2
Conceitos e Tecnolgias Utilizadas

A Web Semântica é um termo concebida por Tim Bernes-Lee para descrever a rede de dados, ela pretende embutir inteligência e contexto nos códigos XML (*Extensible Markup Language* – Linguagem de Marcação) utilizados para confecção de páginas na Internet. Essa abordagem possibilitaria melhorar a forma de interação dos programas computacionais com tais páginas, e também tornaria sua utilização mais intuitiva por parte dos usuários [18].

Web Semântica permite incorporar semântica às informações de tal forma que as máquinas possam compreender a linguagem humana. Ela fornece estruturas e dar significado ao conteúdo das páginas Web, criando um ambiente onde agentes inteligentes e usuários possam trabalhar e interagir de forma cooperativa [18].

O esforço para promover a Web semântica é dirigido por *frameworks*, tais como: o RDF e OWL. Estes *frameworks* permitem que os dados sejam estruturados com metadados semanticamente relevantes, os metadados são anotações e *links* entre fontes de dados relacionadas [18].

Os tópicos discutidos neste documento dependem de algum conhecimento da Web semântica e algumas das suas principais tecnologias subjacentes para entender o objetivo do trabalho desenvolvido. Nas seguintes seções deste capítulo, alguns conceitos da Web Semântica aplicados na conexão de dados serão apresentados. A seção 2.1 traz conceitos acerca de RDF e a linguagem SPARQL. A seção 2.2 apresenta as bases da *Linked Data*. Nas seções 2.3, 2.4 e 2.5, conceitos sobre computação na nuvem, o paradigma de programação MapReduce e o Hadoop *framework* são explicados brevemente para mostrar como esses conceitos estão fortemente relacionados. A seção 2.6 apresenta uma visão de como as necessidades da Web podem ser complementadas com a tecnologia de computação na nuvem.

2.1 Tecnologias da Web Semântica

2.1.1 RDF

O *Resource Description Framework* (RDF) é uma infraestrutura que permite a codificação, troca e reúso de metadados estruturados [19]. A linguagem RDF é um padrão criado e mantido pela *World Wide Web Consortium* (W3C)[1] para descrição de recursos e suas relações na Web, como por exemplo: título, autor, data de publicação de uma página de conteúdo, etc. Esse padrão possibilita que agentes de software acessem não só a informação e a sua formatação, como também os seus meta-dados [19].

Uma das idéias trazidas pelo RDF é a utilização de um identificador único de recursos(*Uniform Resource Identifiers* - URI) [20] para cada recurso disponível na Web. Um recurso é qualquer coisa que possa ser descrita na Web, podendo ser algo diretamente recuperável, como é o caso de um documento eletrônico ou itens de uma loja de eletroeletrônicos.

O RDF possui um esquema de declarações simples para descrição de recursos. Cada declaração é representada através de uma tripla na forma:

$$< sujeito, predicado, objeto > \tag{2-1}$$

- O sujeito é representado por uma URI que define o recurso que se deseja descrever.
- O predicado é o identificador de uma propriedade associada ao recurso.
- O objeto é o valor dessa propriedade, podendo ser um literal ou uma outra URI.

O benefício real de URIs é que eles podem ser dereferenciáveis (do inglês *dereferenceable*) para adquirir informação adicional sobre o recurso que identificam. O RDF combina URI com anotações entendíveis por humanos, encapsulando está combinação dentro do *framework* o que facilita o entendimento de máquinas. Isto faz do RDF um ferramenta poderosa para publicar e conectar dados na Web [18].

[1] http://www.w3.org/

2.1.2 SPARQL

Existe uma linguagem projetada especificamente para modelagem e consultas sobre grafos RDF, conhecida como SPARQL (*Simple Protocol and RDF Query Language*) [9]. SPARQL é a linguagem de consulta padrão da Web Semântica, tornando-se uma recomendação da W3C em 2008. No entanto, SPARQL não é somente uma linguagem de consulta declarativa, também é um protocolo [21] usado para enviar consultas e recuperar resultados através do protocolo HTTP. Fontes de dados na *Linked Data* tipicamente fornecem um SPARQL *endpoint*. O SPARQL *endpoint* é um serviço Web com suporte ao protocolo SPARQL e possui uma URI específica para receber requisições HTTP com consultas SPARQL e retornar os resultados dessas consultas.

A sintaxe de uma consulta SPARQL é muito parecida com a de SQL no uso do **SELECT** e o **WHERE**, cláusulas que definem como os dados devem ser selecionados dado um conjunto de parâmetros. No entanto, enquanto SQL trabalha sobre banco de dados relacionais, onde os dados são armazenados em coleção de tabelas, SPARQL percorre grafos RDF em busca triplas que contêm padrões fornecidos nos parâmetros das consultas. Para poder consultar os grafos RDF, eles devem estar armazenados em um banco de dados especializado em armazenamento de triplas, e disponíveis através de um SPARQL *endpoint*.

Uma das dificultades que se apresentam ao consultar uma fonte de dados RDF é a quantidade de resultados retornados pelo serviço SPARQL *endpoint*. O tempo de requisição de uma consulta pode exceder o tempo configurado no serviço. É por isso que SPARQL oferece duas cláusulas que tratam estas limitações, **OFFSET** e **LIMIT**. A cláusula **LIMIT** restringe a quantidade de resultados que a consulta pode recuperar do conjunto de dados RDF consultado. A cláusula **OFFSET** opera sobre o conjunto de registros que a consulta representa e define o início do subconjunto de dados a ser recuperados.

Existem mais recursos de SPARQL que auxiliam a consultar grafos RDF com uma variedade de níveis de especificação, tornando SPARQL uma ferramenta útil no contexto da Web Semântica.

2.2 Linked Open Data

2.2.1 Principios de Linked Data

Como parte do desenvolvimento da Web Semântica, surgiu o conceito de *Linked Data* que pode ser definido como um conjunto de boas práticas para publicar e conectar conjuntos de dados estruturados na Web, com o intuito de criar uma "Web de Dados" [1]. Estas práticas auxiliam na migração dos Web atual para a Web Semântica. A *Linked Open Data* (LOD) é uma iniciativa que tem como objetivo a padronização da forma como os dados são publicados e interligados na Web semântica, baseada em uma licença aberta [1]. Essa iniciativa tem ganhado força nos últimos anos com a publicação de um número significativo de conjunto de dados (do inglês *datasets*) de escala considerável [22]. Juntos, esses conjuntos de dados formam a Web de Dados e podem ser visualizados como uma nuvem de conjuntos interligados. Uma representação da nuvem de LOD pode ser encontrada em [23].

Os princípios fornecidos pela *Linked Data* são os seguintes [1]:

1. Uso de URIs para identificação de qualquer tipo de recurso que possa ser descrito na web. Esses recursos podem variar desde conceitos concretos, como pessoas, animais ou objetos, a conceitos abstratos, como amor, amizade, compaixão, etc.

2. Toda URI deve ser dereferenciável, ou seja, quando são acessadas através da web via protocolo HTTP, deve ser possível obter o recurso ou alguma descrição dele.

3. As informações obtidas pelo acesso a uma URI devem estar em um dos formatos padrões da web semântica (RDF, RDFS).

4. Os conjuntos de recursos devem estar conectados à URIs de recursos pertencentes a outros conjuntos publicados na LOD, permitindo a descoberta de um maior número de informações através da exploração dessas conexões (*links*).

2.2.2 Descrição do conjunto de dados na Linked Open Data

Criar uma descrição compacta de um conjunto de dados em RDF que se pretende publicar, e ao mesmo tempo conseguir níveis de precisão detalhada das estatísticas de seu conteúdo, é um processo caro e difícil de manter, devido às mudanças constantes que sofrem os dados. Entretanto, conseguir uma estatística certa de uma fonte de dados na nuvem da *Linked Open Data*, significa criar consultas consistentes para obter mais informação, além de poder ajudar a resolver o problema de conexões de dados na nuvem de *Open Data* [24].

O vocabulário VoiD[2] permite descrever conjuntos de dados e dados conectados, como também habilitar uma série de tarefas a serem automatizadas de um modo escalável. O VoiD é um vocabulário composto por um conjunto de instruções que possibilita o descobrimento e o uso de um conjunto de dados conectados [24].

O princípio do VoiD é usar necessidades reais para orientar o escopo de um projeto, e reutilizar vocabulários existentes, sempre que for possível, no lugar de criar um próprio. O VoiD envolve quatro tipos da metadados [24]:

1. **Metadados gerais:** Segue o modelo de *Dublin Core*[3], que visa descrever objetos digitais, tais como: vídeos, sons, imagens, textos e *sites* na Web.

2. **Acesso aos Metadados:** Descreve como os dados em RDF podem ser acessados, como por exemplo, através do protocolo HTTP, arquivos RDF *dumps* e SPARQL *endpoint*.

3. **Metadados estruturais:** Descrevem a estrutura e o esquema da fonte de dados RDF. Isto é útil para tarefas de consultas federadas e integração de dados.

4. **Descrição de *links* entre fontes de dados:** Estas descrições são úteis para entender como múltiplas fontes de dadas são relacionadas e como podem ser usadas juntas.

Outra abordagem para descrever conjuntos de dados é a *Semantic Sitemaps*[4] [25], uma extensão do *Sitemap Protocol*[5](uma abordagem bem sucedida para indexar a *Deep Web*). Ela fornece uma efetiva e simples solução para publicar e descrever dados, de forma que os usuários possam escolher o método de

[2]http://semanticweb.org/wiki/VoiD
[3]http://dublincore.org/
[4]http://sw.deri.org/2007/07/sitemapextension/
[5]http://www.sitemaps.org/protocol.html

acesso mais adequado. O *Sitemap Protocol* é baseado na noção estendida de *authoritative information* [25].

2.2.3 Descoberta de links entre conjunto de dados

A publicação de conjuntos de dados na LOD pode ter certo grau de automatização. Ferramentas como a D2R Server [26] e o Open-Link Virtuoso [27] são utilizadas para mapear uma estrutura de dados em um conjunto de triplas em RDF.

Atualmente, a quantidade de dados publicados na Web semântica ultrapassa os 25 bilhões de triplas e os *links* contribuem apenas com 5% desse total. Isso mostra que, mesmo utilizando essas ferramentas, ainda é necessário desenvolver mecanismos automatizados para estabelecer *links* entre fontes de dados triplificadas e as já publicadas na LOD [24].

O problema de encontrar *links* entre indivíduos similares é normalmente conhecido na literatura como *Matching* [28]. Um processo de *Matching* clássico possui como entrada:

1. Dois conjuntos de dados de indivíduos **F** e **A**, referidos respectivamente como conjuntos fonte e destino.

2. Uma métrica de distância entre indivíduos, a qual fornecerá um valor correspondente ao grau de similaridade entre os mesmos.

3. Um limiar **t**, onde todo par de indivíduos que possuir grau de similaridade abaixo de t não é considerado um *match* e, todo par de indivíduos que possuir grau acima de **t** é considerado um *match*, sendo possível estabelecer um *link* de similaridade entre esses indivíduos.

Um dos desafios em estabelecer *links* entre indivíduos similares está em decidir quais informações são relevantes para se identificar um *match*. Uma das técnicas frequentemente encontradas na literatura consiste em selecionar uma determinada propriedade, ou um conjunto de propriedades, que têm como alvo um literal de um determinado tipo. Depois pode-se aplicar uma métrica de similaridade para obter um valor numérico, que expresse o grau de proximidade dos indivíduos comparados. Essa técnica pode falhar em casos onde apenas os literais associados não sejam suficientes para identificar um indivíduo. Nesse caso é necessário levar em consideração as relações entre os indivíduos, onde será preciso considerar, o contexto no qual o indivíduo está inserido para o cálculo de similaridade [29].

2.3 Computação na Nuvem

A computação na nuvem está tornando-se uma das palavras chaves da indústria de Tecnologia da Informação. A nuvem é uma metáfora para a Internet ou infraestrutura de comunicação entre os componentes arquiteturais, baseada em uma abstração que oculta a complexidade de sua infraestrutura. Cada parte desta infraestrutura é provida como um serviço que é alocado em centros de dados, utilizando *hardware* compartilhado para computação e armazenamento [30].

Para utilizar esses serviços, os usuários necessitam apenas ter nas suas máquinas um sistema operacional, um navegador e acesso à internet. Todos os recursos computacionais estão disponíveis na nuvem e as máquinas dos usuários não necessitam ter altos recursos computacionais, diminuindo o custo na aquisição de máquinas. Todo hardware pode ser utilizado para realizar alguma tarefa que seja adequada ao seu poder de processamento. Além disso, novos recursos de *hardware* podem ser adicionados a fim de aumentar o poder de processamento e cooperar com os recursos já existentes [31].

A infraestrutura do ambiente de computação na nuvem normalmente é composta por um grande número, (centenas ou milhares de máquinas físicas ou nós físicos de baixo custo) conectadas por meio de uma rede, como ilustra a Figura 2.1. Cada máquina física tem as mesmas configurações de software, mas pode ter variação na capacidade de hardware em termos de CPU, memória e armazenamento em disco [32]. Dentro de cada máquina física existe um número variável de máquinas virtuais (VM) ou nós virtuais em execução, de acordo com a capacidade do hardware disponível na máquina física [31].

O modelo de computação na nuvem foi desenvolvido com o objetivo de fornecer serviços de fácil acesso, baixo custo e com garantias de disponibilidade e escalabilidade. Este modelo visa fornecer basicamente três benefícios. O primeiro benefício é reduzir o custo na aquisição e composição de toda infraestrutura requerida para atender as necessidades das empresas, podendo essa infraestrutura ser composta sob demanda e com recursos heterogêneos e de menor custo. O segundo é a flexibilidade que esse modelo oferece no que diz respeito à adição e substituição de recursos computacionais, podendo escalar tanto em nível de recursos de *hardware* quanto de *software* para atender as necessidades das empresas e usuários. O último benefício é prover uma abstração e a facilidade de acesso aos usuários destes serviços. Neste sentido, os usuários dos serviços não precisam conhecer aspectos de localização física e de entrega dos resultados destes serviços [31].

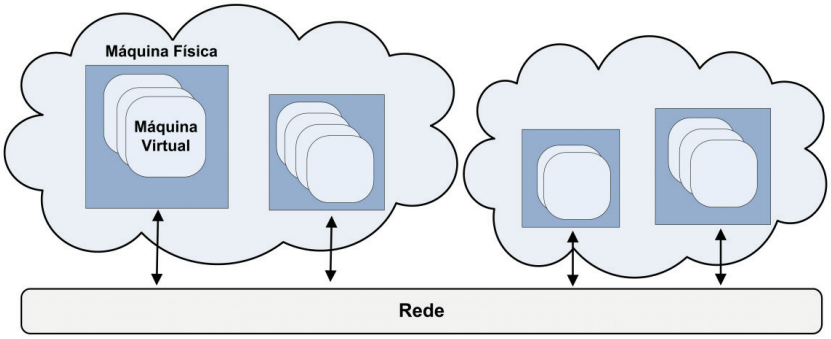

Figura 2.1: Ambiente de Computação na Nuvem [31]

2.4 Modelo MapReduce

2.4.1 Definição

MapReduce é um modelo de programação distribuída originalmente projetado e implementado pela Google para o processamento e geração de grandes conjunto de dados [33].

O modelo é baseado em duas funções básicas, *map* e *reduce*, que são similares as funções "map" e "reduce" presentes na linguagem funcional Lisp [34]. O modelo de programação de MapReduce da Google provou ser eficaz com o simples princípio de "mapeamento e redução" permitindo um alto grau de paralelismo com pouco custo operacional. Hoje o *framework* de MapReduce é usado na Google para processar dados medidos em *petabytes* sobre uma rede de milhares de computadores [34]. Neste modelo de programação toda a informação é codificada como tuplas da forma:

$$< chave, valor > \tag{2-2}$$

O fluxo de trabalho de uma tarefa no MapReduce pode ser resumida da seguinte forma:

1. A função *map* processa a entrada de tuplas retornando um outro conjunto de tuplas intermediárias *<chave2, valor2>*.

2. As tuplas intermediárias são agrupadas de acordo com o valor da sua *chave*.

3. Cada grupo gerado é processado pela função *reduce* que produzirá novas tuplas da forma *<chave3, valor3>*.

2.4.2 Exemplo: Contar o número de ocorrências de uma palavra

O problema do contador de ocorrências de palavras é um exemplo comum usado para descrever como MapReduce trabalha. O problema consiste em contar as ocorrências de palavras simples dentro de um texto e pode ser resolvido lançando uma simples tarefa MapReduce. Inicialmente, o texto usado como entrada para a tarefa é transformado em sequências de tuplas que tem como *chave* e *valor* as mesmas palavras do texto. Por exemplo, a sentença "MapReduce é um modelo de programação" é transformada em seis tuplas, cada uma contendo uma palavra da sentença tanto como *chave* quanto como *valor*.

O algoritmo MapReduce é definido a seguir:

1. O texto é transformado em tuplas da forma <*palavra, palavra*>, a função *map* processa a entrada e retorna um outro conjunto de tuplas intermediárias <*palavra, 1*>. A chave das tuplas é a palavra em questão, com um valor representativo de *1*.

2. Depois que a função *map* acaba de processar toda a entrada de dados, as tuplas intermediárias são agrupadas pelo valor da *chave*.

3. A função *reduce* faz a contagem das tuplas no grupo, o número de tuplas reflete o número de vezes que as funções *map* (*mappers*) encontram a palavra no texto. A saída da função *reduce* é uma tupla da forma <*palavra, contagem*>. Esta última tupla codifica o número de ocorrências da palavra no texto.

O algoritmo da tarefa MapReduce é apresentado na lista 2.1, e um quadro geral da execução da tarefa é apresentada na Figura 2.2

Lista 2.1: Algoritmo: Contar o número de ocorrências de uma palavra [34]

```
map (key, value):
// key : palavra
// value: palavra
output . collect(key , 1)

reduce (key , iterator values):
// key : palavra
// count : contagem
count = 0
for value in values
count = count + 1
output.collect(key, count)
```

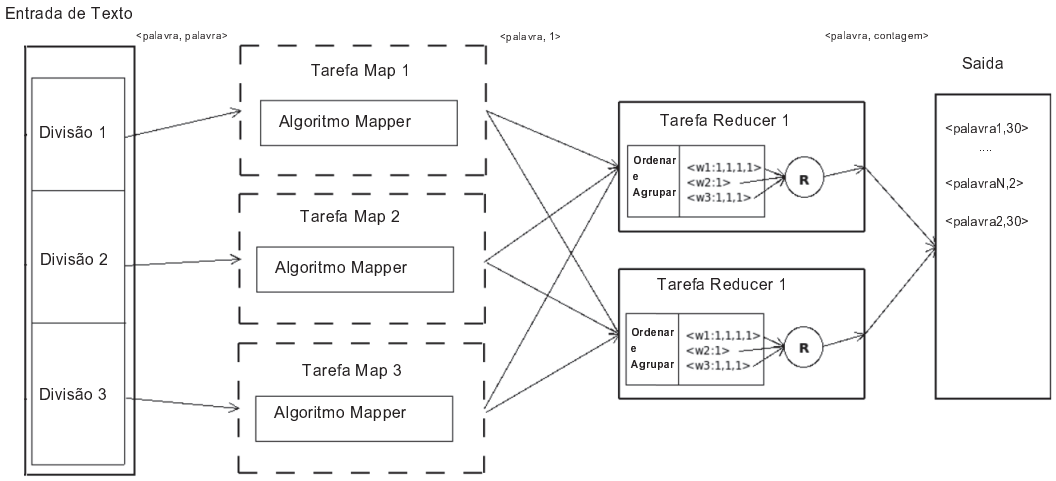

Figura 2.2: Execução do exemplo de contar palavras [34]

2.4.3 Paralelismo e execução de tarefas

As funções *map* e *reduce* trabalham somente com uma pequena fração da entrada e não precisam acessar outros dados. Por consequência, a execução destas duas funções pode ser eficientemente distribuída sobre vários nós após a entrada de dados ser divida em pedaços de mesmo tamanho. Com um *framework* como Hadoop, o usuário pode submeter algumas tarefas MapReduce e o *framework* as executará sobre os nós disponíveis levando em conta dos detalhes técnicos [34].

Uma típica execução de uma tarefa MapReduce é ilustrada na Figura 2.3. Primeiro o mestre (do inglês *master*) que em Hadoop é nomeado de *jobtracker*, divide a entrada em vários pedaços e atribui aos trabalhadores (do inglês *workers*) a execução de tarefas *map*, que são as responsáveis em processar a entrada dividida. Os trabalhadores lêem a entrada dividida que lhes foi atribuída, executam o código do algoritmo da função *map* e armazenam localmente a sua saída. A saída é particionada de acordo com às *chaves* das tuplas e cada partição é processada por uma particular tarefa de tipo *reduce*. O *framework* atribui cada uma das tarefas *reduce* aos trabalhadores e eles remotamente acessam à informação que precisam, executam o algoritmo da função *reduce*, e retornam um conjunto de tuplas como saída [34].

2.5 O Hadoop framework

O Hadoop *framework* é um projeto de código aberto (do inglês *open source project*) desenvolvido em Java, apoiado pela *Apache Software Fouda-*

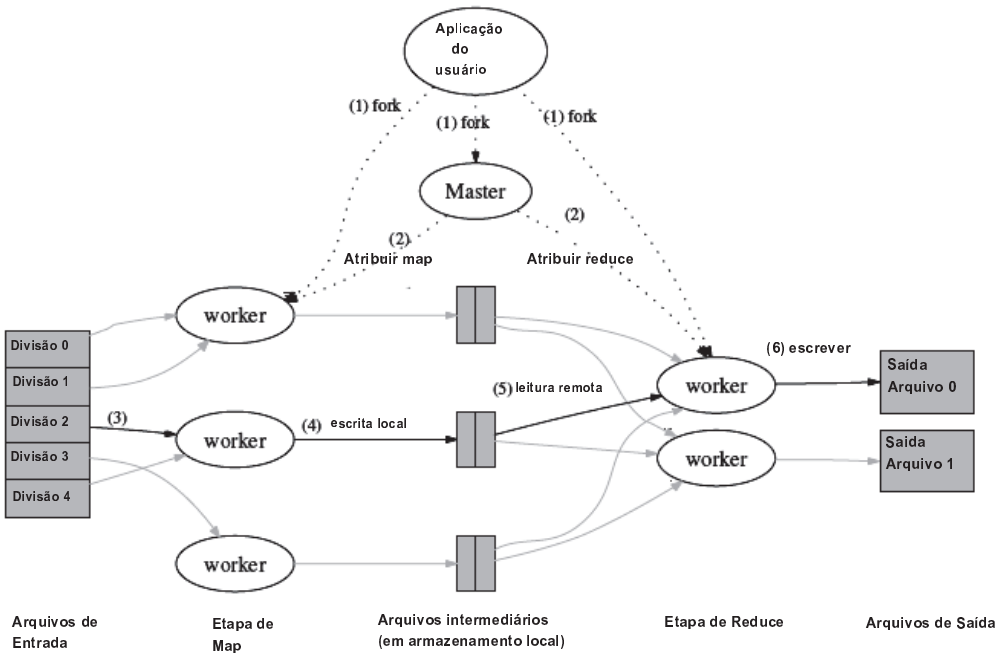

Figura 2.3: Execução paralela de uma tarefa MapReduce [34]

tion[6] [35]. Hadoop é uma das maiores contribuições do Yahoo! que implementa o modelo de programação MapReduce, e também uma livre alternativa ao *framework* de MapReduce desenvolvido pela Google, que não é de acesso público [34, 36].

Atualmente Hadoop é o mais popular dos *frameworks* que implementam MapReduce, isto pela sua licença de código aberto e o desenvolvimento intensivo que foi feito sobre ele nos últimos anos.

O Hadoop *framework* roda sobre uma rede de computadores e usa um sistema de arquivos distribuído para armazenamento dos dados. O usuário pode lançar algumas tarefas de MapReduce em cima do *framework* e é ele que distribui a execução das tarefas entre os nós da rede. Uma tarefa MapReduce tipicamente lê as entradas do sistema de arquivos distribuído, processa a entrada e escreve a saída no mesmo sistema de arquivos [34].

Hadoop fornece sistema de arquivos distribuído nomeado *Hadoop Distributed File System* (HDFS). O HDFS está fortemente conectado ao resto do *framework*, sendo fácil de instalar e configurar. O HDFS não é a única opção de armazenamento de dados, Hadoop pode usar outros sistemas de arquivos, como por exemplo, o Amazon Simple Storage Service (S3) [34].

A primeira ação de Hadoop é analisar a entrada de texto e dividi-la em alguns fragmentos, onde cada um deles será processado por um simples

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

mapper na rede de computadores. Estes fragmentos são igualmente divididos, assim todos os *mappers* recebem a mesma quantidade de entradas.

No *framework* existem quatro tipos de programas em execução que auxiliam no processamento dos dados. O primeiro é nomeado *jobtracker* e atua como mestre, submete e coordena a execução das tarefas em diferentes unidades computacionais. O *namenode* é o correspondente do *jobtracker* para o sistema distribuído HDFS. O *namenode* é o responsável pela replicação dos blocos, organizando e armazenando informação da atividade dos nós. O *tasktracker* é o programa que realiza a computação das tarefas. O *tasktracker* aceita a tarefa atribuída pelo *jobtracker* e reporta os resultados obtidos ao mesmo *jobtracker*. Cada nó computacional na rede contém um *tasktracker* sendo executado. O *tasktracker* é visto como o trabalhador (*workers*) da rede. O último programa é o *datanode* que o análogo do *tasktracker* para o sistema distribuído HDFS. Uma instalação de Hadoop contém pelo menos um *jobtracker* e vários *tasktracker*. No caso de usar HDFS, devemos ter um *namenode* e vários diferentes *datanodes*. Normalmente, os *datanode* são executados no mesmo computador junto com os *tasktracker*, assim eles podem agir contemporaneamente como escravos (*slaves*) no processo de armazenamento de HDFS e na computação de Hadoop, respectivamente [34, 36].

Uma tarefa MapReduce antes de ser executada deve ser corretamente configurada. O *framework* exige que o usuário indique qual *mapper* e que *reducer* serão usados e como a entrada e a saída devem ser processadas.

Uma tarefa MapReduce trabalha com informação codificada em tuplas e a natureza da data é oculta. Pode ser que a entrada seja composta de simples arquivos ou uma coleção de tabelas de bancos de dados. Uma tarefa MapReduce só vê a informação codificada em tuplas e a fonte real dos dados é desconhecida. É por esta razão que Hadoop usa tipos de objetos específicos para processar a entrada e saída (I/O), convertendo os dados de seu formato original a uma sequência de tuplas. O usuário deve configurar a tarefa indicando os tipos de dados que devem ser usados [34].

2.6 Web semântica e computação na nuvem

Com o crescimento do número de conjunto de dados conectados na nuvem da LOD, a automatização de determinadas tarefas, tais como a descoberta, seleção e otimização, tornam-se cada vez mais importantes [24].

Mesmo assim, tendo uma abundância da informação disponível, descrever metadados de forma adequada é requisito essencial para a publicação e consumo de dados abertos. Com frequência, a publicação de dados abertos

está relacionada com um conjunto de diretrizes que permitem fácil reuso e integração semântica [37].

O principal desafio é entender o conjunto de dados, o esquema e as conexões que são usadas dentro dele. Usar um navegador RDF para explorar dados interligados pode ser um exercício tedioso e muito demorado para grandes quantidades de dados. Uma abordagem do tipo *trial-and-error* sobre um esquema que representa um conjunto de dados RDF, consiste em testar estruturas do esquema, tornado-se difícil quando os dados usam um esquema de forma completa, ou usam vários esquemas [3]. Na nuvem de LOD, isto é ainda mais evidente devido às ontologias heterogêneas usadas em cada fonte de dados, então é necessário procurar outra abordagem. Tudo isto para entender a estrutura dos dados e identificar cenários onde estão envolvidos recursos de uma ou mais fontes de dados, com o objetivo de contribuir com conexões entre elas [3].

Nesse sentido, a geração de metadados e de estatísticas de grandes quantidades de dados é útil em muitos cenários [37]. O caso mais óbvio é quando o engenheiro de dados se depara as seguintes questões:

(a) Como procurar informação sobre um tópico especifico?

(b) Como ele pode conhecer que dados estão disponíveis na fonte?

(c) Como ele pode rapidamente descobrir conexões de uma fonte de dados para outra?

(d) Que conexões usadas ele já tem?

A programação de *crawling* e consultas sobre dados conectados é outro cenário, onde as estatísticas e metadados podem dar suporte à tomada de decisões e ajudar alcançar melhores resultados.

Para o processamento e entendimento da Web de dados é necessário definir os aspectos que a caracterizam [38]:

1. Por um lado a Web de dados é interligada e a análise de suas conexões têm sido fundamental na obtenção de um maior entendimento da sua natureza implícita.

2. Ao mesmo tempo, os dados são distribuídos e consistentes, mas somente dentro dos limites de um *site* independente.

3. A Web está composta de uma enorme quantidade de dados, sendo o *pentabyte* sua unidade de medida.

Para processar este tipo de dados, os principais provedores de busca na internet tem sido pioneiros nas formas de processar grandes quantidades dados, sobre os *clusters* de computadores interconectados com a tecnologia de rede dominante. As publicações do Google [33], sobre o paradigma de MapReduce, e a iniciativa de Yahoo! com a implementação de Hadoop, tem um incremento da atenção entre os desenvolvedores e usuários.

O estilo da computação de MapReduce é compatível com as possibilidades oferecidas pelo novo paradigma de computação na nuvem. Este estilo de computação geralmente fornece uma abstração útil para que os desenvolvedores possam focar na tarefa em questão (processamento paralelo e distribuído, armazenamento ou indexação de dados). A execução da computação "nas nuvens" refere-se ao modelo no qual um aplicativo solicita recursos computacionais de um provedor de serviços sem necessidade de se preocupar com os detalhes da oferta computacional.

Um exemplo que recentemente ganho muita popularidade é o serviço *Amazon Elastic Computing* [39], o qual permite que os recursos computacionais sejam alocados em poucos minutos e incrementados dinamicamente conforme a necessidade. O MapReduce é agnóstico com respeito ao tamanho do *cluster* onde ele está rodando. Executar Hadoop nessa arquitetura de nuvem é uma estratégia atrativa para a computação de dados intensivos, sendo antecipadamente minimizado ou otimizado o investimento de infraestrutura [38].