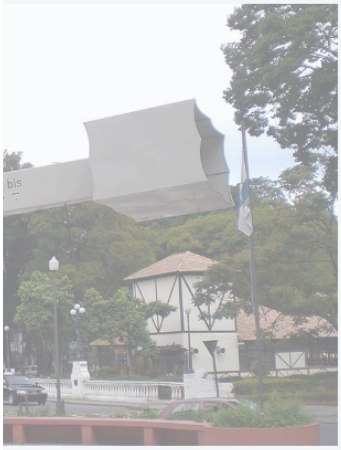
SBBD 2015

XXX SIMPÓSIO BRASILEIRO DE BANCO DE DADOS

13 A 16 DE OUTUBRO DE 2015

PETRÓPOLIS - LABORATÓRIO NACIONAL DE COMPUTAÇÃO CIENTÍFICA - LNCC



TÓPICOS EM GERENCIAMENTO DE DADOS E INFORMAÇÕES 2015

CARMEM S. HARA, FABIO PORTO E EDUARDO OGASAWARA (ORG.)

Realização

Laboratório Nacional de Computação Científica

CEFET/RJ

TÓPICOS EM GERENCIAMENTO DE DADOS E INFORMAÇÕES 2015

1ª Edição

Editora

Sociedade Brasileira de Computação

Organizadores

Carmem S. Hara, Fábio Porto e Eduardo Ogasawara

ISBN: 978-85-99961-18-6

Simpósio Brasileiro de Banco de Dados (30. 2015, out. 13-16 : Petrópolis, RJ)
S612t Tópicos em gerenciamento de dados e informações 2015 [recurso eletrônico] / [Organização : Carmen S. Hara, Fábio Porto e Eduardo Ogasawara]. Petrópolis, RJ. : Laboratório Nacional de Computação Científica, 2015.
xvii, 128p. : il.

ISBN : 978-85-99961186

1. Banco de dados - Encontros 2. Gerência de informações 3. SBBD I. Hara, Carmen S. (Org.) II. Porto, Fábio (Org.) III. Ogasawara, Eduardo (Org.) IV. Título.

CDD – 005.74

Editorial

Os capítulos deste livro foram escritos pelos autores dos minicursos apresentados no XXX Simpósio Brasileiro de Banco de Dados (SBBD 2015). Os minicursos tem como objetivo apresentar temas relevantes da área de Banco de Dados e promover discussões sobre os fundamentos, tendências e desafios relacionados ao tema abordado. Os minicursos tem três horas de duração e constituem uma excelente oportunidade de atualização para acadêmicos e profissionais que participam do evento.

Nesta edição, foram selecionadas três propostas para serem apresentadas durante o SBBD 2015. A seleção das propostas foi realizada por um Comitê de Avaliação formado por três avaliadores. Durante o processo de seleção, as propostas submetidas foram avaliadas por todos os membros do comitê. O primeiro minicurso, "Processamento de Grafos em Big Data", apresenta métodos e ferramentas que tem como objetivo dar suporte ao grande volume de dados que são naturalmente representados na forma de grafos, tais como os relacionamentos em redes sociais e os *links* da Web. No segundo minicurso, "Publicação e Consumo de Dados na Web: Conceitos e Desafios", são discutidos conceitos relacionados aos dados na Web, comparando o processo de publicação de dados neste meio com os bancos de dados convencionais. São tratados também o ciclo de vida dos dados, envolvendo dados abertos e conectados, e um conjunto de boas práticas para o processo de publicação e consumo de dados na Web. O terceiro minicurso, "Tecnologias para Gerenciamento de Dados na Era do Big Data", apresenta, de forma introdutória, técnicas de processamento de grandes volumes de dados utilizando Hadoop e tecnologias associadas, tendo como exemplo o problema de contagem de triângulos.

Gostaríamos de agradecer aos autores pela submissão das propostas e geração dos textos finais, e ao Comitê de Avaliação, pela dedicação e eficiência em todo o processo de seleção dos minicursos.

Carmem S. Hara (UFPR)
Coordenadora de Minicursos do SBBD 2015

Fabio Porto (LNCC)
Eduardo Ogasawawara (CEFET/RJ)
Coordenadores Locais do SBBD 2015

XXX Simpósio Brasileiro de Banco de Dados

13 a 16 de Outubro 2015
Petrópolis – RJ – Brasil

MINICURSOS

Promoção

Sociedade Brasileira de Computação – SBC
Comissão Especial de Banco de Dados (CEBD) da SBC

Organização

Laboratório Nacional de Computação Científica – LNCC
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca - CEFET/RJ

Comitê Diretivo do SBBD

Mirella M. Moro (DCC - UFMG), Coordenadora da CEBD
Altigran Soares da Silva (UFAM)
Caetano Traina Jr. (ICMC - USP)
Cristina Ciferri (USP - ICMC - USP)
Javam C. Machado (DC - UFC)
Marco A. Casanova (DI - PUC-Rio)
Vanessa Braganholo (UFF)

Coordenadores do SBBD 2015

Coordenadora do Comitê Diretivo
Mirella M. Moro (DCC - UFMG)

Coordenadores Locais
Fabio Porto (LNCC) e Eduardo Ogasawara (CEFET/RJ)

Coordenadoras do Comitê de Programa
Vanessa Braganholo (UFF)

Coordenadores da Sessão de Demos e Aplicações
Valéria C. Times (UFPE)

Coordenadores do Workshop de Teses e Dissertações em Banco de Dados
Ricardo Torres (UNICAMP)

Concurso de Teses e Dissertações em Banco de Dados
José Palazzo M. de Oliveira (UFRGS)

Coordenadora de Minicursos
Carmem S. Hara (UFPR)

Coordenadora de Tutoriais
Javam C. Machado (UFC)

Comitê de Organização Local

Fabio Porto, LNCC (coord.)
Eduardo Ogasawara, CEFET/RJ (coord.)
Adolfo Simões (LNCC)
Daniel S. Kaster, UEL (consultor)
Fernando Fereira dos Santos (consultor)
Paulo Pires (UFRJ, financeiro)

Comitê de Avaliação de Minicursos

Carmem Hara, UFPR (coordenadora)
Cristina Ciferri, USP São Carlos
Javam Machado, UFC

Sumário

cap:1

Capítulo 1

Processamento de Grafos em Big Data

Regis P. Magalhães, Luís Gustavo C. do Rêgo, José Antônio F. de Macêdo e Vânia M. P. Vidal

Abstract

Graphs are ubiquitous data structures in many areas of applications. With the advent of the Big Data phenomenon, processing and storage of graphs have required the use of new techniques and tools. In recent years, many systems for processing large-scale graphs were developed. In this context, this chapter provides an overview of the state of the art in graphs processing and details the main characteristics and limitations of these systems.

Resumo

Grafos são estruturas de dados ubíquas em inúmeros domínios de aplicações. Com o advento do fenômeno Big Data, o processamento e armazenamento dos grafos tem exigido o uso de novas técnicas e ferramentas. Nos últimos anos, diversos sistemas para processamento de grafos em larga escala foram desenvolvidos. Neste contexto, este capítulo apresenta uma visão geral do estado da arte no processamento de grafos e detalha as principais características e limitações desses sistemas.

1.1. Introdução

A teoria de grafos fornece uma abstração simples e elegante para modelar entidades e seus relacionamentos. Formalmente, um grafo ou rede consiste em um conjunto de vértices, os quais podem representar entidades, e um conjunto de arestas, que servem para ligar dois vértices, representando um relacionamento entre duas entidades. Através desta abstração é possível representar diversos fenômenos do mundo real.

De acordo com [Newman 2003] grafos têm sido aplicados para modelar quatro principais domínios da ciência: estruturação de informação, redes tecnológicas, eventos biológicos e ciências sociais. No domínio da estruturação das informações, os grafos também têm sido utilizados para entender a estrutura das páginas e *links*

da *World Wide Web* [Huberman 2003], assim como estruturas de relações entre artigos científicos [Chen et al. 2012]. As redes tecnológicas, tais como redes elétricas [Watts e Strogatz 1998], redes de ruas [Coelho da Silva et al. 2014] e linhas aéreas [Amaral et al. 2000], têm sido analisadas com o suporte dos grafos. Da mesma forma, grafos biológicos têm sido usados para entender fenômenos da natureza, tais como interação entre proteínas [Jeong et al. 2001] e redes neurais [White et al. 1986]. Nos estudos de ciências sociais, os grafos têm sido utilizados para modelar os relacionamentos entre indivíduos, ou grupo de indivíduos, e analisar as interações entre eles.

Nos últimos anos, o tamanho dos grafos tem crescido exponencialmente, alavancado principalmente pelo fenômeno *Big Data*. Por exemplo, a rede social do *Facebook*, a qual pode ser representada por um grande grafo social, atingiu um tamanho no segundo trimestre de 2015 da ordem de 1,5 bilhões de usuários[1], crescendo 13% a cada ano. É fácil imaginar que este grafo possuirá uma quantidade de arestas com pelo menos duas ordens de grandeza maior que o número de vértices. Outro exemplo de grande grafo é o grafo da *World Wide Web*[2], extraído em 2012, que contém 3,5 bilhões de vértices, representando as páginas da *Web*, e 129 bilhões de arestas, representando os *links* entre essas páginas.

O processamento de grandes grafos é um problema complexo, pois muitas técnicas existentes para processar grafos são computacionalmente custosas. No caso de grafos muito grandes, o processamento pode apresentar um tempo de execução muito alto, tornando inviável seu uso em aplicações reais. Além disso, certas aplicações também exigem a atualização de grandes grafos em tempo real [Vaquero et al. 2014]. Para lidar com esses tipos de problemas, são necessárias técnicas e ferramentas específicas. Tal necessidade tem atraído consistentemente a atenção do meio acadêmico e da indústria nos últimos anos. No entanto, poucos trabalhos têm sido publicados apresentando uma visão geral dos desafios e problemas relacionados com o domínio de processamento de grandes grafos. Até o presente momento os trabalhos de [Batarfi et al. 2015] e [Doekemeijer e Varbanescu 2014] são uns dos poucos que fornecem tal visão. Apesar deste esforço, tais trabalhos focam particularmente na experimentação e avaliação dos sistemas de processamento de grandes grafos, não detalhando os métodos e técnicas utilizados por tais sistemas. Este capítulo busca preencher esta lacuna e apresentar em detalhe os principais métodos e técnicas para processamento de grandes grafos.

Este capítulo está organizado da seguinte forma: a Seção 1.2 apresenta uma fundamentação teórica sobre grafos e seus principais algoritmos, conceitos de *Big Data* e definições importantes sobre Banco de Dados de Grafos. A Seção 1.3 aborda propriedades importantes dos Sistemas de Processamento de Grafos em Larga Escala e a Seção 1.4 comenta sobre as principais Ferramentas e Aplicações para esses sistemas de processamento. A Seção 1.5 apresenta as limitações e desafios dessa área de pesquisa. Por fim, a Seção 1.6 faz um *overview* das tecnologias apresentadas de acordo com algumas propriedades principais na forma de uma tabela e a Seção 1.7 apresenta as considerações finais sobre o tema, lições aprendidas e destaca direcionamentos futuros, como as oportunidades e desafios encontrados.

[1]http://investor.fb.com/releasedetail.cfm?ReleaseID=924562

[2]http://webdatacommons.org/hyperlinkgraph/index.html

1.2. Fundamentação Teórica

Esta seção apresenta os principais conceitos necessários ao entendimento deste capítulo e relacionados ao uso de grafos, sistemas de armazenamento, distribuição de processamento, distribuição de dados, *Big Data* e bancos de dados orientados a grafos. Os principais tipos de algoritmos em grafos também são abordados.

1.2.1. Grafos

Um **grafo** é um par ordenado $G = (V, E)$ composto por um conjunto V de **vértices** e um conjunto E de **arestas**. Cada aresta em um grafo interliga dois vértices.

Dependendo da aplicação, as arestas podem ou não ter direção. Pode ser permitido ou não arestas ligarem um vértice a ele próprio e vértices e/ou arestas podem ter um peso (numérico) associado. Se as arestas têm uma direção associada, o grafo é chamado de grafo direcionado, dirigido, orientado ou digrafo.

Um **grafo direcionado** é um par $G = (V, E)$, onde E é um conjunto de pares ordenados de vértices, chamados de arcos, arestas direcionadas ou setas. Ele difere de um **grafo não-direcionado** em que o conjunto de arestas E é definido em termos de pares não ordenados de vértices.

Grafos são geralmente representados através de uma *matriz de adjacência* ou de uma *lista de arestas*.

1.2.2. Algoritmos em Grafos

Esta seção trata dos principais tipos de algoritmos em grafos e da necessidade de paralelizá-los quando são usados no contexto de *Big Data*. Uma visão geral dos algoritmos em grafos é apresentada por [Even 2011]. Ele aborda os algoritmos de caminho mais curto, árvores, busca em profundidade e busca em largura, fluxo de rede e grafos planares.

Um estudo realizado por [Guo et al. 2014b] sobre processamento de grafos obteve os algoritmos mais usados nesse contexto. Foram consultados os artigos das principais conferências de pesquisa nas áreas de bancos de dados, sistemas distribuídos e recuperação de informação. Conferências como *SIGMOD*, *(P)VLDB*, *HPCD* e outras foram consideradas no estudo, por representarem uma amostra significativa dos esforços da academia e da indústria em relação ao processamento de grafos. Para isso, foram extraídas informações de 124 artigos publicados em 10 importantes conferências entre 2009 e 2013. Foram encontrados 149 algoritmos nesses artigos. Eles foram agrupados em seis classes: Estatísticos, Travessia, Componentes Conexos, Detecção de Comunidades, Evolução e outros. Componentes conexos é um tipo de métrica global usada em grafos, que leva em conta todas as arestas do grafo. A Tabela 1.1 resume os algoritmos identificados na pesquisa.

Quase metade dos artigos (46,3%) usam alguma forma de travessia em grafo. Algoritmos relacionados a estatísticas sobre grafos e extração de componentes conexos correspondem a 16,1% e 13,4%, respectivamente. Outros algoritmos variados presentes em menos de 3% dos artigos totalizam 14,8% dos algoritmos usados nos artigos.

Tabela 1.1. Algoritmos em grafos [Guo et al. 2014b]

Classe	Algoritmos	Quant.	%
Estatística	Triangulação [Wang et al. 2010], Diâmetro [Jiang e Agrawal 2011], BC [Shun e Blelloch 2013]	24	16,1
Travessia	BFS, DFS, *Shortest Path*	69	46,3
Componentes Conexos	MIS [Buluç et al. 2013], BiCC [Cong e Makarychev 2012], Alcançabilidade [Cai e Poon 2010]	20	13,4
Detecção de Comunidade	*Clustering*, Vizinhos mais Próximos (kNN)	8	5,4
Evolução do Grafo	*Forest Fire Model* [Leskovec et al. 2005], *Preferential Attachment Model* [Barabási e Albert 1999]	6	4,0
Outros	Amostragem, Particionamento	22	14,8
TOTAL		149	100

Um algoritmo frequentemente usado em artigos é o *PageRank* [Page et al. 1999]. Ele é um tipo de algoritmo iterativo sobre grafos usado para medir a influência e a importância de elementos do grafo.

Aplicações de grafos geralmente consomem muitos recursos computacionais e levaram ao surgimento de várias propostas sobre como paralelizar e otimizar sua execução. [Quinn e Deo 1984, Ramachandran 1993] apresentaram algumas dessas técnicas para paralelizar a execução de algoritmos em grafos. [Harish e Narayanan 2007, Hong et al. 2011] trataram do processamento de grandes grafos através do uso do modelo de programação CUDA em unidades de processamento gráfico (*Graphics Processing Units* - GPUs).

1.2.3. Big Data

Nas duas últimas décadas, o aumento contínuo do poder computacional tem produzido um fluxo enorme de dados [Ji et al. 2012]. A cada dia, 2,5 quintilhões de bytes de dados são criados e 90% dos dados no mundo hoje foram produzidos nos últimos dois anos [Wu et al. 2014]. Como exemplos deste cenário, podem-se citar modernos centros de pesquisa em física, tais como *DZero*, que normalmente geram mais de um *terabyte* de dados por dia. A rede social *Facebook* possui mais de um bilhão de usuário e cem horas de novos vídeos são armazenados a cada minuto no *Youtube*. Estes exemplos exigem armazenamento eficiente, análise de grandes quantidades de dados e tomada de decisão instantânea. Este contexto de gestão e análise de dados tem sido denominado *Big Data*.

De acordo com [Begoli e Horey 2012], *Big Data* é a prática de coleta e processamento de grandes conjuntos de dados, bem como os sistemas e algoritmos utilizados para analisar estes conjuntos de dados massivos. Já [Wu et al. 2014] argumenta que *Big Data* refere-se a grandes volumes heterogêneos, fontes autônomas com controle distribuído e descentralizado, procurando explorar as relações complexas e em evolução entre os dados.

Com o atual uso de infraestruturas massivas de servidores virtualizados para processar um grande volume de dados distribuídos, é possível supor que a maior parte dos dados a serem processados serão armazenados em memória principal, localizados em suas respectivas máquinas virtuais. Sendo assim, a nova métrica para análise da complexidade de processamento deixa de ser o número de acessos a disco para se tornar quantidade de comunicação entre os servidores que processam essas informações em memória. Claramente, os métodos e técnicas de banco de dados deverão levar em consideração esta mudança de complexidade em um ambiente *Big Data*.

A necessidade da computação iterativa é notória em diversos tipos de aplicação no cenário *Big Data*, incluindo clusterização usando *K-Means*, *PageRank*, consultas recursivas relacionais, análise de redes sociais, etc. Essas técnicas visam processar os dados iterativamente até que a computação satisfaça uma determinada condição de parada, ou convergência. Este tipo de computação iterativa não é provido pelas tecnologias de bancos de dados atuais. De fato, são necessárias técnicas mais sofisticadas para lidar com computação iterativa, levando em conta os custos de comunicação.

Bancos de dados paralelos e outras tecnologias de processamento paralelo lidam com a falha dos nós de maneira excepcional. Em um cenário *Big Data*, onde uma grande quantidade de máquinas estão envolvidas, claramente as falhas deixam de ser uma exceção para se tornarem uma regra. Desta maneira, é necessário que o gerenciamento de dados em *Big Data* lide com a falha dos nós como um evento frequente, e não como uma exceção ao processamento.

Em um típico cenário *Big Data*, é necessária uma infraestrutura computacional para lidar com grandes volumes de dados heterogêneos e distribuídos. Considerando o grande volume a ser analisado e que o processamento intensivo dos dados está além da capacidade de qualquer máquina individual, o cenário *Big Data* demanda a disponibilização de novos modelos computacionalmente eficientes [Lin e Dyer 2010]. Neste contexto, a computação baseada em clusters permite aproveitar grande número de processadores em paralelo para resolver um problema de computação [Pavlo et al. 2009]. Claramente, a disponibilização de uma infraestrutura de processamento em larga escala exige uma infraestrutura de software compatível, a qual possa tirar vantagem da grande quantidade de máquinas e mitigar o problema de comunicação entre essas máquinas. Com o crescente interesse em clusters, aumentou a quantidade de ferramentas para utilizá-los, dentre as quais se destacam os frameworks *MapReduce* [Dean e Ghemawat 2008] e *Spark* [Zaharia et al. 2012], utilizados para gerenciar grandes quantidades de dados em clusters de servidores. Eles são atraentes porque oferecem um modelo simples por meio do qual os usuários podem expressar programas distribuídos relativamente sofisticados.

Em muitas situações, o processo de análise deve ser eficiente e quase em tempo real, pois o armazenamento de todos os dados observados é quase inviável [Wu et al. 2014]. Como resultado, um volume de dados sem precedentes necessita de análise eficiente e eficaz. Para tanto, técnicas de mineração de dados podem ser utilizadas para analisar e entender os dados a serem manipulados. A análise é baseada em modelos capazes de sumarizar dados, extrair novos conhecimentos ou realizar predições. Estes modelos podem ser utilizados para construir um software que possibilite identificar o perfil de clientes para conceder empréstimos bancários, aplicações de recomendação

de busca de amigos em redes sociais, que envolvem grafos que possuem bilhões de nós e arestas ou ainda, sistemas de software que identifiquem possíveis ameaças terroristas [Rajaraman et al. 2012].

1.2.4. Bancos de Dados de Grafos

Modelos de bancos de dados de grafos são aqueles cujas estruturas de dados para o esquema e instâncias são modeladas como grafos ou generalizações deles, e cuja manipulação de dados é expressa por operações orientadas a grafos [Angles e Gutierrez 2008].

Sistemas de Gerenciamento de Bancos de Dados de Grafos (SGBDGs) como Neo4j [Neo Technology 2015], OrientDB [Orient Technologies 2015] e Titan [Aurelius 2015] são SGDBs otimizados para estruturas de grafos. Eles possibilitam armazenamento persistente, fácil manipulação, atualização e consulta aos dados. Também provêem uma *API*, linguagens para definição de dados, manipulação e consulta, além de otimizador de consultas, módulo de armazenamento, processamento de transações e ferramentas de gerência (*backup*, recuperação, *tuning*, etc.). Uma comparação entre diversos bancos de dados de grafos foi realizada por [Angles 2012]. Ele analisa características relacionadas à armazenamento de dados, linguagens de consulta, estruturas de dados, restrições de integridade e suporte a consultas tradicionais em grafos. Os bancos de dados analisados foram: AllegroGraph, DEX, Filament, G-Store, HypergraphDB, InfiniteGraph, Neo4J, Sones e vertexDB.

[McColl et al. 2014] também compararam diversos bancos de dados de grafos (Neo4j, OrientDB, InfoGrid, Titan, FlockDB, ArangoDB, InfiniteGraph, AllegroGraph, DEX, GraphBase e HyperGraphDB) usando os quatro tipos de algoritmos em grafos a seguir: travessia (*Single Source Shortest Paths*), componentes conexos (algoritmo Shiloach-Vishkin), iterativos (*PageRank*) e de atualização (conjunto de inserções e atualizações de arestas). Os experimentos usaram grafos de até 256 milhões de arestas. As comparações também incluíram ferramentas baseadas em MapReduce, HDFS, modelo de processamento de dados BSP (ver seção 1.3.2) e pacotes para processamento e/ou visualização de grafos em memória compartilhada (NetworkX, Gephi, MTGL, Boost, uRiKA, e STINGER). As ferramentas de visualização de grafos usadas não alcançam bom desempenho com grafos contendo mais de mil vértices.

[Ciglan et al. 2012] criaram um *benchmark* para operações de travessia em bancos de dados de grafos e o utilizaram para comparar o desempenho de alguns bancos de dados de grafos e, assim, identificar suas vantagens e limitações. Um resultado observado foi que operações que realizam travessia no grafo inteiro são viáveis quando o grafo inteiro está armazenado em memória principal. De modo geral, os bancos de dados de grafos são mais adequados para operação de travessia parcial do grafo.

Uma vantagem dos bancos de dados de grafos em relação aos SGBDs relacionais é que as associações entre as entidades são parte do modelo e não necessitam de junções entre relações para acessar elementos adjacentes [Vicknair et al. 2010]. Eles também geralmente provêem operações para atualização do grafo e alguma linguagem específica de domínio como Cypher, Gremlin ou SPARQL [Pérez et al. 2006] para realização de consultas [Holzschuher e Peinl 2013]. No entanto, o desempenho deles não é adequado para a realização de consultas analíticas em grandes grafos. Por esse motivo, essas consultas

são geralmente executadas através de processos *batch* [McColl et al. 2014] em sistemas de processamento de grafos em larga escala.

1.3. Sistemas de Processamento de Grafos em Larga Escala

O processamento de grafos é uma tarefa não trivial que vem recebendo muita atenção de pesquisadores em todo o mundo, devido a particularidades encontradas nessas estruturas: por exemplo, a velocidade com que essas estruturas crescem, a variedade das informações guardadas e o volume que esses grafos podem ter. Os dispositivos de georreferenciamento estão se tornando cada vez mais ubíquos em nosso dia-a-dia. A todo instante são geradas informações sobre localização das pessoas, seja através de aparelhos celulares, dos sistemas de navegação dos veículos ou até mesmo pelo endereço de rede dos computadores. Cada sistema que coleta essas informações pode armazená-las em diferentes formatos, como em bancos de dados, planilhas ou em arquivos de texto simples. Com a quantidade crescente de dispositivos gerando esses dados, a velocidade com que eles são coletados e armazenados também tende a crescer em igual velocidade. Esse é um cenário típico com vários desafios relacionados ao processamento de grafos em larga escala.

[Doekemeijer e Varbanescu 2014] distingue duas categorias bem claras no processamento de grafos em larga escala: análises *online* e *offline*. Na primeira categoria, também chamada de consultas em grafos, são analisadas apenas algumas partes da estrutura do grafo e busca-se um rápido tempo de resposta. Um exemplo dessa categoria seria uma consulta de caminho mais curto em um grafo dirigido ou não dirigido com arestas de pesos não negativos. É necessário explorar apenas alguns vértices da estrutura para retornar o melhor caminho de maneira eficiente. Já na segunda categoria, também chamada de processamento em *batch*, o grafo por completo é analisado. O algoritmo *PageRank* é um exemplo disso, onde todos os nós da rede são percorridos. Este capítulo tem como foco principal o processamento *offline* de grafos.

Pode-se caracterizar os sistemas de processamento de grafos em larga escala de acordo com algumas particularidades inerentes aos mesmos, como o tipo de armazenamento, o modelo de execução e de programação utilizado e o tipo de particionamento. A subseção seguinte apresenta alguns conceitos fundamentais ao entendimento desses sistemas.

1.3.1. Características dos Sistemas

Atualmente não existe um sistema de processamento capaz de lidar com grafos em larga escala que satisfaça todos os requisitos dos desenvolvedores. O que mais encontra-se na literatura são propostas de sistemas que otimizam uma ou mais características do ambiente em que serão implantados. Por exemplo, há ambientes com um *cluster* de várias máquinas com pouca memória ou um outro ambiente com vários processadores e uma grande quantidade de memória compartilhada; há situações em que o grafo apresenta um alto grau de dinamismo ou situações em que o grafo é praticamente estático; também há o caso de *clusters* com GPUs disponíveis para complementar o processamento do sistema, ou máquinas simples que só utilizam suas próprias CPUs. Esta seção apresenta algumas dessas importantes características dos sistemas de processamento de grafos em larga escala.

Tipo de armazenamento e Unidades de Processamento
[Yoneki e Roy 2013] argumentam que um dos fatores mais determinantes para o bom desempenho de sistemas de processamento de grafos é a velocidade com que um grafo pode ser carregado a partir da memória: seja do disco para a memória ou da memória para a CPU-Cache. Dito isso, precisa-se de uma otimização de leituras e escritas para cada tipo de mídia utilizada.

As estruturas de grafos podem ser armazenadas tanto em Disco Rígido (*Hard Drive* - HD) quanto em Discos de Estado Sólido (*Solid-State Disk* - SSD). Muitas vezes os sistemas são otimizados para cada uma dessas unidades de armazenamento, ao invés de uma implementação para um *hardware* genérico. Por exemplo, alguns sistemas como o FlashGraph [Zheng et al. 2015] possuem implementações específicas para utilização de SSDs, oferecendo um tempo menor de execução de algoritmos.

GPUs também podem ser utilizadas para retirar parte da carga de computação da CPU. Por exemplo, o TOTEM [Gharaibeh et al. 2012] processa vértices com alto grau na GPU e os vértices de menor grau na própria CPU.

Um ponto que também deve ser levado em conta é a heterogeneidade das máquinas em um *cluster*. Muitas vezes o *cluster* a ser trabalhado tem máquinas com *hardware* muito distintos e o sistema de processamento deve ser robusto o suficiente para gerenciar essa diferença de poder de processamento, além de particionar de forma eficiente o grafo entre as máquinas.

Modelos de execução
Uma decisão importante a ser tomada ao realizar o processamento de grafos em larga escala diz respeito à seleção do modelo de execução mais adequado. A diferença básica entre execuções síncronas e assíncronas está no momento em que uma atualização se torna visível para o restante do grafo [Doekemeijer e Varbanescu 2014].

Podem-se descrever esses dois tipos de execuções da seguinte forma:

1. **Síncrono**: o sistema realiza computações sobre conjunto de vértices ativos a cada iteração ou *pontos de sincronismo*, e propaga as atualizações realizadas para os outros vértices no final de cada iteração. Um determinado vértice só terá as novas informações de seus vizinhos ao final de cada iteração.

2. **Assíncrono**: no processo não existem pontos de sincronismo, mas cada vértice pode obter as novas informações de seus vizinhos, assim que possível.

A Figura 1.1 exemplifica um pouco esses dois modos. Alguns sistemas de processamento de grafos em larga escala implementam o suporte para ambos os tipos de execução, como por exemplo PowerGraph (Seção 1.4.5), Trinity/Graph Engine [Shao et al. 2013, Microsoft Research] e GRACE [Prabhakaran et al. 2012].

A execução síncrona abstrai um determinado algoritmo em grafo na forma de uma sequência de iterações na qual todos os vértices ativos no primeiro conjunto de vértices analisados irão executar computações em paralelo utilizando os valores dos vértices vizinhos, que tenham sido atualizados na iteração anterior. Os vértices que foram atualizados nessa iteração serão os próximos elementos a terem computações executadas. Como essas

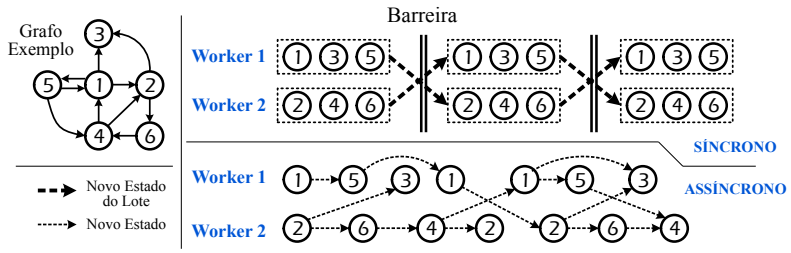


Figura 1.1. Comparação entre os fluxos de execução dos diferentes modos. Adaptado de [Xie et al. 2015].

execuções são feitas em lotes, esse tipo de execução favorece computações com leitura e escrita de dados intensiva [Xie et al. 2015].

A execução assíncrona realiza as computações dos vértices assim que possível, utilizando as informações do grafo e sem ser necessário o fim de uma iteração completa. Esse tipo de execução foi projetada para tirar vantagem das rápidas atualizações dos vértices e da velocidade de convergência de um algoritmo sob recursos de *hardware* suficientes. Além disso, essas execuções podem utilizar *pipelines* de processamento de vértices para amenizar a latência de rede. Entretanto, como a comunicação entre diferentes máquinas em execuções assíncronas se dá a qualquer momento, torna-se difícil o agrupamento de mensagens para reduzir o custo de rede e qualquer falha nesse pacote de mensagens prejudica as atualizações instantâneas características da assincronicidade. Ainda pior, leituras e escritas em um vértice também requerem que a atomicidade das transações seja mantida, o que causa uma significante sobrecarga de escalonamento [Xie et al. 2015].

Modelos de programação

O modelo de programação *MapReduce* tornou-se muito popular para o processamento de dados no cenário atual onde usualmente é necessário lidar com um grande volume de informações. Infelizmente, esse modelo não se mostrou muito adequado para a execução de algoritmos em grafos, que geralmente exigem computações iterativas. Os sistemas de processamento de grafos mais conhecidos atualmente foram desenvolvidos para preencher essa lacuna, como por exemplo o Apache Giraph (Seção 1.4.2), o GraphLab (Seção 1.4.5), Trinity [Shao et al. 2013] e GRACE [Wang et al. 2013, Xie et al. 2013].

O processo mais comum adotado por esses sistemas consiste basicamente em dividir a entrada em subgrafos para que possam ser paralelizados e, em seguida, utilizar um modelo de programação centrado em vértice [Tian et al. 2013]. Nesse modelo, os desenvolvedores expressam seus algoritmos da seguinte forma: cada vértice contém informações sobre ele mesmo e sobre suas arestas de saída. Infelizmente, o modelo centrado em vértice não faz uso de uma importante propriedade da maioria dos sistemas de processamento: a forma como os dados são particionados. Geralmente os grafos são divididos e distribuídos entre máquinas de forma a minimizar a comunicação dentro do sistema. Nesse modelo, cada vértice só tem informações sobre seus vizinhos diretos, o que torna a propagação de informações lenta.

Para superar esse problema, [Tian et al. 2013] desenvolveu o modelo centrado em grafo, envolvendo não somente um único vértice, mas sim todo o subgrafo de uma partição. Nesse modelo existem vértices internos ao subgrafo e também vértices de borda responsáveis pela comunicação entre os subgrafos de diferentes partições. Um vértice é

classificado como interno somente a um único subgrafo (o qual é chamado de dono do vértice), mas pode ser classificado como de borda em qualquer quantidade de partições. Nesse modelo, para cada vértice interno de uma partição há informações dos valores do próprio vértice, das suas arestas de entrada e saída e de todas as mensagens que chegam. Para um vértice de borda, somente informações de seus valores estão disponíveis. Esses valores dos vértices de borda são apenas temporários. O valor real deles estão nos seus respectivos subgrafos originais. Esses valores temporários são essencialmente cópias locais em diferentes partições: eles ainda precisam ser propagados para o vértice principal através da passagem de mensagens.

Tipo de arquitetura

Conhecer bem a plataforma que será utilizada é de fundamental importância para a correta escolha do sistema de processamento de grafos. A plataforma pode ser constituída de processadores com memória compartilhada ou de sistemas distribuídos. Os grafos que serão utilizados podem caber na memória principal ou na memória externa, caso a memória principal seja insuficiente. Esta seção discute esses conceitos e serve de parâmetro para a seleção do sistema de processamento de grafos mais adequado para cada contexto.

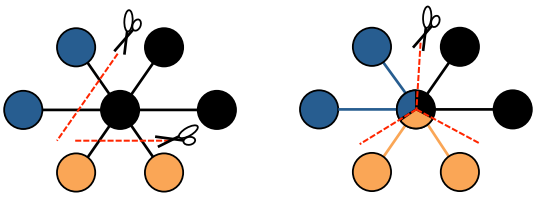
Sistemas de memória compartilhada permitem a comunicação eficiente entre processos, pois múltiplas tarefas acessam a mesma memória. Uma plataforma que utiliza o conceito de memória compartilhada não implica em uma única unidade de processamento: diversos processadores podem compartilhar uma grande memória global. O desafio do processamento de grafos em larga escala torna a arquitetura de memória compartilhada adequada para esse tipo de processamento. Apesar de sua escalabilidade ser limitada, a comunicação é bastante eficiente, com baixo custo e a depuração de código é bem direta. O desenvolvedor não tem que lidar com o gerenciamento de *clusters* e o *framework* não tem que lidar com certas particularidades da computação distribuída, como a tolerância a falhas.

Sistemas distribuídos são formados por várias unidades de processamento, onde cada uma delas tem sua própria memória privada. Os dados são particionados entre todos os nós e a comunicação explícita (por exemplo, por passagem de mensagem) é necessária para o sincronismo das computações. Pode-se escalar esses sistemas basicamente adicionando mais unidades de processamento. Comparado com sistemas de memória compartilhada, sistemas distribuídos são menos dependentes da evolução do *hardware* para serem escalados, mas a comunicação entre as máquinas pode rapidamente se tornar um gargalo em aplicações que utilizam grafos.

O suporte à memória externa se refere à habilidade de trabalhar com dados que não cabem na memória principal. Alguns sistemas de processamento de grafos são capazes de acomodar grafos que excedem o tamanho da memória agregada utilizando (a lenta) memória externa. Para sistemas distribuídos, os *frameworks* de propósito geral dão suporte a essa transferência de dados para o disco. Por exemplo, o *MapReduce* trabalha em fases onde o dado flui da memória principal para o disco e vice-versa. Mapeamentos e reduções são imediatamente escritos em um sistema de arquivos distribuídos. O gasto dessa escrita em disco faz com que o *MapReduce* seja menos utilizado em aplicações iterativas, fazendo com que a maioria dos sistemas de processamento de grafos distribuídos mantenham o grafo na memória agregada.

Tipos de particionamento

Ao contrário das execuções em paralelo que trabalham com dados mais genéricos onde as informações são processadas em nós isolados e independentes, o modelo de paralelização de grafos precisa processar as informações dos vértices ou arestas levando em consideração a influência que seus vizinhos possam ter sobre eles. Como consequência disso, a forma como essas informações são indexadas e particionadas são de fundamental importância para uma execução distribuída eficiente. Como as execuções sobre estruturas de grafos requerem muita comunicação entre seus elementos - nós e arestas tanto do mesmo subgrafo quanto de subgrafos diferentes - o particionamento adequado desse grafo pode minimizar o custo de armazenamento e de comunicação e garantir um melhor balanceamento entre os nós do *cluster*.



(a) Corte em Aresta (b) Corte em Vértice

Figura 1.2. Corte em Aresta e Corte em Vértice. O corte em aresta (a) divide o grafo quebrando determinadas arestas enquanto o corte em vértice escolhe o vértice a ser cortado e outros iguais dependendo do número de máquinas a ser utilizado. Adaptado de [Xin et al. 2013].

A maioria dos sistemas de processamento de grafos utiliza o **corte em aresta** para criar subgrafos e distribuí-los entre os nós. O corte em aresta atribui cada vértice a uma única máquina e distribui as arestas entre as máquinas onde estão os vértices que os compõe. É fácil perceber que o custo de comunicação e armazenamento do corte em aresta é diretamente proporcional ao número de cortes das arestas feito no grafo. Para diminuir esse custo, é preciso minimizar o número de cortes, assim como o número de vértices das máquinas com mais vértices.

Infelizmente, ao trabalhar com grafos em larga escala, construir um particionamento baseado no corte das arestas pode ter um custo muito alto, impossibilitando o processo. Como consequência disso, grande parte dos sistemas tem adotado a estratégia de distribuir aleatoriamente os vértices, criando um corte aleatório das arestas.

Como alternativa ao corte em aresta, que atribui os vértices a cada máquina, o corte em vértice distribui as arestas entre as máquinas e permite que os vértices estejam espalhados entre várias máquinas. O custo de comunicação e de armazenamento do corte em vértice é diretamente proporcional à soma do número de máquinas onde há vértices replicados. Logo, para reduzir esse custo, é preciso minimizar o número de máquinas com replicações de vértices. Em contraste ao corte em aresta, que tem apresentado uma eficiência baixa em relação a grandes grafos, existem resultados teóricos [Albert et al. 2000] e práticos [Gonzalez et al. 2012] que demonstram desempenho satisfatório ao usar cortes em vértices em grafos do mundo real.

Grau de dinamismo

Grafos são estruturas que, em determinadas aplicações, exigem mudanças constantes em

suas estruturas. Um exemplo dessas aplicações são as redes sociais. Estruturas de grafos são a base desses serviços onde arestas entre vértices podem ser criadas quando dois usuários começam uma amizade ou novos vértices são adicionados quando um novo usuário se cadastra. Aplicações também podem exigir atualizações rápidas em seus grafos. Por exemplo, bancos precisam detectar rapidamente tentativas de fraude em seus sistemas. [Vaquero et al. 2014] classifica o grau de dinamismo dessas estruturas em três aspectos:

1. **Alteração dos metadados**: quando informações das arestas e dos vértices são alteradas, como, por exemplo, seus pesos, é preciso propagar essas alterações o mais rápido possível para que futuras consultas sejam sempre efetuadas na versão mais atual do grafo.

2. **Alteração da topologia**: os vértices e as arestas que compõem o grafo podem ser adicionados ou removidos ao longo do tempo, modificando a topologia da estrutura. Quando um sistema altera com frequência a estrutura topológica do grafo, dois grandes problemas podem surgir: (i) o desbalanceamento de carga entre as máquinas utilizadas e (ii) o desbalanceamento do particionamento do grafo, acarretando em um custo de comunicação maior dentro do sistema.

3. **Alteração da demanda**: o tipo e a quantidade de consultas que um grafo recebe pode mudar de acordo com picos randômicos ou sazonais. Em sistemas distribuídos, essa alteração de demanda pode afetar diretamente o desempenho de uma máquina em particular, já que um *hotspot* pode ser identificado. Uma possível solução seria a alocação balanceada do grafo em diversas máquinas.

Com os sistemas atuais, realizar alterações nos metadados dos elementos dos grafos não apresenta desafios significativos. Por outro lado, alterar a topologia da estrutura continua sendo um grande problema. Alguns sistemas atuais [Shao et al. 2013, Cheng et al. 2012] permitem que vértices e arestas sejam adicionados ou removidos, mas não realizam nenhuma pós-otimização quando o grafo é alterado. Dessa forma, o particionamento do grafo entre as máquinas deixa de ser ótimo e o custo de comunicação interna aumenta, piorando o desempenho do sistema.

1.3.2. Modelos de Processamento de Grafos em Larga Escala

Esta seção apresenta alguns modelos de processamento de grafos frequentemente usados nos sistemas de processamento de grafos em Big Data.

MapReduce

O *MapReduce* é um modelo de programação paralelo especialmente criado para executar tarefas complexas e distribuídas. Em geral, esse modelo é composto por duas fases consecutivas, que, para a maioria das tarefas, são repetidas de forma iterativa: o mapeamento (*Map*) e a redução (*Reduce*). A primeira fase processa os dados em nós paralelos, onde a segunda, de redução, agrega o resultado da primeira. Em cada iteração, todo o conjunto de dados é dividido em partes, que, a cada iteração, são usados como entrada para a fase de mapeamento. Cada pedaço pode ser processado por apenas um nó de mapeamento. Uma vez que todos os dados foram processados, eles emitem pares de chave e valor para a fase de redução. Antes do inicio dessa nova fase, os pares são organizados de acordo

com seus valores para que cada nó de redução receba uma lista de valores relacionados a uma determinada chave. A saída da fase de redução é salva em um sistema de arquivos distribuído.

No caso do processamento de algoritmos em grafos iterativos, a estrutura do grafo e outros dados estáticos devem ser transferidos entres os nós que realizam as tarefas de *map* e *reduce*. Isso causa um custo de transmissão de rede muito alto e pode ser a maior desvantagem da utilização do *MapReduce* para o processamento de grafos.

Bulk Synchronous Parallel

O modelo paralelo síncrono em massa (*Bulk Synchronous Parallel* - BSP) consiste de uma sequência de iterações, chamadas de *supersteps* (super-passos). Durante um *superstep* o *framework* chama em paralelo uma função definida pelo usuário para cada vértice. A função especifica o comportamento em um único vértice V e um único *superstep* S. É possível ler mensagens enviadas a V no *superstep* $S-1$, enviar mensagens para outros vértices que serão recebidas no *superstep* $S+1$ e modificar o estado de V e das arestas que saem de V. As mensagens são geralmente enviadas através das arestas que saem do vértice, mas uma mensagem pode ser enviada a qualquer vértice com identificador conhecido.

A Figura 1.3 ilustra o funcionamento do modelo de programação BSP.

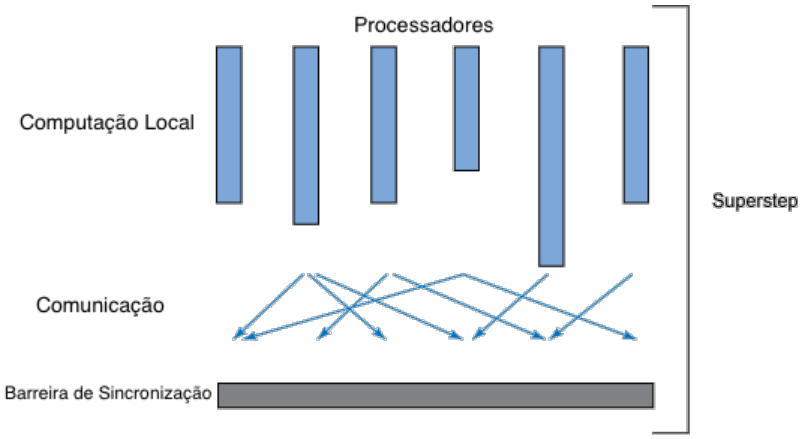


Figura 1.3. Modelo de programação BSP. Adaptado de [Sakr 2013].

No *superstep* 1 todos os vértices recebem o status *ativo*. Todos os vértices ativos executam a função *compute()* em cada *superstep*. Cada vértice pode promover sua própria mudança de status para *inativo* ao votar *parada (halt)* em qualquer *superstep*. O vértice realiza essa mudança de status ao receber uma mensagem. Um vértice pode voltar ao estado ativo ao receber uma mensagem na execução durante a execução de um *superstep* subsequente. Este processo é mostrado na Figura 1.4 e continua até que todos os vértices não possuam mais mensagens a enviar e, assim, se tornam inativos. A execução do programa finaliza quando todos os vértices ficam inativos em algum *superstep*.

A Figura 1.5 mostra um exemplo de mensagens entre um conjunto de vértices para obtenção do valor máximo de vértice. No *superstep* 1 cada vértice envia seu valor para seu vértice vizinho. Se o valor recebido for maior que o valor do vértice, ele atualiza seu valor com o valor recebido e envia seu valor para seus vizinhos. No entanto, se o valor

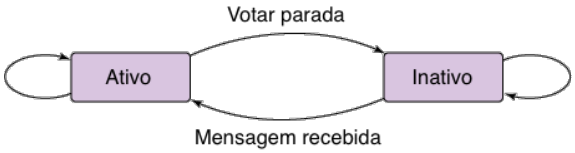


Figura 1.4. Esquema de votação dos vértices no modelo BSP. Adaptado de [Sakr 2013].

recebido for menor, o vértice mantém seu valor e vota parada. Assim, no *superstep* 2 da Figura 1.5, somente o vértice com valor 1 atualiza seu valor para um valor mais alto recebido (5) e envia seu novo valor. Esse mesmo processo ocorre no vértice com valor 2 durante o *superstep* 3. Finalmente, no *superstep* 4 todos os vértices votam por parada e a execução termina.

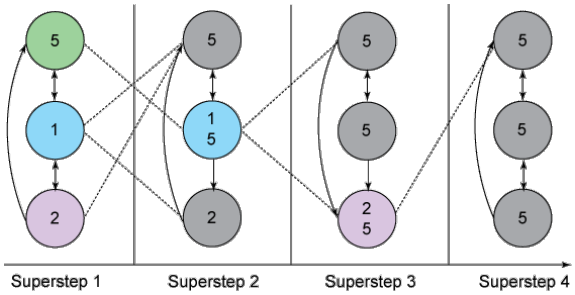


Figura 1.5. Exemplo de computação do valor máximo de vértice através do modelo BSP. Adaptado de [Sakr 2013].

O uso de um modelo de programação síncrono evita *deadlocks* e outros problemas comuns em sistemas assíncronos. Como as aplicações envolvendo grafos geralmente possuem muito mais vértices que máquinas, é importante, porém complexo, balancear o carregamento de máquinas de modo que a sincronização entre elas *supersteps* não adicione latência excessiva.

GAS - *Gather, Apply, Scatter*

O modelo de programação GAS (*Gather, Apply, Scatter*) representa três fases conceituais de um programa centrado em vértice [Gonzalez et al. 2012]. Na fase *gather* (reunir), valores de vértices e arestas adjacentes de um vértice V são reunidos e é realizada uma operação de soma sobre eles. O resultado dessa operação é usado na fase *apply* (aplicar) para atualizar o valor do vértice. Finalmente, a fase *scatter* (disseminar) usa o novo valor do vértice para atualizar dados das arestas adjacentes.

1.3.3. Métodos de Processamento de Grafos em Larga Escala

Parallel Sliding Window

A técnica de janela deslizante paralela (*Parallel Sliding Window* - PWS) processa eficientemente um grafo com valores de arestas atualizáveis a partir do disco, buscando minimizar a quantidade de acessos não sequenciais ao disco e ainda mantendo o modelo de computação assíncrono herdado do GraphLab (Seção 1.4.5). PSW processa o grafo em três estágios: (i) carregamento de um subgrafo a partir do disco; (ii) atualização dos vértices e arestas; (iii) escrita dos valores atualizados para o disco. Na estratégia de PSW, os vértices do grafo são divididos em intervalos. Cada intervalo é associado com um *shard* (fragmento), que armazena todas as arestas que possuem vértice destino naquele

intervalo. O processamento do grafo é realizado em intervalos de execução. São processados os vértices de um intervalo por vez. Quando o PSW move de um intervalo para o seguinte, ele desloca uma janela sobre cada um dos *shards*. Se o grau de distribuição de um grafo não for uniforme, o tamanho da janela será variável. A Figura 1.6 apresenta uma visualização dos estágios de uma interação do PSW. Nesse exemplo, os vértices foram divididos em quatro intervalos, cada um associado a um *shard*. Um subgrafo de vértices é construído a cada intervalo. As arestas que chegam aos vértices são lidas a partir do *shard* em memória (destacados com cor mais escura na figura), enquanto que as arestas que saem dos vértices são lidas de cada um dos *shards* deslizantes. As janelas deslizantes dos *shards* são apresentadas na figura como um retângulo sobre cada um deles.

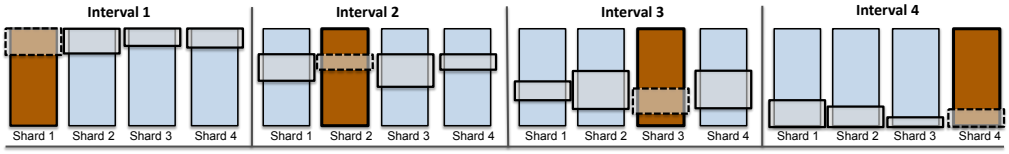


Figura 1.6. Exemplo de Estágios iteração da técnica *Parallel Sliding Window*. Adaptado de [Kyrola et al. 2012]

.

A técnica PSW possui algumas limitações que ocorrem quando o grafo não cabe inteiramente em memória RAM. Primeiramente, PSW não suporta ordenação dinâmica de forma eficiente. Também não é eficiente para travessia de grafo ou consultas de vértices, já que para obter a vizinhança de um único vértice é necessário pesquisar em todo um *shard* armazenado em memória. Desse modo, PSW funciona bem quando é necessário processar todos os vértices, mas o desempenho não é muito satisfatório quando apenas parte dos vértices precisam ser usados no processamento.

Message Passing Interface

Interface de Passagem de Mensagem (*Message Passing Interface* - MPI) é uma interface padrão para construção de uma ampla gama de programas que precisam realizar transferência de mensagens [Salihoglu e Widom 2013]. Há várias implementações *open-source* de MPI [Open MPI 2015, MPICH 2015]. Bibliotecas baseadas em MPI, como [Buluç e Gilbert 2011, Gregor e Lumsdaine 2005, Lugowski et al. 2012], também são usadas para implementar algoritmos para passagem de mensagens de forma paralela. Essas bibliotecas são eficientes, mas requerem manipulação em baixo nível de sincronização, agendamento de tarefas e primitivas de comunicação. Elas também não provêm mecanismos de tolerância a falhas.

1.4. Ferramentas e Aplicações

Esta seção apresenta algumas ferramentas e aplicações usadas atualmente para realizar processamento de grafos em *Big Data*.

1.4.1. Pregel

O framework Pregel [Malewicz et al. 2010], criado pelo Google, apresenta um modelo de computação síncrono, centrado em vértice e adequado para processar de forma eficiente grandes grafos, como o grafo que representa as páginas *Web* ou grafos de redes sociais. Nesse modelo, os programas são expressos como uma sequência de iterações em que um

vértice pode: (i) receber mensagens enviadas na iteração anterior; (ii) enviar mensagens para outros vértices; (iii) modificar seu próprio estado e o estado das arestas que saem dele; (iv) alterar a topologia do grafo. O modelo foi projetado para ser eficiente, escalável e tolerante a falhas, podendo funcionar em clusters com milhares de computadores. Os detalhes de distribuição ficam escondidos através do uso de uma API abstrata, expressiva e fácil de programar.

Apesar de ser proprietario, o Pregel motivou o desenvolvimento de outros frameworks bastante semelhantes que implementam sua API, ou mesmo frameworks bastante diferentes, mas que ainda assim, foram inspirados por ele. [Han et al. 2014] realizaram uma comparação entre alguns desses sistemas de processamento de grafos baseados no Pregel. Os sistemas comparados foram estes: Giraph [The Apache Software Foundation], GPS [Salihoglu e Widom 2013], Mizan [Khayyat et al. 2013] e GraphLab [Low et al. 2010, Low et al. 2012]. Além desses, [Sakr 2013] ainda cita mais dois *frameworks open-source* baseados no Pregel: o GoldenOrb ([GoldenOrb 2015]) e o Signal/Collect ([Stutz et al. 2010]). Esses sistemas serão detalhados a seguir.

1.4.2. Apache Giraph

Em 2012, o Apache Giraph [The Apache Software Foundation , Sakr 2013] foi disponibilizado como projeto *open-source* e implementa os conceitos do Pregel. Giraph usa a infraestrutura de um *cluster* Hadoop e herda características do Hadoop como: eficiência, escalabilidade, tolerância à falha e abstração dos detalhes de distribuição. Ele é executado como uma tarefa Hadoop sem a fase de redução (*reduce*), executando *workers* como mapeadores (*mappers*) especiais que se comunicam entre si para entregar mensagens entre os vértices e sincronizar entre *supersteps*. O sistema de arquivos distribuído HDFS é usado para entrada e saída de dados. Os vértices e as arestas são mantidos em memória e usam transferências em rede somente para mensagens. O modelo encapsula todo o mecanismo de execução, de modo que não é possível obter a ordem de execução dentro de um *superstep*. Toda a comunicação ocorre do *superstep S* para o *superstep* $S+1$. Durante a execução de uma tarefa, vértices são particionados e atribuídos a *workers*. O mecanismo de particionamento padrão é o particionamento *hash*, mas o particionamento personalizado também é suportado. O Giraph possui diferentes estruturas de dados para armazenamento de listas de adjacência de vértices. Por padrão, *array de bytes* é usado por ser eficiente em termos de espaço, além de rápido no carregamento de dados de entrada. No entanto, arestas armazenadas como *array de bytes* não são eficientes para alterações no grafo, como adição ou remoção de vértices ou arestas. Por outro lado, arestas representadas como *hashmap* são menos eficientes no uso da memória, mas bastante eficientes para alterações no grafo realizadas por algoritmos como o DMST (*Distributed Minimum Spanning Tree*).

1.4.3. Apache Hama

Apache Hama [The Apache Software Foundation 2015] é um *framework* disponibilizado oficialmente desde 2012, voltado para análise de *Big Data* através do uso do modelo de programação BSP. Assim como o Giraph, ele também é executado sobre a infraestrutura do Hadoop e possui uma API baseada no Pregel. No entanto, ele não se limita ao pro-

cessamento de grafos. Seu modelo BSP é genérico e pode ser usado de forma isolada (pura), para o processamento de grafos ou ainda para treino e processamento de redes neurais em larga escala. O módulo do Hama voltado para grandes redes neurais baseia-se no *framework* DistBelief [Dean et al. 2012] do Google.

1.4.4. GPS

GPS (*Graph Processing System*) [Salihoglu e Widom 2013] é uma solução *open-source* similar ao Pregel para processamento de grandes grafos. Entretanto, ele possui três recursos não disponíveis no Pregel: (i) uma API estendida para tornar a computação mais facilmente expressa e eficiente; (ii) um esquema de reparticionamento dinâmico, que atribui vértices a diferentes *workers* a partir do reconhecimento de padrões de mensagens; (iii) uma otimização que distribui listas de vértices com alto grau entre todos os nós disponíveis, com o objetivo de melhorar o desempenho. Além disso, o GPS ainda usa a linguagem específica de domínio Green-Marl [Hong et al. 2012] de alto nível projetada para simplificar o desenvolvimento de algoritmos complexos.

O particionamento realizado pelo GPS busca deixar juntos em um mesmo nó vértices que frequentemente enviam mensagens entre si. Uma abordagem usada para reduzir o custo de comunicação no GPS é particionar grandes listas de adjacências (*Large Adjacency List Partitioning* - LALP) entre vários *workers*. Essas listas possuem os vértices com alto grau, ou seja, com grande número de vizinhos. A abordagem LALP pode melhorar o desempenho de algoritmos que possuam essas duas propriedades: (i) vértices usam sua lista de adjacência, somente para envio de mensagem e não para computação; (ii) se um vértice enviar uma mensagem, ele a envia para todos os seus vizinhos. Isso é especialmente útil em algoritmos como *PageRank* e descoberta de Componentes Conexos, onde cada vértice envia a mesma mensagem para todos os seus vizinhos. Se, por exemplo, um vértice V localizado em um nó i tem 1000 vizinhos no nó j, então V envia a mesma mensagem 1000 vezes em cada *superstep* entre os nós i e j. No entanto, se o nó j tivesse a LALP, ele poderia receber uma única mensagem de i e, então, replicá-la aos nós adjacentes de V localizados em j.

[Han et al. 2014] apresentam alguns problemas que foram encontrados ao usar o GPS: a documentação de sua API é escassa se comparada com a documentação do Giraph (Seção 1.4.2) ou GraphLab (Seção 1.4.5); tipos personalizados devem ser codificados diretamente como uma classe de configuração de tarefas para cada algoritmo; *master* e *workers* demoram pelo menos um minuto para liberarem suas portas após uma execução; a definição dos números de portas é realizada apenas via arquivo de configuração; falhas e erros não são apresentados na interface *Web* do GPS ou no console; execuções com falhas não param após um tempo limite (*timeout*).

1.4.5. GraphLab

GraphLab [Low et al. 2010, Low et al. 2012, Gonzalez et al. 2012] difere dos demais frameworks apresentados até aqui por realizar processamento paralelo assíncrono. Ele foi desenvolvido em C++ com o intuito de resolver de forma eficiente problemas de aprendizagem de máquina e mineração de dados (*Machine Learning and Data Mining* - MLDM) sobre um grande volume de dados. Desse modo, o GraphLab fornece um

modelo de programação assíncrono, dinâmico e voltado para processamento paralelo de grafos, que abstrai as complexidades dos sistemas distribuídos. Seus autores afirmam [Low et al. 2012] que outras abstrações paralelas como MapReduce [Chu et al. 2007, Dean e Ghemawat 2008], Dryad [Isard et al. 2007] e Pregel [Malewicz et al. 2010] falharam em fornecer essas propriedades. Embora tenha sido afirmado que o modo de execução do GraphLab é assíncrono, ele também suporta execução síncrona [Gonzalez et al. 2012], que pode ser mais adequada em certos cenários.

Em 2013, o GraphLab passou a ser chamado GraphLab PowerGraph [Dato, Inc. 2015a]. A experiência adquirida com os projetos GraphLab PowerGraph e GraphChi [Kyrola et al. 2012] levaram à criação de um novo produto chamado GraphLab Create [Dato, Inc. 2015b], voltado para o desenvolvimento e publicação de serviços de aprendizagem de máquina para o mercado profissional de ciência de dados. GraphLab PowerGraph não é mais mantido pela equipe que o criou, mas pela comunidade interessada em sua evolução e continuidade.

GraphLab garante a consistência dos dados e alcança um alto desempenho em ambientes distribuídos ou de memória compartilhada. Ele usa versionamento de dados para reduzir o custo de comunicação através da rede e bloqueio distribuído em *pipeline* para reduzir os efeitos da latência da rede. Para lidar com os desafios relacionados à localidade dos dados, o GraphLab propõe a divisão do grafo em partes chamadas átomos para eficientemente disponibilizar o grafo em uma estrutura distribuída. A tolerância a falhas é baseada em uma adaptação do algoritmo de *snapshot* Chandy-Lamport [Chandy e Lamport 1985].

O GraphLab possui dois mecanismos possíveis de execução distribuída: o cromático (*chromatic engine*) e o de bloqueio (*locking engine*). O mecanismo cromático usa coloração de grafos para obter uma execução sequencialmente consistente para escalonadores estáticos. O mecanismo de bloqueio usa bloqueio distribuído em *pipeline* e técnicas para evitar que se perceba a latência (*latency hiding*) em execuções priorizadas dinamicamente.

GraphLab e Pregel usam o modelo GAS de forma bastante diferente [Gonzalez et al. 2012]. No Pregel, a fase *gather* é implementada através de combinadores de mensagens. As fases *apply* e *scatter* ocorrem no código do vértice. Por outro lado, o GraphLab expõe a vizinhança inteira ao programa do vértice e permite ao desenvolvedor definir as fases *gather* e *apply* dentro do programa. Além disso, mudanças feitas ao vértice ou aresta são automaticamente visíveis aos vértices adjacentes.

1.4.6. GraphX

GraphX [Xin et al. 2013, Gonzalez et al. 2014] é um *framework* para processamento de grafos construído sobre o sistema distribuído para processamento de dados Apache Spark [Zaharia et al. 2012]. Spark foi projetado para processamento de dados em larga escala, podendo inclusive ser usado para o processamento de grafos. Entretanto, expressar algoritmos de grafos nativamente no Spark é bastante trabalhoso, levando a junções complexas e excessiva movimentação de dados, por não explorar bem a estrutura do grafo. GraphX estende a abstração do Conjunto de Dados Distribuído Resiliente (*Resilient Distributed Dataset* - RDD) para um Grafo Distribuído Resiliente (*Resilient Distributed Graph*

- RDG), que associa registros com vértices e arestas de um grafo e provê uma coleção de primitivas de manipulação de grafos. O RDG provê um modelo para representação de grafos e explora a estrutura do grafo para minimizar a comunicação e a sobrecarga de armazenamento. Além disso, ele disponibiliza uma larga gama de operações sobre grafos em uma plataforma interativa e tolerante a falhas.

O GraphX possui implementados os *frameworks* Pregel e GraphLab PowerGraph a partir de suas primitivas e dos RDGs. As primitivas também fornecem operações para visualizar, filtrar e transformar grafos, que simplificam bastante o processo de ETL e de análise de grafos. Em relação ao particionamento dos dados distribuídos, ele usa o particionamento de corte em vértice (*vertex-cut*) em uma representação tabular. Esse tipo de particionamento reduz o custo de comunicação em grafos naturais como redes sociais e grafos que representam a estrutura da Web [Gonzalez et al. 2012] (ver Seção 1.3.1).

A representação interna de grafos no GraphX é feita por coleções de um par de vértices e aresta construídos sobre a abstração de RDD do Spark. Essas coleções possuem indexação e particionamento específico para grafo como uma camada sobre os RDDs. A coleção de vértices usa particionamento baseado no *hash* dos identificadores dos vértices. Para ter suporte a junções frequentes entre coleções de vértices, os vértices são armazenados em um índice de *hash* local em cada partição. Uma máscara de *bits* armazena a visibilidade de cada vértice, permitindo remoções virtuais para promover reuso do índice. A coleção de arestas é particionada horizontalmente por uma função de partição definida pelo usuário. Para possibilitar a busca eficiente de arestas pelos seus vértices fonte ou destino, as arestas de uma partição são clusterizadas pelo identificador do vértice fonte, usando uma representação de linha esparsa comprimida (*Compressed Sparse Row* - CSR [Saad 2003]) com indexação *hash* pelo identificador do vértice destino.

1.4.7. Outros sistemas para processamento de dados em Larga Escala

Esta seção apresenta alguns outros sistemas voltados para processamento de dados, mas que não são tão difundidos quanto os sistemas já tratados até aqui.

GraphChi
GraphChi [Kyrola et al. 2012] é um sistema derivado do GraphLab e baseado em disco ou SSD para processamento eficiente de grafos com bilhões de arestas em uma única máquina. Ele usa a técnica de janela deslizante paralela (*Parallel Sliding Window* - PSW - Seção 1.3.3) para segmentar e processar o grafo em pequenas partes.

FlashGraph
FlashGraph [Zheng et al. 2015] é um sistema capaz de processar grafos com bilhões de vértices e arestas em um único computador. Ele adota o modelo de programação centrado em vértice, assim como o Pregel e o GraphLab. Como diferencial, o FlashGraph busca sanar algumas das limitações de PSW através do acesso otimizado a SSDs com mínima perda de desempenho. Para isso, ele implementa um mecanismo de processamento de grafos no espaço de usuário do sistema de arquivos do SSD, especialmente projetado para possibilitar um alto grau de paralelismo e um alto número de operações de entrada/saída por segundo (*Input/Output Operations Per Second* - IOPS). O FlashGraph armazena os estados dos vértices em memória RAM e a lista de arestas em SSDs. A latência é aliviada pela sobreposição entre processamento e operações de entrada/saída (*Input/Output* - I/O).

Para reduzir o consumo de banda de I/O, o FlashGraph somente acessa listas de arestas requisitadas sob demanda pelas aplicações. Para aumentar a vazão e reduzir o uso da CPU para I/O, ele combina requisições de I/O. Segundo seus autores, essas características permitem que o FlashGraph alcance 80% do desempenho de sistemas exclusivamente em memória e que mesmo supere o desempenho do GraphLab e GraphChi na execução de vários algoritmos.

Trinity/Graph Engine
Trinity [Shao et al. 2013] iniciou como um projeto da *Microsoft Research*. Depois de um certo amadurecimento, seu nome mudou para Graph Engine [Microsoft Research]. Basicamente ele é um mecanismo distribuído para manipulação de grafos de propósito genérico. Ele otimiza o gerenciamento de memória e a comunicação em rede para dar suporte à manipulação do grafo e uma computação paralela eficiente. Os padrões de acesso de processamento *online* e *offline* aos grafos são usados para otimizar o desempenho do uso de memória e comunicação. O *Graph Engine* suporta o processamento de consulta *online* e a análise *offline* de grandes grafos. Além disso, ele fornece uma linguagem de especificação de alto nível chamada TSL (*Trinity Specification Language*) para declaração do esquema de dados e dos modelos de comunicação em rede. Graph Engine implementa um armazenamento em memória distribuído e endereçável globalmente.

GoldenOrb
O GoldenOrb [GoldenOrb 2015] implementa a API do Pregel e é executado sobre a infraestrutura do Hadoop. No entanto, o projeto ficou obsoleto, tendo sua última atualização ocorrida em agosto de 2011.

Mizan
Mizan [Khayyat et al. 2013] é uma implementação baseada no Pregel e implementada em C++, que suporta balanceamento de carga entre *supersteps*.A maioria dos sistemas realiza o particionamento do grafo em um passo de pré-processamento para balancear melhor o grafo entre os nós do *cluster*. No entanto, o Mizan usa uma abordagem adaptativa de particionamento, que é realizado de forma dinâmica a partir da detecção de alterações sobre propriedades do sistema. Com isso, ele consegue realizar um balanceamento de carga eficiente, melhor adaptado ao ambiente em que está sendo executado. Diferentemente das demais implementações do Pregel, o Mizan não assume um conhecimento prévio sobre a estrutura do grafo ou sobre o comportamento do algoritmo. Ele realiza uma migração de vértices para balancear de forma adequada computação e comunicação. Os autores do Mizan afirmam que ele provê 84% de melhoria em relação às técnicas que usam particionamento prévio e estático em cargas de trabalho bastante dinâmicas. [Han et al. 2014] relatam que o Mizan possui muitos *bugs* e que ainda não possui muitos recursos úteis. Como exemplo disso, ele assume que valores de vértices e pesos de arestas possuem o mesmo tipo, levando muitas vezes a um consumo elevado de memória. Ele também não suporta a operação *compute()* no nó *master*. Pela forma como foi projetado, as implementações de algoritmos no Mizam são mais suscetiveis a erros de memória que se fossem implementadas no Giraph, GPS ou GraphLab. A documentação de sua API também é escassa, sendo necessário entender o código interno do *framework*.

PEGASUS
PEGASUS [Kang et al. 2009] é uma biblioteca *open-source* para mineração de dados em

grandes grafos implementada sobre a infraestrutura provida pelo *framework* Hadoop. Ele possibilita a realização de tarefas de mineração como computar o diâmetro do grafo, obter o raio de cada vértice e encontrar os componentes conexos. Várias dessas operações de mineração são baseadas em repetidas multiplicações de matrizes. Para realizar essas operações o PEGASUS usa uma primitiva chamada multiplicação GIM-V (*Generalized Iterated Matrix-Vector*), que possui boa escalabilidade e tempo de execução linear de acordo com o número de arestas e cinco vezes mais rápida que a versão não otimizada do GIM-V.

1.5. Limitações e Desafios

Após a caracterização dos métodos e técnicas utilizados pelos sistemas para processamento de grafos, verifica-se que existem algumas questões que merecem mais investigação, sugerindo futuras direções de pesquisa. A primeira questão está relacionada com o fato dos sistemas de processamento de grafos adotarem um ou mais modelos de programação distribuída. De fato, não é fácil verificar o quanto esses modelos são efetivos para cada tipo de problema em grafos. Por exemplo, o modelo *MapReduce* é menos adequado para lidar com problemas em grafo do que o modelo centrado em vértices, como discutido em [Elser e Montresor 2013]. Por outro lado, [Salihoglu e Widom 2014] argumentam que uma implementação eficiente de algoritmos em grafos usando o *framework* Pregel, o qual adota modelo síncrono centrado em vértice usando Hadoop com fase de mapeamento mas sem fase de redução, é uma tarefa muito complexa. Além disso, [Nguyen et al. 2013] executaram experimentos que demonstram que modelos de programação possuem desempenho distinto para diferentes variações de algoritmos e tipos de grafos. Consequentemente, decidir qual modelo de programação é mais eficiente para cada problema em grafo é um problema em aberto.

Para a maior parte dos sistemas analisados, não é claro como cada tipo de algoritmo em grafos pode ser expresso eficientemente. Algoritmos do tipo PageRank mapeiam claramente em algoritmos que utilizam o modelo centrado em vértices. Claramente existe uma lacuna na descrição de quais algoritmos podem ser facilmente mapeados nos modelos de programação existentes. Diferente da questão de desempenho dos modelos frente aos problemas em grafos, o problema aqui tratado se refere à facilidade de representar um problema em grafo no modelo de programação utilizado. Este problema está relacionado com o poder de expressividade do modelo de programação.

A utilização desses sistemas é complexa no que tange às características relacionadas com a arquitetura mais adequada para processar um determinado problema, ou ainda, quantas máquinas são necessárias para processar um grande grafo. As respostas a essas questões poderiam beneficiar bastante os usuários que desejam processar grafos de forma eficiente, poupando tais usuários de conhecimentos técnicos mais aprofundados sobre os sistemas.

A comparação de desempenho dos sistemas propostos é um grande desafio. Como um usuário pode decidir se necessita de um sistema mais específico ou mais geral? Como um usuário pode saber a quantidade de máquinas necessárias para processamento? Neste sentido, *benchmarks* específicos precisam ser criados para ajudar na comparação de cada sistema, permitindo a sua melhor caracterização quanto ao seu

tipo de processamento. Várias tentativas para comparar *frameworks* foram realizadas [Elnozahy et al. 2002, Guo et al. 2014a, Han et al. 2014, Kajdanowicz et al. 2014], mas apenas sobre um pequeno subconjunto dos sistemas apresentados. Claramente a existência de um *benchmark* poderia acelerar o processo de escolha do sistema mais adequado para o problema em grafos a ser atacado.

À parte da avaliação dos sistemas de processamento de grandes grafos e a construção de novos sistemas impõem desafios. Por exemplo, como permitir o processamento de grandes grafos que sofrem constantes mudanças sobre a forma de *stream* de dados? Neste cenário é fundamental prover técnicas e métodos para atualizar incrementalmente os grafos, além de prover processamento incremental somente sobre as partes que foram alteradas do grafo.

Ainda no cenário de construção de novos sistemas, permitir que múltiplas aplicações trabalhem sobre um grafo compartilhado é fundamental para evitar frequentes cargas do mesmo grafo em memória. Além disso, mecanismos de particionamento de grafos eficientes que permitem acelerar o processamento de determinados algoritmos é igualmente fundamental. Claramente, um sistema de processamento de grafos deve prover transparência ao usuário quanto à forma de particionamento dos dados, ao mesmo tempo que deve prover diversas formas de particionamento simultâneas para atender as diferentes necessidades dos algoritmos.

1.6. Comparação entre os sistemas

A tabela 1.2 apresenta uma síntese dos sistemas de processamento de grafos abordados neste capítulo. Esses sistemas são classificados como sendo de Memória **C**ompartilhada, **D**istribuídos, Grafos Direcionados **A**cíclicos, Centrado em **V**értice, Centrado em **G**rafo ou do tipo Master-**W**orker. • representa o suporte explícito do sistema e ○ o suporte explícito, mas com escolha do usuário. Na última coluna, encontra-se a classificação sobre os *frameworks* de base para cada sistema: **H**adoop, **P**regel, **G**raphLab e **O**utros Sistemas.

1.7. Conclusão

Grafos são poderosas ferramentas para representar o relacionamento entre entidades. Devido a simplicidade de abstração de um grafo e os inúmeros algoritmos desenvolvidos sobre tal modelo, diversos ramos da ciência e da indústria têm utilizado grafos para realizar análises sobre fenômenos do mundo real. Nos últimos anos, sob efeito do fenômeno *Big Data*, os grafos continuaram crescendo em tamanho fomentando a necessidade de métodos e técnicas que permitam processá-los de forma eficiente.

Neste capítulo foram apresentados os principais métodos e técnicas adotados pelos sistemas de processamento de grandes grafos disponíveis na literatura. Ao longo deste trabalho, foi demonstrado que esses sistemas precisam lidar com grandes desafios herdados de aplicações de grafos, além dos desafios do cenário imposto pelo fenômeno *Big Data* e da computação paralela e distribuída de larga escala. Grafos possuem uma estrutura irregular e o seu tamanho pode exceder o tamanho da memória de uma única máquina, requerendo mecanismos específicos para lidar com eles e ao mesmo tempo manter um desempenho satisfatório. Além disso, o uso de grandes de grafos possui desafios parti-

	Plataforma		Modelo Computacional			Particionamento		
Sistema	Arquitetura	Mutabilidade do Grafo	Modelo de Programação	Síncrono	Assíncrono	Corte em Vértice	Corte em Aresta	Framework Base
Hama [The Apache Software Foundation 2015]	D		VMA	•				P
Spark [Zaharia et al. 2012]	D	•	A	•			•	O
PEGASUS [Kang et al. 2009]	D		M	•	◦		◦	H
Pregel [Malewicz et al. 2010]	D	•	V	•				P
GoldenOrb [GoldenOrb 2015]	D	•	V	•			•	O
Trinity / Graph Engine [Shao et al. 2013]	D		G	◦	◦		•	O
Mizan [Khayyat et al. 2013]	D	•	V	•			•	P
Giraph [The Apache Software Foundation]	D	•	V	•			•	P
Distributed GraphLab [Low et al. 2012]	D		V	◦	◦		•	G
PowerGraph [Gonzalez et al. 2012]	D		V	◦	◦		•	G
GPS [Salihoglu e Widom 2013]	D	•	V	•			◦	P
Titan-Hadoop [Aurelius 2015]	D	•	V	•			•	H
GraphX [Xin et al. 2013]	D		G	•		•		O
GraphLab [Dato, Inc. 2015a]	S		V	◦	◦			G
GraphChi [Kyrola et al. 2012]	S		V		•		•	G
FlashGraph [Zheng et al. 2015]	S		V		•		•	O

Tabela 1.2. Comparação entre os frameworks de processamento de grafos em larga escala. Adaptado de [Doekemeijer e Varbanescu 2014].

culares relacionados a paralelização do seu processamento, particionamento e localidade dos dados.

Este capítulo também descreveu os principais sistemas para processamento de grafos em larga escala e as principais técnicas e métodos usados por esses sistemas. Os sistemas foram comparados de acordo com a combinação de características que eles apresentam, tais como: Plataforma, Modelo Computacional, Particionamento e *Framework* Base. Conclui-se, portanto, que muitos sistemas possuem uma interface comum, mas apresentam implementações distintas. Além disso, escolher o melhor sistema para um determinado problema em grafos é muito complexo, visto que não é claro como cada tipo de algoritmo pode ser expresso eficientemente nos sistemas apresentados. O desempenho de cada *framework* difere de acordo com a arquitetura, modelo de programação, implementação do algoritmo e dados de entrada. Desta forma, a avaliação do desempenho desses sistemas é uma questão em aberto, o que torna complexa a seleção do melhor sistema para uma determinada aplicação.

Finalmente, verificou-se que existem muitos desafios relacionados com a criação de novos sistemas. A velocidade do *Big Data* é um dos desafios a serem atacados, dado que mecanismos eficientes de particionamento e compartilhamento de grafos são necessários para permitir o processamento de *stream* de dados na análise em tempo real de grandes grafos.

Referências

[Albert et al. 2000] Albert, R., Jeong, H., e Barabási, A.-L. (2000). Error and attack tolerance of complex networks. *nature*, 406(6794):378–382.

[Amaral et al. 2000] Amaral, L. A. N., Scala, A., Barthelemy, M., e Stanley, H. E. (2000). Classes of small-world networks. *Proceedings of the national academy of sciences*, 97(21):11149–11152.

[Angles 2012] Angles, R. (2012). A comparison of current graph database models. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pages 171–177. IEEE.

[Angles e Gutierrez 2008] Angles, R. e Gutierrez, C. (2008). Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1.

[Aurelius 2015] Aurelius (2015). Titan.

[Barabási e Albert 1999] Barabási, A.-L. e Albert, R. (1999). Emergence of scaling in random networks. *science*, 286(5439):509–512.

[Batarfi et al. 2015] Batarfi, O., El Shawi, R., Fayoumi, A. G., Nouri, R., Barnawi, A., Sakr, S., et al. (2015). Large scale graph processing systems: survey and an experimental evaluation. *Cluster Computing*, pages 1–25.

[Begoli e Horey 2012] Begoli, E. e Horey, J. (2012). Design principles for effective knowledge discovery from big data. In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*, pages 215–218. IEEE.

[Buluç et al. 2013] Buluç, A., Duriakova, E., Fox, A., Gilbert, J. R., Kamil, S., Lugowski, A., Oliker, L., e Williams, S. (2013). High-productivity and high-performance analysis of filtered semantic graphs. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 237–248. IEEE.

[Buluç e Gilbert 2011] Buluç, A. e Gilbert, J. R. (2011). The combinatorial blas: Design, implementation, and applications. *International Journal of High Performance Computing Applications*, page 1094342011403516.

[Cai e Poon 2010] Cai, J. e Poon, C. K. (2010). Path-hop: efficiently indexing large graphs for reachability queries. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 119–128. ACM.

[Chandy e Lamport 1985] Chandy, K. M. e Lamport, L. (1985). Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 3(1):63–75.

[Chen et al. 2012] Chen, R., Yang, M., Weng, X., Choi, B., He, B., e Li, X. (2012). Improving large graph processing on partitioned graphs in the cloud. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 3. ACM.

[Cheng et al. 2012] Cheng, R., Hong, J., Kyrola, A., Miao, Y., Weng, X., Wu, M., Yang, F., Zhou, L., Zhao, F., e Chen, E. (2012). Kineograph: taking the pulse of a fast-changing and connected world. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 85–98. ACM.

[Chu et al. 2007] Chu, C., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., e Olukotun, K. (2007). Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281.

[Ciglan et al. 2012] Ciglan, M., Averbuch, A., e Hluchy, L. (2012). Benchmarking traversal operations over graph databases. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pages 186–189. IEEE.

[Coelho da Silva et al. 2014] Coelho da Silva, T., Araujo, A. C., Magalhaes, R. P., Farias, V., de Macedo, J., e Machado, J. (2014). Efficient and distributed dbscan algorithm using mapreduce to detect density areas on traffic data. ICEIS.

[Cong e Makarychev 2012] Cong, G. e Makarychev, K. (2012). Optimizing large-scale graph analysis on multithreaded, multicore platforms. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 414–425. IEEE.

[Dato, Inc. 2015a] Dato, Inc. (2015a). `https://github.com/dato-code/PowerGraph`.

[Dato, Inc. 2015b] Dato, Inc. (2015b). `https://dato.com/products/create/`.

[Dean et al. 2012] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. (2012). Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231.

[Dean e Ghemawat 2008] Dean, J. e Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[Doekemeijer e Varbanescu 2014] Doekemeijer, N. e Varbanescu, A. L. (2014). A survey of parallel graph processing frameworks. Technical Report PDS-2014-003, Delft University of Technology.

[Elnozahy et al. 2002] Elnozahy, E. N., Alvisi, L., Wang, Y.-M., e Johnson, D. B. (2002). A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys (CSUR)*, 34(3):375–408.

[Elser e Montresor 2013] Elser, B. e Montresor, A. (2013). An evaluation study of big-data frameworks for graph processing. In *Big Data, 2013 IEEE International Conference on*, pages 60–67. IEEE.

[Even 2011] Even, S. (2011). *Graph algorithms*. Cambridge University Press.

[Gharaibeh et al. 2012] Gharaibeh, A., Beltrão Costa, L., Santos-Neto, E., e Ripeanu, M. (2012). A yoke of oxen and a thousand chickens for heavy lifting graph processing. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pages 345–354. ACM.

[GoldenOrb 2015] GoldenOrb (2015). `https://github.com/jzachr/goldenorb`.

[Gonzalez et al. 2012] Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., e Guestrin, C. (2012). Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, volume 12, page 2.

[Gonzalez et al. 2014] Gonzalez, J. E., Xin, R. S., Dave, A., Crankshaw, D., Franklin, M. J., e Stoica, I. (2014). Graphx: Graph processing in a distributed dataflow framework. In *Proceedings of OSDI*, pages 599–613.

[Gregor e Lumsdaine 2005] Gregor, D. e Lumsdaine, A. (2005). The parallel bgl: A generic library for distributed graph computations. *Parallel Object-Oriented Scientific Computing (POOSC)*, 2:1–18.

[Guo et al. 2014a] Guo, Y., Biczak, M., Varbanescu, A. L., Iosup, A., Martella, C., e Willke, T. L. (2014a). How well do graph-processing platforms perform? an empirical performance evaluation and analysis. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 395–404. IEEE.

[Guo et al. 2014b] Guo, Y., Varbanescu, A. L., Iosup, A., Martella, C., e Willke, T. L. (2014b). Benchmarking graph-processing platforms: a vision. In *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*, pages 289–292. ACM.

[Han et al. 2014] Han, M., Daudjee, K., Ammar, K., Özsu, M. T., Wang, X., e Jin, T. (2014). An experimental comparison of pregel-like graph processing systems. *Proceedings of the VLDB Endowment*, 7(12):1047–1058.

[Harish e Narayanan 2007] Harish, P. e Narayanan, P. (2007). Accelerating large graph algorithms on the gpu using cuda. In *High performance computing–HiPC 2007*, pages 197–208. Springer.

[Holzschuher e Peinl 2013] Holzschuher, F. e Peinl, R. (2013). Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 195–204. ACM.

[Hong et al. 2012] Hong, S., Chafi, H., Sedlar, E., e Olukotun, K. (2012). Green-marl: a dsl for easy and efficient graph analysis. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 349–362. ACM.

[Hong et al. 2011] Hong, S., Kim, S. K., Oguntebi, T., e Olukotun, K. (2011). Accelerating cuda graph algorithms at maximum warp. In *ACM SIGPLAN Notices*, volume 46, pages 267–276. ACM.

[Huberman 2003] Huberman, B. A. (2003). *The laws of the Web: Patterns in the ecology of information*. mit Press.

[Isard et al. 2007] Isard, M., Budiu, M., Yu, Y., Birrell, A., e Fetterly, D. (2007). Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72. ACM.

[Jeong et al. 2001] Jeong, H., Mason, S. P., Barabási, A.-L., e Oltvai, Z. N. (2001). Lethality and centrality in protein networks. *Nature*, 411(6833):41–42.

[Ji et al. 2012] Ji, C., Li, Y., Qiu, W., Jin, Y., Xu, Y., Awada, U., Li, K., e Qu, W. (2012). Big data processing: Big challenges and opportunities. *Journal of Interconnection Networks*, 13(03n04):1250009.

[Jiang e Agrawal 2011] Jiang, W. e Agrawal, G. (2011). Ex-mate: data intensive computing with large reduction objects and its application to graph mining. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 475–484. IEEE Computer Society.

[Kajdanowicz et al. 2014] Kajdanowicz, T., Kazienko, P., e Indyk, W. (2014). Parallel processing of large graphs. *Future Generation Computer Systems*, 32:324–337.

[Kang et al. 2009] Kang, U., Tsourakakis, C. E., e Faloutsos, C. (2009). Pegasus: A peta-scale graph mining system implementation and observations. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 229–238. IEEE.

[Khayyat et al. 2013] Khayyat, Z., Awara, K., Alonazi, A., Jamjoom, H., Williams, D., e Kalnis, P. (2013). Mizan: a system for dynamic load balancing in large-scale graph processing. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 169–182. ACM.

[Kyrola et al. 2012] Kyrola, A., Blelloch, G. E., e Guestrin, C. (2012). Graphchi: Large-scale graph computation on just a pc. In *OSDI*, volume 12, pages 31–46.

[Leskovec et al. 2005] Leskovec, J., Kleinberg, J., e Faloutsos, C. (2005). Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM.

[Lin e Dyer 2010] Lin, J. e Dyer, C. (2010). Data-intensive text processing with mapreduce. *Synthesis Lectures on Human Language Technologies*, 3(1):1–177.

[Low et al. 2012] Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., e Hellerstein, J. M. (2012). Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727.

[Low et al. 2010] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., e Hellerstein, J. M. (2010). Graphlab: A new parallel framework for machine learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 340–349.

[Lugowski et al. 2012] Lugowski, A., Alber, D. M., Buluç, A., Gilbert, J. R., Reinhardt, S. P., Teng, Y., e Waranis, A. (2012). A flexible open-source toolbox for scalable complex graph analysis. In *SDM*, volume 12, pages 930–941. SIAM.

[Malewicz et al. 2010] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., e Czajkowski, G. (2010). Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM.

[McColl et al. 2014] McColl, R. C., Ediger, D., Poovey, J., Campbell, D., e Bader, D. A. (2014). A performance evaluation of open source graph databases. In *Proceedings of the first workshop on Parallel programming for analytics applications*, pages 11–18. ACM.

[Microsoft Research] Microsoft Research. Graph Engine. `http://www.graphengine.io/`.

[MPICH 2015] MPICH (2015). `http://www.mpich.org/`.

[Neo Technology 2015] Neo Technology (2015). `http://neo4j.com/`.

[Newman 2003] Newman, M. E. (2003). The structure and function of complex networks. *SIAM review*, 45(2):167–256.

[Nguyen et al. 2013] Nguyen, D., Lenharth, A., e Pingali, K. (2013). A lightweight infrastructure for graph analytics. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 456–471. ACM.

[Open MPI 2015] Open MPI (2015). `http://www.open-mpi.org/`.

[Orient Technologies 2015] Orient Technologies (2015). Orientdb. `http://orientdb.com/orientdb/`.

[Page et al. 1999] Page, L., Brin, S., Motwani, R., e Winograd, T. (1999). The pagerank citation ranking: bringing order to the web.

[Pavlo et al. 2009] Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., e Stonebraker, M. (2009). A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 165–178. ACM.

[Pérez et al. 2006] Pérez, J., Arenas, M., e Gutierrez, C. (2006). Semantics and complexity of sparql. In *International semantic web conference*, volume 4273, pages 30–43. Springer.

[Prabhakaran et al. 2012] Prabhakaran, V., Wu, M., Weng, X., McSherry, F., Zhou, L., e Haradasan, M. (2012). Managing large graphs on multi-cores with graph awareness. In *USENIX Annual Technical Conference*, volume 7, page 2.

[Quinn e Deo 1984] Quinn, M. J. e Deo, N. (1984). Parallel graph algorithms. *ACM Computing Surveys (CSUR)*, 16(3):319–348.

[Rajaraman et al. 2012] Rajaraman, A., Ullman, J. D., Ullman, J. D., e Ullman, J. D. (2012). *Mining of massive datasets*, volume 77. Cambridge University Press Cambridge.

[Ramachandran 1993] Ramachandran, V. (1993). Efficient parallel graph algorithms. *Lectures on parallel computation*, (4):67.

[Saad 2003] Saad, Y. (2003). *Sparse Matrices*, chapter 3, pages 73–101. Siam.

[Sakr 2013] Sakr, S. (2013). Processing large-scale graph data: A guide to current technology. `http://www.ibm.com/developerworks/library/os-giraph/`.

[Salihoglu e Widom 2013] Salihoglu, S. e Widom, J. (2013). Gps: A graph processing system. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, page 22. ACM.

[Salihoglu e Widom 2014] Salihoglu, S. e Widom, J. (2014). Optimizing graph algorithms on pregel-like systems. *Proceedings of the VLDB Endowment*, 7(7):577–588.

[Shao et al. 2013] Shao, B., Wang, H., e Li, Y. (2013). Trinity: A distributed graph engine on a memory cloud. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 505–516. ACM.

[Shun e Blelloch 2013] Shun, J. e Blelloch, G. E. (2013). Ligra: a lightweight graph processing framework for shared memory. In *ACM SIGPLAN Notices*, volume 48, pages 135–146. ACM.

[Stutz et al. 2010] Stutz, P., Bernstein, A., e Cohen, W. (2010). Signal/collect: graph algorithms for the (semantic) web. In *The Semantic Web–ISWC 2010*, pages 764–780. Springer.

[The Apache Software Foundation] The Apache Software Foundation. Apache Giraph. `http://giraph.apache.org/`.

[The Apache Software Foundation 2015] The Apache Software Foundation (2015). Apache Hama. `http://hama.apache.org/`.

[Tian et al. 2013] Tian, Y., Balmin, A., Corsten, S. A., Tatikonda, S., e McPherson, J. (2013). From think like a vertex to think like a graph. *Proceedings of the VLDB Endowment*, 7(3):193–204.

[Vaquero et al. 2014] Vaquero, L. M., Cuadrado, F., e Ripeanu, M. (2014). Systems for near real-time analysis of large-scale dynamic graphs. *arXiv preprint arXiv:1410.1903*.

[Vicknair et al. 2010] Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., e Wilkins, D. (2010). A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th annual Southeast regional conference*, page 42. ACM.

[Wang et al. 2013] Wang, G., Xie, W., Demers, A. J., e Gehrke, J. (2013). Asynchronous large-scale graph processing made easy. In *CIDR*.

[Wang et al. 2010] Wang, N., Zhang, J., Tan, K.-L., e Tung, A. K. (2010). On triangulation-based dense neighborhood graph discovery. *Proceedings of the VLDB Endowment*, 4(2):58–68.

[Watts e Strogatz 1998] Watts, D. J. e Strogatz, S. H. (1998). Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442.

[White et al. 1986] White, J., Southgate, E., Thomson, J., e Brenner, S. (1986). The structure of the nervous system of the nematode caenorhabditis elegans: the mind of a worm. *Phil. Trans. R. Soc. Lond*, 314:1–340.

[Wu et al. 2014] Wu, X., Zhu, X., Wu, G.-Q., e Ding, W. (2014). Data mining with big data. *Knowledge and Data Engineering, IEEE Transactions on*, 26(1):97–107.

[Xie et al. 2015] Xie, C., Chen, R., Guan, H., Zang, B., e Chen, H. (2015). Sync or async: Time to fuse for distributed graph-parallel computation. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 194–204. ACM.

[Xie et al. 2013] Xie, W., Wang, G., Bindel, D., Demers, A., e Gehrke, J. (2013). Fast iterative graph computation with block updates. *Proceedings of the VLDB Endowment*, 6(14):2014–2025.

[Xin et al. 2013] Xin, R. S., Gonzalez, J. E., Franklin, M. J., e Stoica, I. (2013). Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, page 2. ACM.

[Yoneki e Roy 2013] Yoneki, E. e Roy, A. (2013). Scale-up graph processing: a storage-centric view. In *First International Workshop on Graph Data Management Experiences and Systems*, page 8. ACM.

[Zaharia et al. 2012] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., e Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association.

[Zheng et al. 2015] Zheng, D., Mhembere, D., Burns, R., Vogelstein, J., Priebe, C. E., e Szalay, A. S. (2015). Flashgraph: Processing billion-node graphs on an array of commodity ssds. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 45–58, Santa Clara, CA.

Sobre os Autores

Regis P. Magalhães:
Bacharel em Ciência da Computação pela Universidade Estadual do Ceará (1995), Especialista em Tecnologias para Web pela Universidade Federal do Piauí (2004), Mestre em Ciência da Computação pela Universidade Federal do Ceará (2012) e doutorando em Ciência da Computação pela Universidade Federal do Ceará (UFC). Atualmente é Professor Assistente da UFC no Campus de Quixadá, atuando principalmente nos seguintes temas: Bancos de Dados, Armazenamento e Processamento de Grafos em Larga Escala, Grafos Dependentes do Tempo, Integração de Dados, Web Semântica e Linked Data. Também tem trabalhado na construção de um framework para manipulação de redes dependentes do tempo.

Luís Gustavo C. do Rêgo:
Mestrando em Ciência da Computação pela Universidade Federal do Ceará. Bacharel em Computação pela UFC (2014). Atualmente trabalha na construção de um framework para processamento de redes dependentes do tempo.

José Antônio F. de Macêdo:
Professor Adjunto do Programa de Mestrado e Doutorado da Universidade Federal do Ceara. Pesquisador nível 2 do CNPQ. Doutor em Informática pela Pontifícia Universidade Católica do Rio de Janeiro (PUC-RIO). Pós-doutorado na École Polytechnique Fédérale de Lausanne (Suíça). Mestre pela PUC-RIO. A sua pesquisa é focada no processamento de dados em larga escala, grafos dependentes do tempo e manipulação de dados espaço-temporais.

Vânia M. P. Vidal:
Professora Associada da Universidade Federal do Ceará onde atua em cursos de graduação e pós-graduação. Possui graduação em Engenharia Civil pela Universidade Federal do Ceará (1979), mestrado em Informática pela Pontifícia Universidade Católica do Rio de Janeiro (1982) e doutorado em Engenharia de Sistemas e Computação pela Universidade Federal do Rio de Janeiro (1994). Realizou estágio pós-doutoral na École Polytechnique Fédérale de Lausanne, Suíça (2007/2008). Suas principais áreas de interesse incluem integração semântica, integração de dados na Web, desenvolvimento de aplicações Web e Servicos Web Semânticos.

cap:2

Capítulo 2

Publicação e Consumo de Dados na Web: Conceitos e Desafios

Bernadette Farias Lóscio, Marcelo Iury S. Oliveira, Ig Ibert Bittencourt

Abstract

The growing interest on publishing open data, the paradigm of WoT (Web of Things) and the OWP (Open Web Platform) initiative confirm the potential of the Web as a platform for sharing and exchanging data. However, several challenges related to publishing and using data on the Web still need to be addressed in order to ensure the consolidation of the Web as platform for data sharing. In this chapter, the main concepts related to data on the Web are discussed, including the Data on the Web Life Cycle, Open Data, Linked Data and a set of best practices proposed to improve the publication and the use of data in this new scenario. Finally, some of the main challenges faced when using the Web as a platform for data sharing are discussed and some directions for future work are presented.

Resumo

A crescente publicação de dados em formato aberto e o advento do paradigma de WoT (Web of Things) e do movimento de OWP (Open Web Platform) confirmam o potencial da Web como plataforma para compartilhamento e troca de dados. Porém, diversos desafios relacionados aos processos de publicação e consumo de dados ainda precisam ser enfrentados a fim de garantir a consolidação desta ideia. Neste capítulo, são discutidos os principais conceitos relacionados aos dados na Web, com destaque para o Ciclo de Vida dos Dados, Dados Abertos, Dados Conectados e um conjunto de Boas Práticas que visam facilitar e aperfeiçoar os processos de publicação e consumo de dados neste novo cenário. Além disso, serão apresentados os principais desafios enfrentados no uso da Web como plataforma para compartilhamento de dados, bem como serão apresentados direcionamentos para trabalhos futuros na área.

1. Introdução

Desde o seu surgimento, a Web tem se destacado como um importante meio para a troca e compartilhamento de informações. Mais recentemente, a crescente publicação de Dados Abertos, o grande volume de dados gerados pelas redes sociais, o paradigma de WoT (*Web of Things*) e o movimento de OWP (*Open Web Platform*) tem confirmado o potencial da

Web como plataforma para compartilhamento e troca de dados. Entretanto, devido, principalmente, à flexibilidade oferecida pela Web, novos desafios precisam ser enfrentados a fim de garantir o seu sucesso como plataforma de compartilhamento de dados. Esses desafios estão relacionados com diferentes subáreas da Ciência da Computação, tais como Banco de Dados, Engenharia de *Software*, Inteligência Artificial, Ontologias, bem como com outras Ciências.

Nesse cenário de grande quantidade de dados disponíveis na Web, dois papéis merecem destaque: os provedores e os consumidores de dados. Em termos gerais, os provedores de dados visam a publicação e o compartilhamento de dados, com acesso livre ou controlado, enquanto que os consumidores de dados (que também podem ser eles mesmos provedores) desejam fazer uso destes dados para a geração de informações úteis e relevantes, bem como para a geração de novos dados.

É importante ressaltar que o interesse na publicação de dados na Web não é algo novo [Berners-Lee *et al.* 1999, Abiteboul *et al.* 2000]. Porém, nos últimos anos, este interesse tem se caracterizado pela publicação de dados de maneira a promover o compartilhamento e a reutilização de dados. Dessa forma, apenas disponibilizar o acesso aos dados não é suficiente. De maneira geral, torna-se necessário publicar dados de forma que possam ser facilmente compreendidos e utilizados por consumidores, além da disponibilização dos dados em formatos que possam ser facilmente processados por aplicações. Porém, a heterogeneidade dos dados e a falta de padrões para descrição e acesso aos conjuntos de dados, tornam o processo de publicação, compartilhamento e consumo de dados uma tarefa complexa.

Neste contexto, este capítulo discute os principais conceitos relacionados à publicação e ao consumo de dados na Web, abordando aspectos relevantes, tanto do ponto de vista da pesquisa quanto da inovação e empreendedorismo. Por um lado, os desafios presentes na Web estão conduzindo pesquisadores da área de Computação a enfrentá-los com todo o *know how* da área. Por outro lado, o potencial da Web como plataforma aberta tem induzido diversas empresas e consultores a empreenderem com alto valor agregado, gerando emprego e renda.

Este documento está organizado como se segue. Na Seção 2, é discutida a relação entre publicação de dados na Web e os sistemas de bancos de dados convencionais. Na Seção 3, é apresentado o conceito de Dado Aberto, juntamente com a classificação em estrelas para os Dados Abertos. Na Seção 4 é apresentado o conceito de Dados Conectados[1]. Na Seção 5, são discutidas algumas técnicas para publicação de dados na Web e, na Seção 6, é apresentado o Ciclo de Vida dos Dados na Web. Na Seção 7, são apresentados alguns dos principais desafios para a publicação e o consumo de dados na Web, enquanto que na Seção 8 é apresentado o trabalho desenvolvido pelo W3C com o intuito de propor Boas Práticas para a publicação e o consumo de dados na Web. Finalmente, na Seção 9, são discutidas algumas questões de pesquisa e, na Seção 10, são apresentadas algumas conclusões.

2. Bancos de Dados Convencionais e Publicação de Dados na Web

Atualmente, destacam-se como plataformas para a publicação e consumo de dados: os Sistemas de Banco de Dados (SBD) e a Web. Os SBD oferecem diversas funcionalidades

[1] Tradução oficial reconhecida pelo W3C Brasil para o termo *Linked Open Data*.

que permitem o compartilhamento de dados, sejam elas, funções para atualização e recuperação de dados, bem como suporte a acesso concorrente e simultâneo [Elmasri e Navathe 2014]. Em especial, um dos motivos da ampla utilização dos SBDs é o provimento de linguagens declarativas que permitem a realização de consultas e manipulação de dados sem a necessidade de conhecimento da estrutura de armazenamento físico dos dados.

Por sua vez, desde o seu surgimento, a Web tem passado por diversas transformações, evoluindo para atender a novas demandas, seja para suportar novas formas de comércio e serviços quer seja para manter um acervo cada vez maior de dados. Entre as evoluções da Web, destaca-se o surgimento do conceito de Web 2.0 que permitiu a interação cada vez maior dos usuários, fazendo com que a quantidade de informação gerada e publicada aumentasse consideravelmente. Devido a sua flexibilidade, a Web possibilita a publicação e o consumo de dados de maneira bastante simples, sem a exigência de sistemas para controlar o acesso concorrente aos dados, por exemplo. Por outro lado, a facilidade de compartilhamento em grande escala requer soluções flexíveis que possibilitem o consumo de dados por grupos de usuários previamente desconhecidos e com diferentes requisitos [Lóscio *et al.* 2015].

Ambas as soluções apresentam funcionalidades similares no sentido de lidar com mecanismos de armazenamento e compartilhamento de dados. Entretanto, há diferenças significativas entre elas no que tange a facilidade de publicação e consumo de dados. No Quadro 2.1 são resumidas algumas das principais diferenças entre a publicação de dados na Web e a abordagem de bancos de dados convencionais.

Quadro 2.1. Comparação entre a publicação de dados na Web e a abordagem de bancos de dados convencionais.

Característica	Bancos de Dados Convencionais	Publicação de dados na Web
Organização dos Dados	Estruturado	Estruturado, semiestruturado ou não-estruturado
Esquema	Fixo	Flexível ou ausente
Mecanimos de Acesso	Interfaces Padronizadas, Drivers e Linguagens Declarativas	Requisições HTTP e Serviços em formato *Web Service* ou *Rest Service*
Controle de Acesso	Controlado	Sem controle de acesso para consumo, mas controlado para publicação
Tipos de Consulta	Multiatributos, faixa de valores, agregações e junções	Dependem da forma de publicação de dados
Controle de Concorrência	ACID	BASE ou inexistente
Escalabilidade	Implementação Complexa	Dependem do modelo de concorrência adotado

Uma diferença fundamental entre as duas abordagens é quanto a descrição e utilização de esquemas. Esquemas descrevem a estrutura de organização de dados de acordo com algum modelo, sejam eles abstratos ou físicos. Em geral, SBDs registram seus dados de forma estruturada, podendo suportar ou não a padronização de um único esquema para toda uma coleção de dados. Por sua vez, comumente, os dados publicados na Web têm como característica a ausência completa ou parcial de um esquema que defina rigorosamente a estrutura dos dados a serem armazenados. Essa flexibilidade facilita o processo de publicação de dados, mas, em contrapartida, torna mais complexo o processo de consumo.

Ainda em relação à definição de esquemas, as soluções também se diferenciam no que diz respeito à semântica dos dados. Considerando que um banco de dados é projetado para um grupo específico de usuários ou para aplicações já conhecidas, tem-se que existe um consenso tácito quanto aos conceitos usados na representação do esquema do banco de dados. Por outro lado, na publicação de dados na Web, como não é possível prever todos os grupos de usuários ou aplicações que irão fazer uso de um mesmo conjunto de dados, torna-se necessário deixar a semântica dos dados explícita. Além disso, a fim de possibilitar o cruzamento de dados armazenados em conjuntos de dados distintos, existe a necessidade da utilização de vocabulários comuns ou termos padrões para a representação dos dados e metadados.

No mais, um banco de dados convencional geralmente é projetado para atender aos requisitos de um grupo de usuários ou de aplicações específicas, sendo praticamente possível prever todas as regras de negócio que devem ser consideradas no banco de dados. No contexto da Web, os conjuntos de dados são publicados de maneira que diferentes grupos de usuários possam ser beneficiados.

Os SBDs e a Web também se diferenciam quanto aos mecanismos de acesso aos dados. É fundamental que os dados sejam facilmente publicados, descobertos e acessíveis. Assim, ambas as soluções devem disponibilizar mecanismos que permitam a programas e usuários externos incluir, atualizar ou consultar dados previamente armazenados. Os SBDs oferecem, usualmente, mecanismos de acesso por meio de APIs (*Application Programming Interface*) ou *drivers* padronizados (a exemplo de JDBC ou ODBC), que permitem a execução de comandos em linguagens declarativas como SQL (*Structured Query Language*) [Elmasri e Navathe 2014]. A partir destes mecanismos, o SBDs permtitem a realização de consultas mais complexas e ricas, como agregações e junções de dados. Na Web, a troca de dados se dá, usualmente, através do protocolo HTTP (*Hypertext Transfer Protocol*), fornecendo acesso no formato de um serviço por meio de padrões *Web Services* ou *Rest Services*. Nesse sentido, as operações podem ser requisitadas por meio de interfaces remotas convencionais ou de interfaces RESTful, que padronizam as operações de manipulação e consulta de dados.

Ainda em relação aos mecanismos de acesso, é possível diferenciar as soluções quanto aos tipos de consultas que podem ser realizados. Consultas podem variar de buscas simples, como *exact match*, a buscas mais complexas, como consultas SQL. Os SBDs convencionais fornecem suporte a diversos tipos de consultas. Essa flexibilidade permite a execução de consultas por faixa de valores, consulta por múltiplos atributos, consultas de junção, consultas de agregação e consultas por *ranking* (também conhecidas como consultas *top-k*). Em contraste, apesar de usar padrões amplamente reconhecidos por diversas plataformas de desenvolvimento, a Web permite usualmente a busca de dados por meio do uso de palavras-chaves que tendem a um resultado impreciso. Dependendo do modelo de dados utilizado, outras formas de consulta mais ricas podem ser realizadas. Por exemplo, bases de dados RDF (*Resource Description Framework*) permitem consultas com múltiplos atributos e junções por meio da linguagem SPARQL (*SPARQL Protocol and RDF Query Language*), bem como acesso através de *SPARQL Endpoints*.

No que tange ao controle de concorrência, as soluções de publicação e consumo de dados devem prover garantias de que o acesso simultâneo por parte de múltiplos usuários não comprometa a consistência dos dados. Os SBDs convencionais, em especial os bancos de dados relacionais, são baseados nos princípios ACID (Atomicidade, Consistência,

Isolamento e Durabilidade) que definem como as transações devem ser executadas, a fim de garantir a consistência e permitir o acesso controlado aos dados. Contudo, a necessidade de gerenciar grandes volumes de dados, estruturados e não estruturados, fez com que os princípios ACID tornassem difíceis de se alcançar [Pritchett 2008] no contexto da Web. Dessa forma, por ser um meio mais flexível, a Web permite o desenvolvimento de novas soluções para o gerenciamento de dados que consigam lidar com as novas exigências de disponibilidade e escalabilidade. Entre os modelos de controle de concorrência usados pela Web destaca-se o paradigma BASE (do inglês, ***B**asically **A**vailable, **S**oft State, e **E**ventual Consistency*), o qual caracteriza-se por prover um modelo de consistência mais fraco (*e.g.* consistência eventual), no qual o sistema permance basicamente disponível, mesmo que parcialmente, na ocorrência de falhas.

Com o cenário de crescimento da quantidade de dados, a escalabilidade e melhoria de desempenho dos sistemas se tornam mais relevantes. Escalabilidade diz respeito à capacidade de um sistema de aumentar a vazão de dados com a inserção de novos recursos para lidar com o aumento de carga. Diversas soluções de SBDs proveem suporte nativo à escalabilidade, por meio do uso de replicação e fragmentação dos dados. Contudo, devido às técnicas de controle de concorrência e ao uso de modelos de consistência mais rigorosos, a garantia de princípios de escalabilidade em SBDs torna-se uma tarefa não trivial. Em contraste, a ausência parcial ou completa de bloqueios no acesso aos dados na Web permite alcançar escalabilidade de forma mais fácil, tornando esta tecnologia adequada para manipular volumes de dados crescentes.

Além das diferenças já apresentadas, é importante observar outros aspectos relacionados à proveniência e à qualidade de dados. Em particular, no contexto da Web, os dados são publicados a partir de diferentes provedores e cobrem diferentes domínios. Além disso, devido à natureza autônoma dos provedores e consumidores de dados, um mesmo dado pode ser simplesmente replicado por um novo provedor ou pode ser transformado e/ou agregado com outros dados para enfim ser publicado novamente. Outra situação possível é o compartilhamento, voluntário ou não, de dados incorretos. Neste sentido, para a publicação de dados na Web, é preciso disponibilizar metadados de proveniência, que permitam descrever a história de produção e publicação dos dados. Além disso, informações de qualidade devem estar disponíveis, a fim de permitir a máquinas avaliarem automaticamente atributos de qualidade e a humanos consultarem informações sobre a qualidade dos dados.

É importante ressaltar que a Web, de forma geral, oferece funcionalidades eficientes e menos complexas para a publicação dos dados, como apresentado na Tabela 2.1. Entretanto, em alguns casos, pode ficar a cargo do desenvolvedor resolver problemas como controle de concorrência ou tolerância a falhas. Na medida em que quantidade de dados a ser mantida aumenta, a implementação dessas funcionalidades pode se tornar um grande desafio.

3. Dados Abertos

O crescente volume de dados publicados em formato aberto tem impulsionado fortemente o interesse no desenvolvimento de soluções para viabilizar a publicação e o consumo de dados na Web. No cenário de Dados Abertos, tanto a publicação quanto o consumo de dados são tarefas fundamentais. Nesse contexto, não basta apenas publicar os dados na Web, é importante que os dados sejam publicados de maneira que sejam úteis para seus potenciais consumidores.

Segundo a *Open Knowledge Foundation* [Open Knowledge Foundation 2012], Dado Aberto consiste em qualquer dado que pode ser livremente utilizado, reutilizado e redistribuído por qualquer um. Assim, dados abertos consistem na publicação e disseminação de informações na Internet, compartilhadas em formatos abertos, legíveis por máquinas, e que possam ser livremente reutilizadas de forma automatizada pela sociedade. Ou seja, a abertura de dados está interessada em evitar um mecanismo de controle e restrições sobre os dados que forem publicados, permitindo que tanto pessoas físicas quanto jurídicas possam explorar estes dados de forma livre [Isotani e Bittencourt 2015]. Um dado é considerado aberto quando apresenta as seguintes características [Open Knowledge Foundation 2012]:

i. *Disponibilidade e acesso*: o dado precisa estar disponível por inteiro. Deve estar num formato conveniente e modificável;

ii. *Reúso e redistribuição*: o dado precisa ser fornecido em condições de reúso e redistribuição podendo ser combinado com outros;

iii. *Participação universal*: todos podem usar, reusar e redistribuir o dado sem restrições de áreas, pessoas ou grupos.

Os dados abertos podem ser classificados de acordo com uma escala, baseada em estrelas, proposta por Tim Berners-Lee [Berners-Lee 2008]. Segundo essa classificação, apresentada na Figura 2.1, um dado publicado na Web em qualquer formato (imagem, tabela ou documento) e associado a uma licença que permita o seu uso e reuso sem restrições é avaliado como sendo 1 Estrela. Apesar de já ser um avanço, os dados com 1 Estrela precisam ser manipulados manualmente ou por meio de extratores construídos especificamente para o acesso aos dados.

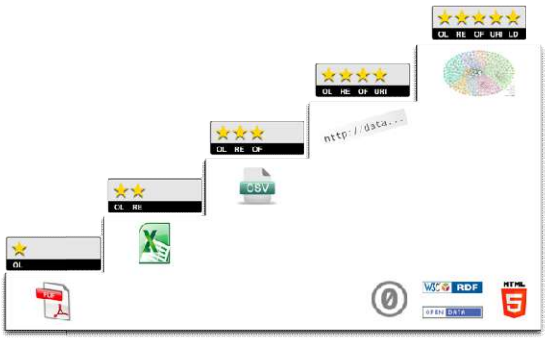


Figura 2.1 – Esquema de 5 estrelas para os Dados Abertos

(Fonte: [Berners-Lee 2008])

A partir do momento em que os dados são publicados em um formato que pode ser processado automaticamente por algum *software* (por exemplo, planilhas Excel ao invés de uma imagem), os dados passam a ser classificados como 2 Estrelas. Por um lado, isso pode facilitar o trabalho do consumidor de dados, porém, por outro lado, pode tornar a tarefa de publicação um pouco mais complexa.

Os dados recebem a classificação de 3 Estrelas quando são publicados em formatos não proprietários (por exemplo, CSV ao invés de Excel). Novamente, a publicação de dados em formatos abertos pode trazer custos adicionais para os provedores. Isso acontece quando o formato de origem é diferente do formato adotado para a publicação, e requer a conversão

dos dados, bem como a manutenção da consistência entre a fonte de dados original e os dados publicados em formato aberto.

A medida em que os dados recebem uma identificação única e são conectados com outros dados, eles podem ser classificados como 4 Estrelas. A criação de *links* entre os dados permite que eles façam parte de uma rede maior de dados abertos e conectados [Bizer *et al.* 2009]. Finalmente, o dado a recebe a classificação 5 Estrelas se estiver conectado com dados já disponíveis na Web. Nesse caso, é necessário identificar dados que representam o mesmo conceito do mundo real a fim de estabelecer os *links* entre os dados. Na próxima Seção, o conceito de Dados Abertos Conectados será discutido.

Seguindo o movimento dos dados abertos, governos de diversos países estão usando a Web como meio para publicação de dados e informações sobre suas administrações. Esses dados, denominados Dados Abertos Governamentais, podem ser facilmente encontrados nos chamados Portais de Dados Abertos, os quais oferecem uma interface mais amigável para catalogação e acesso aos dados. Como exemplos de portais de dados abertos já consolidados, destacam-se o portal dos EUA[2] e o portal do Reino Unido[3]. Diversos países na Europa, como França[4] e Holanda[5], bem como países na América Latina, como Chile[6] e Uruguai[7], também possuem portais de dados governamentais abertos. No caso do Brasil, o Portal de Dados Abertos[8] foi lançado no início de 2012, sendo uma iniciativa liderada pelo Ministério do Planejamento.

A iniciativa de abertura dos dados por parte dos Governos foi impulsionada pela procura de transparência, de colaboração e de participação da sociedade/comunidade [Goldstein 2013]. Com o intuito de chegar a um consenso dos requisitos necessários para se caracterizar uma base de dados abertos, o grupo de trabalho, *Open Government Working Group*, elaborou os oito princípios dos dados governamentais abertos [Tauberer *et al.* 2007]:

- *Completos*: todos os dados devem estar disponíveis e não limitados. Um dado público é o dado que não está sujeito a limitações válidas de privacidade, segurança ou privilégios de acesso.
- *Primários*: os dados devem estar em formato bruto, sem agregação ou modificação.
- *Atuais*: os dados devem ser publicados tão rapidamente quanto necessário para preservar o seu valor.
- *Acessíveis*: os dados devem ser acessíveis pelo maior número possível de usuários e para o maior número possível de finalidades.
- *Processáveis* por máquinas: os dados devem ser razoavelmente estruturados para permitir processamento automatizado.
- *Não-discriminatórios*: os dados devem ser disponíveis para todos, sem necessidade de cadastro.

[2] http://data.gov
[3] http://data.gov.uk
[4] http://data.gouv.fr
[5] http://data.overheid.nl
[6] http://datos.gob.cl
[7] http://datos.gub.uy
[8] http://dados.gov.br

- *Não-proprietários*: os dados devem ser publicados em formato aberto sobre o qual nenhuma entidade tem controle exclusivo.
- *Licenças livres*: os dados não devem estar sujeitos a nenhuma regulamentação de direitos autorais, patentes, propriedade intelectual ou segredo industrial. Restrições sensatas relacionadas à privacidade, segurança e privilégios de acesso podem ser permitidas.

Os dados governamentais abertos dizem respeito a assuntos diversos e podem envolver desde dados sobre despesas e receitas do governo até dados sobre censo escolar, pontos turísticos, reclamações de consumidores, demandas de serviços, entre outros. Em geral, os dados disponibilizados são provenientes de atividades rotineiras realizadas dentro dos órgãos governamentais, como ministérios e secretarias.

Uma vez que os dados governamentais sejam disponibilizados em formato aberto, espera-se que sejam usados no desenvolvimento de aplicativos que possam ser facilmente usados e acessados tanto por cidadãos comuns, bem como pelo próprio governo. Os aplicativos oferecem meios para análise dos dados, por meio de filtros, bem como permitem a visualização de dados de forma simples e criativa. Diversos aplicativos e visualizações já estão disponíveis na Web, os quais resultaram, principalmente, de concursos e *hackathons* promovidos para a divulgação e popularização dos portais de dados abertos. Como exemplo de concursos, destaca-se o "Cidadao Inteligente.Rec"[9], promovido pela Prefeitura de Recife como uma maneira de fomentar o uso dos dados abertos municipais.

4. Dados Conectados

O conceito de Dados Conectados 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" [Bizer *et al.* 2006]. A Web de Dados cria inúmeras oportunidades para a integração semântica dos próprios dados, motivando o desenvolvimento de novos tipos de aplicações e ferramentas, como navegadores e motores de busca [Isotani e Bittencourt 2015]. Ao se observar as camadas da Web Semântica, os Dados Conectados podem ser considerados como descrito na Figura 2.2.

Para um melhor entendimento sobre a Web de Dados, pode-se estabelecer um paralelo entre a Web de Documentos (*i.e.* a Web atual) e a Web de Dados. A primeira faz uso do padrão HTML para acessar dados, enquanto que na segunda os dados são acessados a partir do padrão RDF [Isotani e Bittencourt 2015]. A Web de Documentos é baseada em um conjunto de padrões, incluindo: um mecanismo de identificação global e único, os URIs (*Uniform Resource Identifier*); um mecanismo de acesso universal, o HTTP e um formato padrão para representação de conteúdo, o HTML. De modo semelhante, a Web de Dados tem por base alguns padrões, como: o mesmo mecanismo de identificação e acesso universal usado na Web de Documentos (URIs e HTTP, respectivamente); um modelo padrão para representação de dados, o RDF e uma linguagem de consulta para acesso aos dados, a linguagem SPARQL [Isotani e Bittencourt, 2015].

[9] http://www.cidadaointeligente.rec.br/

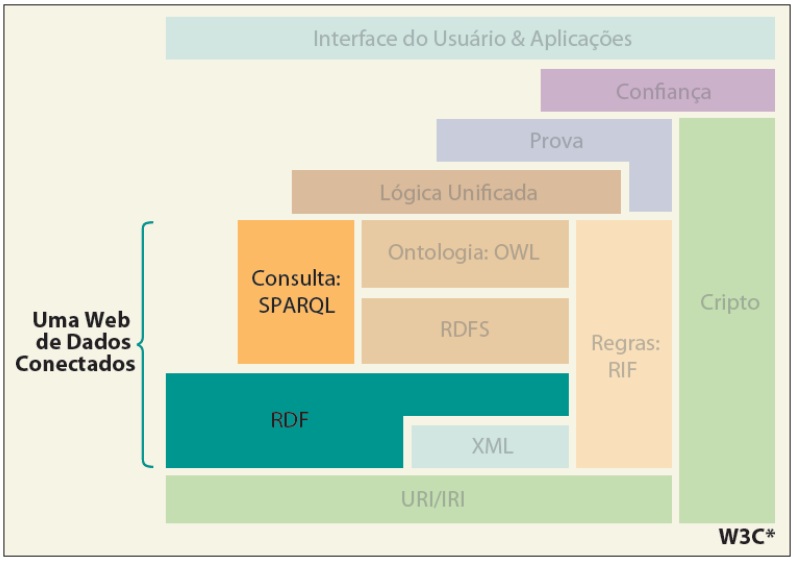


Figura 2.2 – A visão de Dados Conectados segundo as camadas da Web Semântica.

(Fonte: extraída de [Isotani e Bittencourt 2015])

Os Princípios de *Linked Data*, ou seja, o conjunto de Boas Práticas para a publicação de dados estruturados na Web, foram introduzidas por Tim Berners-Lee em [Berners-Lee 2006] e resumem-se em quatro princípios básicos:

1. Usar URIs como nome para recursos;

2. Usar URIs HTTP para que as pessoas possam encontrar esses nomes;

3. Quando uma URI for acessada, garantir que informações úteis possam ser obtidas por meio dessa URI, as quais devem estar representadas no formato RDF;

4. Incluir links para outras URIs de forma que outros recursos possam ser descobertos.

O primeiro princípio defende o uso de URI para identificar, não apenas documentos Web e conteúdos digitais, mas também objetos do mundo real e conceitos abstratos.

O segundo princípio defende o uso de URIs HTTP para identificar os objetos e os conceitos abstratos definidos pelo Princípio 1, possibilitando essas URIs serem dereferenciáveis sobre um protocolo HTTP. Neste contexto, dereferenciar é o processo de recuperar uma representação de um recurso identificado por uma URI, onde um recurso pode ter várias representações como documentos HTML, RDF, XML, entre outros.

A fim de permitir que uma ampla gama de aplicações diferentes possa processar dados disponíveis na Web, é importante que exista um acordo sobre um formato padrão para disponibilização dos dados. O terceiro princípio *Linked Data* defende o uso de RDF como modelo para a publicação de dados estruturados na Web [Klyne e Carol 2004]. Com o RDF, é possível descrever significado sobre recursos, habilitando agentes de *software* a explorar os dados de forma automática, muitas vezes, agregando, interpretando ou mesclando dados.

O quarto princípio diz respeito ao uso de *links* para conectar não apenas os documentos da Web, mas qualquer tipo de recurso. Por exemplo, um *link* pode ser criado

entre uma pessoa e um lugar, ou entre um local e uma empresa. Em contraste com a Web clássica onde os hiperlinks são em grande parte não tipados, hiperlinks que conectam os recursos em um contexto de *Linked Data* são capazes de descrever a relação entre eles. Hiperlinks no contexto de *Linked Data* são chamados de *links* RDF, a fim de distingui-los dos hiperlinks existentes na Web convencional [Heath e Bizer 2011].

É importante destacar que, atualmente, já existe um grande volume de dados abertos conectados disponível na Web. Como exemplo, destacam-se os conjuntos de dados abertos publicados pelo projeto LOD[10].

Como mencionado anteriormente, os dados conectados contribuem para a geração de uma Web de Dados, sendo, portanto, a opção mais almejada para a publicação de dados na Web. Recentemente, o *W3C Government Linked Data Working Group* propôs um conjunto de Boas Práticas para publicação de dados conectados a fim de prover diretrizes para auxiliar o acesso e o reúso de dados governamentais abertos[11].

5. Técnicas para Publicação de Dados na Web

A medida em que a Web se consolidou como plataforma para publicação e compartilhamento de documentos, organizações passaram a ter interesse no uso da Web como plataforma para publicação de dados. Durante os últimos anos, diversas técnicas emergiram para a publicação de dados na Web que vão desde o uso de formulários para a realização de consultas a um banco de dados até a publicação de Dados Conectados, *i.e.*, dados publicados de acordo com os princípios de *Linked Data* [Ceri *et al.* 2013, Ferrara *et al.* 2011]. Nesta seção, algumas dessas técnicas para a publicação de dados são apresentadas [Ceri *et al.* 2013, Ferrara *et al.* 2011], incluindo o acesso a partir de formulários Web, o uso de Web APIs, a inserção de dados diretamente nas páginas HTML e as ferramentas para criação de catálogos de dados.

5.1. Acesso a partir de formulários e Web APIs

Atualmente, a Web oferece um grande volume de informações disponível em páginas HTML. Porém, acredita-se que existe um volume muito maior de informações que estão ocultas, ou seja, que não podem ser facilmente encontradas e indexadas pelos mecanismos de busca. Dentre essas informações, destacam-se aquelas que só podem ser acessadas por meio de formulários de acesso e consulta à banco de dados. Como ilustrado na Figura 2.3, tais informações não podem ser acessadas de forma estática por meio de URLs, sendo necessário o uso de páginas Web dinâmicas, criadas a fim de apresentar respostas para consultas submetidas por meio de interfaces de consulta que oferecem acesso a um banco de dados [He e*t al.* 2007]. Na Web, como os mecanismos de busca existentes não podem efetivamente fazer *crawling* em bancos de dados, então os dados são considerados invisíveis e, portanto, estão ocultos para os usuários, fazendo parte da *Deep Web*.

Como mencionado anteriormente, uma das grandes limitações da *Deep Web* é a dificuldade na indexação dos dados por parte dos mecanismos de busca. Por exemplo, um *Web Crawler* pode acessar o formulário de acesso aos dados, porém irá se deparar com os seguintes problemas: *i)* como distinguir os formulários que dão acesso ao banco de dados de outros formulários com propósitos distintos? *ii)* como incluir informações no formulário

[10] http://lod-cloud.net/
[11] http://www.w3.org/TR/ld-bp/

a fim de ter acesso ao banco de dados? *iii)* como interpretar os resultados? e *iv)* como obter *links* permanentes para os itens acessados por meio do formulário? [Ceri *et al.* 2013]

Apesar de ser amplamente difundida e ter um baixo custo, o acesso aos bancos de dados a partir de formulários apresenta muitas limitações. Uma outra forma de publicação de dados na Web consiste em utilizar Web *Application Program Interfaces* (Web APIs). Uma das primeiras propostas para padronização de APIs para a Web foram os Web Services [Alonso *et al.* 2004], inspirados no paradigma de RPC (*Remote Procedure Call*) [Nelson 1981] e no uso de XML (*eXtensible Markup Language*) para a troca de dados. Posteriormente, surgiu o paradigma REST (*REpresentational State Transfer*) e o formato JSON (*JavaScript Object Notation*) [Mandel 2008] passou a ser amplamente adotado. Este novo tipo de API é conhecido como RESTful service.

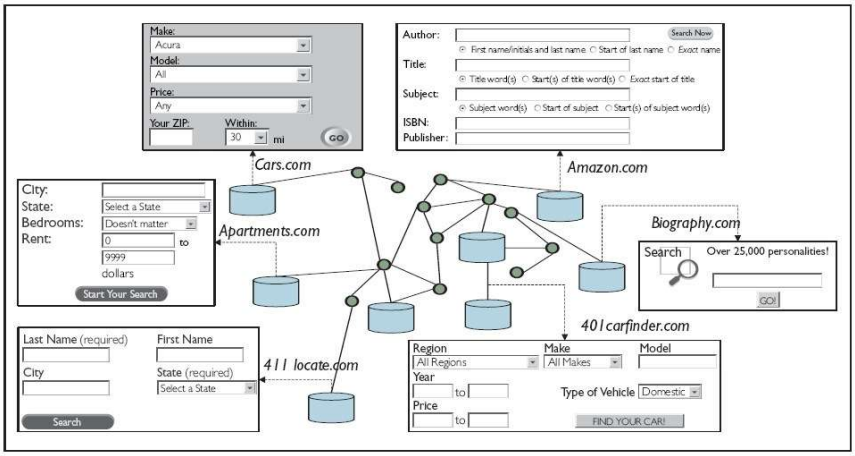


Figura 2.3. Visão Conceitual da *Deep Web*

(FONTE: [He *at al.* 2007])

Em geral, dados expostos por meio de APIs não podem ser encontrados pelos mecanismos de busca. Uma das razões para isso é que em muitos casos é necessário realizar uma autenticação antes de ser possível acessar a API. Além disso, existem restrições quanto ao uso da API a fim de evitar acessos exaustivos aos dados. Sendo assim, é possível dizer que os dados disponíveis por meio de APIs são semelhantes aos dados disponíveis na *Deep Web*, ou seja, não podem ser facilmente encontrados e indexados. Porém, a razão para isso acontecer é bem diferente e consiste na necessidade dos provedores em controlar o acesso aos dados por aplicações externas.

5.2. Enriquecimento de páginas HTML

Uma outra forma de publicar de dados na Web consiste em fazer a inclusão dos dados nas páginas HTML. Isso é possível com o uso de microformatos, ou seja, marcadores (*tags*) específicos que tornam explícita a semântica dos dados. O uso de microformatos permite aos mecanismos de busca identificar os dados disponíveis nas páginas HTML e, assim, apresentar melhores resultados aos usuários. Além disso, os provedores de dados podem alcançar maior visibilidade. Diversos microformatos foram desenvolvidos pela comunidade para a publicação de dados de diferentes domínios, incluindo: *hCalendar* para eventos,

hReview para revisões e *ratings*, *hRecipe* para receitas culinárias e *hCard* para dados pessoais[12].

O uso de Microformatos é uma solução simples para a publicação de dados na Web, porém também apresenta algumas limitações: *i)* o uso de diferentes microformatos em uma mesma página pode levar a conflitos de nomes (por exemplo, a class url de CSS e o termo url do microformato hCalendar), *ii)* não permite a criação de especializações e generalizações e *iii)* cada microformato requer um *parser* específico.

Esses problemas podem ser solucionados com o uso de RDFa[13], uma solução que permite a especificação de atributos para descrição de dados estruturados em qualquer linguagem de marcação, em particular XHTML[14] e HTML. Enquanto os microformatos combinam a sintaxe para incluir os dados estruturados nas páginas HTML com a própria semântica dos dados, RDFa preocupa-se apenas com a sintaxe para inclusão dos dados estruturados. Para a semântica dos dados, RDFa permite o uso de vocabulários específicos, como o schema.org[15]. RDFa permite que múltiplos vocabulários sejam utilizados em conjunto sem a necessidade de *parsers* específicos para cada um deles. Na Figura 2.4, é possível visualizar um diagrama apresentando a relação que existe entre Web Semântica, Dados Conectados, RDF e os diversos formatos de dados estruturados.

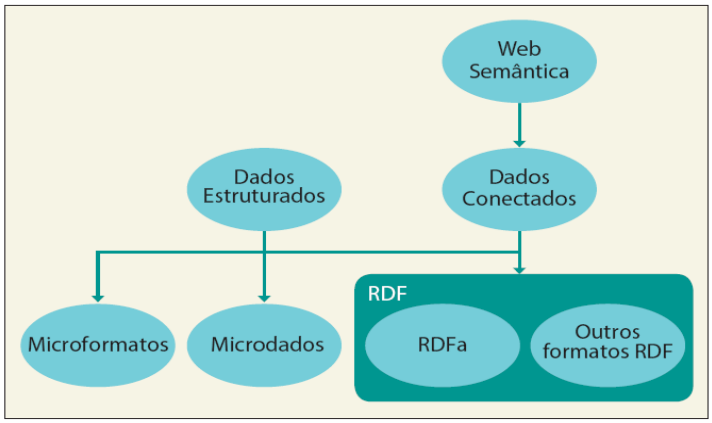


Figura 2.4 – Ecossistema de Dados Estruturados.

(Fonte: Extraído de [Isotani e Bittencourt, 2015])

5.3. Ferramentas para Catalogação de Dados

Com o crescente interesse na publicação de dados abertos, em especial os dados abertos governamentais, uma nova forma de publicação de dados na Web ganhou destaque: as ferramentas para catálogos de dados, como CKAN[16] e Socrata[17]. A partir dessas plataformas, são criados os portais de dados abertos, os quais oferecem acesso a conjuntos de dados previamente catalogados. Os conjuntos de dados são organizados como uma série

[12] http://microformats.org/
[13] http://www.w3.org/TR/rdfa-primer/
[14] http://www.w3.org/TR/xhtml1/
[15] http://schema.org/
[16] http://ckan.org/
[17] http://www.socrata.com/

de recursos e podem ser classificados de acordo com *tags* que explicitam o domínio dos dados.

Os portais de dados são uma ótima ferramenta para indexação de conjuntos de dados, mas deixam a desejar quanto à busca de dados, uma vez que não permitem fazer buscas nos conjuntos de dados propriamente ditos. Em alguns casos, as ferramentas de catalogação oferecem APIs de acesso aos dados, mas isso é feito de forma bastante simplificada. Os conjuntos de dados disponíveis nos catálogos podem ser encontrados pelas ferramentas de busca, porém ainda não é possível encontrar itens de dados específicos armazenados em um conjunto de dados.

Apesar da grande disseminação dos portais de dados abertos, estas soluções apresentam diversas limitações, dentre elas destacam-se: a dificuldade em manter os dados atualizados, a falta de padrões de metadados para descrição dos conjuntos de dados e a impossibilidade de realização de consultas sobre os dados. Além disso, como os conjuntos de dados publicados nos portais geralmente encontram-se disponíveis em diversos formatos, ou seja, múltiplos arquivos para um mesmo conjunto de dados, também pode haver redundância de dados.

6. Ciclo de Vida dos Dados na Web

O processo de publicação e consumo de dados na Web envolve várias fases que vão desde a seleção e publicação dos dados até o uso dos dados e *feedback* sobre os dados utilizados. Esse conjunto de fases que compõem o processo de publicação e consumo dos dados é chamado de Ciclo de Vida dos Dados na Web.

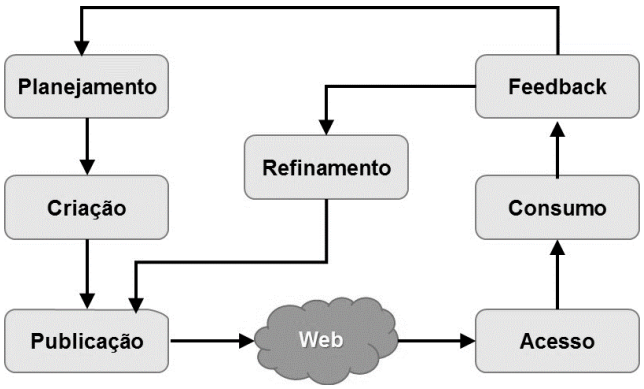


Figura 2.5 – Ciclo de Vida dos Dados na Web

Em geral, tanto os responsáveis pela publicação quanto os consumidores de dados ou não têm conhecimento sobre o ciclo de vida dos dados ou não existe um consenso entre eles sobre tais fases e seu significado. Esta seção apresenta uma proposta de ciclo de vida para os Dados na Web, a qual, além de descrever as principais fases envolvidas no processo de publicação e consumo de dados, também oferece uma terminologia comum que auxilia a comunicação entre provedores e consumidores de dados.

O ciclo de vida proposto é uma instância do *Abstract Data Lifecycle Model* (ADLM) proposto em [Möller 2010]. ADLM é um modelo genérico para representação de ciclo de vida para dados e metadados, o qual foi derivado a partir de uma coleção de modelos de ciclo de vida para domínios centrados em dados também apresentados em [Möller 2010]. O ADLM compreende um conjunto de fases, características e papéis que

podem ser usados para classificar e comparar modelos de ciclo vida centrados em dados, bem como para a construção de novos modelos de ciclo de vida. De acordo com esse modelo abstrato, um ciclo de vida para domínios centrados em dados deve ser composto pelas seguintes fases: Desenvolvimento de Ontologia, Planejamento, Criação, Arquivamento, Refinamento, Publicação, Acesso, Uso Externo, *Feedback* e Término.

É importante notar que o ciclo de vida para os Dados na Web não contempla todas as fases definididas no modelo ADLM. Especificamente, Desenvolvimento de Ontologia, Arquivamento e Término não fazem parte do ciclo de vida proposto. A criação de ontologias é considerada uma atividade independente e por isso não foi incluída. Da mesma forma, Arquivamento e Término não foram consideradas porque acredita-se que, uma vez que o dado foi publicado na Web, ele sempre estará disponível.

A Figura 2.5 apresenta as fases do ciclo de vida dos Dados na Web, as quais são brevemente descritas a seguir.

- **Planejamento:** Esta fase se estende desde o momento em que surge a intenção de publicar os dados até a seleção dos dados que serão publicados. Vale lembrar que não existem regras que determinem a prioridade dos dados a serem publicados, porém é sempre importante levar em consideração a relevância dos dados, ou seja, dados que possuem um grande potencial de utilização deveriam ter prioridade no momento da escolha. Dessa forma, sempre que possível, é importante fazer uma consulta prévia juntos aos potenciais consumidores de dados para identificar a relevânica dos dados.
- **Criação:** Diz respeito ao momento em que os dados são criados, ou seja, compreende a etapa de extração dos dados de fontes de dados já existentes até a sua transformação para o formato adequado para publicação na Web. Durante a etapa de criação, além dos dados propriamente ditos, também devem ser criados os metadados que irão descrever os dados. Na etapa de criação, também será feita a escolha dos formatos de dados a serem usados para a publicação de dados e metadados. Além disso, é sempre bom considerar a publicação de dados em diferentes formatos, minimizando a necessidade de transformação dos dados por parte dos consumidores.
- **Publicação:** Compreende o momento em que os dados serão disponibilizados de forma pública na Web. Para isso, podem ser usadas ferramentas de catalogação de dados, como CKAN e Socrata, e APIs. Em ambos os casos, o provedor de dados deverá oferecer toda a informação necessária para que o consumidor tenha fácil acesso aos dados. Além disso, é importante garantir que os dados serão atualizados de acordo com uma frequência pré-determinada, a qual deverá ser disponibilizada juntamente com os dados.
- **Acesso:** Denota o momento do ciclo de vida em que os usuários ganham acesso aos dados. Uma vez que os dados foram publicados, será necessário informar que os dados estão disponíveis e divulgar que o acesso aos dados está liberado.
- **Consumo:** Implica o momento em que os dados são usados para a criação de visualizações, como gráficos e mapas de calor, bem como para aplicações que permitem o cruzamento e a realização de análises sobre os dados. Esta etapa do ciclo de vida está diretamente relacionada ao consumidor de dados, que pode ser desde uma grande empresa interessada em usar os dados disponíveis na Web para a melhoria de seus produtos e serviços, até um único desenvolvedor interessado em usar os dados para criar uma aplicação que irá melhorar a qualidade de vida na sua cidade.

- ***Feedback:*** Esta fase compreende o momento em que os consumidores devem prover comentários sobre os dados e metadados previamente utilizados. Esta fase é de fundamental importância, pois a partir do *feedback* dos consumidores será possível identificar melhorias e realizar correções nos dados previamente publicados. Além disso, esse canal de comunicação entre consumidores e provedores de dados também facilita a identificação de novos dados relevantes que devem ter prioridade no momento da escolha de novos dados a serem publicados.
- **Refinamento:** Esta fase compreende todas as atividades relacionadas a adições ou atualizações nos dados que já foram publicados. É muito importante garantir a manutenção dos dados previamente publicados, a fim de oferecer maior segurança para aqueles que irão consumir os dados. A manutenção pode ser feita de acordo com o *feedback* dos consumidores ou novas versões podem ser geradas a fim de garantir que os dados não fiquem obsoletos. Para isso, é importante fazer o correto gerenciamento das diferentes versões dos dados e garantir que os consumidores tenham acesso à versão correta dos dados.

Com relação aos atores que participam do ciclo de vida dos dados na Web, estes podem desempenhar dois papéis principais: os provedores de dados e os consumidores de dados. O papel de provedor de dados pode ser desempenhado por vários atores, os quais são responsáveis por realizar atividades como criação de metadados, criação e publicação de dados. Os consumidores de dados são atores que recebem e consomem os dados. É importante notar que os consumidores de dados também podem ser provedores de dados, uma vez que os consumidores podem realizar melhorias e refinamentos nos dados a fim de oferecê-los novamente para a comunidade. É importante notar que o ciclo de vida proposto não requer que todas as etapas sejam seguidas até que uma nova iteração seja iniciada.

7. Desafios para a Publicação e o Consumo de Dados na Web

Considerando as especificidades da publicação e do consumo de dados na Web, esta seção apresenta alguns dos principais desafios enfrentados neste novo cenário. Estes desafios dizem respeito a assuntos que possuem questões em aberto e foram identificados a partir de uma série de *use cases* que descrevem situações reais de publicação e consumo de dados na Web, apresentados no documento *Data on the Web: Best Practices, Use Cases & Requirements* [Lee *et al.* 2015]. De forma geral, os desafios envolvem a resolução tanto de questões técnicas, como seleção de formato de dados usado e provimento de interfaces de acesso, quanto não-técnicas, como licenciamento, criação de um processo organizacional para produção de dados e engajamento dos usuários.

7.1. Metadados

Os metadados podem ser definidos como "dados que descrevem os dados" e são usados para auxiliar a descoberta e a reutilização de dados e/ou conjunto de dados. Metadados podem ser atribuídos considerando diferentes níveis de granularidade, ou seja, podem ser aplicados ao conjunto de dados como um todo, mas também podem estar relacionados a algum item específico do conjunto, bem como a todos os conjuntos de dados de uma determinada organização.

Os metadados podem ser de diferentes tipos e podem ser classificados de acordo com diferentes taxonomias com diferentes critérios de agrupamento. Por exemplo, uma taxonomia específica poderia definir três tipos de metadados: descritivos, estruturais e

administrativos. Metadados descritivos são úteis para identificar um conjunto de dados, metadados estruturais auxiliam na comprensão da estrutura dos dados e metadados administrativos fornecem informações sobre a versão e a proveniência de um conjunto de dados, por exemplo. Uma outra taxonomia poderia definir tipos de metadados considerando as tarefas para as quais metadados são utilizados, tais como descoberta e reutilização.

Metadados devem ser fornecidos de maneira a possibilitar tanto a sua compreensão por usuários quanto o seu processamento automático por aplicações. Metadados podem ter uma estrutura estabelecida e padronizada por alguma especificação ou consórcio, tais como DCAT[18] e MPEG-7[19], ou podem não ter estrutura. Além disso, o uso de metadados padronizados contribui para a diminuição dos custos envolvidos no desenvolvimento de aplicações na medida em que reduzem o retrabalho. Contudo, devido à natureza heterogênea dos dados publicados na Web e de seus respectivos provedores, é importante que os metadados sejam flexíveis de tal forma a permitir a descrição de dados distintos. Atualmente, não existe um consenso sobre quais metadados devem ser considerados na publicação de conjuntos de dados na Web nem qual a melhor forma de disponibilizar metadados que possam ser facilmente compreendidos por humanos e processáveis por máquinas.

7.2. Licença de Dados

Uma licença é uma especificação ou contrato que determina as condições de utilização de algo. No caso de dados, as licenças podem ser usadas para explicitar as condições e as possíveis formas de utilização de um determinado conjunto de dados. Como definido pela *Dublin Core Metadata Initiative* [DCMI 2013], uma licença de dados é um documento legal que oferece uma permissão oficial para a utilização do dado ao qual está associada. Assim, o provedor de dados pode liberar ou restringir o uso e compartilhamento dos dados por meio de licenças.

Mesmo para dados abertos, que não devem estar sujeitos a "*regulações de direitos autorais, marcas, patentes ou segredo industrial*", é fundamental que sejam especificadas as licenças para garantir que "*qualquer indivíduo tenha a liberdade de usá-lo, reutilizá-lo e redistribui-lo*" [Open Knowledge Foundation 2012]. Isso é necessário para que seja legalmente admitido realizar, de forma livre, agregações, estatísticas, análises e cruzamentos de dados de diferentes fontes. Dessa forma, o provedor de dados pode desenvolver sua própria licença ou pode fazer uso de licenças padronizadas para especificar as condições de utilização de dados. Diversos padrões e famílias de licenças foram criadas para determinar as condições de uso e compartilhamento de *software*, conteúdo e dados, entre elas destacam-se a General Public License (GPL)[20], a família Creative Commons[21] e a Open Data Commons[22].

Porém, para as licenças de dados, é muito importante considerar a necessidade de compatibilidade da licença escolhida e as condições desejadas para a utilização e compartilhamento dos dados. Por exemplo, algumas licenças da família Creative Commons violam os preceitos de dados abertos, sendo consideradas abertas apenas as variações CC-Zero, na qual o autor renuncia a todos os direitos, CC-BY, que determina a citação de

[18] http://www.w3.org/TR/vocab-dcat/
[19] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=34228
[20] http://www.gnu.org/licenses/gpl.html
[21] http://creativecommons.org/licenses/
[22] http://opendatacommons.org/

autoria ao compartilhar, e CC-BY-SA, que obriga o uso da mesma licença ao redistribuir ou modificar [Open Definition 2015]. A legislação nacional de cada país também pode impor restrições de compartilhamento de dados. No contexto brasileiro, tanto a Constituição quanto a Lei de Acesso à Informação determinam o direito à privacidade do cidadão, fazendo com que alguns dados, como dados fiscais, bancários e telefônicos sejam privados, não podendo assim ser publicados na Web.

No contexto de dados na Web, a licença de um conjunto de dados pode ser especificada como parte do próprio conjunto de dados ou em um documento separado. No entanto, da mesma forma que os dados, as licenças também devem ser facilmente acessíveis e localizadas. Além disso, a licença deveria permitir o processamento automático por máquina com o intuito de verificar se os dados estão realmente atendendo às condições especificadas e se as licenças atendem às normas maiores, tais como princípios de dados abertos, padrões ISO ou legislação nacional. Contudo, o problema com esta abordagem é que ainda não há uma forma amplamente aceita e genérica o suficiente para todos os casos, fazendo com que o processamento e checagem automática de licenças seja uma tarefa complexa a ser desenvolvida.

7.3. Proveniência de Dados

Proveniência de dados consiste em um conjunto de informações que oferece detalhes sobre a história dos dados aos seus usuários, possibilitando o rastreamento da origem dos dados. Informações de proveniência são particularmente importantes quando dados são compartilhados entre usuários que não são previamente conhecidos, ou seja, em situações onde o provedor e o consumidor de dados não possuem um relacionamento. Dessa forma, torna-se fundamental que os provedores apresentem detalhes sobre o processo de criação e origem dos dados. As informações de proveniência também contribuem para aumentar a credibilidade dos dados.

Informações de proveniência podem ser capturadas em maior ou menor nível de granularidade, registrando todo o *workflow* de atividades, computacionais ou não, de geração e/ou derivação dos dados, bem como os agentes envolvidos nessas atividades (aplicações, ambientes e pessoas). É importante ressaltar que o processo de captura das informações de proveniência pode ser tão ou mais complexo que o próprio *workflow* de geração de dados. Neste sentido, existem trabalhos que propõem soluções que buscam facilitar a captura, o armazenamento e a análise integrada de informações de proveniência em cenários de ambientes heterogêneos e distribuídos, tais como [Marinho *et al.* 2012] e [Imran e Hlavacs 2012]. Apesar de úteis, esses trabalhos possuem escopo limitado a um domínio de dado específico (*e.g.* dados médicos) e várias destas soluções tratam-se de protótipos. Assim como os metadados, as informações de proveniência devem ser armazenadas de forma estruturada e devem ser descritas por termos padronizados com o intuito de facilitar a realização de atividades de processamento e verificação de dados.

A fim de prover uma representação padrão para as informações de proveniência, o W3C propôs uma família de documentos[23] que definem vários aspectos necessários para promover a troca de informações de proveniência em ambientes heterogêneos, como a Web. Dentre estes documentos, destaca-se o modelo que apresenta os principais conceitos utilizados para descrever as informações de proveniência[24] e a ontologia[25], denominada

[23] http://www.w3.org/TR/2013/NOTE-prov-primer-20130430/
[24] http://www.w3.org/TR/2013/REC-prov-dm-20130430/

PROV-O, que especifica este modelo. Entretanto, apesar do PROV-O descrever como os dados de proveniência devem ser estruturados e organizados, ainda existe um desafio em aberto que é a falta de concordância e o nível de detalhe do que deve ser capturado no que diz respeito ao tipo de informação de proveniência.

7.4.Versionamento

Os conjuntos de dados publicados na Web podem sofrer atualizações ao longo do tempo, de tal forma que alguns conjuntos de dados possuem uma frequência de atualização fixa, como os dados do censo ou dados do orçamento público, e outros conjuntos de dados ou são estáticos ou são atualizados com uma baixa frequência (por exemplo, para a correção de erros). Tendo em vista que os dados publicados na Web são compartilhados em grande escala, torna-se fundamental oferecer informações sobre o versionamento dos conjuntos de dados, bem como garantir que os conjuntos de dados estão sendo atualizados de acordo com a frequência de atualização previamente estabelecida. Além dos dados, as APIs de acesso também podem sofrer atualizações ao longo do tempo, sendo necessário o versionamento das mesmas.

É importante garantir a consistência das informações de versionamento a fim de manter um bom nível de confiabilidade por parte dos consumidores dos dados. Semelhante aos dados de proveniência e de qualidade, também é importante que as informações sobre versionamento sejam fornecidas em um formato que possa ser facilmente processado pelas aplicações. Dessa forma, será possível fazer checagens automáticas sobre o uso adequado das diferentes versões de um mesmo conjunto de dados. Apesar da relevância das informações de versionamento, atualmente, não existe um padrão para descrição de versões que seja amplamente aceito e que possa ser facilmente processado por máquinas. Além disso, no contexto de Dados na Web, também não existe um consenso sobre o conceito de versão de conjunto de dados.

7.5. Qualidade de Dados

Qualidade de dados é comumente definido como "adequação ao uso" para uma determinada aplicação ou caso de uso. Considerando que a qualidade dos dados pode influenciar consideravelmente a qualidade das aplicações que fazem uso dos dados, é muito importante ter informações de qualidade associadas aos conjuntos de dados publicados na Web. Em cenários onde os dados estão disponíveis para compartilhamento em grande escala, a qualidade dos dados é um aspecto fundamental. Considerando que os dados serão reutilizados por diferentes grupos de consumidores com propósitos diversos, a publicação de dados de baixa qualidade pode levar a sérios problemas. A propagação de dados de baixa qualidade ou a tomada de decisão com base nesses dados devem ser evitados.

Geralmente, a avaliação da qualidade envolve diferentes dimensões de qualidade [Wang e Strong 1996], de maneira que cada dimensão representa características que são relevantes para os provedores ou consumidores de dados. Cada uma das dimensões ou critérios de qualidade deve estar associada a uma métrica que será usada para avaliar o critério ou dimensão em questão. As métricas podem ser objetivas ou subjetivas. Métricas objetivas definem um método para avaliação do critério e métricas subjetivas necessitam da intervenção humana para mensurar o critério.

[25] http://www.w3.org/TR/2013/REC-prov-o-20130430/

Diversos trabalhos têm sido propostos na literatura com o intuito de prover classificações para as dimensões de qualidade, bem como métricas específicas para a avaliação da qualidade. Dentre esses trabalhos, destaca-se o trabalho [Wang e Strong 1996], no qual é proposto um *framework* para descrição de 15 critérios de Qualidade da Informação agrupados em quatro categorias. Também merece destaque o trabalho de Zaveri e colegas [Zaveri *et al.* 2013], no qual é apresentada uma proposta para avaliação de conjuntos de dados conectados a partir de 26 critérios de qualidade (dois deles exclusivos para *linked data – interlinking* e performance) divididos em 6 categorias. Esse trabalho também reuniu um conjunto de métricas (objetivas e subjetivas) como forma de mensurar cada critério de qualidade.

Entretanto, para a avaliação dos dados publicados na Web, é preciso levar em consideração a diversidade dos modelos e formatos de dados usados para a representação dos dados, bem como a criação de novos critérios ou dimensões de qualidade. Além disso, as informações de qualidade também precisam ser armazenadas de forma estruturada para permitir o monitoramento e verificação automática por máquinas. Apesar das diversas propostas para descrição de dimensões e critérios de qualidade, não há solução amplamente aceita pela comunidade e, principalmente, não há consenso sobre quais dimensões e métricas são fundamentais para avaliação de dados e conjuntos de dados publicados na Web. Nesse sentido, vale a pena ressaltar que, atualmente, o W3C está trabalhando em um vocabulário para descrição da qualidade de conjuntos de dados publicados na Web[26].

7.6. Identificação

A Web provê um sistema de identificação único baseado no conceito de URI. Uma URI é um identificador que pode ser usado para qualquer recurso, incluindo aqueles que não estão disponíveis na Web, tais como pessoas ou imóveis. Existem diversos esquemas de URI, porém nem todos podem levar a identificação de recursos que podem ser localizados na Internet. Por exemplo, *doi:10.1103/PhysRevD.89.032002* é um exemplo de URI, mas o recurso associado a essa URI não poderá ser encontrado na Internet a partir desse identificador. Para dados na Web, apenas HTTP(s) URIs são relevantes.

A descoberta e o uso de dados na Web dependem fundamentalmente do uso de URIs HTTP. É importante notar que as URIs são strings que não carregam semântica e tem como função apenas a identificação única dos recursos. Entretanto, ao publicar dados na Web é preciso fazer um esforço para criar boas URIs para identificação dos recursos. Para isso, devem ser feitas algumas considerações, incluindo [Sauermann 2011]: *i)* Usar URIs HTTP para tudo, tornando-as passíveis de serem dereferenciadas; *ii)* Evitar URIs com detalhes de implementação ou do ambiente em que estão publicadas. Por exemplo, a URI *http://www.bad.url.org:8080/~fulano/index.php* é um exemplo do que deve ser evitado, pois possui detalhes sobre o autor, a porta 8080 usada em seu ambiente de publicação e o *script* implementado em PHP necessário para sua execução e *iii)* Manter as URIs estáveis e persistentes, pois a alteração das URIs pode levar à quebra de *links* já estabelecidos, criando um problema para a localização de recursos.

Qualquer URI HTTP deve ser capaz de ser dereferenciada, o que significa que clientes HTTP podem procurar por uma URI usando um protocolo HTTP e recuperar uma descrição do recurso que é identificado pela URI. A recuperação da representação mais adequada para o usuário é feita por meio da negociação de conteúdo. Assim, clientes HTTP

[26] http://www.w3.org/TR/vocab-dqv/

enviam, por meio dos cabeçalhos HTTP (Get, Head, Post, Put, Delete, Trace, Options, Connection), o pedido para indicar qual tipo de conteúdo deseja obter. Dessa forma, a URI http://example.com/dataset deverá oferecer o mesmo conjunto de dados em diferentes formatos, por exemplo CSV, JSON ou XML. O servidor retornará o formato adequado de acordo com a negociação de conteúdo.

Uma das principais preocupações com a identificação de recursos na Web diz respeito ao uso de URIs persistentes, ou seja, como garantir que os recursos sempre poderão ser utilizados, indepentemente do seu status, disponibilidade e formato, por exemplo. Questões relativas à identificação de conjuntos de dados e versões de conjuntos de dados também estão em aberto, uma vez que não existe um consenso de como essa identificação deve ser realizada.

7.7. Mecanismos de Acesso

A insfraestrutura da Web oferece diferentes métodos de acesso aos dados por meio de protocolos de comunicação como HTTP. A abordagem mais simples é a organização dos dados em arquivos e publicação dos mesmos em páginas HTML. O acesso, neste caso, pode ser realizado através do *download* individual de cada arquivo. Contudo, é muito comum que grandes volumes de dados estejam distribuídos em múltiplos arquivos ou estejam disponíveis em fluxo contínuo (*streams*). Um consumidor de dados deve ser capaz de requisitar a recuperação de arquivos específicos ou conjuntos de dados inteiros a partir de um determinado domínio de informação, sendo, assim, necessário o suporte de outras abordagens de acesso aos dados, como *download* simultâneo de múltiplos arquivos (também conhecido como *bulk download*) e as Web APIs.

No sistema de *bulk download*, os usuários podem selecionar vários conjuntos de dados e efetuar a recuperação em um formato específico, podendo ser determinado ou não pelo usuário. Uma vez selecionados os arquivos, o sistema inicia o processo de escalonamento das operações de *download*, que pode aplicar algum critério de priorização de dados. Todavia, essa abordagem pode levar a problemas de escalabilidade na medida em que aumenta o consumo de banda passante dos servidores Web utilizados para hospedar os dados. Assim, é recomendada a utilização de políticas de controle que busquem evitar a escassez de recursos de rede e o colapso da plataforma utilizada. Por sua vez, as Web APIs permitem que desenvolvedores façam uso de dados da Web sem a necessidade de efetuar a extração (frequentemente manual) de dados a partir de páginas HTML. Para isso, é essencial que as APIs sejam fornecidas de forma gratuita e aberta, sem exigência de autenticação. Em contrapartida, as APIs trazem consigo cuidados adicionais com segurança e escalabilidade.

Apesar de existirem diversas soluções que permitem o acesso aos dados publicados na Web, ainda não existe um consenso sobre qual forma de publicação é mais adequada. Ademais, ainda existem questões em aberto sobre como disponibilizar acesso a dados em tempo real e como garantir acesso aos dados atualizados.

7.8. Formatos de Dados

Os formatos nos quais os dados estão disponíveis para os consumidores de dados são de grande importância para garantir um bom nível de utilidade, ou seja, para garantir que de fato os dados são úteis para os consumidores. Atualmente, existe uma grande variedade de formatos dados disponível para publicação e troca de dados. Contudo, nem todos os

formatos proveem estrutura adequada que facilite o uso e reúso. Dentre os formatos estruturados, destacam-se CSV, JSON e XML.

Embora, a maioria dos conjuntos de dados sejam publicados em formatos (semi) estruturados, ainda há questões inerentes que precisam ser resolvidas pelos consumidores para que possam efetuar processamento automático por máquinas, como a ausência de metadados estruturais e a presença de erros estruturais. Nem todos os formatos de dados apresentam metadados que descrevem a organização dos dados e tipo de informação que armazenam. Por exemplo, o formato CSV permite a descrição opcional do nome dos campos de dados em um cabeçalho e utilizam caracteres delimitadores para separar campos e novos registros de dados. Dessa forma, a descrição sobre o significado e os tipos de dados usados em cada campo devem ser disponibilizados em um documento separado dos dados. Como essa documentação é opcional, tornam-se mais complexas as tarefas de compreensão dos dados e checagem de consistência da estrutura utilizada.

No mais, os conjuntos de dados devem estar publicados em formatos que não introduzam quaisquer obstáculos tecnológicos para uso ou processamento. Assim, devem preferencialmente ser publicados em formatos não-proprietários. Além disso, considerando que os dados devem estar disponíveis para grupos de usuários previamente desconhecidos, é importante que um mesmo conjunto de dados esteja disponível em diferentes formatos. Isso permite aumentar a interoperabilidade e reduzir os custos envolvidos no processamento dos dados. Contudo, é importante ressaltar que a publicação de dados em múltiplos formatos pode gerar redundância. Apesar dos benefícios inerentes de aumento da disponibilidade e tolerância a falhas, criar tais cópias, contudo, aumenta o consumo de capacidade de armazenamento e pode levar a inconsistências.

7.9.Vocabulários

Em qualquer cenário no qual os dados são oriundos de fontes de dados diversas, a interoperabilidade semântica é um desafio que deverá ser enfrentado. A representação adequada dos conceitos que descrevem o domínio dos dados sendo publicados é de fundamental importância para garantir que diferentes provedores e consumidores de dados compartilhem a mesma visão da realidade.

Nesse contexto, o uso de vocabulários é muito importante para o processo de publicação e consumo de dados na Web. Vocabulários são usados para especificar os termos comuns de domínios específicos, bem como os relacionamentos entre esses termos e possíveis restrições que podem ser aplicadas aos termos. A partir dessa especificação, é possível estabelecer um canal de comunicação entre provedores e consumidores de dados, no qual ambos compartilham a mesma visão da realidade. A ausência desse canal aumenta a possibilidade de inconsistências na compreensão dos conceitos do mundo real. Apesar de requerer um esforço adicional por parte dos provedores de dados, o uso de vocabulários comuns traz grandes benefícios ao longo prazo. A facilidade de compreensão e manipulação dos dados incentivará o uso dos dados, aumentando assim a sua utilidade e seu valor.

Os vocabulários são importantes tanto para a especificação de metadados descritivos, de informações de proveniência e de qualidade, quanto para a descrição dos esquemas dos dados propriamente ditos (metadados estruturais). Em todos os casos, é importante que sejam utilizados vocabulários amplamente conhecidos ou padrões já pré-estabelecidos, a fim de facilitar sua manipulação e compreensão.

Vocabulários podem ser especificados com diferentes níveis de formalismo e complexidade, originando diversas categorias de vocabulários, como vocabulários controlados, taxonomias, *thesaurus* e ontologias [Garshol 2004]. Vocabulários controlados podem ser definidos simplesmente como uma lista de termos, de tal maneira que cada termo é um nome particular para um conceito específico. É importante ressaltar que os termos de um vocabulário controlado devem fazer referência a um único conceito, a fim de evitar ambiguidades. De maneira geral, o objetivo de um vocabulário controlado é evitar o uso ou definição de termos poucos significativos (muito específicos ou muito gerais), bem como minimizar inconsistências na grafia de um mesmo termo.

As taxonomias são classificações que permitem organizar os termos de um vocabulário controlado em uma hierarquia. As taxonomias possibilitam o agrupamento e categorização dos termos, facilitando a identificação dos termos mais adequados para a descrição de um dado objeto. Assim como o termo taxonomia, o termo *thesaurus* tem sido utilizado para fazer referência a diversos tipos de estrutura e classificações. De acordo com [Garshol 2004], *thesaurus* estendem as taxonomias a fim de torná-las mais expressivas para a descrição dos conceitos do mundo real, permitindo a definição de outros relacionamentos entre os conceitos, além daqueles especificados na hierarquia. Um *thesaurus* permite, por exemplo, especificar que um termo é preferido com relação a outro termo ou que um termo possui um significado mais amplo do que outro termo. Finalmente, as ontologias podem ser consideradas como a forma mais expressiva para a definição de vocabulários. Ao contrario das taxonomias e dos *thesaurus*, nos quais existe um conjunto fixo de possíveis relacionamentos entre os conceitos, as ontologias permitem a especificação de qualquer relacionamento entre os conceitos.

As ontologias têm sido indicadas como uma solução para o problema de interoperabilidade semântica, uma vez que permitem expressar o significado de conceitos de um domínio, bem como de seus relacionamentos, de maneira clara e explícita [Gruber 1993; Bittencourt *et al.* 2009]. Entretanto, o desenvolvimento de ontologias é uma tarefa complexa e que demanda a presença de especialistas do domínio. Além disso, dependendo do domínio a ser modelado, chegar um consenso não é trivial. Diversas metodologias para engenharia de ontologias já foram propostas na literatura, as quais combinam etapas como modelagem conceitual, projeto e implementação [Casella 2011]. No contexto de publicação de dados na Web, o uso de ontologias se faz necessário para facilitar a compreensão e o cruzamento de dados de diferentes fontes, bem como para auxiliar a comunicação entre provedores e consumidores de dados.

Apesar de já existirem diversos vocabulários disponíveis[27] na Web, a escolha de vocabulários ainda é uma tarefa difícil. Além disso, quando os vocabulários existentes não são suficientes para a descrição do domínio dos dados a serem publicados, então é preciso enfrentar um outro desafio: a criação de um novo vocabulário. Para isso, é importante levar em consideração, sempre que possível, a criação de relacionamentos ou extensões de vocabulários existentes. No mais, semelhante aos dados publicados na Web, os vocabulários devem ser devidamente documentados e publicados em formato aberto.

7.10. *Feedback*

Publicar dados na Web possibilita o compartilhamento de dados em grande escala com um amplo público, o qual pode possuir diferentes níveis de *expertise*. Nesse contexto, os

[27] http://lov.okfn.org/dataset/lov/

provedores de dados desejam ter a garantia de que os dados publicados estão atendendo aos requisitos dos consumidores de dados e, para alcançar esse objetivo, a aquisição de *feedback* dos usuários torna-se fundamental. O *feedback* de consumidores auxilia os provedores de dados a melhorar a qualidade e integridade dos dados publicados. Além disso, também pode contribuir para a publicação de novos dados ao descreverem suas experiências com o uso dos dados, além de suas preferências e requisitos.

Sempre que possível, o *feedback* deve estar disponível de forma pública, a fim de que outros consumidores possam analisá-lo. Ambientes colaborativos, no qual diferentes consumidores podem compartilhar opiniões e esclarecer questões sobre os dados, também podem ser criados. Por fim, é importante que o *feedback* seja organizado de forma estruturada para, em linhas gerais, permitir o armazenamento e, consequentemente, viabilizar o processamento automático do mesmo.

8. Boas Práticas para Dados na Web

O crescente interesse na publicação de Dados Abertos tem motivado discussões quanto à abordagem mais adequada para a publicação de dados na Web. Entretanto, ainda não existe um consenso quanto a isso. Tendo em vista a relevância do assunto, o W3C criou um grupo de trabalho, denominado *Data on the Web Best Practices*, com o objetivo de propor uma recomendação que possa ser utilizada como guia para a publicação e o consumo de dados na Web.

A ideia principal desse guia consiste em propor um conjunto de Boas Práticas a serem implementadas pelos provedores de dados com o intuito de produzir conjuntos de dados que sejam úteis e atendam às expectativas dos consumidores de dados. As Boas Práticas não consideram abordagens ou tecnologias específicas para publicação de dados e propõem diretrizes que devem ser seguidas a fim de garantir uma melhor comunicação entre provedores e consumidores de dados.

Em geral, as Boas Práticas para Dados na Web [Lóscio *et al.* 2015] aplicam-se a conjuntos de dados e distribuições. De acordo com a especificação do vocabulário DCAT [Maali *et al.* 2014], um conjunto de dados pode ser definido como sendo uma "*coleção de dados disponível para acesso e download em um ou mais formatos*" [Maali *et al.* 2014], no qual os dados são fatos conhecidos que podem ser armazenados e que possuem um significado implícito [Elmasri e Navathe 2015]. A fim de atender aos requisitos de diferentes grupos de usuários é necessário que os conjuntos de dados estejam disponíveis em diferentes distribuições. De acordo com a especificação do vocabulário DCAT, uma distribuição "*representa uma forma específica de tornar um conjunto de dados disponível. Cada conjunto de dados pode estar disponível em diferentes formatos e com diferentes mecanismos de acesso. Um arquivo CSV, uma API ou um feed RSS são exemplos de distribuições*".

Dentre os princípios gerais que regem as Boas Práticas, dois merecem destaque. O primeiro deles diz respeito ao compartilhamento de dados em grande escala e o segundo diz respeito ao uso da Web como plataforma para publicação e compartilhamento de dados [Jacobs e Walsh 2004].

O compartilhamento de dados em grande escala permite que os conjuntos de dados sejam acessados por consumidores com diferentes requisitos e expectativas sobre os dados publicados. Dessa forma, torna-se fundamental oferecer os dados em múltiplos formatos, bem como prover metadados capazes de descrever os conjuntos de dados e os dados

propriamente ditos, suas estruturas, informações sobre a licença de uso, os métodos de acesso, frequência de atualização, informações de proveniência e qualidade dos dados, entre outros.

Quadro 2.2 – Exemplo de Boas Práticas para os Dados na Web – BP1 [Lóscio *et al.* 2015]

Best Practice 1: Provide metadata
Metadata must *be provided for both human users and computer applications*
Intended Outcome:
- It must be possible for humans to understand the metadata, which makes it *human readable metadata*.
- It should be possible for computer applications, notably user agents, to process the metadata, which makes it *machine readable metadata*.

Possible approach to implementation:
Possible approaches to provide *human readable metadata:*
- to provide metadata as part of an HTML Web page
- to provide metadata as a separate text file

Possible approaches to provide *machine readable metadata:*
- machine readable metadata may be provided in a serialization format such as Turtle and JSON, or it can be embedded in the HTML page using [JSON-LD], or [HTML-RDFA] or [Microdata]. If multiple formats are published separately, they should be served from the same URL using content negotiation. Maintenance of multiple formats is best achieved by generating each available format on the fly based on a single source of the metadata.
- when defining machine readable metadata, reusing existing standard terms and popular vocabularies are strongly recommended. For example, Dublin Core Metadata (DCMI) terms [DC-TERMS] and Data Catalog Vocabulary [VOCAB-DCAT] should be used to provide descriptive metadata (see Section 9.9 Data Vocabularies).

Quadro 2.3 – Exemplo de Boas Práticas para os Dados na Web – BP2 [Lóscio *et al.* 2015]

Best Practice 2: Provide descriptive metadata
The overall features of a dataset* must *be described by metadata
Intended Outcome
- It should be possible for humans to understand the nature of the dataset.
- It should be possible for user agents be able to automatically discover the dataset.

Possible Approach to Implementation
Descriptive metadata should include the following overall features of a dataset:
- The title and a description of the dataset.
- The keywords describing the dataset.
- The date of publication of the dataset.
- The entity responsible (publisher) for making the dataset available.
- The contact point of the dataset.
- The spatial coverage of the dataset.
- The temporal period that the dataset covers.
- The themes/categories covered by a dataset.

The machine readable version of the discovery metadata may be provided according to the vocabulary recommended by W3C to describe datasets, i.e. the Data Catalog Vocabulary [VOCAB-DCAT]. This provides a framework in which datasets can be described as abstract entities.

As Boas Práticas para Dados na Web foram organizadas de acordo com os desafios apresentados na Seção 6, de maneira que uma ou mais práticas foram propostas para cada desafio. Para fins de ilustração, nos Quadros 2.2 e 2.3, são apresentados exemplos de Boas Práticas propostas para metadados. A *Best Practice 1* é mais geral e diz respeito à forma como os metadados devem ser oferecidos pelos provedores de dados. A *Best Practice 2* é

mais específica e está relacionada com os metadados descritivos que devem ser disponibilizados para os conjuntos de dados.

Para cada prática são descritos o resultado esperado com a aplicação da prática (*Intended Outcome*) e possíveis formas de implementação da prática (*Possible approach to implementation*). Na versão completa das Boas Práticas, disponível em [Lóscio *et al.* 2015], também são descritos a motivação para o uso da prática (*Why*) e quais testes podem ser realizados para verificar se a prática foi implementada de forma adequada (*How to test*). No mais, são apresentados exemplos e *links* para os requisitos que justificam a necessidade de cada uma das práticas propostas. Esses requisitos foram identificados e definidos no documento *Data on the Web: Best Practices, Use Cases & Requirements* [Lee *et al.* 2015].

9. Questões de Pesquisa

Nesta seção, são apresentados alguns temas que possuem questões em aberto que precisam ser solucionadas a fim de garantir que a publicação de dados produza resultados úteis e relevantes.

9.1.Técnicas e Soluções de Publicação de Dados

Coletar dados e publicá-los não é uma tarefa simples nem barata. Fazer isso de forma sistemática é ainda mais complexo, pois requer, frequentemente, o envolvimento de pessoas que são efetivamente os provedores dos dados. Todo o processo necessita de conhecimentos tanto de Informática no que tange as operações de armazenamento e publicação dos dados, quanto conhecimentos mais específicos em relação à natureza dos dados para, quando necessário, incorporar informações semânticas. Há ferramentas já desenvolvidas e trabalhos que propõem soluções para facilitar partes desse processo. Mas, nenhuma delas se tornou um padrão "*de facto*". Por isso, deve-se investir em estruturas e técnicas de coleta e anotação de dados que facilitem a correta publicação dos dados de forma contínua e que não seja fruto apenas de esforços pontuais.

Por outro lado, o crescimento recente do volume de dados na Web também foi alcançado através de contribuições de pessoas comuns. Isso pode ser visto com uma tendência ou apenas como uma moda. Um sistema de *ranking* com *billing* que contabilizasse o consumo e a contribuição de dados poderia incentivar a participação como colaboradores mantendo a base dos sistemas mais atuais e confiáveis.

9.2. Descoberta automática

Um dos grandes desafios para a publicação de dados na Web diz respeito à descoberta automática tanto dos conjuntos de dados publicados quanto das aplicações que fazem uso dos conjuntos de dados. Um grande esforço é realizado para a publicação de dados na Web e, portanto, é importante garantir que os os conjuntos de dados possam ser facilmente buscados e encontrados pelos possíveis consumidores. Além disso, é importante que os consumidores possam acompanhar as diferentes formas como os conjuntos de dados estão sendo publicados.

A disponibilização de metadados que descrevam os conjuntos de dados, bem como as diversas formas de utilização dos dados, em formato processável por máquina, facilitam esta descoberta automática. Como já mencionado, atualmente existem algumas iniciativas para criação de vocabulários para a descrição de dados de proveniência, qualidade e uso dos

dados[28]. Porém, novos padrões ainda precisam ser definidos, como vocabulários para a descrição de informações sobre versionamento. Finalmente, é preciso estabelecer mecanismos para a associação dos metadados aos conjuntos de dados e aplicações.

9.3. Integração de Dados

Uma possibilidade de exploração de dados na Web é a integração de diversos conjuntos de dados com o intuito de recuperar tanto informações implícitas quanto conjuntos de dados mais completos sobre um determinado assunto. A integração de dados tem sido tema frequente de pesquisa na área de Banco de Dados e tem como principal objetivo oferecer uma visão integrada de dados distribuídos em múltiplas fontes de dados [Lóscio 2003]. No contexto de publicação de dados na Web, o processo de integração de dados torna-se mais complexo devido ao grande volume de dados disponível em bases de dados distintas, heterogêneas e desconexas. Ainda há desafios adicionais, tais como conjuntos de dados com alta frequência de atualização, pouca representação ou ausência de metadados e um alto grau de autonomia dos provedores de dados. Isso faz com que os usuários sejam frequentemente responsáveis pela integração dos dados a fim obter respostas mais completas para suas consultas.

Como os conjuntos de dados podem ser não estruturados ou semiestruturados, os processos de descoberta e integração de dados podem ficar mais difíceis, uma vez que, em geral, não são fornecidas informações suficientes em relação à semântica dos dados. Além disso, os cojuntos de dados podem utilizar modelos de dados distintos e diferentes representações para as abstrações de conceitos do mundo real, tornando mais complexos os processos de integração de esquemas, bem como limpeza e transformação dos dados.

Nesse contexto, alguns problemas ainda precisam ser investigados: *i)* como implementar de forma escalável atividades de processamento e transmissão de dados, que são inerentes à integração de dados? *ii)* como realizar os processos de mapeamento de correspondências entre os conjuntos de dados? *iii)* Como reconhecer que instâncias de dados armazenadas em diferentes conjuntos representam uma mesma entidade ? *iv)* Como resolver conflitos, tais como valores distintos para um mesma instância de dados? *v)* Como reduzir o custo, financeiro ou de recursos, associado à execução de processos de integração?

A utilização de Dados Conectados enriquecidos por ontologias pode ser um caminho para atacar o desafio de integração de dados [Alcântara *et al.* 2015]. As ontologias fornecem suporte para a evolução de vocabulários e para processar e integrar a informação existente sem problemas de indefinição ou conflito de terminologia. A aplicabilidade de ontologias se estende a diversos propósitos relativos a integração de dados, destacando a especificação de esquema de mediação, representação de metadados e suporte para consultas de alto nível.

9.4. Granularidade e escopo dos conjuntos de dados

Uma das dificuldades enfrentadas na publicação de dados abertos está relacionada ao nível de granularidade no qual os dados devem ser publicados. Nos casos em que os dados são oriundos de bancos de dados convencionais, planilhas ou documentos já existentes, torna-se necessário determinar quais dados são relevantes para a publicação e como esses dados serão organizados nos conjuntos de dados. Considerando a publicação de dados de acordo com o modelo de dados tabular, é necessário definir se os dados seguirão a mesma

[28] http://www.w3.org/TR/vocab-duv/

organização ou se os dados devem passar para um processo de "desnormalização", por exemplo.

Além disso, também é preciso levar em consideração a publicação de séries temporais, como os dados de Receitas e Despesas, que devem ser publicados com uma frequência pré-estabelecida. A decisão consiste em determinar se novos conjuntos de dados deverão ser criados ou se conjuntos de dados existentes devem ser atualizados a fim de incluir os novos dados.

Outro aspecto que pode deteminar o nível de granularidade é o escopo geográfico. Os conjuntos de dados podem ser organizados de acordo com um escopo local ou global. Por exemplo, considerando dados a respeito de escolas de um determinado estado, por exemplo, os conjuntos de dados podem ser organizados de acordo com os municípios ou apenas um conjunto de dados pode ser disponibilizado com dados de todo o estado. Nesse contexto, torna-se necessária a definição de metodologias que auxiliem o projeto dos conjuntos de dados a serem publicados e que facilitem a definição dos diferentes níveis de escopo e granularidade que podem ser considerados para a publicação dos dados.

10. Conclusão

O interesse na publicação de dados na Web não é algo novo. Entretanto, o crescente interesse no uso da Web como plataforma para compartilhamento de dados trouxe novos desafios para a publicação de dados de forma estruturada. Em cenários onde os consumidores de dados não são previamente conhecidos, a publicação de dados deve ser realizada de maneira a atender grupos de consumidores com requisitos e perfis diversos.

Neste contexto, além dos aspectos básicos de disponibilização de dados, devem ser levados em consideração outros aspectos que dizem respeito à compreensão, à confiabilidade e ao processamento dos dados de forma automática. Por um lado, os provedores devem fornecer informações que auxiliem no entendimento dos dados, como metadados estruturais, mas também devem prover informações que permitem aos consumidores conhecer a proveniência e a qualidade dos dados. Por outro lado, os conumidores devem ser capazes de prover *feedback* sobre os dados que foram usados, a fim de contribuir para a melhoria do processo de publicação. Além disso, os consumidores devem prover informações sobre o uso dos dados, ou seja, juntamente com a aplicação ou visualização que foi gerada a partir dos dados publicados, devem ser disponibilizadas informações sobre os dados que foram usados.

Diversas iniciativas estão sendo desenvolvidas com o intuito de facilitar a publicação de dados e promover canais de comunicação entre os provedores e consumidores de dados. Dentre estas iniciativas, destacam-se o trabalho que vem sendo desenvolvido pelo Data on the Web Best Parctices Working Group e pela Open Knowledge Foundation. Porém, apesar dessas inciativas, ainda existem diversas questões em aberto que precisam ser resolvidas a fim de que a publicação e consumo de dados na Web atinja todo o seu potencial. A partir deste capítulo, espera-se que este assunto seja amplamente discutido na comunidade de Banco de Dados, contribuindo, dessa forma, a geração de soluções inovadoras para os desafios encontrados nesse novo cenário de compartilhamento de dados na Web.

Referências

Abiteboul, S., Buneman, P., e Suciu, D. (2000). Data on the Web: from relations to semistructured data and XML. Morgan Kaufmann.

Alcantara, W., Bandeira, J., Barbosa, A., Teixeira, A., Ávila, T., Bittencourt, I.I., e Isotani, I. (2015). Desafios do uso de Dados Abertos Conectados na Educação Brasileira. Workshop de Desafios da Computação aplicado à Educação (DesafIE'15). Anais do Congresso da Sociedade Brasileira de Computação (CSBC).

Alonso, G., Casati, F., Kuno, H., e Machiraju, V. (2004). Web services (pp. 123-149). Springer Berlin Heidelberg.

Berners-Lee, T. (2006). Linked data-design issues (2006). Recuperado em 04 de agosto de 2015, do URL http://www. w3. org/DesignIssues/LinkedData. html.

Berners-Lee, T. (2010). The 5 stars of open linked data. Recuperado em 04 de agosto de 2015, doAcessado em Agosto de 2015, de http://inkdroid. org/journal/2010/06/04/the-5-stars-of-open-linked-data

Berners-Lee, T., Connolly, D., e Swick, R. R. (1999). Web architecture: Describing and exchanging data. Recuperado em 04 de agosto de 2015, do http://www.w3.org/1999/04/WebData.

Bittencourt, I. I., Costa, E., Silva, M., e Soares, E. (2009). A computational model for developing semantic web-based educational systems, Knowledge-Based Systems, v. 22, n. 4, p. 302-315, 2009.

Bizer, C., Heath, T., e Berners-Lee, T. (2009). Linked data-the story so far. Semantic Services, Interoperability and Web Applications: Emerging Concepts, 205-227.

Casellas, N. (2011) Methodologies, Tools and Languages for Ontology Design. Legal Ontology Engineering: Methodologies, Modelling Trends, and the Ontology of Professional Judicial Knowledge. Law, Governance and Technology. Berlin: Springer, 2011.

Ceri, S., Bozzon, A., Brambilla, M., Della Valle, E., Fraternali, P., e Quarteroni, S. (2013). Web Information Retrieval. Springer Science & Business Media.

DCMI, Dublin Core Metadata Initiative. (2013). Dublin core metadata element set, version 1.1. DCMI recommendation. Recuperado em 04 de agosto de 2015, do http://dublincore.org/documents/dces/

Elmasri, R., e Navathe, S. B. (2014). Fundamentals of database systems. Pearson.

Ferrara, E., De Meo, P., Fiumara, G., e Baumgartner, R. (2014). Web data extraction, applications and techniques: A survey. Knowledge-Based Systems, 70, 301-323.

Gama, K. S. e Lóscio, B. F., (2014). Towards Ecosystems based on Open Data as a Service. In 17th International Conference on Enterprise Information Systems (ICEIS), Barcelona, Spain, 2014.

Garshol, L. M. (2004). Metadata? Thesauri? Taxonomies? Topic maps! Making sense of it all. Journal of information science, 30(4), 378-391.

Goldstein, B. e Dyson, L., (2013). Beyond Transparency: Open Data and the Future of Civic Innovation, Code For America Press.

Gruber, T. R. (1993) A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition. 5, v. 2, p. 199-220, 1993.

He, B., Patel, M., Zhang, Z., e Chang, K. C. C. (2007). Accessing the deep web. Communications of the ACM, 50(5), 94-101.

Heath, T., e Bizer, C. (2011). Linked data: Evolving the web into a global data space. Synthesis lectures on the semantic web: theory and technology, 1(1), 1-136.

Huijboom, N., e Van den Broek, T., (2011). Open Data: an international comparison of strategies, European Journal of ePractice, No. 12, March/April 2011.

Imran, M., e Hlavacs, H. (2012). Provenance in the cloud: Why and how. In The Third International Conference on Cloud Computing, GRIDs, and Virtualization (pp. 106-112).

Isotani, I, e Bittencourt, I.I. (2015). Dados Abertos Conectados. Ed. Novatec. 176p

Jacobs, I., e Walsh, N. (2004). Architecture of the World Wide Web, Volume One. 15 December 2004. W3C Recommendation, Recuperado em 04 de agosto de 2015, do http://www.w3.org/TR/webarch/

Klyne, G., e Carroll, J. J. (2006). Resource description framework (RDF): Concepts and abstract syntax. Recuperado em 04 de agosto de 2015, do http://www.w3.org/TR/rdf-concepts/

Lee, D., Lóscio, B.F., e Archer, P. Data on the Web Best Practices Use Cases & Requirements Public (2015), Recuperado em 04 de agosto de 2015, do http://www.w3.org/TR/dwbp-ucr/

Lóscio, B.F., Burle, C., e Calegari, N.: Data on the Web Best Practices. W3C Second Public Working Draft (2015), Recuperado em 04 de agosto de 2015, do http://www.w3.org/TR/2015/WD-dwbp-20150224/

Lóscio, B. F. (2003) Managing the Evolution of XML-based Mediation Queries. Ph.D. Thesis, Federal University of Pernambuco, Brazil, 2003.

Maali, F., Erickson, J., e Archer, P. (2014). Data catalog vocabulary (DCAT). W3C Recommendation. Recuperado em 04 de agosto de 2015, do http://www.w3.org/TR/vocab-dcat/

Mandel, L. (2008). Describe REST Web services with WSDL 2.0. Rational *Software* Developer, IBM.

Marinho, A., Murta, L., Werner, C., Braganholo, V., Cruz, S. M. S. D., Ogasawara, E., e Mattoso, M. (2012). ProvManager: a provenance management system for scientific workflows. Concurrency and Computation: Practice and Experience, 24(13), 1513-1530.

Maximilien, E. M., Ranabahu, A., e Gomadam, K. (2008). An online platform for web apis and service mashups. Internet Computing, IEEE, 12(5), 32-43.

Möller, K. (2013). Lifecycle models of data-centric systems and domains: The abstract data lifecycle model. Semantic Web 4, 1, 67-88.

Nelson, B. J. (1981). Remote procedure call. PhD-Thesis, CMU-CS-81-119, Carnegie-Mellon University.

Oliveira, L. e Lóscio, B.F. (2014). Uma Abordagem para Captura de Informações sobre Aplicações que fazem uso de Dados Abertos, III Simpósio Brasileiro de Tecnologia da Informação (SBTI 2014), Maceió, Alagoas, 2014.

Open Defintion (2015): Conformant Licenses, Recuperado em 04 de agosto de 2015, do http://opendefinition. org/licenses

Open Knowledge Foundation. (2012). The Open Data Handbook. Recuperado em 04 de agosto de 2015, do http://opendatahandbook.org

Papazoglou, M. P. (2003). Service-Oriented Computing: Concepts, Characteristics and Directions, 4th International Conference on Web Information Systems Engineering (WISE'03), Rome, Italy, 2003.

Pras, A., Schönwälder, J., Burgess, M., Festor, O., Perez, G. M., Stadler, R., e Stiller, B. (2007). Key research challenges in network management.Communications Magazine, IEEE, 45(10), 104-110.

Pritchett, D. (2008). Base: An acid alternative. Queue, 6(3), 48-55.

Sauermann, L., Cyganiak, R., e Völkel, M. (2011). Cool URIs for the semantic web. Recuperado em 04 de agosto de 2015, do http://www.w3.org/TR/cooluris/

Tauberer, J. (2007). 8 Principles of Open Government Data. Recuperado em 04 de agosto de 2015, do http://opengovdata.org/

Truong, H.L., e Dustdar, S. (2009). On analyzing and specifying concerns for data as a service. Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific. IEEE, 2009.

Wang, R. Y., e Strong, D. M. (1996). Beyond accuracy: What data quality means to data consumers. Journal of management information systems, 5-33.

Yu, S., e Woodard, C. J. (2009). Innovation in the programmable web: Characterizing the mashup ecosystem. In Service-Oriented Computing–ICSOC 2008 Workshops (pp. 136-147). Springer Berlin Heidelberg.

Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., Auer, S., e Hitzler, P. (2013). Quality assessment methodologies for linked open data. Submitted to Semantic Web Journal.

Sobre os Autores

Bernadette Farias Lóscio:
Possui doutorado em Ciência da Computação pelo CIn/UFPE (2003) e desde 2010 atua como Professora Adjunto dessa mesma instituição. Suas principais áreas de atuação são Dados na Web, Dados Abertos, Integração de Dados e Web Semântica. Desde 2012, atua como consultora na área de Dados Abertos, além de ter ministrado diversos cursos e palestras sobre esse tema. Dentre os trabalhos realizados, destacam-se a consultoria em Dados Abertos para o Governo Federal de El Salvador e Governo Federal da Costa Rica. Também merece destaque a consultoria para a Prefeitura de Recife, que culminou com o lançamento do Portal de Dados Abertos da cidade. Faz parte do grupo de trabalho Data on the Web Best Practices do W3C (World Wide Web Consortium), sendo editora de três documentos que estão sendo produzidos pelo grupo: "Data on the Web Best Practices Use Cases & Requirements", "Data on the Web Best Practices" e "Dataset Usage Vocabulary".

Marcelo Iury S. Oliveira:
Possui graduação em Bacharelado em Ciência da Computação pela Universidade Estadual do Ceará e mestrado em Ciência da Computação pela Universidade Federal de Campina Grande. Trabalhou como pesquisador da Universidade Federal de Campina Grande e como analista de sistemas do Instituto Atlântico. Atualmente, é Professor Adjunto da Universidade Federal Rural de Pernambuco e doutorando em Ciência da Computação pela Universidade Federal de Pernambuco. Tem experiência na área de Ciência da Computação, com ênfase em Engenharia de Software, Sistemas Distribuídos e Banco de Dados. Suas atuais áreas de interesse são Dados Abertos e Web das Coisas.

Ig Ibert Bittencourt:
É pós-doutor em Computação pela Unicamp, Bolsista de Produtividade em Desenvolvimento Tecnológico e Extensão Inovadora do CNPq, Representante Consultivo do W3C, Professor Adjunto da Universidade Federal de Alagoas e Co-Diretor do Núcleo de Excelência em Tecnologias Sociais. Desde 2011, dedica-se à pesquisa, inovação e empreendedorismo social em Informática na Educação e Ontologias. Ig Bittencourt defende o empreendedorismo social inovador como modelo de desenvolvimento econômico e social necessário para o crescimento sustentável do Brasil. É um dos fundadores da Start-up MeuTutor Soluções Educacionais (ganhadora de prêmios de inovação regionais e nacionais e desenvolve produtos educacionais baseada em ontologias e dados abertos) e um dos criadores da Plataforma JOINT (Java Ontology INtegrated Toolkit, que simplifica o desenvolvimento de aplicações baseadas em ontologias e o consumo de dados conectados).

cap:3

Capítulo 3

Tecnologias para Gerenciamento de Dados na Era do Big Data

Victor Teixeira de Almeida e Vitor Alcântara Batista

Abstract

This chapter is an overview of big data management, and intends to explore and diferentiate several recent technologies. A classic problem of the database community will be used as a background for the examples given throughout this course: triangle counting on graphs. This problem has been chosen because it is being extensively used to identify the importance of individuals on social networks. Also, since it can be described by an algorithm that is simple to understand and yet complex to execute in terms of performance, the differences between technologies in design and performance will be easily demonstrated.

Resumo

Este capítulo pretende explorar e diferenciar de forma introdutória diversas tecnologias recentes para gerenciamento de dados na era do big data. Será utilizado como pano de fundo para os exemplos um problema clássico da comunidade de bancos de dados: a contagem de triângulos em grafos. Ele foi escolhido por ser um problema atual e prático, frequentemente utilizado para identificar a importância de indivíduos em redes sociais. Além disso, ele é de fácil representação e alta complexidade de execução. Através do seu uso, é possível demonstrar as diferenças entre as tecnologias em termos de expressividade e desempenho.

3.1. Introdução

A era da informação, com o advento da Internet (Web 2.0), está terminando. Estamos entrando na era da computação voltada ao consumidor, chamada por Grossman de *Device Era* [Grossman 2012] e por Wirfs-Brock de *Ambient Computing Era* [Wirfs-Brock 2011] (Figura 3.1). Nesta era, focada no indivíduo, as pessoas possuem um ambiente rico em recursos e comunicação, principalmente através de dispositivos móveis. A computação é somente um alicerce para a entrega e melhoria de vida do indivíduo, que é seu principal objetivo.

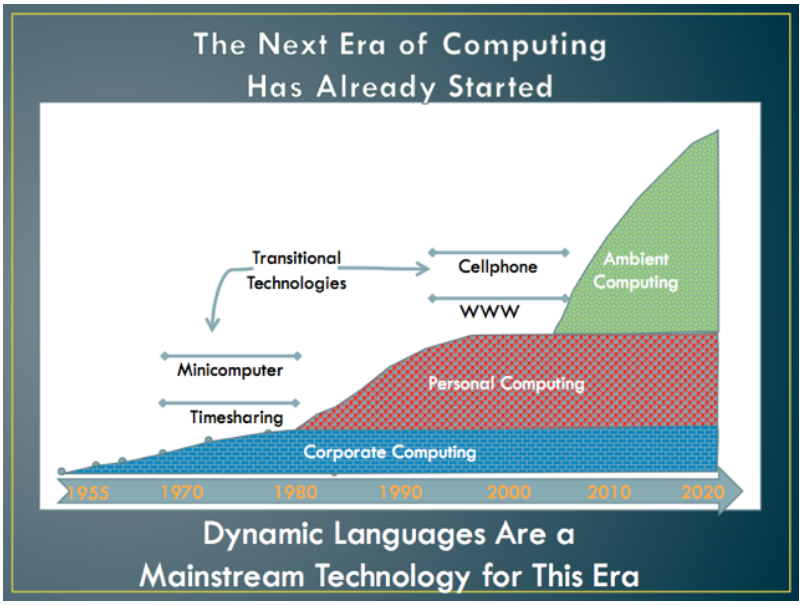


Figura 3.1. Eras da computação. Retirado do blog de Wirfs-Brock [Wirfs-Brock 2011].

O que está acontecendo neste momento é que a quantidade de informação que se está gerando está em crescimento explosivo. Tecnologias consideradas de *big data* tornam-se necessárias para poder eficientemente armazenar, gerenciar, processar ou visualizar essa quantidade massiva de informação. Talvez não na mesma velocidade, mas também com um extraordinário crescimento, novas tecnologias têm surgido para suprir tal necessidade.

Este capítulo pretende explorar e diferenciar de forma introdutória diversas tecnologias recentes para o gerenciamento de dados na era do big data. Será utilizado como pano de fundo para os exemplos, um problema clássico da comunidade de bancos de dados: a contagem de triângulos. Esta problema é perfeito para explicar as diferenças entre as tecnologias em termos de expressividade e desempenho, pois pode ser representado por uma simples consulta SQL com duas junções, contudo extremamente complexa de ser executada eficientemente. A partir deste problema, é possível identificar como cada tecnologia se comporta para representar sua solução, bem como seu desempenho.

O capítulo está subdividido por tecnologia e na Seção 3.9 são apresentadas considerações finais sobre o que ainda há de tecnologia a ser detalhada e que não foi descrito nesse capítulo. A Seção 3.2 detalha o problema da contagem de triângulos, que irá permear as próximas seções de tecnologia. A Seção 3.3 mostra como funcionam bancos de dados relacionais tradicionais. As Seções 3.4 e 3.5 mostram as abordagens de bancos de dados orientados a colunas e em memória, respectivamente. A Seção 3.6 apresenta bancos de dados paralelos e introduz os conceitos de paralelismo de dados e computação. A Seção 3.7 que apresenta o Hadoop e alguns dos seus principais frameworks para análise de dados.

3.2. Contagem de triângulos

Nesta seção é descrito o problema da contagem de triângulos, que será utilizado ao longo deste capítulo. Este problema é extremamente interessante para a avaliação de tecnologias uma vez que é de simples descrição e implementação, logo didádico, e complexo em termos de execução, desempenho.

O problema, como o próprio nome diz, consiste em contar triângulos em um grafo, ou seja, contar os subgrafos t_i de um grafo G contendo 3 diferentes vértices conectados entre si (triângulos).

Uma das grandes aplicações da contagem de triângulos é o cálculo do coeficiente de agrupamento de um nó em um grafo ou do grafo como um todo. Esta métrica possui diversas aplicações práticas em análises de redes sociais. O coeficiente de agrupamento de um nó v expressa a probabilidade de dois nós vizinhos a v serem também vizinhos entre si. Para o grafo G como um todo, o coeficiente de agrupamento é a média dos coeficientes de cada vértice do grafo. Altos valores para este coeficiente significam uma comunidade coesa (*small world community*).

Implementações de algoritmos eficientes para este problema abundam na literatura. O Algoritmo 1, adapdado de [Chu e Cheng 2012] para retornar somente a contagem de triângulos, apresenta uma execução eficiente para o problema, e é a base para as principais implementações de algoritmos que assumem que os dados cabem na memória.

Algoritmo 1 Contagem de triângulos em memória

Input: Grafo $G = (V, E)$
Output: c, a contagem de triângulos em G

```
1: c ← 0
2: para cada v ∈ V faça
3:    para cada u ∈ adj_G(v), dado que u > v faça
4:       para cada w ∈ (adj_G(v) ∩ adj_G(u)), dado que w > u faça
5:          c ← c + 1
6:       fim para
7:    fim para
8: fim para
9: return(c)
```

Este algoritmo começa por inicializar a variável c de contagem de triângulos com 0. Então, para todos os vértices do grafo (nomeados v), o algoritmo tenta resgatar um vértice que seja adjacente a v, nomeado de u, e outro vértice que seja ao mesmo tempo adjacente a v e a u, nomeado de w. Cada vez que esses três vértices conectados por arestas são encontrados, o algoritmo incrementa o contador de triângulos c. Há mais um ponto a ser explicado aqui que é um teste para remover duplicatas que garante que $u > v$ e $w > u$, assumindo que os vértices possuam identificadores únicos no domínio dos números naturais, por exemplo. A complexidade deste algoritmo depende da implementação da busca pelos vértices u e w nas listas de adjacências de v e u, respectivamente.

Nos exemplos desse capítulo, serão utilizados dois conjuntos de dados, que foram também adotados por Leskovec et al. [Leskovec e Mcauley 2012] e encontram-se dispo-

nibilizados pelo Projeto SNAP (*Stanford Network Analyst Project*). Eles contém usuários do Facebook e do Twitter, com estatísticas descritas na Tabela 3.1. Deve-se atentar para o fato de que o Twitter e o Facebook possuem políticas diferentes; no Twitter é possível que um usuário siga algum outro sem que este também o siga de volta, enquanto que no Facebook a contrapartida é exigida. Isto torna o grafo do Twitter direcionado e o do Facebook não direcionado.

Tabela 3.1. Estatísticas dos conjuntos de redes sociais (Facebook e Twitter) utilizados.

Conjunto de dados	Facebook	Twitter
Vértices	4.039	81.306
Arestas	88.234	1.768.149
Triângulos	1.612.010	13.082.506

O que torna este problema de contagem de triângulos interessante, em termos de complexidade de execução, é a natureza dos dados. Normalmente, esses dados de redes sociais possuem distorção (*skew*), seguindo uma função de distribuição chamada *power-law*. Tentando explicar de uma forma simples, o que acontece é que a grande maioria dos vértices possuem muito poucas ligações, mas uma pequena fração possui muitas ligações. O gráfico abaixo mostra a distribuição dos dados. O eixo das abcissas mostra o grau do vértice e o eixo das ordenadas mostra quantos vértices possuem tal grau.

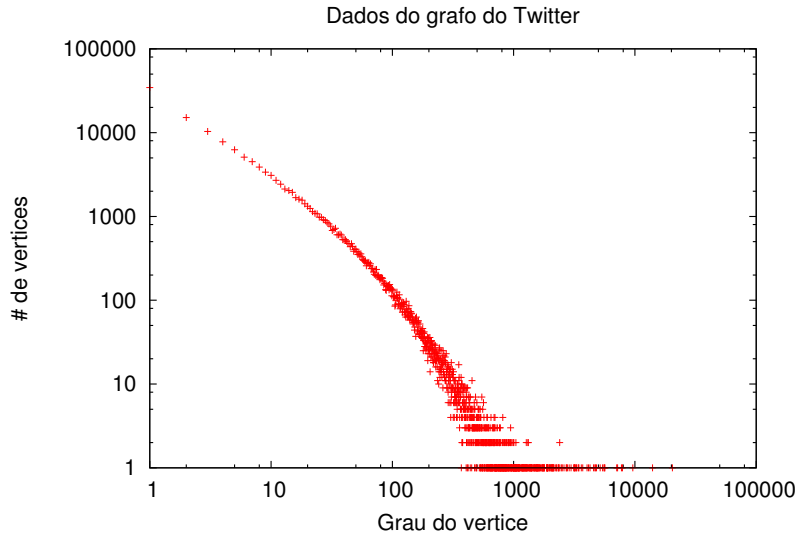


Figura 3.2. Distribuição de dados de redes sociais (*power-law*) com dados do Twitter

3.3. Bancos de dados relacionais

Bancos de dados relacionais são sistemas que implementam o modelo de dados relacional proposto por Codd na década de 70 [Codd 1970], rapidamente implementado por um sistema chamado System R da IBM [Astrahan et al. 1976]. A principal ideia é a de representar os dados em forma de relações (tabelas) e as operações são definidas a partir de uma álgebra: a álgebra relacional. O sistema recebe comandos através de uma linguagem

declarativa (SQL) [Chamberlin e Boyce 1974], os converte em operadores da álgebra relacional para então executá-los nos dados persistentes nas relações.

recID	fname	lname	gender	city	country	birthday
...	...	...	...	...	...	...
39	John	Smith	m	Chicago	USA	12.03.1964
40	Mary	Brown	f	London	UK	12.05.1964
41	Jane	Doe	f	Palo Alto	USA	23.04.1976
42	John	Doe	m	Palo Alto	USA	17.06.1952
43	Peter	Schmidt	m	Potsdam	GER	11.11.1975
...	...	...	...		...	...

Figura 3.3. Representação de uma tabela em um banco de dados relacional.

A Figura 3.3 mostra a representação de uma relação contendo informações sobre indivíduos em um formato de tabela. Esta relação possui os seguintes atributos do indivíduo : (i) *fname*, o nome; (ii) *lname*, o sobrenome; (iii) *gender*, o sexo, *m* para masculino e *f* para feminino; (iv) *city*, a cidade onde nasceu; (v) *country*, o país onde nasceu; e (vi) *birthday*, a data de nascimento.

Os principais operadores da álgebra relacional são: projeção (Π), seleção (σ), junção ($\bowtie$). Sejam duas relações R e S com atributos $r_1, \ldots, r_m$ e $s_1, \ldots, s_n$, representadas por $R(r_1, \ldots, r_m)$ e $S(s_1, \ldots, s_n)$. Uma projeção em R recebe como argumento uma lista de atributos subconjunto dos atributos de R e retorna a relação contendo somente os atributos desta lista. A projeção é representada por $\Pi_{lista\ de\ atributos}(R)$. Uma seleção em R recebe como argumentos uma condição em forma de predicado e retorna todas as tuplas em R que satisfazem o predicado. A seleção é representada por $\sigma_{predicado}(R)$. A junção é um operador binário e é aplicado sobre duas relações R e S. Ele recebe como argumento uma condição em forma de predicado (usualmente a igualdade entre dois atributos das relações R e S), e retorna a relação correspondente ao produto cartesiano das duas relações R e S cujas tuplas satisfazem o predicado da junção. A junção é representada por $R \bowtie_{predicado} S$.

A linguagem SQL expressa de forma declarativa operações sobre os dados armazenados nas relações a partir da álgebra relacional. Uma versão simplificada da linguagem SQL é mostrada a seguir:

```
SELECT <lista de atributos>
FROM <lista de tabelas>
WHERE <predicado>;
```

Após o sistema receber uma consulta na linguagem SQL, esta deve ser expressada segundo operadores da álgebra relacional da forma mais eficiente possível. Um otimizador de consultas é o responsável por esta tradução. A ordem dos operadores influencia no resultado; normalmente seleções (σ) são as primeiras a serem executadas, pois reduzem o tamanho das relações resultantes. A ordem das junções também é de extrema importância; uma junção entre as relações R, S e T pode ser executada nas seguintes ordens: (i) $R \bowtie S \bowtie T$, (ii) $R \bowtie T \bowtie S$ e (iii) $S \bowtie T \bowtie R$. Adicionalmente, existem inúmeros al-

goritmos para a execução eficiente das junções que devem coexistir no sistema e serem utilizados em circunstâncias em que são os mais eficientes [Mishra e Eich 1992].

O problema da contagem de triângulos da Seção 3.2 pode ser expresso na seguinte consulta SQL, assumindo que há uma relação chamada Twitter com os attributos *follower* e *followee* contendo números inteiros de identificadores de usuários representando relacionamentos na rede social, onde *follower* segue *followee*.

```
SELECT COUNT(*)
FROM TWITTER R, TWITTER, S, TWITTER T
WHERE R.followee = S.follower AND
      S.followee = T.follower AND
      T.followee = R.follower AND
      R.follower > S.follower AND
      S.follower > T.follower;
```

Uma ordem de execução desta consulta, expressa em operadores da álgebra relacional é

$$\sigma_{S.follower>T.follower}(\sigma_{R.follower>S.follower}((R \bowtie_{R.followee=S.follower} S) \bowtie_{(S.followee=T.follower)\wedge(T.followee=R.follower)} T)) \quad (1)$$

Este plano de execução irá seguir os seguintes passos:

1. Primeira junção entre as tabelas R e S com predicado $R.followee = S.follower$
2. Seleção com predicado $R.follower > S.follower$ aplicado ao resultado anterior
3. Junção do resultado anterior com a tabela T com predicado $(S.followee = T.follower) \wedge (T.followee = R.follower)$
4. Seleção com predicado $S.follower > T.follower$

3.4. Bancos de dados colunares

A ideia por traz dos bancos de dados colunares não é nova. Uma das primeiras propostas de organizar os dados de um banco de dados por colunas, em vez da tradicional representação por linhas dos bancos de dados relacionais, apareceu em 1969 [Estabrook e Brill 1969]. De forma resumida, enquanto um banco de dados relacional tradicional armazena cada registro ou tupla em um espaço contínuo do disco (ou memória), os bancos de dados colunares armazenam cada coluna em espaços contínuos [Abadi et al. 2009].

A grande vantagem da representação colunar é a compressão de dados, uma vez que o domínio dos dados é preservado em espaços contíguos de disco ou memória. Uma das principais formas de compressão de dados é por dicionário, onde cada coluna de uma tabela (ou relação) é dividida em um índice com os valores distintos ordenados e um vetor com a codificação dos valores de cada tupla na mesma sequência em que aparecem na tabela. Esta codificação utiliza um numero mínimo de bits do domínio de valores de cada

registro, baseada no dicionário. Com os dados comprimidos, menos bytes trafegam entre o disco e a memória principal, tornando operações de consulta e agregação mais rápidas. A Figura 3.3 mostra a representação tradicional de uma tabela num banco de dados relacional tradicional (orientado por linhas) e a Figura 3.4 mostra como a primeira coluna da relação é organizada em um banco de dados colunar com compressão por dicionário.

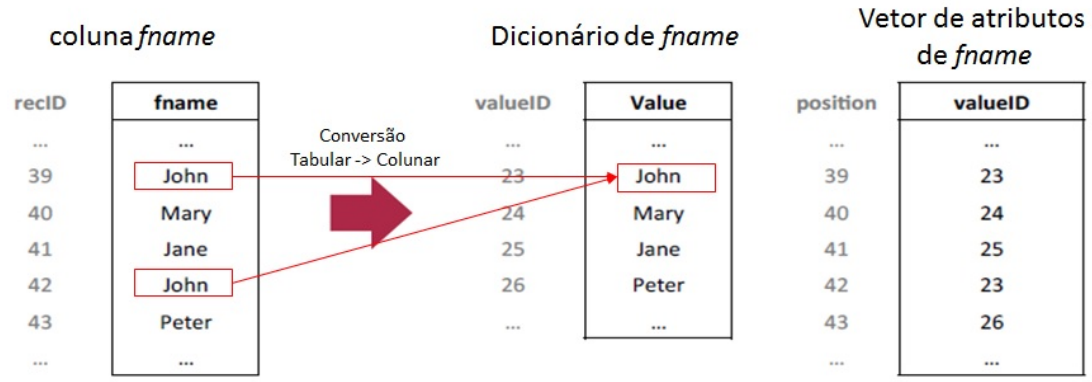


Figura 3.4. Organização da coluna fname em um banco de dados colunar.

Para explicar melhor o mecanismo de compressão, suponha que a coluna *fname* tenha 100 caracteres representados por 1 byte cada. Suponha também que a tabela em questão trata-se da lista de todos os cidadãos brasileiros, ou seja, há aproximadamente 200 milhões de registros. Logo, o espaço de armazenamento no modelo tradicional é de $200 * 10^6 * 100$ $Bytes \cong 18,62$ $GBytes$. No armazenamento em formato de colunas, os dados são armazenados em um dicionário que contém uma entrada para cada valor distinto. Imagine que há 10 mil nomes (primeiro nome) distintos no Brasil. Com isso, para o dicionário, são necessários $10 * 10^3 * 100$ $bytes \cong 1$ $Mbyte$. Já o vetor de valores conterá os 200 milhões de registros, mas codificados pela quantidade mínima de bits necessários para se codificar os 10 mil valores distintos $log_2(\ 10.000) \cong 13,2$, ou seja, 14 bits. Nesse caso, o vetor de valores terá $200 * 10^6 * 14$ $bits \cong 2,6$ $GBytes$. Nesse exemplo simples é possível observar um fator de compressão de $18,62/(0,001+2,6) \cong 7,2$ vezes.

A recuperação de um determinado registro (tupla) passa por um acesso direto à posição dele em cada um dos vetores de valor das colunas da tabela e mais um acesso em cada dicionário para traduzir a codificação para o valor original.

O grande problema nessa representação são operações de remoção e inserção, que usualmente causam uma reorganização do dicionário e consequentemente uma reorganização de todos os dados das colunas cujo dicionário foi reordenado. Por isso, o uso desse tipo de tecnologia deve ser preferível em conjuntos de dados cuja operação predominante seja de consulta. Plattner [Plattner 2009] alega que os bancos de dados corporativos possuem uma carga predominantemente de consulta e com isso, pode-se unificar os repositórios OLTP e OLAP na mesma estrutura, sugerindo para tal, a representação colunar.

Os sistemas modernos que utilizam a representação colunar lançam mão de diversas estratégias para minimizar esses efeitos de reorganização dos dicionários. O SAP HANA, por exemplo, divide os dados em principal e diferencial [Plattner e Zeier 2012].

Novos registros e atualizações de valores são incluídos no conjunto diferencial, que é mantido em tamanho pequeno. As operações de consulta juntam os dados do conjunto principal com o conjunto diferencial. Embora a busca no conjunto diferencial seja ineficiente, ela é realizada sobre um conjunto pequeno de dados. De tempos em tempos, os conjuntos são mesclados resultando em um novo conjunto principal e um conjunto diferencial praticamente vazio, que conterá apenas as operações realizadas durante a operação de mesclagem.

Há também outros mecanismos de compressão para outros domínios de dados, como por exemplo, números inteiros e de ponto flutuante [Abadi et al. 2006, Zukowski et al. 2006]. É importante ressaltar que todos estes mecanismos de compressão de dados tentam balancear a taxa de compressão de dados com o processamento para compressão e descompressão. Em geral, os mecanismos mais eficientes em termos de taxa de compressão não são utilizados, uma vez que o processamento (CPU) para descompressão dos dados durante a execução das consultas pode ser um fator determinante. Adicionalmente, em geral, todos os mecanismos de compressão de dados preservam as propriedades de igualdade e ordenação dos dados, ou seja, valores $=$, $<$ ou $>$ no domínio original dos dados são também $=$, $<$ ou $>$ no domínio de compressão. Esta propriedade é extremamente importante, pois a execução de diversos algoritmos da álgebra relacional que envolvem a comparação e ordenação dos dados, como a junção, podem ser executados no domínio dos dados comprimidos, evitando assim processamento desnecessário de descompressão dos dados. É fácil observar esta propriedade no exemplo acima da compressão da coluna *fname*, desde que o dicionário esteja ordenado por nome.

Atualmente, os principais fornecedores de bancos de dados relacionais tradicionais também fornecem soluções de armazenamento colunar, como Microsoft, Oracle, IBM, SAP e outros. Entretanto, há sistemas gerenciadores de bancos de dados puramente colunares, os comerciais SAP IQ e Actian Vectorwise, e os de código aberto C-Store e MonetDB.

3.5. Bancos de dados em memória

Sistemas de banco de dados em memória são aqueles em que a fonte primária dos dados reside em memória principal (RAM). Esses dados têm, usualmente, cópia em disco, para eventuais falhas no hardware ou simplesmente falta de energia. Embora os sistemas de bancos de dados tradicionais também mantenham dados em memória na forma de *cache*, a principal diferença é que, nos bancos de dados em memória, a fonte de dado primária (ou principal) está armazenada na memória RAM, enquanto nos tradicionais, está armazenada em disco [Garcia-Molina e Salem 1992, DeWitt et al. 1984].

Esse tipo de tecnologia ganhou força nos últimos anos devido aos avanços nas arquiteturas de hardware que, atualmente, permitem sistemas com Terabytes de memória RAM compartilhada entre vários processadores. Além disso, houve um barateamento no custo desses equipamentos, o que tornou viável os sistemas de banco de dados em memória.

Como o acesso à memória principal chega a ser cerca de 1.000 vezes mais rápido que os discos modernos como SSD (disco de estado sólido), esse tipo de tecnologia é bem convidativo. O leitor mais atento pode se perguntar: e se um sistema de banco de dados

tradicional tiver um *cache* grande o suficiente para caber todo o volume de dados, qual seria a diferença? Ainda assim, os sistemas tradicionais são projetados de forma não ótima para uso da memória, pois há ainda a indireção do *cache*. Por exemplo, será necessário consultar um gerenciador do *cache* toda vez que for acessar o dado; outro exemplo são as estruturas dos índices, que estão em estruturas onde o acesso não é imediato Árvores B+.

Embora existam hardwares especializados com memória não volátil, fontes de energia redundantes e outros mecanismos para minimizar falhas no hardware, ainda é virtualmente impossível garantir que o dado em memória principal esteja seguro. Por isso, uma questão muito importante para bancos de dados em memória é a recuperação de falhas. A estratégia tipicamente utilizada é persistir em disco cada uma das transações em um *log*. Uma transação só é concluída após a escrita da mesma no disco (*log*). De tempos em tempos, são criados *checkpoints*, onde uma cópia da memória principal é realizada em disco, podendo assim, descartar os *logs* de transações anteriores ao *checkpoint*. A frequência da criação desses *checkpoints* depende da confiabilidade do hardware e da volatilidade dos dados.

Outras questões importantes do projeto de sistemas de bancos de dados em memória, como controle de concorrência, processamento de transações, organização dos dados, métodos de acesso, processamento de consultas, desempenho e clusterização são discutidos em mais detalhes por Garcia-Molina e Salem [Garcia-Molina e Salem 1992].

Atualmente, praticamente todos os grandes fornecedores de soluções tradicionais de bancos de dados relacionais também possuem versões em memória de seus produtos, como a Microsoft, Oracle e IBM, mas também há fornecedores especializados nesse tipo de solução, sendo os principais comerciais o SAP HANA e o VoltDB, e o MemSQL de código aberto. Alguns desses produtos também possuem características de bancos de dados colunares apresentados na seção 3.4, principalmente com o objetivo de atingir boa compressão dos dados e melhor utilização da memória.

3.6. Bancos de dados paralelos

Bancos de dados paralelos ou massivamente paralelos (MPP, do inglês *massively parallel processing*) também não são tecnologias recentes, discussões sobre esse assunto e implementações de sistemas datam da década de 80-90 [DeWitt e Gray 1992, Fushimi et al. 1986]. Nestes sistemas, os dados das tabelas são distribuídos em diversos nós de um cluster e o processamento da consulta é paralelizado.

Os dados podem ser distribuídos através de particionamento vertical, onde as tabelas são quebradas por colunas; horizontal, onde as tabelas são quebradas por tuplas; ou ambos. A distribuição dos dados entre os nós pode utilizar o conhecimento prévio das consultas mais utilizas no sistema (*query workload*), e o otimizador de consultas, nestes casos, conhecendo a forma como os dados estão distribuídos, pode repassar a parte das consultas para os nós específicos que devem processá-la. O problema geral desta abordagem é que existe a possibilidade de desbalanceamento de nós (*data skew*). O particionamento dos dados por funções de hash é bastante utilizado, pois através de boas funções de hash é possível evitar este desbalanceamento.

A arquitetura dos bancos de dados paralelos pode ser de compartilhamento de

memória (*shared-memory*, onde a memória é compartilhada entre os nós do cluster; compartilhamento de disco (*shared-disk*), onde cada nó do cluster possui sua própria memória, mas o disco é compartilhado entre todos os nós; ou sem compartilhamento (*shared-nothing*), onde cada nó possui sua própria memória e disco e o compartilhamento de dados entre nós do cluster se faz através de transferência por mensagens. As arquiteturas com compartilhamento (*shared-memory* e *shared-disk*) mostraram-se ineficientes em escalabilidade para grandes implantações [DeWitt e Gray 1992, Stonebraker 1986]. A Figura 3.5 mostra um banco de dados paralelo com um nó mestre e arquitetura sem compartilhamento.

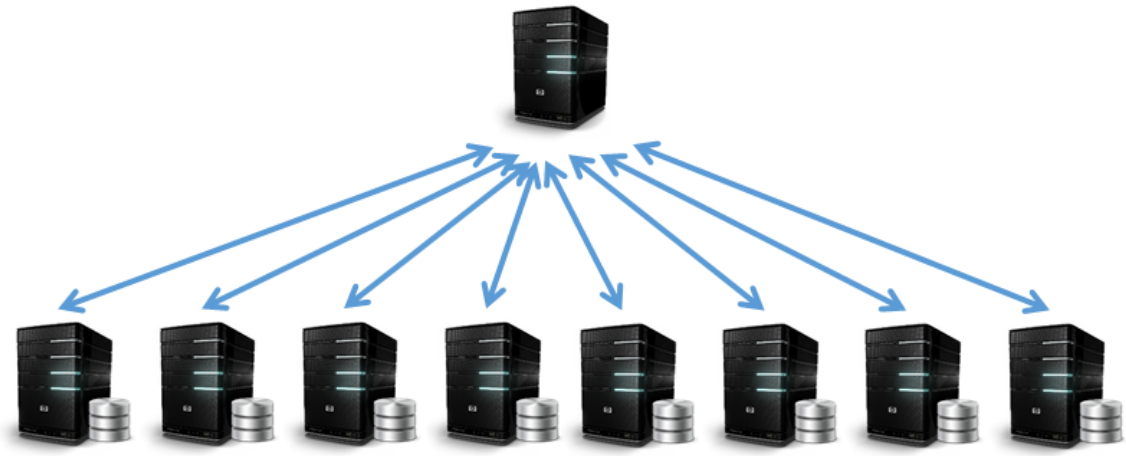


Figura 3.5. Banco de dados paralelo e a arquitetura sem compartilhamento.

Os operadores da álgebra relacional são todos paralelizáveis, principalmente a junção, com algoritmos de distribuição dos dados por função hash [Schneider e DeWitt 1989]. A ideia principal destes algoritmos é a de que cada nó lê sua parcela dos dados, aplica uma função hash nos atributos da junção para determinar o nó de destino e envia os dados para os nós específicos. Desta forma, quando os nós recebem os dados, há a garantia (pela função de hash) de que os nós que irão participar do resultado da junção estão fisicamente localizados juntos, neste passo. Cada nó então executa um algoritmo linear de junção, tipicamente uma junção por hash em memória, e reporta o resultado para o nó coletor. Este processo é demonstrado na Figura 3.6. Nesta figura, é mostrada uma arquitetura com quatro nós sem compartilhamento, e estes quatro nós estão replicados em duas camadas para melhor mostrar a fase de espalhamento dos dados por função de hash (*shuffle*).

A solução do problema da contagem dos triângulos apresentada na Seção 3.2, utilizando banco de dados paralelos é a mesma utilizada em bancos de dados relacionais, uma vez que tal tecnologia implementa a álgebra relacional. Na Seção 3.3 é mostrada a consulta SQL que resolve o problema. Esta consulta envolve duas junções, que serão executadas em paralelo, e as seleções de remoção de duplicatas que serão aplicadas após as junções no fluxo de execução das consultas (*pipeline*).

Os principais sistemas gerenciadores paralelos de bancos de dados comerciais são Pivotal Greenplum Database, HP Vertica, Teradata, Microsoft SQL Server Parallel Data

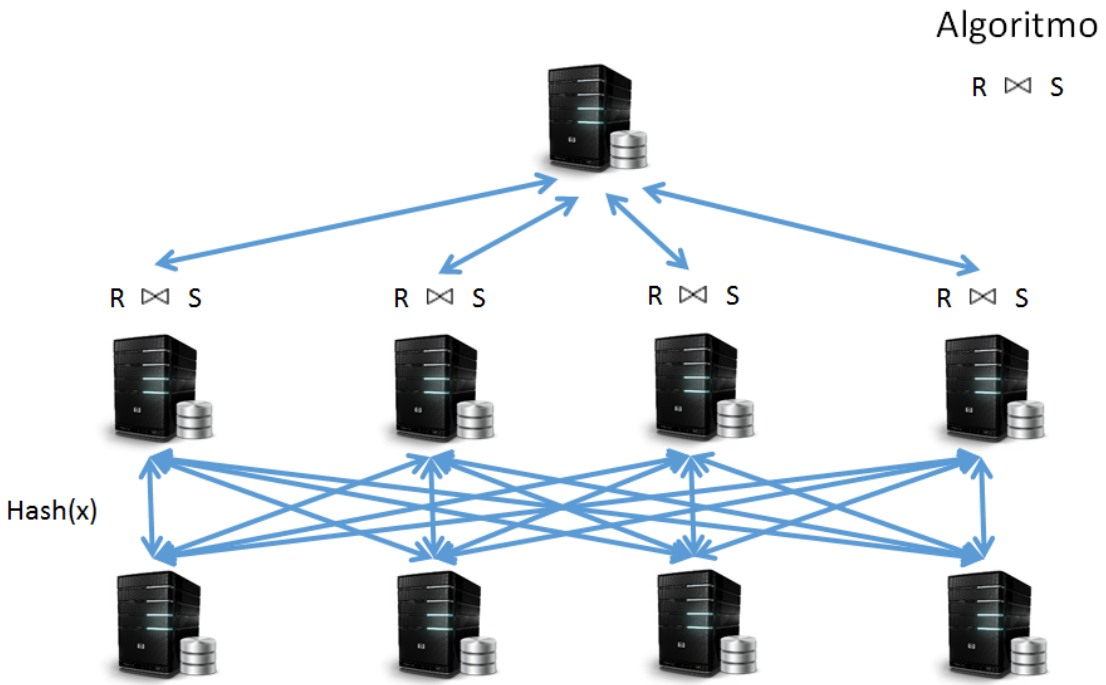


Figura 3.6. Junção por hash em um banco de dados paralelo sem compartilhamento.

Warehouse e IBM Netezza. Dentre os de código aberto pode-se citar: AsterixDB, Stratosphere (atualmente Apache Flink), Myria e Presto.

3.7. Hadoop

3.7.1. Breve histórico

O problema era simples: como criar um índice para uma máquina de busca de toda a Internet? Foi com esse desafio que Mike Cafarella e Doug Cutting resolveram desenvolver o Apache Nutch. Rapidamente o *crawler* e a máquina de busca ficaram prontos, mas eles perceberam que a arquitetura não escalaria para criar um índice de mais de um bilhão de páginas da Internet. Na mesma época, a equipe do Google publicou um artigo que explicava a arquitetura do GFS (Google FileSystem) [Ghemawat et al. 2003], que era um sistema de arquivos distribuído usado em sua máquina de busca. Doug e Mike decidiram criar uma implementação *open source* dessa arquitetura e a chamaram de NDFS (Nutch Distributed FileSystem).

Em 2004, a equipe do Google publicou um novo artigo detalhando como era possível criar um índice de toda a Internet usando o mecanismo de processamento paralelo denominado MapReduce [Dean e Ghemawat 2008]. Com base nesse trabalho, os desenvolvedores do Nutch migraram a maior parte de seus algoritmos para executar sobre o MapReduce e o NDFS. Mais tarde, Doug Cutting foi trabalhar no Yahoo! liderando uma equipe que construiu a nova geração de máquina de busca deles. Depois, o NDFS (posteriormente chamado de HDFS ou *Hadoop Distributed Filesystem*) e o MapReduce tornaram-se um projeto da Apache Software Foundation sob o nome de Apache Hadoop.

Desde então, o Hadoop tem sido usado mundialmente para processar enormes quantidades de dados. Vários frameworks foram construídos usando a sua infraestrutura, como será visto a seguir. Diversos fornecedores criaram suas próprias distribuições do Hadoop, como Microsoft, IBM, EMC, Oracle e outras empresas especializadas como Cloudera e Hortonworks.

3.7.2. Funcionamento do HDFS

O HDFS, como mencionado acima, é um sistema de arquivos distribuídos, projetado para armazenar arquivos muito grandes[1] executando em computadores padrão de baixo custo.

Assim como em qualquer sistema de arquivos, um arquivo é dividido em **blocos** de dados. Enquanto tipicamente um sistema de arquivos tradicional armazena dados em pequenos blocos (e.g. 512 bytes ou 1 kbyte), o HDFS usa, por padrão, blocos de 64MB. Isso torna o HDFS ineficiente para uso em arquivos muito pequenos e numerosos. Para garantir disponibilidade e leitura em paralelo, cada um dos blocos é replicado em um dos nós de um *cluster* HDFS. Quando um disco ou um dos nós do *cluster* falha, além do bloco poder ser lido de outro nó, o sistema de arquivos automaticamente recria os blocos presentes naquele disco em outros nós do *cluster*.

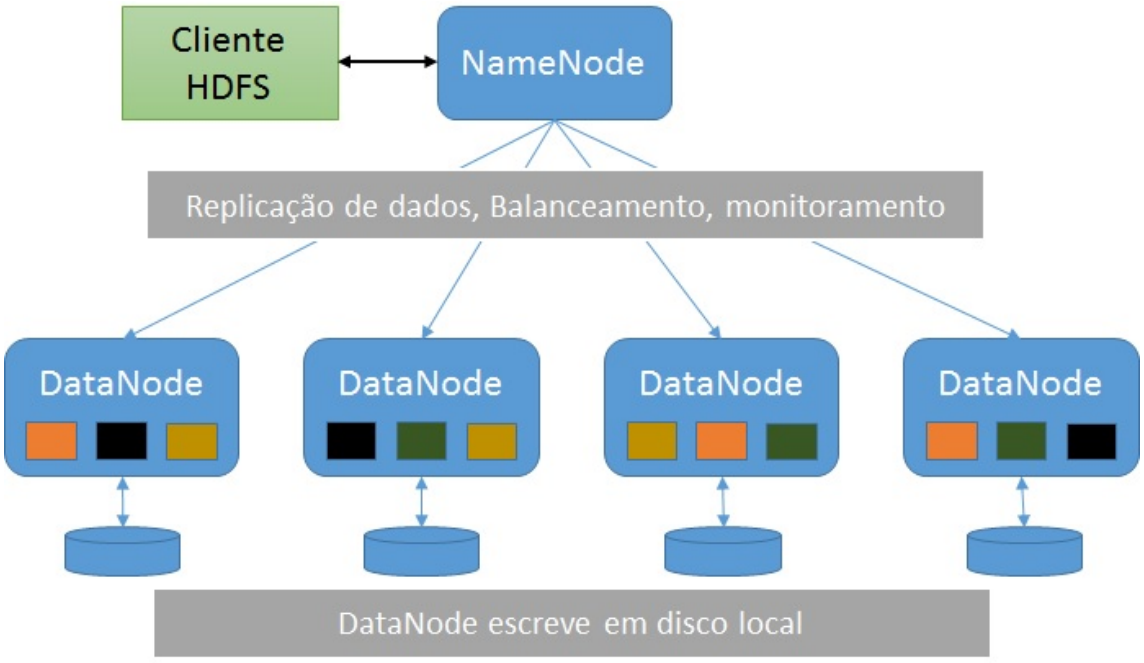


Figura 3.7. Visão geral da arquitetura do HDFS

A arquitetura de um *cluster* HDFS divide-se em *NameNode* e *DataNode*. O primeiro armazena um índice de arquivos e de seus blocos e o segundo armazena os dados (blocos). Um cliente que queira ler arquivos no HDFS, primeiro consulta o *NameNode*, que então diz de quais nós do cluster os blocos serão lidos, garantindo um balanceamento de carga na leitura. Arquivos criados no HDFS só podem ser modificados anexando conteúdo no final. Não é possível modificar blocos já escritos. A falha do *DataNode* implica

[1] Atualmente há instâncias do HDFS armazenando PetaBytes de dados.

na indisponibilidade de todo o HDFS e, por esse motivo, é importante mantê-lo resiliente a falha com mecanismos de redundância. Uma visão geral dessa arquitetura pode ser vista na figura 3.7.

3.7.3. O ecossistema Hadoop

Atualmente o Hadoop conta com um ecossistema com diversos *frameworks*. Uma lista, não exaustiva, de alguns dos principais projetos que compõem o ecossistema pode ser vista abaixo.

- Ambari: Uma ferramenta web para aprovisionamento e gerenciamento de um cluster Hadoop e de diversos de seus componentes.
- HBase: Um banco de dados não relacional (NoSQL) e colunar que utiliza a infraestrutura do Hadoop como mecanismo de armazenamento.
- Hive: Uma infraestrutura de armazém de dados com suporte a sumarização de dados e consultas.
- Pig: Uma linguagem de alto nível para fluxo de dados e um *framework* de execução de computação distribuída.
- Spark: Uma *engine* rápida e de propósito geral para processamento de dados em memória baseados nos dados do HDFS. O Spark oferece um modelo de programação simples e poderoso para executar uma enorme gama de atividades como ETL, aprendizagem de máquina, processamento contínuo de dados, processamento de grafos, etc.
- Sqoop: uma ferramenta para transferência massiva de dados entre bancos de dados relacionais e o HDFS.
- Mahout: Um conjunto de bibliotecas para executar algoritmos de aprendizagem de máquina e mineração de dados. Os coordenadores do projeto decidiram mover a implementação dos algoritmos de MapReduce para o Spark.

3.7.4. MapReduce

O Hadoop MapReduce é um *framework* para facilitar a escrita de programas de computador para processar uma enorme quantidade de dados de forma paralela, distribuída e resiliente a falhas. Os dados de entrada para um *job* MapReduce, por estarem armazenados no HDFS, são também processados de forma distribuída, aproveitando dos dados disponíveis localmente em um nó do *cluster*. A Figura 3.8 apresenta um desenho esquemático de um *job* MapReduce, que é, de forma resumida, composto por duas fases:

1. *Map*: quando os dados são processados e produzem saídas como tuplas no formato (*chave*, *valor*);
2. *Reduce*: quando as tuplas com mesma *chave* são agrupadas para alguma atividade de agregação.

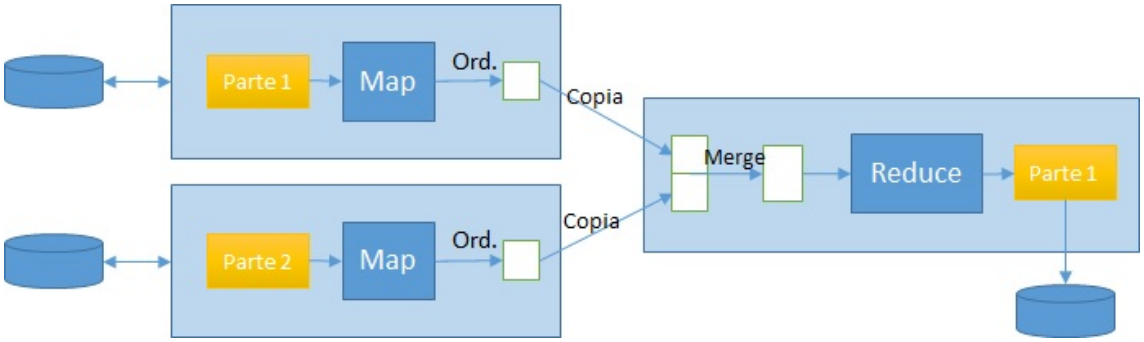


Figura 3.8. Um *job* MapReduce

Um exemplo bem simples para entender o MapReduce é um job para contar a ocorrência de cada palavra em um texto. Na fase de *Map*, cada linha lida do arquivo é dividida em suas palavras que produzem uma saída {PalavraA, 1}. Note que se uma palavra aparece duas vezes na mesma linha duas tuplas idênticas serão produzidas. Na faze de *Reduce*, as tuplas com mesma chave serão agrupadas e os valores 1 serão somados.

Em geral, o programador não precisa se preocupar com comunicação de dados, tratamento de concorrência e eventuais falhas em algum nó que está processando um determinado *job*. Esse é um dos grandes diferenciais do Hadoop MapReduce.

Uma possível solução para contagem de triângulos utilizando MapReduce, baseada na proposta feita por Suri e Sergei [Suri e Vassilvitskii 2011], envolve o encadeamento de 3 *jobs* MapReduce. O primeiro constrói um conjunto de todas as tríades (par de arestas que compartilham um vértice) de um grafo. Essas tríades, juntamente com as arestas originais são gerados como linhas, mas com um atributo identificador dizendo se é uma aresta original do arquivo ou se foi gerada pelo primeiro passo. Essa saída serve de entrada para o segundo *job*, que conecta as tríades com as arestas originais, formando os triângulos. Por fim, o terceiro *job* conta o número de triângulos gerados pelo passo anterior. Vamos olhar um exemplo para melhor entendimento.

Dado o grafo não direcionado de entrada descrito na Tabela 3.2, onde os nomes identificam os vértices e cada linha uma aresta que liga esses vértices, podemos identificar 3 triângulos: (Bernardo, Chico, Renato), (Renato, Chico, Roberto) e (André, Tamara, Marcelo).

Após a etapa de *Map* do primeiro *job*, são gerados um conjunto de pares (*chave*, *valor*) conforme a Tabela 3.3. A chave é escolhida entre os vértices da aresta, mas isso pode ocorrer de diversas maneiras. Para simplificar, usaremos o primeiro nome em ordem alfabética.

A etapa de *Reduce* do primeiro *job* é a mais importante. Nessa etapa, para cada chave da Tabela 3.3, são gerados as arestas originais (valor na Tabela 3.3) bem como as tríades baseadas nessas arestas. O resultado, já ordenado, pode ser visto na Tabela 3.4.

O segundo *job* tem apenas a etapa de *Reduce* e toma de entrada o resultado produzido na Tabela 3.4. Para cada grupo de chaves (arestas), é verificado se há uma aresta original (Gerado = 0). Se sim, é verificada em quantas tríades essa aresta fecha um tri-

Tabela 3.2. Exemplo de grafo de entrada.

Origem	Destino
André	Bernardo
André	Marcelo
André	Tamara
Bernardo	Chico
Bernardo	Renato
Chico	Renato
Chico	Roberto
Chico	Livia
Marcelo	Tamara
Roberto	Renato

Tabela 3.3. Resultado da etapa de Map do primeiro *Job*.

Chave	Valor (arest)
André	André, Marcelo
André	André, Tamara
André	André, Bernardo
Marcelo	Marcelo, Tamara
Bernardo	Bernardo, Chico
Bernardo	Bernardo, Renato
Chico	Chico, Roberto
Chico	Chico, Livia
Chico	Chico, Renato
Roberto	Roberto, Renato

Tabela 3.4. Resultado da etapa de Reduce do primeiro *Job*.

Chave	Gerado	Tríades
André, Bernardo	0	
André, Marcelo	0	
André, Tamara	0	
Bernardo, Chico	0	
Bernardo, Tamara	1	{André,Bernardo}, {André,Tamara}
Bernardo, Renato	0	
Bernardo, Marcelo	1	{André,Bernardo}, {André,Marcelo}
Chico, Livia	0	
Chico, Roberto	0	
Chico, Renato	1	{Bernardo,Chico}, {Bernardo,Renato}
Chico, Renato	0	
Livia, Roberto	1	{Chico, Livia}, {Chico, Roberto}
Livia, Renato	1	{Chico, Livia}, {Chico, Renato}
Marcelo, Tamara	1	{André,Marcelo}, {André,Tamara}
Marcelo, Tamara	0	
Roberto, Renato	1	{Chico,Roberto}, {Chico,Renato}
Roberto, Renato	0	

ângulo, emitindo uma contagem parcial para cada chave. Pelo nosso exemplo, podemos ver que isso ocorre nas chaves (Chico, Renato), (Roberto, Renato) e (Marcelo, Tamara), totalizando nossos 3 triângulos.

A última etapa (terceiro *Job*) é uma tarefa trivial de somar as contagens parciais emitidas pelo passo anterior.

3.7.5. Hive

O Hadoop MapReduce, como descrito anteriormente na Seção 3.7.4, é um framework para ser utilizado e desenvolvido em uma linguagem de programação, e.g. Java, com as complexidades inerentes de um desenvolvimento de software em linguagens de programação. Assim surgiu rapidamente a necessidade de especificação de programas MapReduce em linguagens declarativas de alto nível.

O Hive [Thusoo et al. 2009] é uma das iniciativas neste sentido, no qual a especificação de *jobs* MapReduce é feita em uma linguagem similar ao SQL: o HiveQL. O Hive foi inicialmente desenvolvido pelo Facebook e atualmente seus principais contribuidores são o próprio Facebook e o Netflix.

O Hive interpreta arquivos do HDFS como tabelas com estrutura, de forma similar a um SGBD. Entretanto, nenhuma estrutura adicional de SGBD é criada pelo Hive, exceto os metadados sobre o formato dos arquivos em tabelas. No HiveQL há uma linguagem de definição destes formatos.

Como um exemplo, imagine que há um arquivo contendo os dados do Twitter no HDFS contendo os dados da tabela do twitter descrita na Seção 3.3 com atributos

follower e *followee*, ou seja, um arquivo texto em formato CSV contendo dois números inteiros (delimitados por vírgula). O comando HiveQL abaixo cria os metadados para a tabela Twitter a partir deste arquivo.

```
CREATE TABLE twitter (follower INT, followee INT);
LOAD DATA INPATH '/input/twitter' OVERWRITE INTO TABLE twitter;
```

Embora o HiveQL não implemente a linguagem SQL em todas as suas funcionalidades, a consulta da Seção 3.3 que resolve o problema da contagem dos triângulos pode ser aplicada diretamente no console do Hive.

É importante notar que o Hive interpreta o SQL e gera *jobs* MapReduce em Java, que são compilados e executados sob demanda.

3.7.6. Pig

De forma similar ao Hive, o Yahoo! criou o Pig [Olston et al. 2008] em 2006 e o moveu como um projeto para o Hadoop em 2007, com o intuito de simplificar o desenvolvimento de *jobs* MapReduce através de uma linguagem procedural de alto nível (Pig Latin).

O Pig Latin, diferentemente do HiveQL, é uma linguagem procedural e não declarativa. É possível efetuar atribuições e manipulação de variáveis intermediárias. Em geral, cada comando PigLatin corresponde a um operador da álgebra relacional.

O exemplo abaixo mostra a declaração da tabela do Twitter na linguagem Pig Latin.

```
Twitter = LOAD '/input/twitter' USING PigStorage(',')
  AS (follower: int, followee: int);
```

Note que, diferentemente do Hive, o Pig Lating cria a tabela Twitter e ao mesmo tempo instancia a mesma lendo do arquivo de input com o formato especificado.

O código abaixo executa a contagem dos triângulos, uma vez mapeada a tabela Twitter.

```
Twitter_1 = LOAD '/input/twitter' USING PigStorage(' ')
  AS (follower:int, followee:int);
Twitter_2 = LOAD '/input/twitter' USING PigStorage(' ')
  AS (follower:int, followee:int);
Twitter_3 = LOAD '/input/twitter' USING PigStorage(' ')
  AS (follower:int, followee:int);

Twitter_SelfJoin = JOIN Twitter_1 by $1,
                        Twitter_2 by $0;

Twitter_FullJoin = JOIN Twitter_3 by ($1, $0),
                        Twitter_SelfJoin by ($0, $3);

Triangles = FILTER Twitter_FullJoin
  BY ($0 < $1 AND $1 < $3) OR ($0 > $1 AND $1 > $3);

Triangles_Grp = GROUP Triangles ALL;
Triangles_Cnt = FOREACH Triangles_Grp GENERATE COUNT(Triangles);

DUMP Triangles_Cnt;
```

As três primeiras linhas atribuem as tabelas Twitter_1, Twitter_2 e Twitter_3. Esta atribuição em três tabelas não deveria ser necessária, mas na versão do Pig (antiga) em que foi criado e executado este código não era possível fazer uma junção da tabela com ela mesma (*self join*).

As próximas duas linhas executam as duas junções; a próxima, um filtro que elimina as duplicatas; e as duas penúltimas executam um agrupamento para contagem. A última linha imprime o valor final. Somente a partir de um comando DUMP que o Pig começa a construir os *jobs* MapReduce, compilar e executá-los.

3.8. Apache Spark

O Apache Spark é uma plataforma para computação distribuída que foi projetada para ser de propósito geral e muito eficiente [Zaharia et al. 2012, Karau et al. 2015]. A principal diferença em relação ao MapReduce é que toda a computação é feita e armazenada em memória, sem necessidade de salvar em disco resultados intermediários.

A unidade básica de dados do Spark é o *Resilient Distributed Dataset* (RDD). O conceito é semelhante ao bloco de dados do HDFS, mas trata-se de coleções de dados que estão na memória RAM dos nós do *cluster*. Na prática, o Spark carrega os dados de um bloco do HDFS na memória RAM do nó em que o bloco está. Os programas Spark fazem dois tipos de operação com um RDD:

- **Transformações**: Transformam um RDD em outro RDD. Entre as transformações mais comuns podemos encontrar as operações de *Map* e *ReduceByKey* (mesmo conceito do MapReduce), ordenação e operações de conjuntos, como união, interseção e diferença.

- **Ações**: Produzem algum resultado a partir de um RDD. Ações típicas consistem em enumerar alguma quantidade de itens de um RDD, contar e somar.

O Spark utiliza o conceito de *execução preguiçosa*, para que só quando realmente um resultado tenha de ser produzido (uma **ação** é executada), e toda a sequência de passos necessária é conhecida, o Spark realmente lê os dados e faz cálculos na memória. Com isso, o motor de execução do Spark consegue otimizar tudo que será executado, escolhendo os melhores nós, a melhor sequência, etc.

O Apache Spark também vem com um conjunto de bibliotecas com algoritmos para aprendizado de máquina, grafos, fluxo contínuo de dados e SQL. Algumas dessas são vistas a seguir.

3.8.1. Utilizando o console python

O Apache Spark possui consoles interativos nas linguagens Python e Scala e seus *jobs* podem ser submetidos em batch também em Java. A API do Spark é acessada a partir de um objeto central denominado `SparkContext`. Esse objeto contém a conexão com uma instância do cluster e a partir dele todos os outros objetos e métodos são acessados. Ao se iniciar um console do Spark, o objeto já está automaticamente disponível através da variável `sc`. Para compreender a simplicidade do modelo de programação do Spark, o trecho de código abaixo lê um arquivo txt e faz a contagem de ocorrência das palavras.

```
#produz um RDD onde cada item e uma linha do arquivo texto
arquivo = sc.textFile('hdfs://servidor:10001/arquivo.txt')

#para cada linha produz N itens no novo RDD, uma para cada
   palavra.
palavras = arquivo.flatMap(lambda linha : linha.split(' '))

#cria novo RDD com tuplas do formato (palavra, 1)
palavrasCV = palavras.map(lambda palavra : (palavra, 1))

#executa o Reduce usando a funcao add para os valores das
   tuplas.
contagemPalafras = palavrasCV.reduceByKey(add)

#somente nesse comando toda a computacao e feita de forma
   otimizada.
contagemPalavras.collect()
```

Para resolver o problema de contagem de triângulos utilizando o Spark, a ideia é semelhante à usada com o modelo do MapReduce.

3.8.2. Spark SQL

O Spark SQL [Xin et al. 2013b, Armbrust et al. 2015], anteriormente chamado de Shark, é um módulo do Apache Spark para processamento de dados estruturados (relacionais).

Ele utiliza uma abstração denominada *DataFrame*, e também serve como uma máquina de execução de consultas distribuídas baseada em SQL.

Um *DataFrame* do Spark SQL tem as mesmas características de um *DataFrame* em R ou em Pandas (Python), e pode ser criado a partir de diversas fontes de dados, como um arquivo JSON, um arquivo texto, um RDD do Spark, uma tabela do Hive, ou qualquer fonte que possua um driver JDBC. O esquema de dados de um *DataFrame* pode ser inferido através de reflexão ou programaticamente.

O trecho de código abaixo mostra como executar a contagem de triângulos utilizando o SQL proposto na seção 3.2.

```
# Importa os modulos e cria o contexto
from pyspark.sql import SQLContext, Row
sqlContext = SQLContext(sc)

# Carrega o arquivo de arestas para um RDD
linhas =
   sc.textFile("hdfs://servidor:10001/data/triangles/twitter_combined.txt")
partes = linhas.map(lambda l: l.split())
arestas = partes.map(lambda p: Row(follower=int(p[0]),
   followee=int(p[1])))

# Infere o schema e registra o DataFrame como tabela
schemaWiki = sqlContext.createDataFrame(arestas)
schemaWiki.registerTempTable("arestas")

# Executa o SQL
triangulos = sqlContext.sql("SELECT count(*) FROM arestas R,
   arestas S, arestas T WHERE R.follower = S.followee AND
   S.follower = T.followee AND T.follower = R.followee AND
   R.follower > S.follower AND
     S.follower > T.follower")
print triangulos.collect()
```

3.8.3. Spark GraphX

O Spark GraphX é um novo módulo do Apache Spark que fornece um conjunto de abstrações e ferramentas para processamento paralelo de algoritmos em grafos [Xin et al. 2013a]. Nas abstrações de vértices e arestas é possível incluir propriedades, como pesos, capacidades máximas e mínimas; ou qualquer outra propriedade que seja relevante para modelar um problema.

O Spark GraphX possui um conjunto de operações essenciais para diversos algoritmos de análise de grafos, como operações em paralelo sobre os vértices/arestas dos grafos, obtenção de subgrafos, inversão de arestas e agregação de vizinhos. Esse último, por exemplo, pode ser usado para calcular o grau de cada vértice de um grafo. Mais

detalhes desse módulo podem ser obtidos na documentação oficial do produto [2].

Como é recente seu desenvolvimento, ainda são poucos os algoritmos implementados e a única linguagem suportada é o Scala. Atualmente ele conta com algoritmos de PageRank [Brin e Page 2012], identificação de componentes conectados e contagem de triângulos, que é demonstrada com trecho de código abaixo. Comparado com as estratégias de MapReduce, a contagem de triângulos utilizando Spark GraphX é de várias ordens de grandeza mais rápido e eficiente em consumo de memória.

```
# Carrega o grafo a partir de um arquivo cujas linhas sao pares
   de identificadores dos nos, definindo uma aresta
val graph = GraphLoader.edgeListFile(sc,
   "graphx/data/followers.txt",
   true).partitionBy(PartitionStrategy.RandomVertexCut)

# conta o numero de triangulos
val triCounts = graph.triangleCount().vertices

# imprime o resultado
println(triCounts.mkString("\n"))
```

3.9. Considerações finais

Neste capítulo foram apresentadas algumas tecnologias recentes da era do big data. Entretanto, a lista de tecnologias com certeza não foi exaustiva. Não foram abordadas muitas outras tecnolgias existentes. Como exemplo, não foram citadas as tecnologias de bancos de dados não relacionais (NoSQL) [Han et al. 2011], e a comparação entre estas e o modelo relacional [Cattell 2011, Stonebraker 2010, Stonebraker 2012]. Não é objetivo deste capítulo prover um *survey* sobre todas as possíveis tecnologias de gerenciamento de big data, mas de mostrar de forma introdutória algumas das principais tecnologias já maduras de mercado.

O que também está fora do escopo deste capítulo é a comparação técnica mais profunda e de desempenho entre cada uma das tecnologias [Pavlo et al. 2009], como por exemplo, comparação entre bancos de dados orientados a colunas ou a linhas [Abadi et al. 2008] e a comparação entre modelos de paralelismo Hadoop ou bancos de dados [Stonebraker et al. 2010].

Referências

[Abadi et al. 2006] Abadi, D., Madden, S., e Ferreira, M. (2006). Integrating compression and execution in column-oriented database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 671–682. ACM.

[Abadi et al. 2009] Abadi, D. J., Boncz, P. A., e Harizopoulos, S. (2009). Column-

[2] http://spark.apache.org/docs/latest/graphx-programming-guide.html

oriented database systems. *Proceedings of the VLDB Endowment*, 2(2):1664–1665.

[Abadi et al. 2008] Abadi, D. J., Madden, S. R., e Hachem, N. (2008). Column-stores vs. row-stores: how different are they really? In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 967–980. ACM.

[Armbrust et al. 2015] Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A., et al. (2015). Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM.

[Astrahan et al. 1976] Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., Eswaran, K. P., Gray, J. N., Griffiths, P. P., King, W. F., Lorie, R. A., McJones, P. R., Mehl, J. W., et al. (1976). System R: relational approach to database management. *ACM Transactions on Database Systems (TODS)*, 1(2):97–137.

[Brin e Page 2012] Brin, S. e Page, L. (2012). Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer networks*, 56(18):3825–3833.

[Cattell 2011] Cattell, R. (2011). Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27.

[Chamberlin e Boyce 1974] Chamberlin, D. D. e Boyce, R. F. (1974). Sequel: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, pages 249–264. ACM.

[Chu e Cheng 2012] Chu, S. e Cheng, J. (2012). Triangle listing in massive networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(4):17.

[Codd 1970] Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387.

[Dean e Ghemawat 2008] Dean, J. e Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[DeWitt e Gray 1992] DeWitt, D. e Gray, J. (1992). Parallel database systems: the future of high performance database systems. *Communications of the ACM*, 35(6):85–98.

[DeWitt et al. 1984] DeWitt, D. J., Katz, R. H., Olken, F., Shapiro, L. D., Stonebraker, M. R., e Wood, D. A. (1984). *Implementation techniques for main memory database systems*, volume 14. ACM.

[Estabrook e Brill 1969] Estabrook, G. F. e Brill, R. C. (1969). The theory of the TAXIR accessioner. *Mathematical Biosciences*, 5(3):327–340.

[Fushimi et al. 1986] Fushimi, S., Kitsuregawa, M., e Tanaka, H. (1986). An overview of the system software of a parallel relational database machine grace. In *VLDB*, volume 86, pages 209–219.

[Garcia-Molina e Salem 1992] Garcia-Molina, H. e Salem, K. (1992). Main memory database systems: An overview. *Knowledge and Data Engineering, IEEE Transactions on*, 4(6):509–516.

[Ghemawat et al. 2003] Ghemawat, S., Gobioff, H., e Leung, S.-T. (2003). The google file system. In *ACM SIGOPS operating systems review*, volume 37, pages 29–43. ACM.

[Grossman 2012] Grossman, R. (2012). *The structure of digital computing: from mainframes to big data*. Open Data Press, LLC.

[Han et al. 2011] Han, J., Haihong, E., Le, G., e Du, J. (2011). Survey on nosql database. In *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, pages 363–366. IEEE.

[Karau et al. 2015] Karau, H., Konwinski, A., Wendell, P., e Zaharia, M. (2015). *Learning Spark: Lightning-Fast Big Data Analysis*. "O'Reilly Media, Inc.".

[Leskovec e Mcauley 2012] Leskovec, J. e Mcauley, J. J. (2012). Learning to discover social circles in ego networks. In *Advances in neural information processing systems*, pages 539–547.

[Mishra e Eich 1992] Mishra, P. e Eich, M. H. (1992). Join processing in relational databases. *ACM Computing Surveys (CSUR)*, 24(1):63–113.

[Olston et al. 2008] Olston, C., Reed, B., Srivastava, U., Kumar, R., e Tomkins, A. (2008). Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM.

[Pavlo et al. 2009] Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., e Stonebraker, M. (2009). A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 165–178. ACM.

[Plattner 2009] Plattner, H. (2009). A common database approach for oltp and olap using an in-memory column database. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 1–2. ACM.

[Plattner e Zeier 2012] Plattner, H. e Zeier, A. (2012). *In-Memory data management: Technology and applications*. Springer Science & Business Media.

[Schneider e DeWitt 1989] Schneider, D. A. e DeWitt, D. J. (1989). *A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment*, volume 18. ACM.

[Stonebraker 1986] Stonebraker, M. (1986). The case for shared nothing. *IEEE Database Eng. Bull.*, 9(1):4–9.

[Stonebraker 2010] Stonebraker, M. (2010). Sql databases v. nosql databases. *Communications of the ACM*, 53(4):10–11.

[Stonebraker 2012] Stonebraker, M. (2012). Newsql: An alternative to nosql and old sql for new oltp apps. *Communications of the ACM. Retrieved*, pages 07–06.

[Stonebraker et al. 2010] Stonebraker, M., Abadi, D., DeWitt, D. J., Madden, S., Paulson, E., Pavlo, A., e Rasin, A. (2010). Mapreduce and parallel dbmss: friends or foes? *Communications of the ACM*, 53(1):64–71.

[Suri e Vassilvitskii 2011] Suri, S. e Vassilvitskii, S. (2011). Counting triangles and the curse of the last reducer. In *Proceedings of the 20th international conference on World wide web*, pages 607–614. ACM.

[Thusoo et al. 2009] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., e Murthy, R. (2009). Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629.

[Wirfs-Brock 2011] Wirfs-Brock, A. (2011). The third era of computing.

[Xin et al. 2013a] Xin, R. S., Gonzalez, J. E., Franklin, M. J., e Stoica, I. (2013a). GraphX: A resilient distributed graph system on Spark. In *First International Workshop on Graph Data Management Experiences and Systems*, page 2. ACM.

[Xin et al. 2013b] Xin, R. S., Rosen, J., Zaharia, M., Franklin, M. J., Shenker, S., e Stoica, I. (2013b). Shark: Sql and rich analytics at scale. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data*, pages 13–24. ACM.

[Zaharia et al. 2012] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., e Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association.

[Zukowski et al. 2006] Zukowski, M., Heman, S., Nes, N., e Boncz, P. (2006). Super-scalar ram-cpu cache compression. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 59–59. IEEE.

Sobre os Autores

Victor Teixeira de Almeida:
É analista de sistemas na Petrobras, atua na Arquitetura Tecnológica de TIC. Mestre em bancos de dados pela COPPE/UFRJ; doutor em bancos de dados pela Universidade de Hagen, Alemanha; passou um ano (2013) como pós-doutor na Universidade de Washington, Seattle, EUA, no Projeto Myria, uma plataforma de big data como serviço na nuvem. Especialista no gerenciamento de grandes volumes de dados e tecnologias de big data. Recentemente empossado como Professor Adjunto 20h no Instituto de Ciência da Computação da UFF.

Vitor A. Batista:
Possui Mestrado e Graduação em Ciência da Computação pela Universidade Federal de Minas Gerais. Possui também duas pós-graduações Lato-sensu em Gestão de Negócios e Finanças pela Fundação Dom Cabral. Tem trabalhado com desenvolvimento de software desde 1997, utilizando diversas tecnologias e frameworks. Atuou como analista de negócios e requisitos, implementador e gerenciou projetos de sistemas de médio e grande porte. No passado foi gerente de processos do Synergia Engenharia de Software e Sistemas (UFMG). Atualmente trabalha como Engenheiro de Software na Petrobras, avaliando novas tecnologias que possam contribuir com a inovação de processos na Companhia. Lecionou diversos cursos de graduação e pós-graduação e possui diversos artigos publicados em conferências e periódicos de Engenharia de Software. Possui certificações técnicas do IEEE (Certified Software Development Professional), da Microsoft (MCSD, MCSE) e da OMG (UML Certified Professional).

Patrocínio Diamante

Prata

Bronze

Organização

Promoção