responsável por dividir as requisições para os servidores *slaves*. O grupo de servidores *slaves* por sua vez consultam os índices indexados pelo Lucene no sistema de arquivos do Hadoop e devolvem o resultado para o servidor *master*.

A execução do Katta necessita que exista um ambiente previamente configurado com sistema operacional Linux e o cluster Hadoop funcionando plenamente.

3 MATERIAIS E MÉTODOS

Nesta seção são apresentados os matérias e métodos utilizados para o desenvolvimento deste trabalho, sendo subdividida da seguinte maneira: Local e Período, Materiais e Metodologia.

3.1. Local e período

Este trabalho foi desenvolvido nos laboratórios do complexo de informática do CEULP/ULBRA e em residência, na cidade de Palmas – TO. O período de desenvolvimento teve início no segundo semestre de 2012 e conclusão no primeiro semestre de 2013.

3.2. Materiais

Os materiais utilizados são divididos em: software e fontes de pesquisa.

3.2.1. Fonte de pesquisa

As fontes de pesquisa utilizadas no embasamento deste trabalho foram: artigos, dissertações de mestrado, teses de doutorado, trabalhos de conclusão de curso e páginas *web*, todos coletados através da internet.

3.2.2. Software

Para produção deste trabalho foram utilizados diversos softwares, tais como:

- Virtualização:
  - Vmware Workstation 9. (http://www.vmware.com/virtualization/);
- Cliente Windows SSH:
  - SSHSecureShellClient-3.2.9 (http://www.ssh.com/), *software* cliente para conexão SSH nos servidores Linux ;
- Monitor de desempenho para sistemas Linux:
  - Nmon (http://nmon.sourceforge.net)
- *Frameworks* para indexação e computação distribuída:
  - Apache Lucene – API para indexação de documentos;

    - Apache Hadoop – *Framework* que implementa o armazenamento distribuído;
    - Katta – *Framework* que possibilita a implementação dos índices da API Lucene em sistemas distribuídos.
- iText – Biblioteca para manipulação de documentos em formato pdf (http://itextpdf.com/) .
- Sistema Operacional
    - Linux Ubuntu Server 10.04 LTS (http://www.ubuntu.com/);
- Linguagem de Programação:
    - Java, versão JDK 1.6 (http://www.oracle.com/technetwork/java/)
- Desenvolvimento:
    - Eclipse Juno: IDE para desenvolvimento em java (http://www.eclipse.org/juno/).
    - Tomcat 6: Servidor para *Java Server Pages* (http://tomcat.apache.org/).

A seguir, são detalhados os cenários utilizando máquinas virtuais e a configuração e instalação dos *frameworks* utilizados.

3.2.3. Criação e configuração dos cenários

A máquina física que recebe a instalação do sistema de virtualização é chamada de *host*, e as máquinas virtuais executando sobre o *host*, são chamadas de *guests.* Cada máquina criada para utilização nesse trabalho recebeu uma configuração igual. Na Figura 18 é apresentada a tela de configuração de máquina virtual do software de virtualização Vmware *workstation* 9.

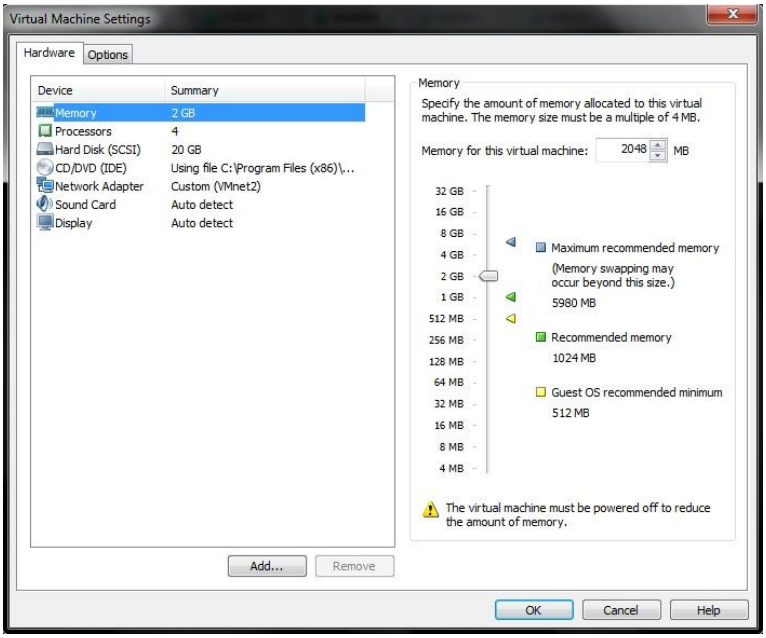

Figura 18 - Recursos de máquina virtual.

Na Figura 18 são exibidos os recursos disponíveis de uma máquina virtual. Essa configuração é padrão para todas as máquinas virtuais criadas para utilização no trabalho: 2GB de memória RAM, disco rígido no tamanho de 20GB e uma placa de rede. O sistema operacional instalado é o Linux Ubuntu Server 10.04 LTS.

O primeiro ambiente virtual criado possui apenas um servidor simples de indexação, ou seja, executando apenas uma máquina virtual. A Figura 19 ilustra o cenário criado.

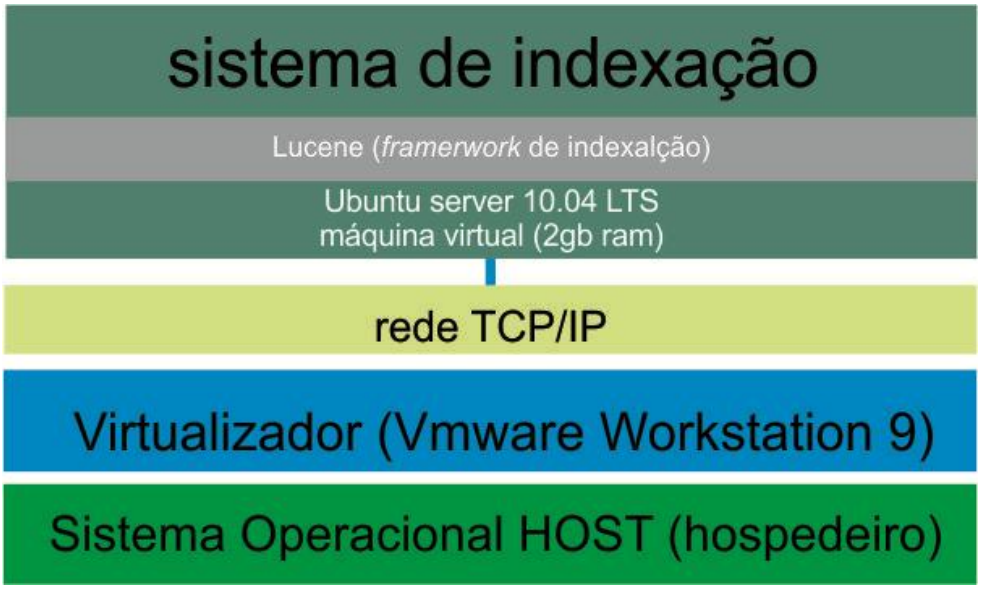

Figura 19 - Ambiente com servidor único.

No cenário apresentado na Figura 19, um servidor com 2GB de memória RAM é virtualizado utilizando o Vmware. Esse servidor roda o sistema operacional Ubuntu 10.04 LTS e o sistema de indexação utilizado é o Apache Lucene. Este cenário é utilizado para executar a indexação utilizando o Apache Lucene em um servidor simples.

O segundo cenário criado dá suporte para execução do cluster de computação distribuída. Para tal, foi definida a quantidade de quatro nós, sendo assim, quatro máquinas virtuais. A Figura 20 representa graficamente esse cenário.

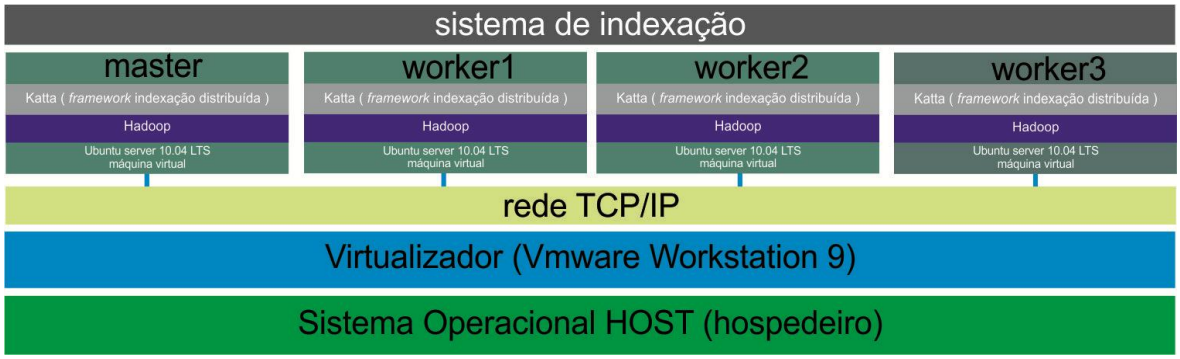

Figura 20 - Ambiente virtual computação distribuída.

No segundo ambiente virtual criado, como apresentado na Figura 20, o cluster de computação distribuída é composto por quatro nós, caracterizados por quatro máquinas virtuais emuladas através do Vmware Workstation 9. Neste cenário, cada nó recebe a instalação do Hadoop e do Katta que resulta no cluster de indexação. No cenário 1, é apenas utilizado o Lucene para indexação. A partir da configuração do ambiente virtual concluída, é necessário configurar o *framework* que atua como *middleware* de computação distribuída. Na seção seguinte é apresentada a configuração do Apache Hadoop.

3.2.4. Hadoop

O Hadoop é o *framework* que possibilita a utilização de armazenamento distribuído de arquivos. Os pré-requisitos para sua execução são: sistema operacional Linux e o Java JDK configurado. Como apresentado anteriormente, nesse trabalho são utilizados o Linux Ubuntu Server 10.04 LTS e o Java versão 1.6. A versão do Hadoop utilizada é a 1.0.6.

No Hadoop existem dois tipos de nós: *master* e *slave*. Na Tabela 2 é apresentada a lista de nós existentes no cluster criado para o trabalho e o tipo de cada um.

Tabela 2 - *Hosts* do cluster

Nome do *host*(nó)	**Tipo de nó**
master	*Master*
worker1	*Slave*
worker2	*Slave*
worker3	*Slave*

Apesar de cada nó do cluster estar configurado com um IP estático, é necessário definir um nome do *host*, pois a comunicação do Hadoop entre os nós utiliza-o como configuração. Essa comunicação é feita automaticamente através de conexão SSH entre os nós. No entanto, é necessária a criação de usuário e grupo específico para uso do Hadoop em cada nó para permitir a execução do *framerwork* e criação das chaves publicas para conexão do SSH. Esse procedimento foi realizado executando os seguintes comandos no console do Linux:

Tabela 3 - Criação de usuário no Linux.

```
1  sudo addgroup hadoop
2  sudo adduser --ingroup hadoop hduser
```

Depois de concluída a criação do grupo e usuário, é feita a geração da chave pública para conexão entre os nós via SSH. Esta ação é exibida na Figura 21.

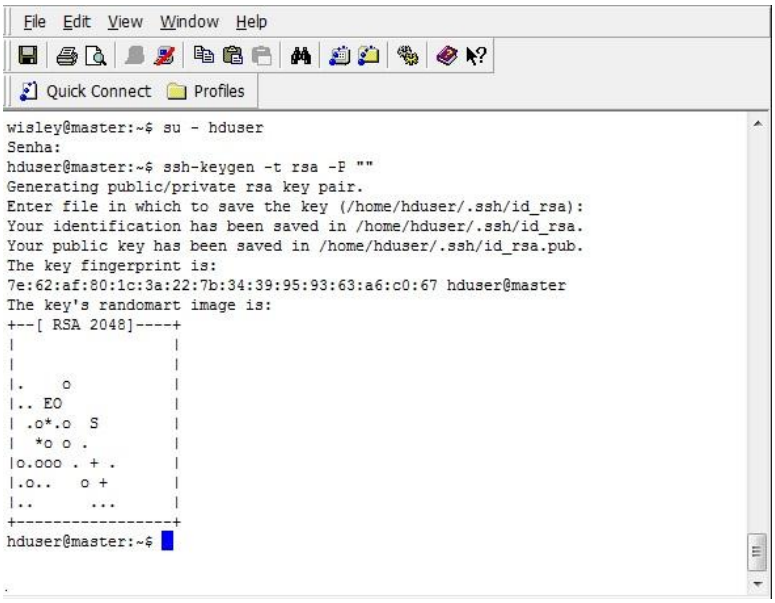

Figura 21 - Criação chave SSH

A chave é criada sem senha para que seja feita a conexão sem interrupções entre os nós do Hadoop. Após criada a chave, deve ser permitido acesso SSH para a máquina local executando o seguinte comando apresentado na Tabela 4:

Tabela 4 - Comando para permissão ssh.

1	`cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys`

Este procedimento apresentado na Tabela 4 deve ser executado em todos os nós do sistema, independente do seu tipo, seja *master* ou *slave.* Com a chave devidamente criada, é feita a cópia da chave para cada nó *slave* do cluster. Os comandos executados são ilustrados na Tabela 5:

Tabela 5 – Cópia das chaves para os nós.

1	`hduser@master:~$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@worker1`
2	`hduser@master:~$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@worker2`
3	`hduser@master:~$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@worker3`

Concluída a configuração da comunicação, são necessárias algumas alterações nas configurações do Hadoop. Os arquivos do Hadoop foram colocados no diretório `/usr/local/hadoop/` e foi dada permissão de leitura e escrita para o usuário criado anteriormente `hduser`. Ainda com o usuário `hduser` foi criado o diretório `/app/hadoop/tmp`, esse diretório armazena os dados processados pelo hadoop.

O nó designado como *master* deve receber as configurações sobre a estrutura do cluster, adicionando o nome de cada nó no arquivo `/usr/local/hadoop/conf/slaves`. Cada linha representa o nome do *host* de um nó. O conteúdo do arquivo é apresentado na Tabela 6.

Tabela 6 – Conteúdo do arquivo de configuração `slaves` do Hadoop.

1	`master`
2	`worker1`
3	`worker2`
4	`worker3`

O arquivo de configuração na Tabela 6 contém quatro nós, esse arquivo deve conter em cada linha, o nome de cada nó do cluster.

Para que os comandos do Hadoop funcionem, é preciso adicionar as variáveis de ambiente na inicialização do sistema editando o arquivo `$HOME/.bashrc`. O trecho a ser adicionado é exibido na Tabela 7.

Tabela 7 – Trecho de código adicionado ao arquivo `$HOME/.bashrc`.

```
1  # Diretorio do hadoop
2  export HADOOP_HOME=/usr/local/hadoop
3
4  # Diretorio do Java
5  export JAVA_HOME=/usr/lib/jvm/java-6-sun
6
7  # Funcoes e comandos do hadoop
8  unalias fs &> /dev/null
9  alias fs="hadoop fs"
10 unalias hls &> /dev/null
11 alias hls="fs -ls"
12 #compressao lzohead
13 lzohead () {
14     hadoop fs -cat $1 | lzop -dc | head -1000 | less
15 }
16
17 #Adicionando diretorio do hadoop na variavel PATH
18 export PATH=$PATH:$HADOOP_HOME/bin
```

Depois de finalizadas as configurações do Hadoop, o cluster está pronto para receber o *framework* de indexação distribuída. Na seção seguinte é apresentada a configuração do *framework* Katta.

3.2.5. Katta

O Katta é o *framework* utilizado na computação distribuída que implementa os índices do Apache Lucene em um sistema de arquivos HDFS. Os requisitos do *framework* são: sistemas de arquivos de arquivos HDFS previamente configurados (Hadoop) e o Java (Java 1.6)

A comunicação entre os nós utiliza SSH da mesma forma que Hadoop e assim utiliza o mesmo usuário e grupo criados além das chaves públicas do SSH.

Os arquivos do Katta estão situados no diretório `/usr/local/katta/` que possui permissão de leitura e escrita para o usuário `hduser`. Para que o *framerwork* reconheça os nós do cluster, é necessária a alteração do arquivo `/usr/local/katta/conf/masters`, exibido na Tabela 8.

Tabela 8 - Conteúdo do arquivo de configuração `masters` do Katta

1	`master`

Na Tabela 9 é exibido o conteúdo do arquivo de configuração do Katta `/usr/local/katta/conf/nodes.`

Tabela 9 - Conteúdo do arquivo de configuração `nodes` do Katta

1	`worker1`
2	`worker2`
3	`worker3`

No arquivo de configuração apresentado na Tabela 9, cada linha deve conter o nome do *host* que será um nó do cluster do Katta.

É necessário, ainda, configurar o diretório do Java no arquivo `/usr/local/katta/conf/katta-env.sh`, adicionando a seguinte linha, como é exibido na Tabela 10.

Tabela 10 - Conteúdo do arquivo de configuração `katta-env.sh` do Katta

1	`export JAVA_HOME=/usr/lib/jvm/`

Para utilizar os ambientes de indexação criados, foi desenvolvido um sistema de indexação para a automatização da indexação utilizando os *frameworks* apresentados. A seguir é apresentada a metodologia para o desenvolvimento do trabalho.

3.3. Metodologia

Para facilitar o entendimento, a metodologia deste trabalho foi dividida em etapas. A primeira etapa consistiu na busca de informações sobre Recuperação da Informação e Computação Distribuída. Esse processo ocorreu por meio da coleta de trabalhos científicos na *web*, tais

como artigos científicos, dissertações de mestrado e teses de doutorado. Assim, com base nas informações adquiridas, foi confeccionada a revisão de literatura do trabalho, abordando os conceitos sobre a Recuperação da Informação, Sistemas de Recuperação de Informação e seus processos, Computação Distribuída e os tipos de Índices Distribuídos.

A etapa seguinte consistiu no desenvolvimento prático deste trabalho que, para facilitar o entendimento, foi dividida em sub-etapas, as quais são descritas a seguir:

- Criação da base de documentos: esta base foi utilizada no processo de indexação e foi criada a partir de documentos do acervo de edições em formato pdf do diário oficial do Estado do Tocantins (http://diariooficial.to.gov.br).
- Desenvolvimento do sistema de indexação: foi criada uma aplicação, baseada na API Apache Lucene, para fazer a indexação de documentos. A aplicação seleciona a base de documentos a ser indexada, faz a leitura e executa a remoção de *stopwords* e aplicação do *stemming* nos documentos selecionados. Ao final, cria os índices e apresenta as saídas do sistema.
- Desenvolvimento dos cenários virtuais: para o desenvolvimento prático deste trabalho seria necessário uma estrutura física de servidores e cabeamento de rede para que fossem configurados os servidores para execução dos *frameworks* de computação distribuída e indexação. No entanto, a utilização de uma estrutura física é inviável, o que levou a escolha pela utilização da virtualização de servidores. Deste modo, utilizando um software de virtualização, foi possível emular um hardware de um servidor, definindo quantidade de memória RAM, tamanho de disco e capacidade de processamento, bem como emular a rede TCP/IP para comunicação entre os nós.
    - Foram criados dois cenários virtuais para execução do sistema. O primeiro cenário é composto por um servidor com 2GB de memória RAM, disco rígido de 20GB e sistema operacional Linux Ubuntu 10.04 LTS. O segundo cenário é formado por quatro servidores, caracterizando quatro nós de um cluster de computação distribuída, todos com a mesma configuração: 2GB de memória RAM, disco rígido de 20GB e sistema operacional Linux Ubuntu 10.04 LTS.
- Definição dos critérios de comparação: para comparar os cenários de execução do sistema, foram definidos alguns critérios a serem analisados: processamento, uso de memória RAM e tempo médio gasto para indexação, esses critérios foram adotados para poder apresentar os recursos utilizados no processo de indexação apresentado no referencial teórico.

- Aplicação dos cenários de teste: nos dois cenários apresentados foi executada a operação de indexação da base de documentos apresentada. Deste modo, foram criados os índices utilizando um sistema normal de indexação e o sistema de indexação distribuído.
- Apresentação dos resultados: no final, apresentou-se o sistema de indexação em um ambiente de computação distribuída, os resultados da execução desse sistema comparando com a execução de um sistema normal de indexação e, ainda, todos os materiais produzidos nesse processo de desenvolvimento.

4 RESULTADOS E DISCUSSÃO

Esta seção apresenta os resultados obtidos no desenvolvimento do trabalho, relacionando os conceitos apresentados na revisão de literatura com a prática. Assim, são apresentados a base de dados utilizada para o estudo de caso do trabalho, o sistema de indexação implementado e os ambientes de execução do sistema, utilizando um servidor não distribuído um cluster de computação distribuída, demonstrado os procedimentos práticos de indexação nos dois ambientes, envolvendo desde a configuração dos ambientes até a execução do sistema.

4.1. Base de Documentos

A base de documentos utilizada nesse trabalho é composta de edições em pdf do diário oficial do estado do Tocantins, disponíveis em http://diariooficial.to.gov.br/. A coleção de documentos compreende em edições do ano de 2005 a 2010, com quantidade de páginas e tamanho variado.

O tamanho total da coleção de documentos é de 2,2 GB, distribuído em 1224 documentos em pdf. Cada edição do diário corresponde a um dia útil do ano, contendo Projetos de Leis, Portarias, Leis, Decretos, publicações de Contratos, avisos, convocações, além de diversas informações sobre o Executivo do Estado do Tocantins.

Para que os documentos em pdf fossem indexados, foi necessário executar o processo de *parse*, que consiste basicamente em extrair o texto plano dos documentos em formato pdf para que possam ser criados os índices. Este processo é apresentado na seção a seguir, bem como o sistema de indexação.

4.2. Sistema de indexação

O sistema de indexação proposto foi desenvolvido utilizando a biblioteca do *framework* Lucene para a criação e tratamento dos índices. A responsabilidade do sistema é realizar a indexação de documentos para um sistema de recuperação da informação. Na Figura 22 é exibida a estrutura do processo de recuperação da informação.

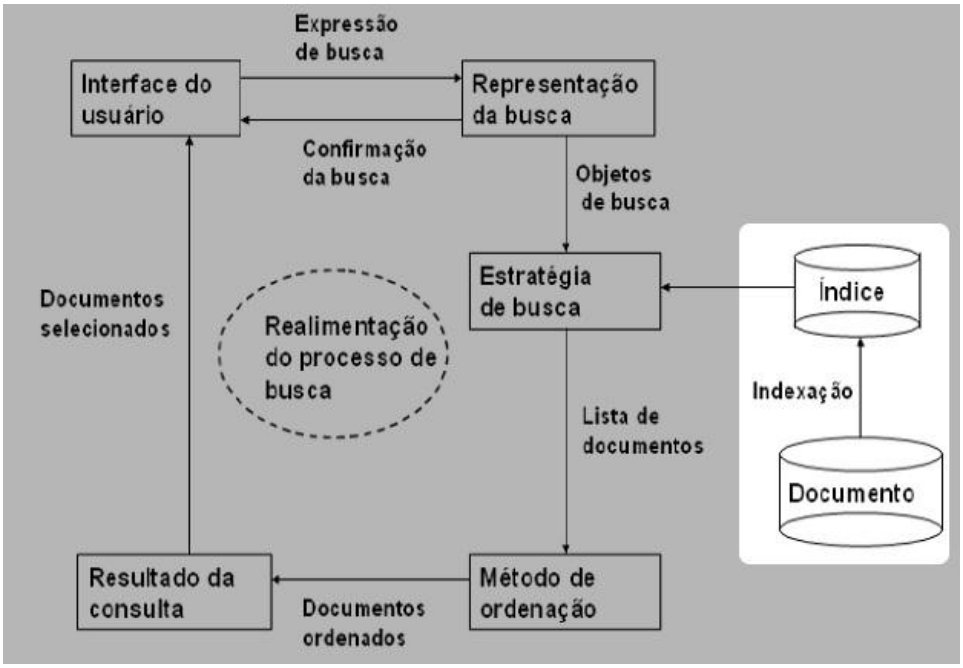

Figura 22 - Estrutura da Recuperação da Informação.

A Figura 22 exibe os passos do processo de recuperação de informação e, em destaque, é apresentado o foco do desenvolvimento deste trabalho que é executar a indexação de documentos para um sistema de recuperação da informação. A funcionalidade do sistema é percorrer uma coleção de documentos em um diretório, tratar cada documento em pdf e indexar esse documento criando os índices no disco. A Figura 23 apresenta o diagrama das classes utilizadas para o funcionamento do sistema.

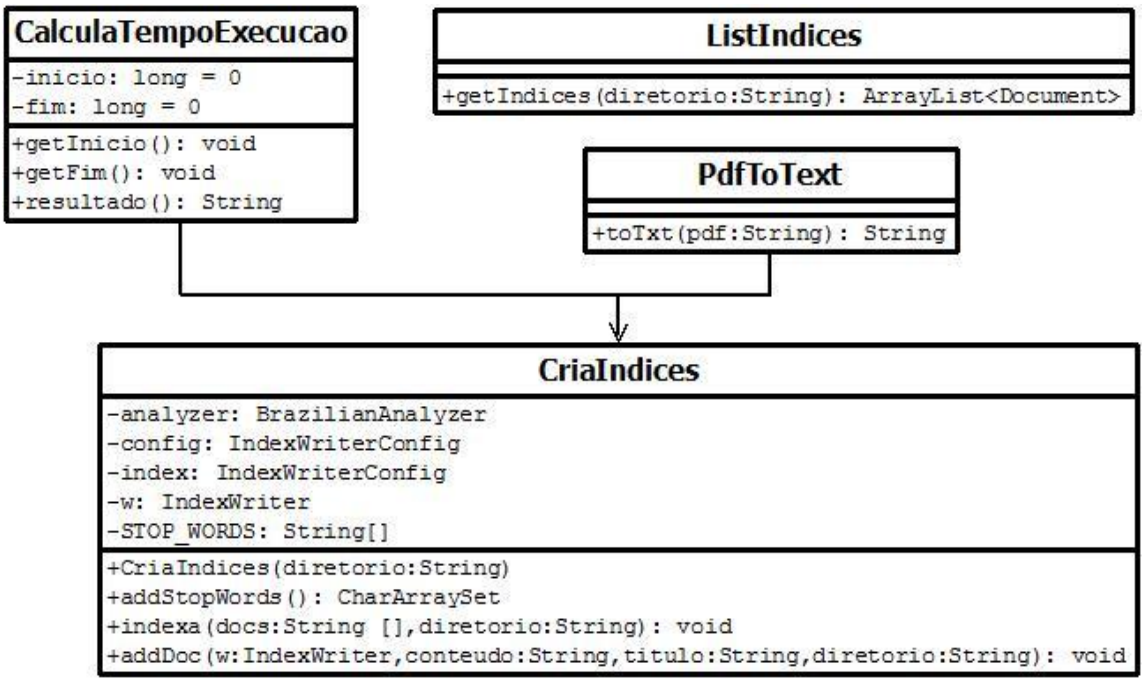

Figura 23 - Diagrama de classes.

Na Figura 23 são apresentadas quatro classes implementadas por meio da linguagem de programação java: `CalculaTempoExecucao`, `PdfToText`, `CriaIndices` e `ListIndices`, cada classe e seu funcionamento será apresentado a seguir.

- A classe `CalculaTempoExecucao` é utilizada para calcular o tempo de execução de uma determinada ação. Possui três métodos em seu escopo, sendo eles: `getInicio()`, que inicia a contagem do tempo, `getFim()` para encerrar o cálculo do tempo gasto e `resultado()` que retorna uma *string* com o resultado do tempo gasto.
- A classe `PdfToText` realiza o processo de *parse*, responsável por realizar a extração do texto do documento em pdf, assim, possibilitar que o documento seja tratado e possa ser criado seus índices para a indexação. A Figura 24 apresenta o código desta classe.

PdfToText.java

```
1  package classes;
2
3  import com.itextpdf.text.pdf.PdfReader;
4  import com.itextpdf.text.pdf.parser.PdfReaderContentParser;
5  import com.itextpdf.text.pdf.parser.SimpleTextExtractionStrategy;
6  import com.itextpdf.text.pdf.parser.TextExtractionStrategy;
7
8  public class PdfToText  {
9      //retornar texto do pdf
10     public String toTxt(String pdf) {
11
12         try
13         {
14                 String doc = "";
15                 PdfReader reader = new PdfReader(pdf);
16                 PdfReaderContentParser parser = new PdfReaderContentParser(reader);
17
18                 TextExtractionStrategy strategy;
19                 for (int i = 1; i <= reader.getNumberOfPages(); i++) {
20                     strategy = parser.processContent(i, new SimpleTextExtractionStrategy());
21                     doc += (strategy.getResultantText());
22                 }
23                 reader.close();
24             return doc;
25         }
26         catch(Exception e)
27         {
28             return e.toString();
29         }
30     }
31 }
```

Figura 24 - PdfToText.java

No código ilustrado na Figura 24, nas linhas 3 a 7 é feito o *import* do iText, biblioteca de manipulação e tratamento de documentos em pdf. Na linha 17 é instanciado o método `toTxt(string pdf)`, onde o parâmetro passado para o método é o endereço do documento pdf que terá o texto extraído. O retorno do método é uma *string*.

- A classe `criaIndices` contém a lógica para criação dos índices dos documentos no sistema de indexação. A classe possui três métodos: `addStopWords()`, `indexa(String [] docs,String diretorio)` e `addDoc(IndexWriter w, String conteudo, String titulo, String diretorio)`. Esta classe, bem como seus métodos são apresentados a seguir na Figura 25.

CriaIndices.java

```
1  package classes;
2
3  import java.io.File;
4  import java.io.IOException;
5
6  import org.apache.lucene.analysis.br.BrazilianAnalyzer;
7  import org.apache.lucene.analysis.util.CharArraySet;
8  import org.apache.lucene.document.Document;
9  import org.apache.lucene.document.Field;
10 import org.apache.lucene.document.StringField;
11 import org.apache.lucene.document.TextField;
12 import org.apache.lucene.index.IndexWriter;
13 import org.apache.lucene.index.IndexWriterConfig;
14 import org.apache.lucene.store.FSDirectory;
15 import org.apache.lucene.util.Version;
16
17
18 public class CriaIndices {
19
20      BrazilianAnalyzer analyzer;
21      static IndexWriterConfig config;
22      static FSDirectory index;
23      IndexWriter w ;
24      String STOP_WORDS [] = {};//lista de stopwors adicionais
25
26
27     public CriaIndices(String diretorio) throws IOException
28     {
29         analyzer = new BrazilianAnalyzer(Version.LUCENE_40,addStopWords());
30         config = new IndexWriterConfig(Version.LUCENE_40, analyzer);
31         index = FSDirectory.open(new File(diretorio));//cria indice no disco
32         w = new IndexWriter(index, config);
33     }
34
35     //stopwords adicionais
36     private CharArraySet addStopWords()
37     {
38
39         CharArraySet aux = new CharArraySet(Version.LUCENE_40, 0, true);
40         for(int i=0;i<STOP_WORDS.length;i++)
41           aux.add(STOP_WORDS[i]);
42         return aux;
43     }
44
45     //------------------------
46
47     public void indexa(String [] docs,String diretorio) throws IOException
48     {
49         PdfToText pdf = new PdfToText();
50         for (String doc: docs)
51         {
52             addDoc(w, pdf.toTxt(diretorio+"/"+doc), doc, diretorio);
53
54         }
55         w.close();
56     }
57      //metodo para add documento no indice
58     private void addDoc(IndexWriter w, String conteudo, String titulo, String diretorio) throws IOException
59         Document doc = new Document();
60         doc.add(new TextField("conteudo", conteudo, Field.Store.YES));
61         // string field para nao tokenizar o titulo
62         doc.add(new StringField("titulo", titulo, Field.Store.YES));
63         doc.add(new StringField("diretorio", diretorio, Field.Store.YES));
64         w.addDocument(doc);
65       }
66     //---------------------
67 }
```

Figura 25 - CriaIndices.java

Nas linhas 6 a 15 do código exibido na Figura 25 são importadas as bibliotecas do Lucene, utilizadas no sistema de indexação. O construtor da classe na linha 27 recebe como parâmetro o diretório que guarda os índices do sistema. O método `indexa(String [] docs,String diretorio)` na linha 47 recebe como parâmetro a lista de documentos e o diretório onde estão os documentos a serem indexados. O método percorre a lista, faz a conversão de cada pdf para texto, e adiciona no índice.

O método `addDoc()` recebe quatro parâmetros: `IndexWriter w,` que contém os método para salvar os índices no disco e a instância do `BrazilianAnalyzer` que executa o processo de remoção de *stopwords* e aplicação de *stemming*, `String conteudo` que recebe o conteúdo a ser indexado, `String titulo` que recebe o título do documento a ser indexado e `String diretorio` que recebe o diretório que contém os documentos indexados.

Ainda na Figura 25, na linha 6 é ilustrado o código que importa a biblioteca do Lucene (`org.apache.lucene.analysis.br.BrazilianAnalyzer`). Essa biblioteca implementa o Lucene *Analyzer* para o idioma português do Brasil que inclui a remoção de *stopwords* e aplicação do processo de *stemming*. A biblioteca tem uma lista pré-definida de *stopwords*, apresentada na Figura 26.

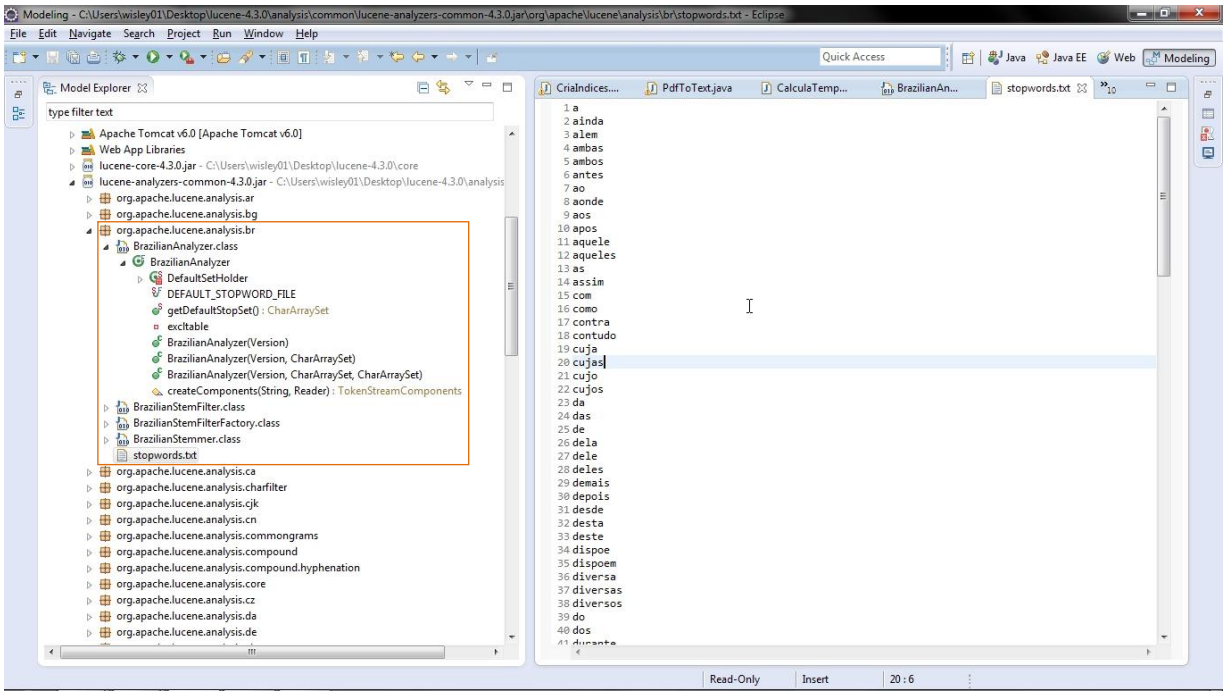

Figura 26 - BrazilianAnalyzer.

A Figura 26 ilustra o arquivo com a lista de *stopwords* a serem removidos no processo de criação de índices do sistema de indexação. No entanto, é possível adicionar mais palavras nessa lista para perfeita adaptação do domínio da aplicação em tempo de execução. Essas palavras podem ser adicionadas em uma lista de *strings* na variável `STOP_WORDS,` na linha 24 no código apresentado na Figura 25.

A classe `ListIndices` é utilizada para listar os índices já criados no disco e possui o método `getIndices(String diretorio)` que recebe como parâmetro o diretório onde se encontram salvos os índices e retorna um `arrayList` de documentos do tipo `org.apache.lucene.document.Document`. A Figura 27 exibe o código da classe.

ListIndices.java

```
1  package classes;
2  import java.io.File;
45
46
47 public class ListIndices
48 {
49
50
51     public static ArrayList<Document> getIndices(String diretorio) throws IOException
52     {
53         File indexDirectory = new File(diretorio);
54
55         IndexReader reader1 = IndexReader.open(FSDirectory.open(indexDirectory));
56
57
58         ArrayList<Document> lista = new ArrayList<Document>();
59
60         for ( int i = 0; i <reader1.maxDoc(); i++)
61         {
62             lista.add(reader1.document(i));
63
64         }
65         reader1.close();
66
67         return lista;
68     }
69
70 }
```

Figura 27 - ListIndices.java

No código apresentado na Figura 27 é implementada uma estrutura de repetição para percorrer o diretório dos índices e retorna a listagem dos mesmos. A camada de apresentação do sistema é feita utilizando *Java Server Pages (JSP),* A seguir é exibida a camada de apresentação do sistema e a sua execução nos cenários criados.

4.3. Sistema de indexação no Cenário 1

O cenário 1 compreende a execução do sistema de indexação em um único servidor, conforme configuração anteriormente apresentada na seção 3.2.3 deste trabalho. O apache Tomcat 6 é utilizado para servir as páginas *JSP* que executam o sistema. A Figura 28 exibe a estrutura do sistema de indexação no cenário 1.

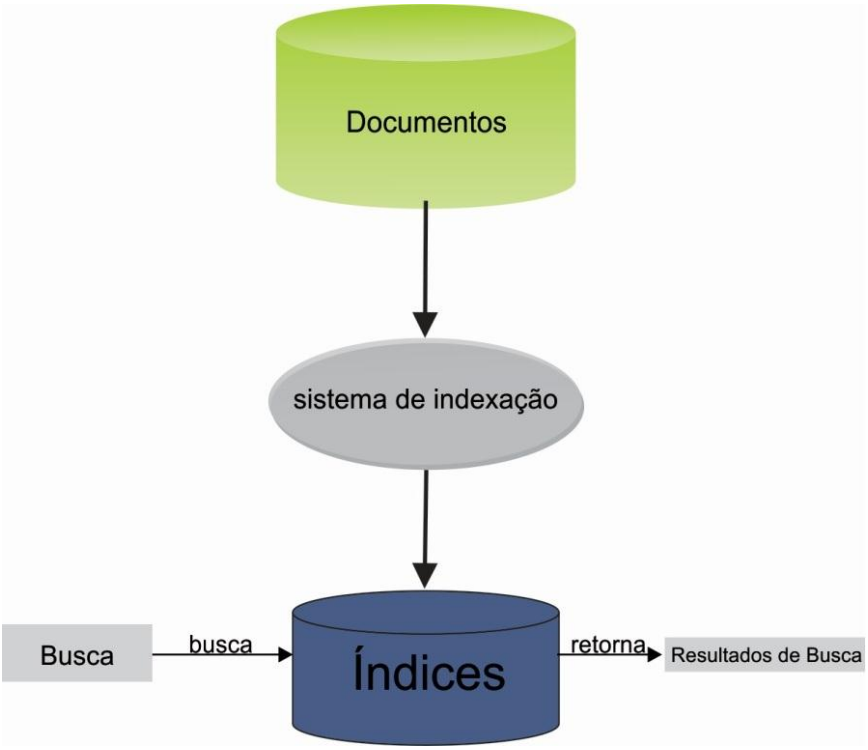

Figura 28 - Estrutura de indexação normal.

Na Figura 28 é apresentada a estrutura do sistema de indexação utilizado no cenário 1 deste trabalho. Nesse cenário, o sistema indexa a coleção de documentos e cria um banco de índices salvo no disco em formato padrão do Lucene.

A tela inicial do sistema apresenta o endereço do diretório que contém a coleção de documentos a ser indexada, a listagem dos documentos e a quantidade de documentos no diretório. Ao abrir a página web no navegador, a tela inicial do sistema é apresentada como pode ser observado na Figura 29.

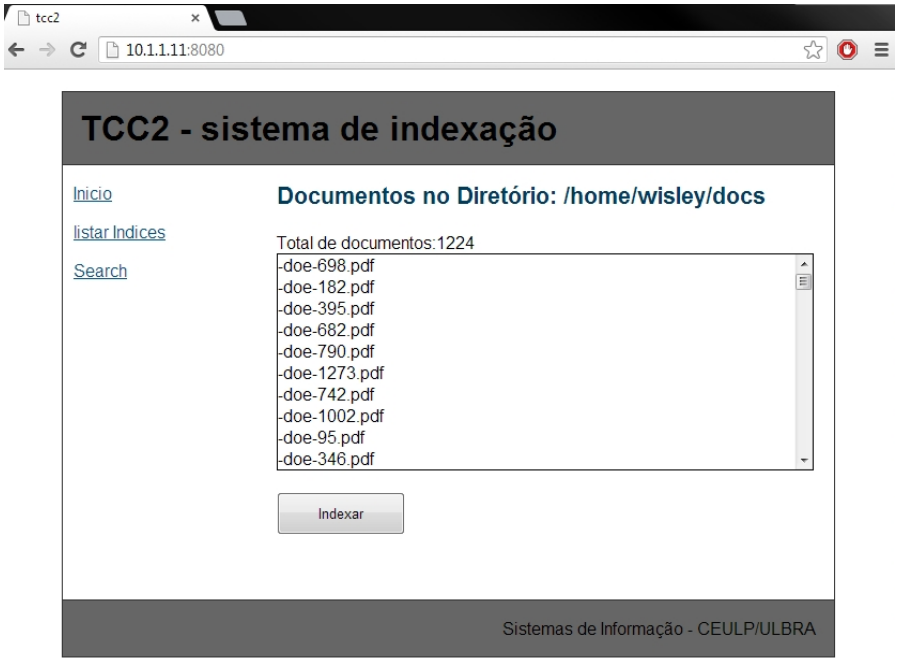

Figura 29 - Página inicial do sistema – cenário 1.

A Figura 29 exibe a tela principal do sistema. Ao acessar o botão indexar, é chamada uma instância da classe `criaIndices`. A seguir, na Figura 30 é apresentado o trecho do código fonte que executa esse processo.

```
47      String diretorio = request.getParameter("diretorio");
48      if (diretorio != null)
49      {
50          try
51          {
52              CalculaTempoExecucao.inicio();//começa a contar tempo de execução
53
54              String dirIndices = getServletContext().getRealPath("indices");
55              CriaIndices criaIndices = new CriaIndices(dirIndices);
56
57              File arquivos = new File(diretorio);
58              String lista[] = arquivos.list();
59              criaIndices.indexa(lista,diretorio);
60          %><div style="border: 1px solid #000; height: 400px; width: 500px;overflow: auto;"><%
61
62              for(String doc:lista)
63              {
64                  out.println(doc+"<br/>");
65              }
66              %>
67          </div>
68          <%
69              CalculaTempoExecucao.Fim();//finaliza contagem do tempo de execuçaõ.
70              out.println(lista.length+" indices criados com sucesso."+"<br/>");
71              out.println("Diretorio dos indices: "+ dirIndices +"<br/>");
72              out.println("Tempo médio gasto: "+CalculaTempoExecucao.Resultado());
73              out.println("<script>closeProgress();</script>");
74          }
75          catch(Exception e)
76          {
77              out.println("<script>closeProgress();</script>");
78              out.println("deu erro: "+e);
79          }
80
81      }
82      else
83      {
84          out.println("<script>closeProgress();</script>");
85          out.println("diretorio inexistente");
86      }
```

Figura 30 - Trecho de código do sistema

No código apresentado na Figura 30, na linha 55 é instanciada a classe `criaIndices`, passando como parâmetro o diretório onde estão guardados os índices no disco. Nesse caso, o diretório é obtido através do código na linha 54, que retorna o caminho do diretório chamando `indices` dentro da raiz da aplicação.

Na linha 59 é chamado o método `criaIndices.indexa`(), passando como parâmetro a lista de documentos e o diretório que contém a coleção de documentos. Nas linhas 70 a 72 o sistema imprime na tela o resultado do processo e o tempo médio gasto.

4.3.1. Execução do sistema

O servidor não distribuído estava ocioso no momento em que o sistema foi executado. Utilizando a ferramenta Nmon é possível obter as informações de processamento e memória do servidor de indexação. A Figura 31 ilustra o retorno do comando `nmon`.

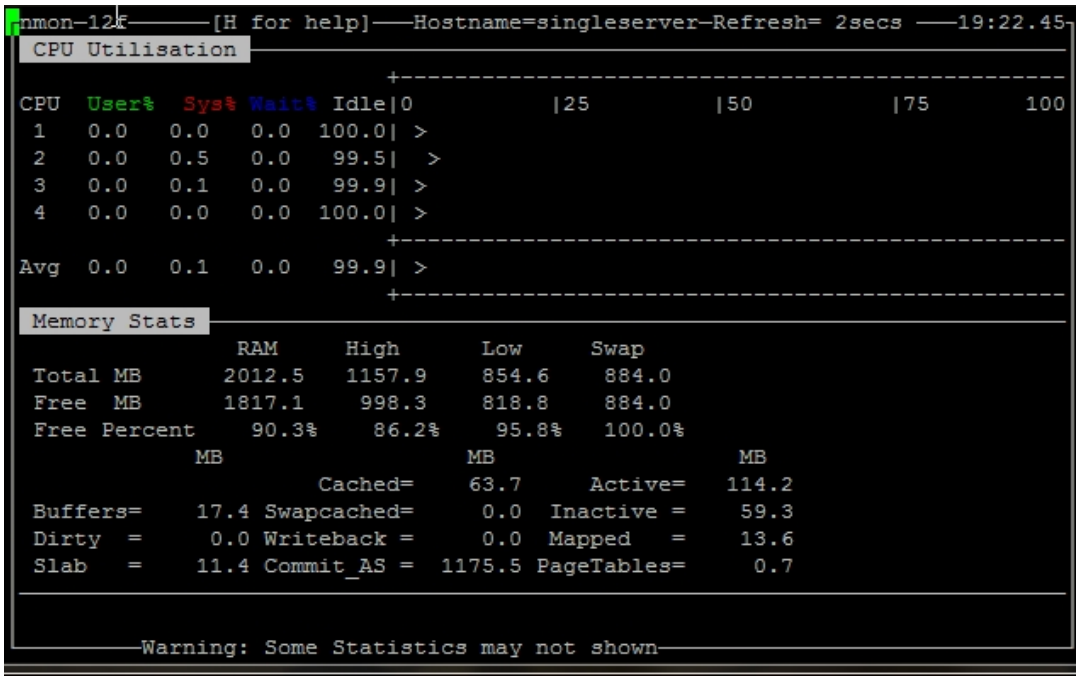

Figura 31 - Monitor de recursos do servidor em espera

O monitor de recursos exibido na Figura 31 apresenta o estado dos recursos do servidor antes de executar o sistema. Nesse momento, o servidor possui 90.3% de memória RAM livre e o processador está com 99.9% da sua capacidade livre.

Ao executar a aplicação, o sistema inicia o processo de indexação da coleção de documentos. Ao terminar o processo, o sistema exibe uma tela com o resultado da operação. Na Figura 32 é apresentada a tela do sistema com o resultado.

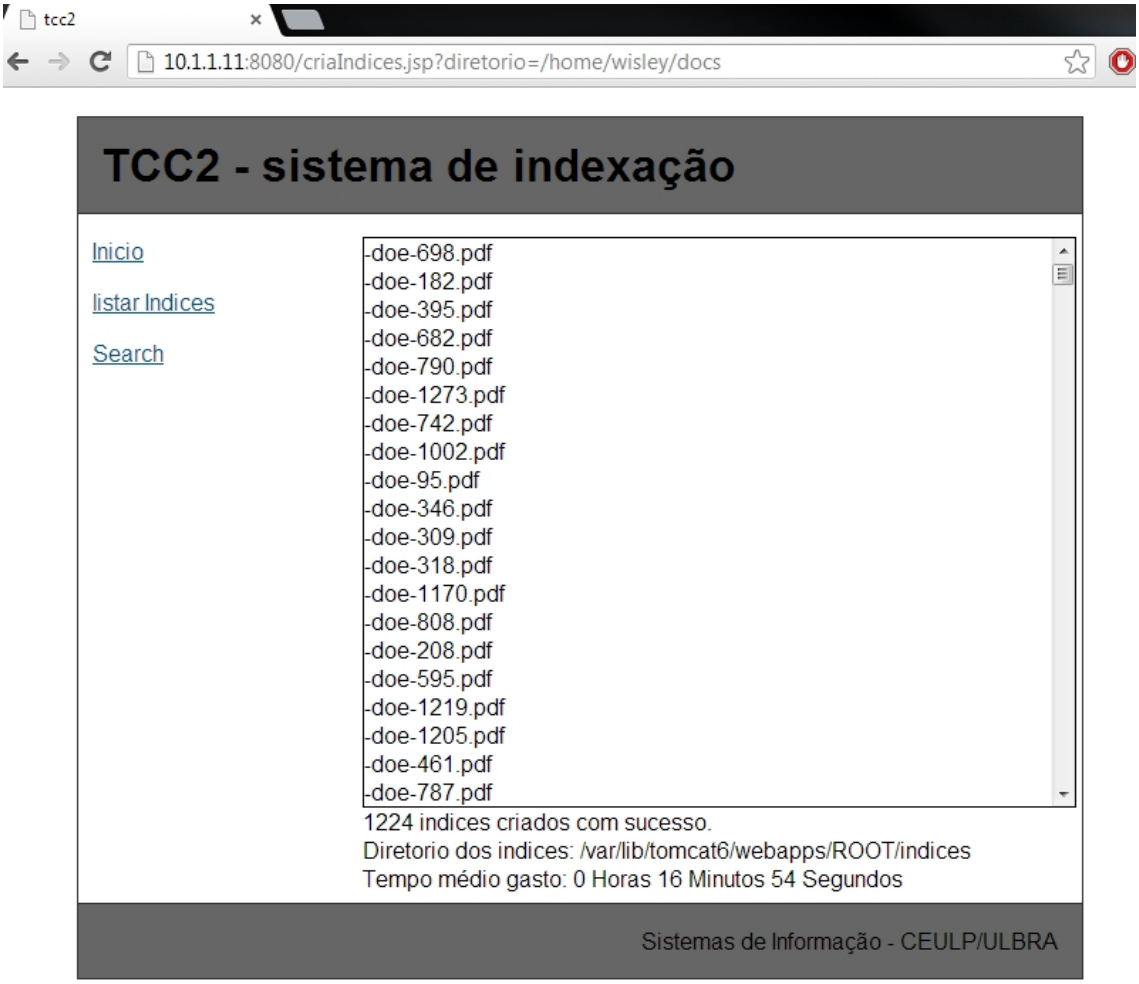

Figura 32 - Resultado execução do sistema – cenário 1.

A tela apresentada na Figura 32 exibe o resultado da indexação, concluída após a execução do sistema. São exibidos a quantidade de documentos indexados com sucesso, o diretório que foi salvo os índices e o tempo gasto com a operação.

Durante a execução do sistema, foram coletados os dados do monitor de recursos em três momentos aleatórios para que pudesse ser calculada a média de recursos utilizados no processo de indexação. A Figura 33 apresenta a captura da tela do monitor de recursos em: 01:00 (um minuto), 08:05 (oito minutos e cinco segundos) e 16:22 (dezesseis minutos e vinte e dois segundos).

Figura 33 - Captura dos dados do monitor de recurso.

A Figura 33 exibe a captura de tela em três tempos distintos do monitor de recursos do servidor. O monitor de recursos apresenta a quantidade de memória RAM total, a quantidade de memória RAM livre e o uso do processador. A Tabela 11 informa os dados utilizados para calcular a média do uso dos recursos usados na execução do sistema.

Tabela 11 - Recursos utilizados do servidor - cenário 1.

Tempo	Uso memória RAM em MB	Uso processador em %
01:00	1001,2	58,7
08:05	1932,0	33,4
16:22	1928,7	28

Média	**1620,6**	**40,0**

A Tabela 11 apresenta o uso de memória RAM e a percentagem usada do processador em cada momento da coleta das informações. O servidor, que possui 2012.5MB de memória RAM disponível, durante a tarefa de indexação de 1224 documentos consumiu em média 1620,6 MB de memória RAM e 40% da capacidade total de processamento. O tempo médio gasto para a execução da indexação foi de 16 minutos e 54 segundos.

Os dados apresentados foram obtidos através da execução do sistema de indexação em um único servidor. A próxima seção apresenta a execução do sistema no cenário 2, utilizando computação distribuída.

4.4. Sistema de indexação no Cenário 2

O cenário 2 compreende a execução do sistema de indexação utilizando computação distribuída através de um cluster de alta performance utilizando índices locais. A configuração do cenário foi apresentada na seção 3.2.3 deste trabalho. O apache Tomcat 6 é utilizado para servir as páginas *JSP* que executam o sistema.

Utilizando computação distribuída, o sistema de indexação usa o *framework* Lucene para o índices e o *framework* Katta para implementar os índices distribuídos. A Figura 34 ilustra a estrutura de funcionamento do sistema de indexação distribuído.

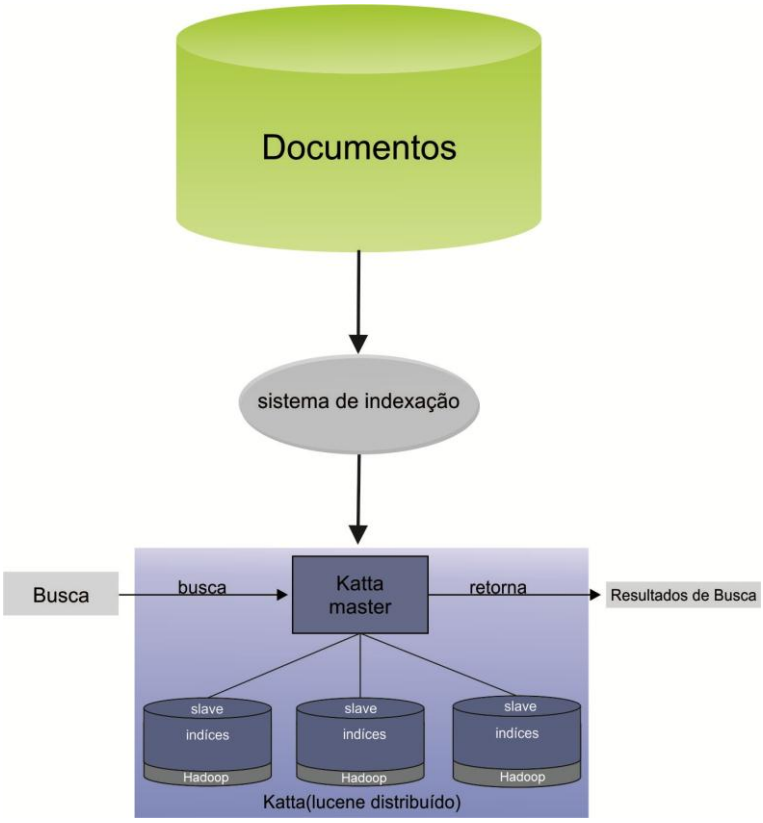

Figura 34 - Indexação distribuída.

Na estrutura apresentada na Figura 34, o sistema realiza a leitura da coleção de documentos e realiza o processo de indexação no cluster de computação distribuída, através do Katta. O sistema de indexação executa o processo de tratamento dos documentos e cria os índices que serão processados pelo Katta e são salvos no sistema de arquivos do Hadoop.

Para utilizar o processo de *MapReduce* do Hadoop, foi implementado um *job* [4] para executar a operação de *parse* dos documentos em pdf e criação dos índices. A seguir, na Figura 35 é exibido o trecho do código fonte que executa essas operações.

IndexJobKatta.java

```
86    public void run(RecordReader<LongWritable, Text> reader, OutputCollector<Text, Text> output, final Reporter report)
87            throws IOException {
88      LongWritable key = reader.createKey();
89      Text value = reader.createValue();
90
91      String tmp = _conf.get("hadoop.tmp.dir");
92      long millis = System.currentTimeMillis();
93      String shardName = "" + millis + "-" + new Random().nextInt();
94      File file = new File(tmp, shardName);
95      report.progress();
96
97      CriaIndices criaIndice = new CriaIndices(FSDirectory.open(file));
98
99      report.setStatus("Iniciando indexação.");
100     while (reader.next(key, value))
101     {
102       report.progress();
103       PdfToText pdf = new PdfToText();
104       FileSplit fileSplit = (FileSplit)report.getInputSplit();
105       String titulo = fileSplit.getPath().getName();
106       String diretorio= fileSplit.getPath().toString();
107       criaIndice.indexa(pdf.toTxt(value.getBytes()), titulo, diretorio);
108     }
109
110     report.setStatus("Documentos indexados.");
111     Thread t = new Thread() {
112       public boolean stop = false;
113
114       @Override
115       public void run() {
116         while (!stop) {
117           report.progress();
118           try {
119             sleep(10000);
120           } catch (InterruptedException e) {
121             //não executa nada
122             stop = true;
123           }
124         }
125       }
```

Figura 35 - Trecho de código IndexJobKatta.java

O código apresentado na Figura 35 faz parte da classe `IndexJobKatta.java`, que implementa um *job* da biblioteca *MapReduce* do Hadoop. Quando esse *job* é executado, a tarefa é dividida em vários pedaços e enviados para os nós do cluster para serem processadas paralelamente. Na linha 97 é instanciada a classe `CriaIndices` passando como parâmetro

[4] Um *job* é a tarefa executada no processo *MapReduce* no cluster Hadoop
.

o diretório onde serão salvos os índices. Na linha 103 é instanciada a classe para efetuar o *parse* do documento em pdf. Na linha 107 é executado o método que cria o índice no disco, passando como parâmetro o texto extraído do pdf, o título e o diretório que contém o documento.

A tela inicial do sistema segue o padrão do sistema apresentado na Figura 29, no cenário 1. No entanto, algumas modificações no código foram necessárias para que fosse adaptado ao sistema distribuído utilizado no cenário 2. Estas modificações consistem em adicionar o trecho de código para importar os arquivos para o sistema *HDFS* e a alteração nos parâmetros de execução do sistema.

Ao executar o sistema, são iniciados dois processos: primeiro o sistema copia a coleção de documentos para o sistema de arquivos *HDFS* e, em seguida, é iniciado o *job* de indexação e o *deploy*[5] dos índices no Katta. A Figura 36 apresenta o código que faz a cópia dos documentos para o HDFS.

```
56      String diretorio = request.getParameter("diretorio");//recebe o diretorio dos documentos
57      if (diretorio != null)
58      {
59          //copia dados para hdfs do hadoop
60          String source = diretorio;
61          String dest = "/home/hduser/hadoop-docs/"+diretorio;
62          boolean cond = false;
63          //copia do diretorio local para diretorio do hadoop
64          Configuration conf = new Configuration();
65          conf.addResource(new Path("/usr/local/hadoop/conf/core-site.xml"));
66          conf.addResource(new Path("/usr/local/hadoop/conf/hdfs-site.xml"));
67          conf.addResource(new Path("/usr/local/hadoop/conf/mapred-site.xml"));
68
69          FileSystem fileSystem = FileSystem.get(conf);
70          Path srcPath = new Path(source);
71          Path dstPath = new Path(dest);
72          try
73          {
74              fileSystem.copyFromLocalFile(srcPath, dstPath);
75              cond = true;
76          }catch(Exception e)
77          {
78              cond = false;
79          }
80          //
```

Figura 36 - Trecho de código para copiar documentos no HDFS.

Nas linhas 64 a 67 na Figura 36, são passadas as configurações do cluster Hadoop . Na linha 74 é executado o método `fileSystem.copyFromLocalFile` passando como parâmetro o endereço do diretório local com os documentos que serão indexados e o diretório de destino no sistema de arquivos do Hadoop. Depois de realizada a cópia dos documentos, é

[5] Adicionar arquivos de índices no Katta.

executado o *job* de indexação. O trecho de código que realiza esse procedimento é demostrado na Figura 37.

```
82          try
83          {
84              if(cond)
85              {
86                  CalculaTempoExecucao.inicio();//começa a contar tempo de execução
87
88                  String masterHDFS = "hdfs://master:54310";
89                  IndexJobKatta jobIndexa = new IndexJobKatta();
90                  String diretorioDocumentos = masterHDFS+ dest;
91                  String indices = masterHDFS+"/home/hduser/indicesTcc/"+diretorio;
92                  jobIndexa.startIndexer(diretorioDocumentos, indices);
93
94                  ZkConfiguration zkConfig = new ZkConfiguration();
95                  DeployClient deployIndices = new DeployClient(ZkKattaUtil.startZkClient(zkConfig, 60000), zkConfig);
96                  deployIndices.addIndex("indicesTcc2", indices, 4);
97
98                  File arquivos = new File(diretorio);
99                  String lista[] = arquivos.list();
100
101                 CalculaTempoExecucao.Fim();//finaliza contagem do tempo de execução.
102                 out.println(lista.length+" indices criados com sucesso."+"<br/>");
103                 out.println("Diretorio dos indices: "+ indices +"<br/>");
104                 out.println("Tempo médio gasto: "+CalculaTempoExecucao.Resultado());
105                 out.println("<script>closeProgress();</script>");
106             }
107             else
108             {
109                 throw new Exception("Erro ao copiar arquivos para HDFS");
110             }
111         }
112         catch(Exception e)
113         {
114             out.println("Erro: "+e);
115         }
```

Figura 37 - Trecho de código que executa o *job* e o *deploy*.

No trecho de código apresentado na Figura 37, a linha 88 contém a variável que recebe o endereço do nó *master* do Hadoop. As linhas 90 e 91 contêm as variáveis que respectivamente recebem o diretório *HDFS* que contém a coleção de documentos e o diretório *HDFS* que receberá os índices. Na linha 92 é executado o método `startIndexer` passando como parâmetro o diretório dos documentos e o diretório dos índices.

Na linha 95 é instanciada a classe `DeployClient` da biblioteca do *framework* Katta passando como parâmetro as configurações do cluster. Na linha 97 é executado o método `addIndex` passando como parâmetro o identificador dos índices no Katta, o diretório HDFS dos índices e quantidade de replicações do índice. Nas linhas 102 a 104 são impressos na tela os resultados da operação. A seção seguinte apresenta a execução do sistema em ambiente distribuído e os resultados obtidos.

4.4.1. Execução do sistema

Antes da execução do sistema, utilizando a ferramenta Nmon, foram coletadas informações dos recursos livres de memória RAM e processamento livre dos nós do cluster. A Tabela 12 apresenta os dados obtidos da tela capturada do monitor de recursos em anexo na Figura 39.

Tabela 12 - Recursos livres do cluster.

nó	Memória livre em MB	Processamento livre em %
master	1852,4	100
worker1	1927,9	100
worker2	1928,4	100
worker3	1928,6	100
Total	**7637,3**	**-**
Média	**1909,33**	**100**

Nos dados apresentados na Tabela 12, o cluster possui o total de 7637,3 MB de memória RAM disponível, totalizando a média de 1909,33 de memória RAM para cada nó do cluster.

O sistema é instalado no nó *master* do cluster, rodando no apache Tomcat. Após a execução, o sistema exibe uma tela com os resultados do processo realizado. Na Figura 38 é apresentada a tela com os resultados da execução do sistema.

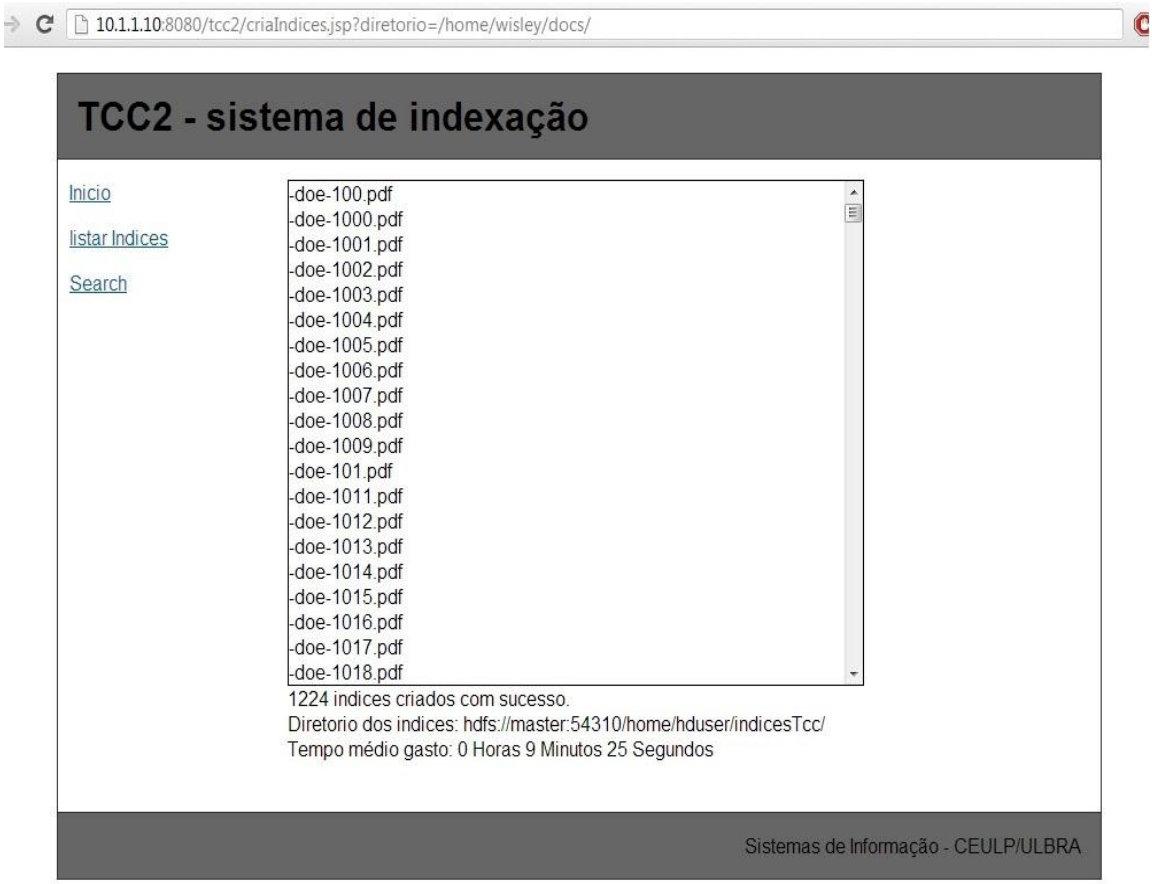

Figura 38 – Resultado execução do sistema computação distribuída**.**

A tela apresentada na Figura 38 exibe o resultado da indexação concluída após a execução do sistema. São exibidos na sequência a quantidade de documentos indexados, o diretório que foi salvo os índices e o tempo gasto com a operação.

Durante a execução do sistema, foram coletados os dados do monitor de recursos dos nós do cluster em três momentos diferentes para que pudesse ser calculada a média do uso desses recursos. Na Tabela 13 são apresentados os dados obtidos utilizando o monitor de recursos Nmon. A captura das telas do monitor de recursos encontra-se em anexo na Figura 40, Figura 41 e Figura 42.

Tabela 13 - Recursos utilizados por nó do cluster.

Tempo	**Média de recursos utilizados por nó.**		
	nó	**Memória usada em MB**	**Processamento usado em %**
01:00	master	1327,05	97,7
	worker1	195,85	49,8
	worker2	306,05	62,3
	worker3	211,75	66,6
	Total	**2040,7**	**-**
	Média	**510,18**	**69,1**
04:50	**nó**	**Memória usada em MB**	**Processamento usado em %**
	master	1710,75	98,6
	worker1	200,95	77,1
	worker2	380,85	98,8
	worker3	507,05	92,6
	Total	**2799,6**	**-**
	Média	**699,90**	**91,78**
08:40	**nó**	**Memória usada em MB**	**Processamento usado em %**
	master	1830,85	84,6
	worker1	330,25	88,7
	worker2	402,25	90,1
	worker3	586,45	97,1
	Total	**3139,77**	**-**
	Média	**787,45**	**90,13**
Média Geral		**665,84**	**83,67**

Os dados apresentados na Tabela 13 foram capturados em três momentos, referentes ao tempo de execução gasto pelo sistema, sendo: 01:00 (um minuto), 04:50 (quatro minutos e cinquenta segundos) e 08:40 (oito minutos e quarenta segundos). Considerando que cada nó do cluster tinha a média de 1909,33 MB de memória RAM e 100% de processamento disponível, foi utilizada a média de 665,84 MB de memória RAM e 83,67% de processamento

por nó para a tarefa de indexação, o tempo médio gasto para a execução da indexação foi de 9 minutos e 25 segundos. Analisando o cluster como um todo, a tarefa de indexação de 1224 documentos utilizou a média de 2660 MB de memória e a média 83,7% da capacidade de processamento do cluster. Na Tabela 14 são apresentados os recursos utilizados pelo cluster como um todo.

Tabela 14 - Recursos utilizados pelo cluster.

Tempo	Uso memória em MB	Uso processador em %
01:00	2040,7	69,1
04:50	2799,6	91,78
08:40	3139,77	90,13
Média	2660,0	83,7

Na Tabela 14 são exibidos os recursos utilizados de cada nó somados, totalizando a quantidade de recurso utilizado pelo cluster no momento em que os dados foram capturados. Antes de iniciar a execução do sistema, a memória RAM disponível do cluster era de 7637 MB e 100% de processamento disponível. Na seção seguinte, são apresentadas algumas considerações sobre a execução dos dois cenários apresentados.

4.5. Considerações Finais dos Resultados

Analisando os resultados obtidos a partir da execução dos cenários apresentados neste trabalho, algumas conclusões podem ser apresentadas. A estrutura utilizada para execução dos cenários é virtualizada, dessa forma, para que se obtenham resultados mais apurados seria necessário utilizar ambientes físicos com *hardware* real. A seguir na **Erro! Fonte de referência não encontrada.**, é apresentado o tempo médio gasto em cada execução dos cenários apresentados.

Tabela 15 - Tempo médio gasto

Cenário	Tempo Médio Gasto
Cenário 1 - Não distribuído	16 minutos e 54 segundos
Cenário 2 - Distribuído	9 minutos e 25 segundos

Na **Erro! Fonte de referência não encontrada.** é exibido o tempo gasto em cada cenário de execução do sistema de indexação. No cenário 1, que consiste em um servidor não distribuído, o tempo médio gasto para executar a indexação dos documentos foi de 16 minutos

e 54 segundos. No cenário 2, que compreende em um cluster de computação distribuída com 4 nós, o tempo médio gasto na indexação dos documentos foi de 9 minutos e 25 segundos

Na comparação da execução dos dois cenários, o cenário 2 teve melhores resultados na execução da tarefa de indexação da coleção de documentos, consumindo menos recursos e obtendo mais agilidade no processo. É visível o aumento do uso de processamento, pois são utilizadas mais operações pelo processo *MapReduce*, que divide as tarefas para os nós do cluster, que faz o processamento da tarefa, e devolve o dado processado. Na execução do cenário 1, os resultados foram inferiores, apresentando maior consumo de recursos e maior tempo para conclusão da tarefa. O sistema no cenário 1 executa as tarefas em sequência, diferente do cenário 2, que executa em paralelo, distribuindo as tarefas entre os nós do cluster.

5 CONSIDERAÇÕES FINAIS

Neste trabalho foram abordados os conceitos sobre Recuperação da Informação, Sistemas de Recuperação da Informação, Índices distribuídos e computação distribuída. A compreensão destes conceitos foi de grande importância para que se fosse entendido o procedimento de indexação no processo de recuperação da informação e o funcionamento dos sistemas distribuídos. Foram ainda apresentados os *frameworks* Lucene, Hadoop e Katta, para desenvolvimento da aplicação de indexação.

Com base nesses estudos, foi desenvolvido um sistema de indexação. Porém, antes foi criada uma base de documentos a ser indexada, composta por uma coleção de documentos (1224 arquivos no formato pdf) resultante de edições do diário oficial do Estado do Tocantins. Posteriormente, dois cenários para execução do sistema foram criados utilizando um sistema de virtualização: no primeiro cenário o sistema foi executado em um servidor não-distribuído, no segundo cenário o sistema foi executado no cluster composto por 4 nós.

A indexação da base de documentos utilizada foi executada nos dois cenários criados, e durante a execução do sistema, foram coletadas as informações sobre o uso de memória RAM e processamento dos servidores.

A média de uso de recursos utilizados para indexação dos documentos no sistema executado no primeiro cenário foi de cerca de 80% da memória RAM disponível e 40% da capacidade de processamento, levando 16 minutos e 54 minutos para concluir a tarefa. Na execução do sistema no segundo cenário para indexação dos documentos pelo cluster, o uso dos recursos foi de cerca de 30% da memória RAM disponível e 90% do processador, levando cerca de 9 minutos e 25 segundos.

O uso do sistema de indexação utilizando computação distribuída apresentou melhores resultados na sua execução, consumindo menor quantidade de recursos e apresentando maior agilidade na execução da indexação. Comparando com a quantidade de recursos utilizados para execução da indexação no primeiro cenário, o sistema distribuído teria condições de executar a indexação de uma base de dados três vezes maior do que a utilizada, sem ultrapassar o limite de recursos disponíveis do cluster.

Deste modo, utilizando a estrutura de computação distribuída, o sistema de indexação desenvolvido nesse trabalho consegue indexar bases de dados maiores com maior eficiência,

com a vantagem de ser um sistema escalável, pois a medida que a demanda do sistema de indexação se torna maior, podem ser adicionados nós no cluster para suprir a necessidade de recursos computacionais.

Para trabalhos futuros, propõe-se criar a estrutura de computação distribuída em um cluster utilizando servidores físicos para, assim, utilizar melhor os recursos de cada nó utilizando o sistema de indexação distribuída. Também, implementar um módulo para permitir a indexação de fontes de dados externas como, por exemplo, bancos de dados relacionais e um módulo de busca.

6 REFERÊNCIAS BIBLIOGRÁFICAS

AUGUSTO, Luiz Daniel Creão. **Arquitetura e implementação de um sistema distribuído de recuperação de informação**. 2010. 85 p. Tese (Mestrado em Ciência da Computação) – Instituto de Matemática e Estatística da Universidade de São Paulo.

BAEZA-YATES, R.; RIBEIRO-NETO,B. **Modern Information Retrieval**. New York: Addson Wesley Longman, 1999.

BORKO, H. Information science: what is it? **American Documentation**, v. 19, n. 1, 1968.

BORGES, Graciane Silva Bruzinga. **Indexação Automática de Documentos Textuais: Proposta de Critérios Essenciais.** 2009. 107 p. Tese (Mestrado em Ciência da Informação) – Programa de Pós-graduação em Ciência da Informação da Escola de Ciência da Informação da Universidade Federal de Minas Gerais.

BOTELHO, Fabiano C. **Algoritmos e Estruturas de Dados**, 2008, Online, Disponível em: <http://homepages.dcc.ufmg.br/~fbotelho/en/courses/aeds-2-2-2008/tp2/tp2.html/>, acesso em 08 de Setembro de 2012.

BRANDT, Mariana Baptista. **Etiquetagem e Folksonomia:** uma análise sob a óptica dos processos de organização e recuperação da informação na web**.** 2009. 142 p. Tese (Mestrado em Ciência da Informação) – Universidade de Brasília.

BUYYA, R, **High Performance Cluster Computing:** Architectures and Systems, Prentice Hall, volume 1 (1999).

CONTI, Fabieli de. **Grades Computacionais para Processamento de Alto desempenho**. 2009, Laboratório de Sistemas de Computação – LSC. Universidade Federal de Santa Maria (UFSM) - Santa Maria, RS – Brasil. Disponível em: <http://www-usr.inf.ufsm.br/~andrea/elc888/artigos/artigo3.pdf>, acesso em 09 de Setembro de 2012.

GOMES, Diego. **O que é escalabilidade**, 2010, Online, Disponível em: <http://escalabilidade.com/2010/01/31/o-que-e-escalabilidade/>, acesso em 09 de Setembro de 2012.

GOMES, H. E; CAMPOS, M. L. A. Tesauro e normalização terminológica: o termo como base para intercâmbio de informações. **Datagramazero**. v. 5, n. 6, dez. 2004.

GONZALES, M.; LIMA, V. L. S. Recuperação de Informação e Processamento da Linguagem Natural. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 23. , 2003, Campinas. **Anais da III Jornada de Mini-Cursos de Inteligência Artificial**, Volume III, Rio de Janeiro: 2003, [s.n.].

GOSPODNETIC, Otis; HATCHE, Erik. **Lucene in Action: Meet Lucene**, 2005, Online, Disponível em: <http://www.webreference.com/programming/lucene/index.html>, acesso em 10 de Setembro de 2012.

HADOOP, Apache. **Documentação *online***, 2012, Online, Disponível em: <http://hadoop.apache.org/>, acesso em 09 de Setembro de 2012.

KATTA. **Documentação *online***, 2008, Online, Disponível em:< http://katta.sourceforge.net/>, acesso em 10 de Setembro de 2012.

LEITE, Maria Angelica de Andrade. **Modelo *Fuzzy* para Recuperação de Informação Utilizando Múltiplas Ontologias Relacionadas**. 2009. 165 p. Tese (Doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

LUCENE, Apache **Documentação *online***, 2012, Online, Disponível em: < http://lucene.apache.org/core/>, acesso em 10 de Setembro de 2012.

OLIVEIRA, F. L. **Clusterização de Consultas em um Modelo de Ordenação Web Baseado na Relevância por Tempo em Domínio Aberto.** 2005. 78 p. Tese (Mestrado em Ciência da Informação) – Universidade Federal de Santa Catarina.

ROCHA, Bruno P. S., REZENDE, Cristiano G., SANTOS, Rodrygo L. T. **Arquivos Invertidos Distribuídos.** Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, MG Brasil, 2006.

RODRIGUES, Mauro S. M. . **Cluster**: Uma Abordagem À Computação Paralela de Alta Velocidade, Universidade Federal de Itajubá, 2007.

SARACEVIC, T. **Ciência da informação:** origem, evolução e relações. Perspectivas em Ciências da Informação., Belo Horizonte, v. 1, n. 1, p. 41-62, jan./jun. 1996.

SILVA, André Correia da. **Clusters de Alta Disponibilidade**, 2011, Online, Disponível em: <http://andrecorreiati.wordpress.com/2011/06/10/clusters-de-alta-disponibilidade//>, acesso em 10 de Novembro de 2012.

TANENBAUM, Andrew S.; STEEN, Maarten Van. **Distributed Systems:** Principles and Paradigms, 2. Ed., 2007, 669 p.

TEIXEIRA, Fábio Augusto Guimarães. **A Recuperação da Informação e a colaboração de usuários na Web.** 2010. 159 p. Dissertação de Mestrado em Ciência da Informação) – Universidade de Brasília.

VALIATI, Eliane Regina de Almeida. **Avaliação de Usabilidade de Técnicas de Visualização de Informações Multidimensionais**. Tese (Doutorado) - Programa de Pós-Graduação em Ciência da Computação. Universidade Federal do Rio grande do Sul. Porto Alegre, 2008.

ZHOU, Deng Peng. **Delve inside the Lucene indexing mechanism**, 2006, Online, Disponível em: <http://www.ibm.com/developerworks/library/wa-lucene/>, acesso em 15 de Setembro de 2012.

ANEXOS

Master

CPU Utilisation

CPU	User%	Sys%	Wait%	Idle	0	25	50	75
1	0.0	0.0	0.0	100.0	>			
2	0.0	0.0	0.0	100.0	>			
3	0.0	0.0	0.0	100.0	>			
4	0.0	0.0	0.0	100.0	>			
Avg	0.0	0.0	0.0	100.0	>			

Memory Stats

	RAM	High	Low	Swap
Total MB	2012.5	1157.9	854.6	884.0
Free MB	1852.4	1038.1	814.2	884.0
Free Percent	92.0%	89.7%	95.3%	100.0%

worker1

CPU Utilisation

CPU	User%	Sys%	Wait%	Idle	0	25	50	75
1	0.0	0.5	0.0	99.5	>			
2	0.0	0.0	0.0	100.0	>			
3	0.0	0.0	0.0	100.0	>			
4	0.0	0.0	0.0	100.0	>			
Avg	0.0	0.0	0.0	100.0	>			

Memory Stats

	RAM	High	Low	Swap
Total MB	2012.5	1157.9	854.6	884.0
Free MB	1927.9	1108.7	819.2	884.0
Free Percent	95.8%	95.8%	95.9%	100.0%

worker2

CPU Utilisation

CPU	User%	Sys%	Wait%	Idle	0	25	50	75
1	0.0	0.0	0.0	100.0	>			
2	0.0	0.0	0.0	100.0	>			
3	0.0	0.0	0.0	100.0	>			
4	0.0	0.0	0.0	100.0	>			
Avg	0.0	0.0	0.0	100.0	>			

Memory Stats

	RAM	High	Low	Swap
Total MB	2012.5	1157.9	854.6	884.0
Free MB	1928.4	1108.9	819.6	884.0
Free Percent	95.8%	95.8%	95.9%	100.0%

worker3

CPU Utilisation

CPU	User%	Sys%	Wait%	Idle	0	25	50	75
1	0.0	0.0	0.0	100.0	>			
2	0.0	0.0	0.0	100.0	>			
3	0.0	0.0	0.0	100.0	>			
4	0.0	0.0	0.0	100.0	>			
Avg	0.0	0.0	0.0	100.0	>			

Memory Stats

	RAM	High	Low	Swap
Total MB	2012.5	1157.9	854.6	884.0
Free MB	1928.6	1108.9	819.8	884.0
Free Percent	95.8%	95.8%	95.9%	100.0%

Figura 39 - Recursos do cluster ocioso.

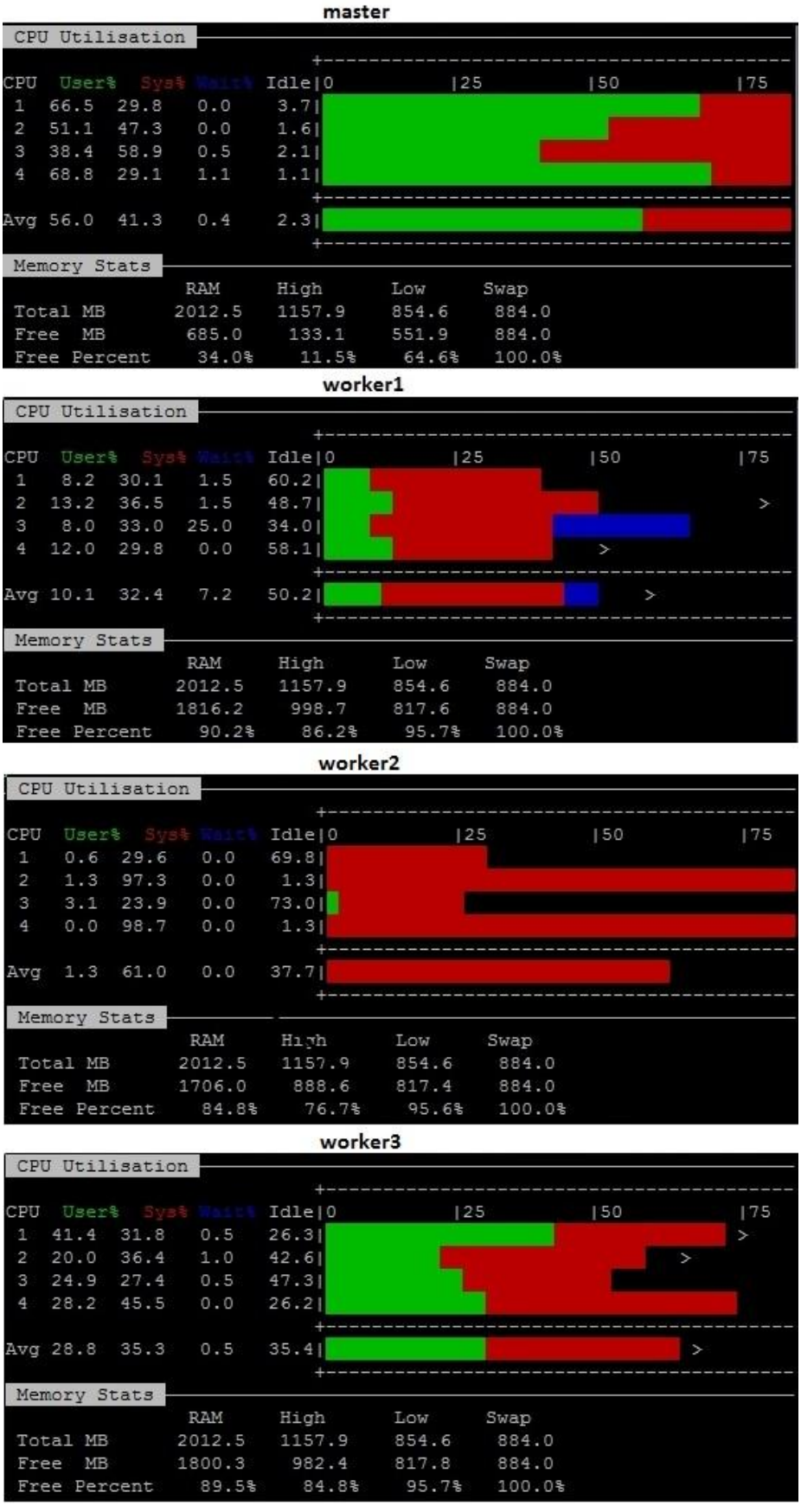

Figura 40 - Captura do monitor de recursos em 01:00.

Figura 41 - Captura do monitor de recursos em 04:50.

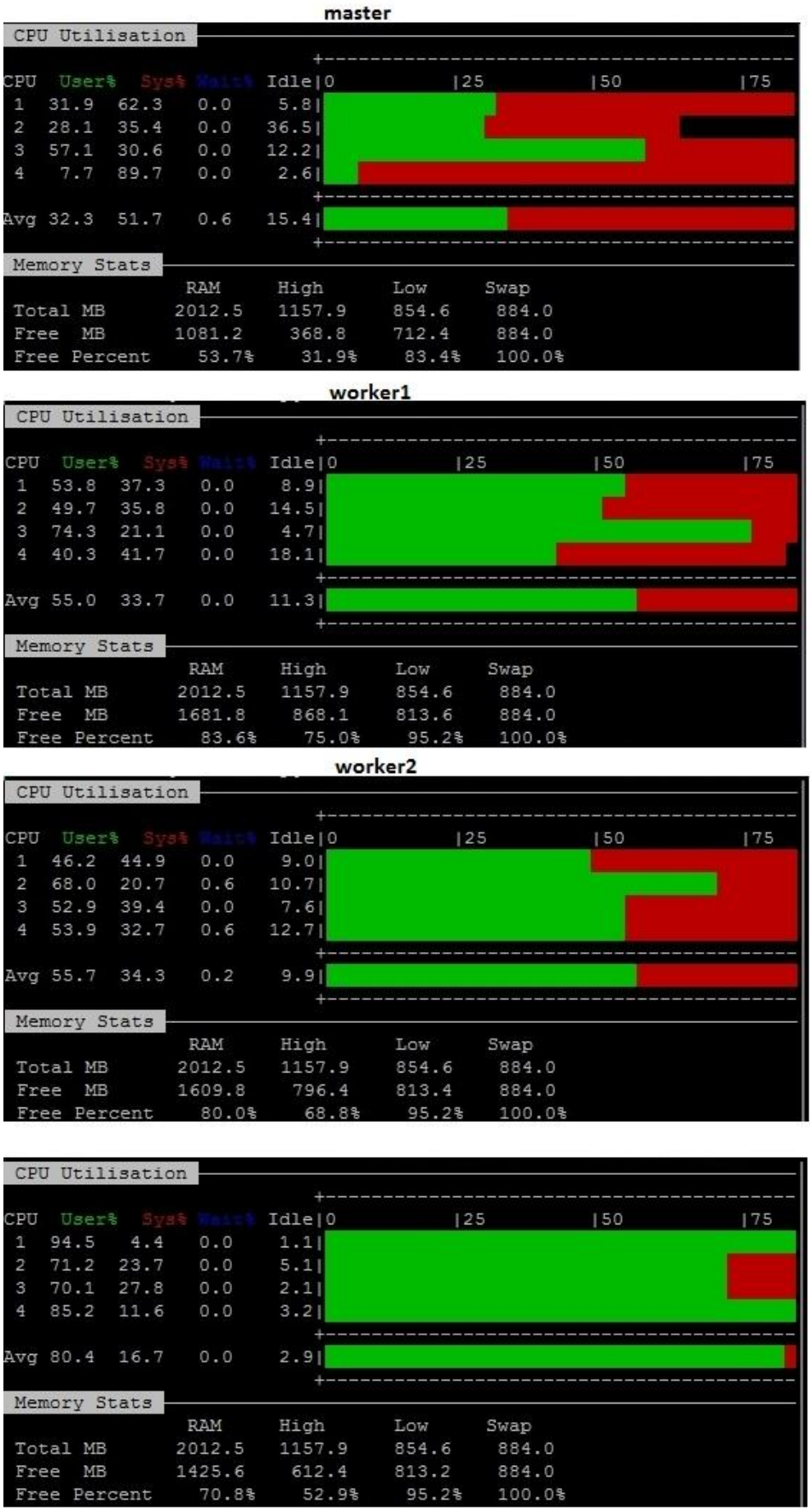

Figura 42 - Captura do monitor de recursos em 08:40.