Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Uma Abordagem em Paralelo para Matching de Grandes Ontologias com Balanceamento de Carga

Tiago Brasileiro Araújo

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sistemas de Informação e Banco de Dados

Prof. Dr. Carlos Eduardo Santos Pires
(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

A663a Araújo, Tiago Brasileiro.
Uma abordagem em paralelo para *matching* de grandes ontologias com balanceamento de carga / Tiago Brasileiro Araújo. – Campina Grande, 2016.
78f. : il. color.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
"Orientação: Prof. Dr. Carlos Eduardo Santos Pires".

1. Ciência da Computação. 2. *Matching* de ontologias. 3. Computação Distribuída. 4. Computação Paralela. 5. Balanceamento de Carga. I. Pires, Carlos Eduardo Santos. II. Título.

CDU 004(043)

"UMA ABORDAGEM EM PARALELO PARA MATCHING DE GRANDES ONTOLOGIAS COM BALANCEAMENTO DE CARGA"

TIAGO BRASILEIRO ARAÚJO

DISSERTAÇÃO APROVADA EM 07/03/2016

CARLOS EDUARDO SANTOS PIRES, Dr., UFCG
Orientador(a)

CLAUDIO ELIZIO CALAZANS CAMPELO, PhD., UFCG
Examinador(a)

MARIA CLAUDIA REIS CAVALCANTI, Dra., IME
Examinador(a)

CAMPINA GRANDE - PB

Resumo

Atualmente, o uso de grandes ontologias em diversos domínios do conhecimento está aumentando. Uma vez que estas ontologias podem apresentar sobreposição de conteúdo, a identificação de correspondências entre seus conceitos se torna necessária. Esse processo é chamado de *Matching* de Ontologias (MO). Um dos maiores desafios do *matching* de grandes ontologias é o elevado tempo de execução e o excessivo consumo de recursos de computacionais. Assim, para melhorar a eficiência, técnicas de particionamento de ontologias e paralelismo podem ser empregadas no processo de MO. Este trabalho apresenta uma abordagem para o *Matching* de Ontologias baseado em Particionamento e Paralelismo (MOPP) que particiona as ontologias de entrada em subontologias e executa as comparações entre conceitos em paralelo, usando o *framework* MapReduce como solução programável. Embora as técnicas de paralelização possam melhorar a eficiência do processo de MO, essas técnicas apresentam problemas referentes ao desbalanceamento de carga. Por essa razão, o presente trabalho propõe ainda duas técnicas para balanceamento de carga (básica e refinada) para serem aplicadas junto à abordagem MOPP, a fim de orientar a distribuição uniforme das comparações (carga de trabalho) entre os nós de uma infraestrutura computacional. O desempenho da abordagem proposta é avaliado em diferentes cenários (diferentes tamanhos de ontologias e graus de desbalanceamento de carga) utilizando uma infraestrutura computacional e ontologias reais e sintéticas. Os resultados experimentais indicam que a abordagem MOPP é escalável e capaz de reduzir o tempo de execução do processo de MO. No que diz respeito às técnicas de balanceamento de carga, os resultados obtidos mostram que a abordagem MOPP é robusta, mesmo em cenários com elevado grau de desbalanceamento de carga, com a utilização da técnica refinada de balanceamento de carga.

Palavras-chave: *Matching* de Ontologias, Grandes Ontologias, Computação Distribuída, Computação Paralela, Balanceamento de Carga, MapReduce.

Abstract

Currently, the use of large ontologies in various areas of knowledge is increasing. Since, these ontologies can present contents overlap, the identification of correspondences among their concepts is necessary. This process is called Ontologies Matching (OM). One of the major challenges of the large ontologies matching is the high execution time and the computational resources consumption. Therefore, to get the efficiency better, partition and parallel techniques can be employed in the MO process. This work presents a Partition-Parallel-based Ontology Matching (PPOM) approach which partitions the input ontologies in sub-ontologies and executes the comparisons between concepts in parallel, using the framework MapReduce as a programmable solution. Although the parallel techniques can get the MO efficiency process better, these techniques present problems concerning to the load imbalancing. For that reason, our work has proposed two techniques to the load balancing - the basic and the fine-grained one – which are supposed to be applied together with the PPOM approach, in order to orientate the uniform distribution of the comparisons (workload) between the nodes of a computing infrastructure. The performance of the proposed approach is assessed in different settings (different sizes of ontologies and degrees of load imbalancing) using a computing infrastructure and real and synthetic ontologies. The experimental results have indicated that the PPOM approach is scalable and able to reduce the OM process execution time. Referring to the load balancing techniques, the obtained results have shown that the PPOM approach is robust, even in settings with a high load imbalancing, with the fine-grained load balancing technique.

Keywords: Ontology Matching, Large Ontologies, Distributed Computing, Parallel Computing, Load Balancing, MapReduce.

Agradecimentos

Quero agradecer, em primeiro lugar, a Deus por sua infinita misericórdia e fidelidade nos momentos de dificuldade. Agradeço ao Senhor por tantas bênçãos alcançadas e pelos dons do Espírito, que foram fundamentais para a realização do mestrado. Abaixo, os demais agradecimentos:

- a família por sempre estar ao meu lado, especialmente minha mãe que em meio às dificuldades sempre prezou pela minha educação;
- a Polyanna, minha namorada, por todo cuidado e carinho nessa etapa da minha vida;
- ao meu orientador professor Dr. Carlos Eduardo Pires pela imensa dedicação e presteza durante todas as etapas do mestrado;
- aos meus amigos, que sempre vieram em meu auxílio nos momentos de comemoração e de dificuldades;
- aos professores e funcionários da COPIN;
- a Universidade Federal de Campina Grande.

Lista de Símbolos

DFS - *Distributed File System*

EM - *Entity Matching*

GO - *Gene Ontology*

IVOA - *International Virtual Observatory Alliance*

LOD - *Linked Open Data*

MBC - *Matriz de Blocos de Comparação*

MDB - *Matriz de Distribuição de Blocos*

MO - *Matching de Ontologias*

MOPP - *Matching de Ontologias baseado em Particionamento e Paralelismo*

MR - *MapReduce*

OAEI - *Ontology Alignment Evaluation Initiative*

OBO - *Open Biomedical Ontologies*

OWL - *Web Ontology Language*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

Atualmente, existe uma grande quantidade de informações sobre diferentes domínios do conhecimento como, por exemplo, saúde, agricultura e astronomia. Uma maneira de descrever o conhecimento de um determinado domínio é por meio da utilização de ontologias [11; 28; 6]. Em uma ontologia, as informações são representadas como um conjunto de conceitos, instâncias e relações entre conceitos [13].

Como as ontologias podem apresentar sobreposição de conteúdo, a identificação (semi-) automática de similaridades entre os conceitos torna-se necessária. O processo que determina os pares de conceitos semelhantes (correspondências) entre ontologias é chamado de Matching de Ontologias (MO) [11; 29; 1; 5], ilustrado na Figura 1.1.

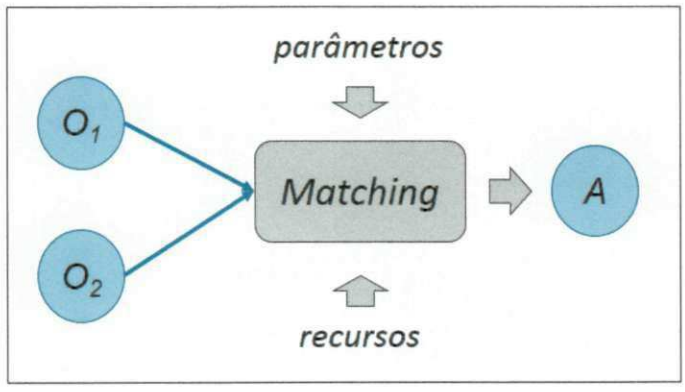

Figura 1.1: Processo de *matching* de ontologias, adaptado de [11].

No processo de MO tradicional, duas ontologias O_1 e O_2 (que normalmente modelam domínios semelhantes) são recebidas como entrada e, para determinar a similaridade entre

os conceitos, são realizadas comparações entre pares de conceitos guiadas pelo produto Cartesiano, ou seja, todos os conceitos da ontologia O_1 são comparados com todos os conceitos de O_2. Parâmetros (por exemplo, pesos, métricas e/ou limiares) e recursos (por exemplo, dicionários ou outras ontologias) podem ser fornecidos ao processo, a fim de ajudar na comparação dos conceitos.

Para calcular a similaridade entre conceitos, múltiplos algoritmos de *matching* (*matchers*) são aplicados [9; 27]. Cada *matcher* gera um valor de similaridade que varia entre 0 (nenhuma semelhança) e 1 (total similaridade). Medidas de agregação (por exemplo, média e média ponderada) são usadas para combinar os valores de similaridade. O valor de similaridade resultante é comparado com um limiar de similaridade para determinar as correspondências que formarão o alinhamento (A) entre as ontologias O_1 e O_2.

Em geral, cada *matcher* explora uma determinada propriedade dos conceitos envolvidos, por exemplo, rótulo e relação *is-a*. Como resultado, para cada par de conceitos, múltiplas comparações são realizadas. Além disso, um único *matcher* pode realizar várias comparações em relação a um mesmo par de conceitos como, por exemplo, um *matcher* que explora a propriedade relação *is-a* por meio da comparação dos subconceitos do par em questão.

Para um melhor entendimento, é utilizada como exemplo a comparação do par de conceitos *Bone Organ* ⋈ *Bone*, ilustrada na Figura 1.2. Para um *matcher* que explora a propriedade rótulo, serão comparados apenas os rótulos "*Bone Organ*" e "*Bone*". Já para um *matcher* que explora a propriedade relação *is-a* dos conceitos envolvidos, serão realizadas 6 comparações, uma vez que serão comparados os rótulos "*Flat Bone*" e "*Short*" (subconceitos de "*Bone Organ*") com os rótulos "*Skull*", "*Flat Bone*" e "*Phalanx*" (subconceitos de "*Bone*").

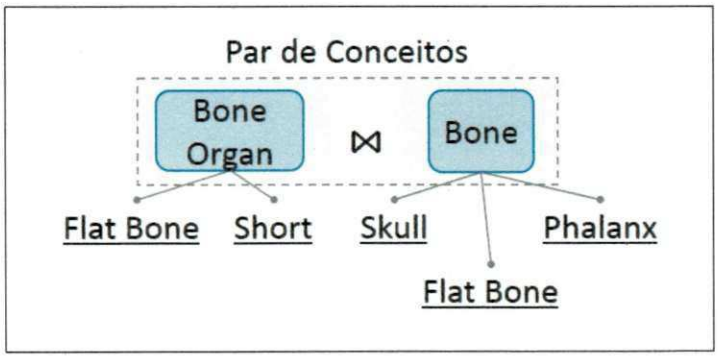

Figura 1.2: Par de conceitos a ser comparado: múltiplas comparações.

O processo de MO é empregado em diferentes áreas como, por exemplo, integração de dados ou esquemas de bancos de dados, ligação de dados e consultas eletrônicas na web [11]. O aumento no tamanho das ontologias tem proporcionado vários desafios relacionados ao tempo de execução do processo e à qualidade dos alinhamentos gerados [12]. Desafios que envolvem diferentes aspectos do MO (por exemplo, desempenho e qualidade) motivaram a criação de competições tais como a organizada pela *Ontology Alignment Evaluation Initiative* (OAEI[1]). Esta competição reúne empresas e universidades de todo o mundo, a fim de obter soluções mais rápidas e eficazes para o processo de MO.

1.1 Motivação

Muitos trabalhos [11; 2; 29; 12] abordam os desafios envolvendo o processo de MO. Dentre esses trabalhos, podem ser destacados: *matching* de ontologias em larga em escala; *matching* de ontologias com base em conhecimento externo (por exemplo, outras ontologias, *thesaurus*, dicionário de dados); seleção, combinação e *tuning* de *matchers*; *matching* social e colaborativo (pessoas podem julgar a veracidade das correspondências); armazenamento e suporte aos alinhamentos (possibilitar o armazenamento, compartilhamento e extração de informações contidas nos alinhamentos).

Segundo Euzenat [11], a eficiência dos *matchers* é de primordial importância no contexto de MO, especialmente quando existem restrições de tempo para execução do processo ou a quantidade de memória dos recursos computacionais disponíveis é limitada. Atualmente, a maioria das abordagens para MO são executadas utilizando apenas um computador (ou máquina virtual). De uma forma geral, essas abordagens não utilizam nenhuma estratégia que melhore a eficiência do processo de MO. Esse fato evidencia que otimizações relacionadas à escalabilidade e eficiência são desafios promissores no contexto de MO.

Ontologias que modelam domínios com uma grande quantidade de informações tendem a conter milhares de conceitos e, assim, são consideradas grandes ontologias (mais de 10.000 conceitos) [29]. Ao lidar com grandes ontologias, o processo de MO tradicional (baseado no produto Cartesiano) torna-se lento e demanda um alto consumo de memória [34]. Nesse contexto, o processo pode demorar várias horas ou dias para ser concluído, além de, em

[1] http://oaei.ontologymatching.org/2015/

alguns casos, ultrapassar o limite de memória disponível (*"out of memory"*). Por esse motivo, técnicas de otimização tornam-se necessárias, principalmente com o intuito de reduzir o espaço de busca e/ou executar o processo de MO em paralelo.

Técnicas para particionamento de ontologias [20; 38; 15] visam reduzir o espaço de busca do *matching* a partir da divisão de cada ontologia de entrada em subontologias (partições) e comparar apenas os conceitos contidos em pares de subontologias similares. Essas técnicas evitam realizar comparações desnecessárias, ou seja, comparações entre conceitos claramente distintos e que, consequentemente, possuem poucas chances de resultarem em correspondências. Dessa forma, com a redução na quantidade de conceitos comparados, o processo de MO torna-se mais rápido. A desvantagem dessas técnicas é a possibilidade de redução na qualidade do alinhamento gerado, visto que pares de conceitos verdadeiramente correspondentes (similares) podem não serem comparados.

Uma vez que o *matching* de grandes ontologias pode ser considerado um processo intensivo de dados com elevado consumo de tempo e recursos, abordagens de MO em paralelo [12; 2; 35; 40; 41] surgem como uma solução para reduzir o tempo de execução e otimizar o consumo de recursos. Essas abordagens visam distribuir as comparações dos conceitos entre os vários recursos (por exemplo, computadores ou máquinas virtuais) de uma infraestrutura computacional.

As abordagens de OM em paralelo também enfrentam desafios, dentre eles, pode-se destacar o desbalanceamento de carga (comparações) entre os recursos e a elevada taxa de replicação dos conceitos. O desbalanceamento de carga ocorre quando alguns nós da infraestrutura executam uma quantidade de comparações superior aos demais nós. Consequentemente, os nós com mais comparações executam por um longo tempo, enquanto os demais nós permanecem ociosos após concluírem as suas comparações. Este problema influencia diretamente o desempenho processo de MO, uma vez que o nó mais lento determina o tempo total de execução do processo. No que diz respeito à taxa de replicação dos conceitos, uma grande quantidade de conceitos replicados sendo transmitidos entre os nós da infraestrutura tende a reduzir o desempenho da abordagem [24].

1.2 Justificativa

Como mencionado anteriormente, a execução do processo de MO pode demandar um elevado consumo de tempo e recursos, sobretudo no contexto de grandes ontologias. Em um cenário onde é realizado o MO tradicional (baseado no produto Cartesiano), a quantidade de comparações entre conceitos cresce proporcionalmente ao tamanho das ontologias envolvidas. Por exemplo, se duas ontologias com 10.000 e 20.000 conceitos forem submetidas ao processo de MO, 200 milhões de pares de conceitos serão comparados. Levando em consideração que cada par de conceitos pode executar múltiplas comparações (dependendo do *matcher*), a quantidade de comparações executadas em um processo de MO pode chegar na ordem de bilhões.

Na trilha de grandes ontologias (*Large Bio*) da competição OAEI, em 2015, apenas oito (LogMapLt, RSDLWB, AML, XMAP, XMAP-BK, LogMap, LogMapC e LogMapBio) dos vinte e dois algoritmos que concorreram nessa trilha conseguiram concluir todos os casos de teste. Os algoritmos foram executados no modo *standalone* em todos os casos de teste. A maioria dos algoritmos ultrapassou o limite de memória disponível ou excedeu o limite de tempo da competição [7]. Dessa forma, técnicas de particionamento e paralelização do processo de MO surgem como possíveis soluções para prover eficiência ao processo.

Embora as técnicas para paralelizar o processo de MO possam melhorar o desempenho do processo, essas técnicas apresentam problemas referentes ao desbalanceamento de carga e a elevada taxa de replicação dos conceitos. No desbalanceamento de carga, um único nó pode concentrar uma grande parte da carga de trabalho (comparações) do processo, enquanto que a alta taxa de replicação de conceitos aumenta o volume de dados transmitidos entre os nós. Ambos os cenários elevam o tempo de execução do processo. Por isso, os ganhos referentes ao desempenho das abordagens de MO em paralelo podem ser seriamente comprometidos em virtude desses problemas. Particularmente, o desbalanceamento de carga pode ocorrer devido a dois principais fatores: distribuição desigual dos pares de conceitos ou das comparações (executadas nos pares de conceitos) entre os nós da infraestrutura. É importante destacar que dentre todas as abordagens estado da arte de MO em paralelo [12; 35; 2; 40] analisadas neste trabalho, nenhuma aplica técnicas de balanceamento de carga.

Claramente, as técnicas de particionamento e paralelização são fundamentais para prover

desempenho ao processo de MO. Porém, essas técnicas são comumente utilizadas separadamente. Neste trabalho é proposta uma abordagem que combina as duas técnicas com o intuito de obter melhores resultados, no que diz respeito ao desempenho e qualidade.

1.3 Objetivos

Nessa seção, serão abordados os objetivos gerais e específicos deste trabalho:

- Objetivo geral

  O objetivo geral deste trabalho é propor uma abordagem para o Matching de Ontologias baseado em Particionamento e Paralelismo (MOPP). Particularmente, a abordagem distribuída, pretende paralelizar a comparação dos pares de conceitos no sentido de reduzir o tempo de execução do processo como um todo.

- Objetivos Específicos

  Tratar o problema de desbalanceamento de carga e elevada taxa de replicação dos conceitos, relacionados à execução do processo de MO em paralelo, sobretudo com a utilização de MR.

  Avaliar a abordagem proposta (MOPP) no que diz respeito ao desempenho do processo e à qualidade dos alinhamentos gerados.

  Avaliar a utilização de técnicas de particionamento de ontologias no sentido de identificar possíveis contribuições que possam, juntamente com as técnicas de paralelização, reduzir o tempo total do processo de MO no contexto de grandes ontologias.

1.4 Principais Contribuições

Em resumo, as principais contribuições deste trabalho são:

- Uma abordagem para o processo de *matching* de ontologias baseada em particionamento e paralelismo com balanceamento de carga. A partir do particionamento das ontologias é possível reduzir a quantidade de comparações a serem realizadas e consequentemente reduzir o tempo de execução do processo. Com o objetivo de melhorar

ainda mais a eficiência do MO, as comparações são executadas em paralelo pelos nós de uma infraestrutura computacional. Para paralelizar, MapReduce foi utilizado como solução programável.

- Foram propostas a utilização de duas técnicas de balanceamento de carga: básica e refinada (com baixo nível de granularidade). A primeira técnica baseia-se na quantidade de pares de conceitos contidos em cada par de subontologias para guiar a distribuição dos pares de conceitos entre os nós [4; 3]. A segunda técnica é mais robusta e refinada, uma vez que ela analisa a quantidade de comparações realizadas em cada par de conceitos. Com base na quantidade de comparações, a técnica refinada de balanceamento distribui uniformemente os pares de conceitos entre os nós, de forma que todos os nós executam uma quantidade semelhante de comparações.

- Análises experimentais envolvendo eficiência do processo de MO e a qualidade dos alinhamentos gerados. A abordagem proposta neste trabalho foi comparada com uma abordagem estado da arte. Além disso, foi analisada a aplicação das duas técnicas de balanceamento de carga propostas neste trabalho na abordagem MOPP. Foram criados diferentes cenários experimentais com o objetivo de analisar os desafios referentes ao desempenho do processo de MO.

1.5 Avaliação e Principais Resultados

Primeiramente, são realizados experimentos que analisam comparativamente o desempenho da abordagem MOPP (proposta neste trabalho) utilizando a técnica básica de balanceamento de carga, a mesma abordagem MOPP utilizando a técnica refinada de balanceamento de carga e uma abordagem estado da arte proposta por Gross [12]. Neste experimento são analisadas duas métricas: tempo de execução e *speed up*. A métrica *speed up* mensura o quanto mais rápida a abordagem em paralelo é quando comparada com a abordagem sendo executada de forma sequencial (utilizando apenas um nó/core). A abordagem MOPP utilizando a técnica refinada de balanceamento apresenta melhor desempenho com relação as demais abordagens em ambas às métricas, chegando a obter um tempo de execução até 24% inferior em relação à abordagem de Gross e 14% em relação à abordagem com a técnica básica de

balanceamento de carga.

Em seguida, foi conduzido um estudo com ontologias sintéticas com o intuito de avaliar comparativamente as duas técnicas (básica e refinada) de balanceamento de carga empregadas no MOPP. As ontologias foram criadas sinteticamente para que possibilitassem manipular a quantidade de propriedades de cada conceito. Como a quantidade de comparações realizadas para um par de conceitos é dada pela quantidade de propriedades contidas em cada conceito do par, a partir das ontologias sintéticas foi possível criar diferentes cenários de desequilíbrio de comparações entre os pares de conceitos. A abordagem MOPP utilizando a técnica refinada de balanceamento de carga obteve melhores tempos de execução e *speed up*, chegando a ser 87% mais rápida do que a abordagem MOPP com a técnica básica de balanceamento de carga.

Foi realizado também um experimento avaliando o impacto da utilização (ou não) de técnicas de particionamento de ontologias como uma etapa de pré-processamento do MO. Nesse experimento, foram analisados o tempo de execução e a qualidade do alinhamento para o processo de MO em paralelo com e sem o particionamento de ontologias. Com a utilização do particionamento de ontologias o tempo de execução chegou a ser metade do tempo gasto pelo processo sem particionamento das ontologias.

Um outro experimento foi realizado com o intuito de avaliar o impacto das duas técnicas (básica e refinada) de balanceamento de carga na taxa de replicação de conceitos da abordagem MOPP. Para realizar essa avaliação, foi calculada a quantidade de conceitos replicados na abordagem MOPP com a técnica básica e na abordagem MOPP com a técnica refinada. A partir dos resultados obtidos, foi possível perceber que a abordagem MOPP com a técnica refinada reduziu significativamente (em até 50%) a quantidade de conceitos replicados em relação à abordagem MOPP com a técnica básica.

Por último, foram realizados experimentos envolvendo ontologias consideradas muito grandes com a aplicação de múltiplos *matchers* no processo de MO em paralelo. Para tal, foram utilizadas ontologias com mais de 60 mil conceitos. No experimento, a utilização de apenas um *matcher* no processo de MO foi comparada com a combinação de dois *matchers*. As métricas de tempo de execução e qualidade do alinhamento gerado foram utilizadas para analisar o impacto da combinação de vários *matchers*. A partir dos resultados coletados, foi possível observar que, aumentando o número de nós disponíveis na infraestrutura, a diferença

entre os tempos de execução do MO com apenas um *matcher* e combinando dois *matchers* diminuiu razoavelmente. A combinação dos *matchers* obteve um alinhamento com qualidade 6% superior à utilização de apenas um *matcher* no processo de MO.

1.6 Estrutura da Dissertação

No Capítulo 2, é apresentada a fundamentação teórica necessária para o entendimento do trabalho. No Capítulo 3, são descritos, comentados e comparados os trabalhos relacionados ao trabalho proposto. A abordagem para o processo de *matching* de ontologias baseada em particionamento e paralelismo com balanceamento (básico e refinado) de carga é apresentada no Capítulo 4. O Capítulo 5 descreve e discute a avaliação da abordagem. As principais conclusões do trabalho, além de perspectivas para trabalhos futuros são apresentadas no Capítulo 6.

Capítulo 2

Fundamentação Teórica

Nesse capítulo, é apresentada a fundamentação teórica necessária para o entendimento do trabalho desenvolvido. Os tópicos necessários para uma melhor compreensão acerca deste trabalho estão organizados neste capítulo em quatro seções. A seção 2.1 fornece uma visão geral sobre ontologias. A seção 2.2 descreve os *matchers* linguísticos utilizados neste trabalho. A seção 2.3 apresenta o modelo de programação MapReduce utilizado na computação distribuída como solução programável. Na seção 2.4 são apresentadas as duas técnicas mais comumente empregadas para melhorar a eficiência do processo de MO: particionamento de ontologias e paralelização do processo de MO. Na seção 2.5 são descritos os dois principais métodos utilizados pelas técnicas de balanceamento de carga: estático e dinâmico.

2.1 Ontologias

Ontologias são utilizadas como forma de modelar e organizar informações referentes a um determinado domínio do conhecimento [11; 13]. A quantidade de ontologias vem crescendo e ganhando destaque em vários domínios do conhecimento [31], tais como Web Semântica [39], Saúde (*Open Biomedical Ontologies Foundry* - OBO[1], *Gene Ontology* - GO[2] e *OpenGalen*[3]), Astronomia (IVOA[4]) e Agricultura (AGROVOC[5]).

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

[2] http://geneontology.org/

[3] http://www.opengalen.org/index.html

[4] http://www.ivoa.net/

[5] http://aims.fao.org/vest-registry/vocabularies/agrovoc-multilingual-agricultural-thesaurus

Para modelar o conhecimento em uma ontologia, a informação é representada como um conjunto de conceitos, instâncias e relações entre conceitos. Um conceito define um grupo de indivíduos que compartilham as mesmas propriedades, considerando estas como relações binárias utilizadas para definir relacionamentos entre indivíduos ou entre indivíduos e valores de dados (por exemplo, *Livro* é um conceito de uma ontologia e título, preço e comentários são propriedades do conceito *Livro*). Instâncias são as representações específicas de um conceito (por exemplo, "A Bíblia" e "O Pequeno Príncipe" são instâncias do conceito do *Livro*). As relações especificam em que sentido um conceito está relacionado com outros conceitos na ontologia (por exemplo, manual, catálogo e enciclopédia são subtipos (relações *is-a*) do conceito *Livro*).

Ontologias que modelam domínios que possuem uma grande quantidade de informações tendem a possuir uma grande quantidade de conceitos. Essas ontologias geralmente possuem dezenas de milhares de conceitos e, por isso, são consideradas grandes ontologias [29]. As grandes ontologias vêm sendo utilizadas cada dia mais e muitas delas se tornaram de suma importância em seus domínios, como é o caso das grandes ontologias OBO, GO, OpenGalen, IVOA e AGROVOC, citadas anteriormente. De maneira particular, a utilização de ontologias como forma de modelar domínios da biomedicina vem crescendo consideravelmente nos últimos anos [11]. Podemos destacar o projeto BioPortal[6], que oferece um repositório com mais de 270 ontologias biomédicas, no qual dezenas dessas ontologias são consideradas grandes ontologias. No contexto de repositórios de ontologias, é comum a necessidade de realizar continuamente o processo de MO, com o intuito de identificar a sobreposição de conteúdo existente entre as ontologias contidas nos repositórios e as novas ontologias que são inseridas no repositório. Quando grandes ontologias são submetidas ao processo de MO, uma enorme quantidade de comparações entre conceitos é realizada, consequentemente, o processo torna-se lento e requer uma quantidade significativa de recursos [32; 16]. Por essa razão, técnicas de otimização são aplicadas ao processo de MO com o objetivo de prover desempenho ao processo.

[6]http://bioportal.bioontology.org/

2.2 Matchers Linguísticos

Os *matchers* linguísticos são os mais utilizados no processo de MO [27]. Neste trabalho, com o intuito de identificar a similaridade entre os conceitos, foram utilizados os seguintes *matchers* linguísticos: *Label*, *Children*, *NamePath* e *Annotation*.

Label. Para cada par de conceitos, esse *matcher* compara linguisticamente os rótulos dos conceitos, por meio de algoritmos de similaridade linguística[7] (por exemplo, distância de *Levenshtein*, Trigrama). Como resultado, o valor de similaridade do par de conceitos é dado pela similaridade linguística entre os rótulos dos conceitos.

Children. Nesse *matcher* são exploradas as relações *is-a* (subconceitos) dos conceitos. Para cada par de conceitos, o *matcher Children* compara linguisticamente todos rótulos dos subconceitos de um conceito com todos os rótulos dos subconceitos do outro conceito, utilizando um algoritmo de similaridade linguística. O valor final de similaridade determinado pelo *matcher* é dado pela média dos valores de similaridade entre os subconceitos.

NamePath. Inicialmente, para cada conceito da ontologia, é gerada uma *string* contendo o caminho hierárquico do conceito. Para gerar o caminho hierárquico de um determinado conceito, são identificados os rótulos dos conceitos antecessores (conceitos pai) do conceito em questão até o conceito raiz. Os rótulos dos conceitos antecessores são concatenados para formar a *string* que representa o caminho hierárquico do conceito em questão. O *matcher NamePath* compara linguisticamente as *strings* contendo o caminho hierárquico dos conceitos, por meio de algoritmos de similaridade linguística, determinado o valor de similaridade entre os conceitos.

Annotation. Nesse *matcher* são exploradas as propriedades anotações dos conceitos. Comumente, as anotações de um conceito representam sinônimos linguísticos do rótulo do conceito. Para cada par de conceitos, o *matcher Annotation* compara linguisticamente todas as anotações de um conceito com todas as anotações do outro conceito, utilizando um algoritmo de similaridade linguística. O valor final de similaridade determinado pelo *matcher Annotation* é dado pelo maior valor de similaridade entre as anotações.

[7] https://sourceforge.net/projects/simmetrics/

2.3 MapReduce

MapReduce é um modelo de processamento (*framework*) distribuído de dados de fácil escalonamento, cujas primitivas de processamento de dados são divididas nas fases de *map* e *reduce*. Na fase de *map*, as funções (tarefas) de *map* transformam registros de entrada em registros temporários, que não são necessariamente do mesmo tipo de dados da entrada. Nas funções de *map* é realizado também o mapeamento das entradas com suas respectivas chaves, com o objetivo de formar um conjunto de pares (chave, valor) de um registro temporário. Na fase de *reduce*, as funções de *reduce* são responsáveis por reduzir um conjunto de valores temporários, que partilham a mesma chave, no resultado final do processo. A fase de *reduce* possui as seguintes etapas:

- *Sort*: esta é a etapa inicial da fase de *reduce*, que tem como entrada as saídas das funções de *map*. O *framework* agrupa as entradas da etapa de *reduce* pelas chaves, ordenando-as;
- *Shuffle*: nesta etapa, o *framework* analisa a parte mais relevante da chave das saídas de todas as funções de *map*;
- *Reduce*: baseado nos pares (chave, valor) gerados pelas funções de *map*, a fase de *reduce* agrupa quais dados (valores) serão processados em uma mesma função de *reduce*, agrupando os dados pela chave.

No MapReduce, os dados são inicialmente particionados entre os recursos disponíveis da infraestrutura e armazenado no sistema de arquivos distribuídos (*Distributed File System* - DFS). Os dados são representados como pares (chave, valor) e os cálculos são expressos usando duas funções principais:

- map : $(chave_{entrada}; valor_{entrada}) \rightarrow lista(chave_{tmp}; valor_{tmp})$
- reduce : $(chave_{tmp}; lista(valor_{tmp})) \rightarrow lista(chave_{saida}; valor_{saida})$

Toda execução envolvendo as fases de *map* e *reduce*, no *framework* MapReduce, é chamada de *job*. Em cada *job*, as funções de *map* são aplicadas em paralelo em diferentes partições dos dados de entrada. Os pares (chave, valor) resultantes das funções de *map* são

particionados pela chave. Em cada tarefa de *reduce*, todas as partições recebidas são mescladas em uma ordem de classificação (*sort*), com base nas chaves. Para cada partição, os pares são classificados segundo a sua chave e, em seguida, submetidos à etapa de *shuffle* (na fase de *reduce*). Todos os valores de pares que compartilham uma determinada chave são enviados para uma única função de *reduce*. A saída de cada função de reduce é gravada em um arquivo distribuído (DFS).

Além das funções de *map* e *reduce*, a estrutura também permite que o usuário forneça uma função *combine*, que é executada como mapeadores internos, logo após a conclusão da fase de *map*. Esta função atua como um redutor local, operando com os pares (chave, valor) locais, permitindo a diminuição da quantidade de dados transmitidos na rede. Por fim, o MapReduce também permite que o usuário configure (programe) as funções *map* e *reduce* de forma que atenda as restrições e o fluxo do processo que está sendo paralelizado, além de personalizar a forma como são feitos o particionamento e a classificação das chaves dos pares (chave, valor) [37].

A Figura 2.1 ilustra um *job* MapReduce que realiza o processo de MO. Neste exemplo, não foi aplicada nenhuma técnica que minimize o problema de desbalanceamento de carga. Inicialmente, o *job* recebe como entrada os pares de subontologias (Par 0 e Par 1) gerados pelo algoritmo de particionamento de ontologias. Os conceitos contidos nos pares de subontologias são lidos e encaminhados para as funções de *map*, onde são determinados os pares (chave, valor) dos conceitos. No exemplo, a chave de cada conceito é formada por dois números, de maneira que os números representam o índice do par de subontologias e o índice da subontologia no par de subontologias, respectivamente. O valor associado a cada par (chave, valor) é representado pelo próprio conceito. Por exemplo, o conceito *Bone* recebe a chave "0.1", onde "0" indica que o conceito pertence ao Par 0 e "1" indica que o conceito pertence a segunda subontologia do par.

Na fase de *reduce*, os pares (chave, valor) são classificados em ordem crescente (*sort*) e particionados, baseados na chave de cada conceito. O particionamento é realizado de modo que os pares de conceitos sejam distribuídos entre as tarefas de *reduce* disponíveis. O particionamento é baseado nos pares de subontologias, ou seja, todos os conceitos de um par de subontologias são enviados para uma única tarefa de *reduce*. Os blocos são alocados (*shuffle*) de acordo com o primeiro número da chave (identificador do par de subontologias),

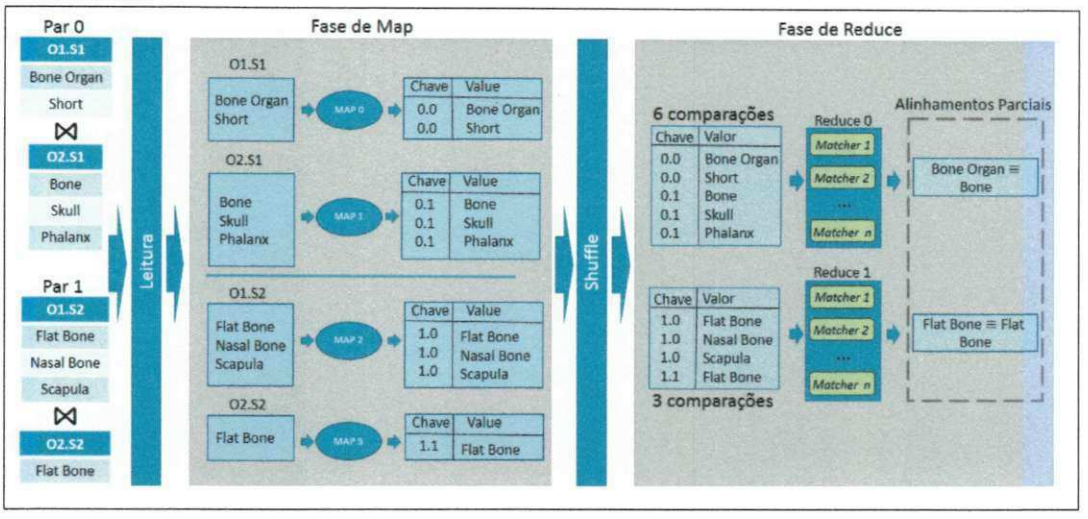

Figura 2.1: Exemplo de *job* MapReduce para o processo de MO.

de maneira que todos os conceitos que possuem o mesmo primeiro número da chave sejam enviados para mesma tarefa de *reduce*. Por essa razão, os conceitos *Bone Organ, Short, Bone, Skull* e *Phalanx* são enviados para a tarefa de *reduce* 0. Após a alocação, os conceitos pertencentes a primeira subontologia (segundo número da chave igual a 0) são comparados com todos os conceitos da segunda subontologia (segundo número da chave igual a 1). No exemplo, é aplicado um *matcher* que explora os rótulos dos conceitos. Dessa forma, os rótulos dos conceitos *Bone Organ* e *Short* são comparados com os rótulos conceitos *Bone, Skull* e *Phalanx*, totalizando seis comparações. Após a conclusão das comparações entre conceitos nas funções de *reduce*, cada função gera um alinhamento parcial como saída. É importante destacar que, no exemplo, existe um desbalanceamento na quantidade de pares de conceitos comparados nas tarefas de *reduce*. Enquanto a tarefa de *reduce* 0 executa seis comparações, a tarefa de *reduce* 1 executa apenas três comparações.

2.4 Técnicas de Otimização para o Matching de Ontologias

Nesta seção são descritas as duas técnicas mais comumente utilizadas no processo de MO, com o intuito melhorar o desempenho do processo.

2.4.1 Particionamento de Ontologias

Técnicas de particionamento de ontologias são aplicadas à abordagens de MO com o objetivo de reduzir a quantidade de comparações a serem realizadas e, consequentemente, o tempo de execução do processo de MO [19; 33; 38]. Nas abordagens de MO baseadas em particionamento, cada ontologia de entrada O_1 e O_2 é individualmente particionada e os conceitos são divididos em subontologias (partições), de forma que não existe sobreposição de conceitos entre as subontologias. As subontologias de O_1 e O_2 que possuem um certo grau de similaridade são pareadas para serem comparadas. Para cada par de subontologias a ser comparado, todos os conceitos de uma subontologia são comparados com todos os conceitos da outra subontologia, ou seja, a comparação é guiada pelo produto Cartesiano [29].

Geralmente, as técnicas de particionamento de ontologias possuem as etapas ilustradas na Figura 2.2:

- Leitura dos arquivos: os arquivos das ontologias (por exemplo, no formato OWL) são lidos e o conteúdo (por exemplo, conceitos, relacionamentos e valores) de cada ontologia é carregado na memória;

- Particionamento das ontologias: cada ontologia de entrada produz um conjunto de subontologias, de maneira que os conceitos de uma ontologia de entrada que são similares, de acordo com uma determinada característica (por exemplo, linguística [8], estrutural [18] ou semântica [27]), são agrupados em uma mesma subontologia;

- Identificação das subontologias semelhantes: identifica quais os pares de subontologias possuem conceitos com um certo grau de similaridade, de maneira que os pares de subontologias são formados por uma subontologia originada de O_1 e uma subontologia originada de O_2. Essa etapa tem o objetivo de evitar que todas as subontologias de O_1 sejam comparadas com todas as subontologias de O_2;

- *Matching* de subontologias: realiza as comparações entre os conceitos (guiadas pelo produto Cartesiano) de cada par de subontologias identificadas como semelhantes (definidos na etapa anterior). A similaridade entre os conceitos é determinada pelos *matchers*. Como resultado, cada par de subontologias comparado gera um alinhamento parcial;

- Combinação e/ou refinamento dos alinhamentos parciais: combina os alinhamentos parciais (por exemplo, união das correspondências) para produzir o alinhamento final. Opcionalmente, os alinhamentos parciais também podem ser refinados (etapa de pós-processamento) com o intuito de remover inconsistências e correspondências duplicadas.

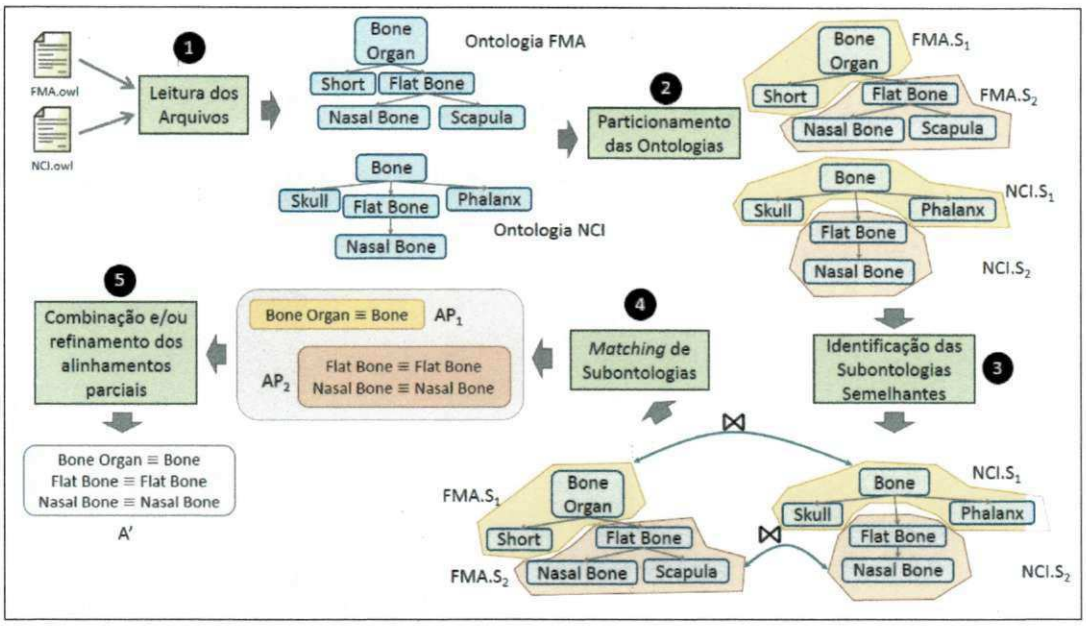

Figura 2.2: Etapas do processo de MO baseado em particionamento [4].

Na literatura, é possível destacar os seguintes algoritmos de particionamento de grandes ontologias: PBM (*Partition-based block matching*) [20], APP (*Anchor, Partition, Partition*) e PAP (*Partition, Anchor, Partition*) [15]. Neste trabalho, foi aplicado o algoritmo PAP para gerar as entradas (pares de subontologias a serem comparados) da abordagem MOPP, uma vez que esse algoritmo obteve melhores resultados quando comparado com os algoritmos APP e PBM, conforme descrito nos experimentos realizados por [15].

Inicialmente, o algoritmo PAP particiona uma das ontologias com base na similaridade linguística dos rótulos dos conceitos. Dessa forma, os conceitos com um certo grau de similaridade são agrupados em uma mesma subontologia. São comparados linguisticamente apenas os rótulos dos conceitos, com o intuito de evitar que o particionamento tenha um elevado tempo de execução. Após o particionamento de uma das ontologias de entrada, são determinados os conceitos âncoras entre as ontologias (aqueles considerados similares em

ambas as ontologias). Os conceitos âncoras também são determinados com base na similaridade linguística dos rótulos dos conceitos. Os conceitos âncoras, selecionados entre as duas ontologias, são usados como guias para o particionamento da outra ontologia e definem quais os pares de subontologias devem ser comparados.

No exemplo, ilustrado na Figura 2.3, os conceito *Bone Organ* e *Short* são agrupados em uma mesma subontologia ($FMA.S_1$). Posteriormente, os pares de conceitos (*Bone Organ*, *Bone*), (*Flat Bone*, *Flat Bone*) e (*Nasal Bone*, *Nasal Bone*) são determinados como conceitos âncora. Influenciados pelos conceitos âncoras, os conceitos *Bone*, *Skull* e *Phalanx* são agrupados em uma mesma subontologia ($NCI.S_1$). Por fim, como os conceitos *Bone Organ* e *Bone* são considerados conceitos âncora (por similaridade linguística dos rótulos), as subontologias $FMA.S_1$ e $NCI.S_1$ são determinadas como um par de subontologias a serem comparadas.

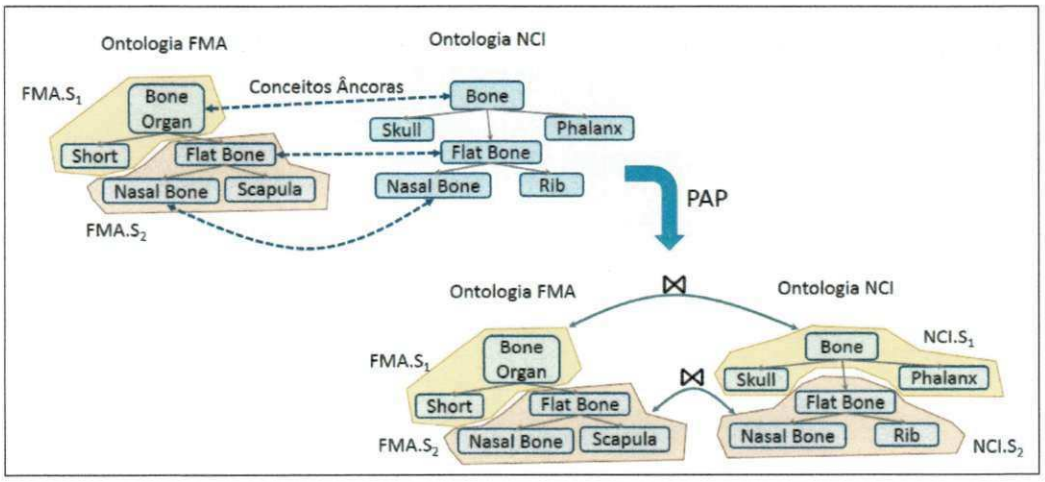

Figura 2.3: Ilustração do funcionamento do algoritmo PAP.

2.4.2 Paralelização do Processo de Matching de Ontologias

Existem basicamente duas maneiras de executar o processo de MO em paralelo, mais precisamente a etapa de *matching* de subontologias (etapa 4 na Figura 2.2). As duas maneiras estão relacionadas ao modo como os *matchers* são executados [6]: *inter-matcher* e *intra-matcher*. No modo *inter-matcher* (ilustrado na Figura 2.4), cada *matcher* (distinto) é executado em um nó (por exemplo, computador ou máquina virtual) de uma infraestrutura computacio-

nal distribuída (por exemplo, uma *cloud* ou *cluster* de máquinas), de maneira que todas as comparações entre os conceitos são enviadas para cada nó, onde o *matcher* (do nó) executa as comparações. No modo *intra-matcher* (ilustrado na Figura 2.5), todos os *matchers* são executados em cada nó. As comparações entre conceitos são distribuídas entre os nós disponíveis e todos os *matchers* realizam as comparações sequencialmente, ou seja, os nós executam paralelamente os *matchers* de forma sequencial.

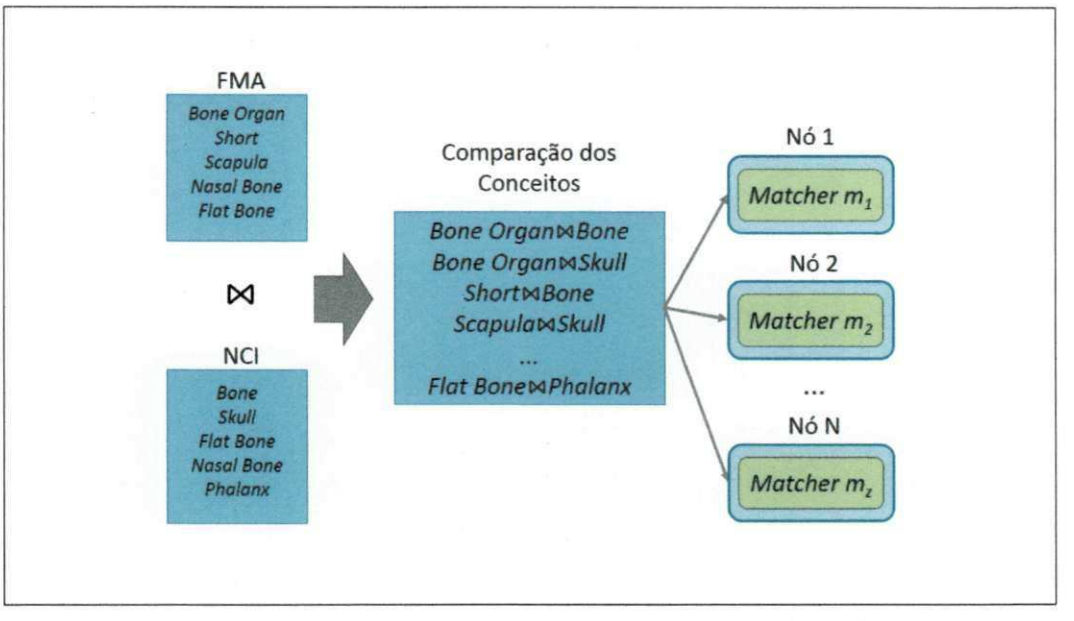

Figura 2.4: Modo *inter-matcher* de paralelização.

A desvantagem do modo *inter-matcher* está relacionada com as diferentes complexidades dos *matchers*, no qual a complexidade de um *matcher* refere-se ao custo computacional para executar uma comparação. Para uma infraestrutura homogênea (nós com a mesma configuração), o nó que hospeda o *matcher* mais complexo provavelmente executará por um longo tempo, enquanto os outros nós (que hospedam *matchers* menos complexos), após concluírem a execução das comparações, permanecerão ociosos. Esse desequilíbrio afetará diretamente o tempo total de execução do processo de MO. Contudo, esse problema está fora do escopo do presente trabalho, pois é assumido que todos os nós executam todos os *matchers* (*intra-matcher*), ou seja, todos os nós são semelhantes quanto à complexidade dos *matchers* executados.

No entanto, o modo *intra-matcher* apresenta dois problemas relevantes: a elevada taxa

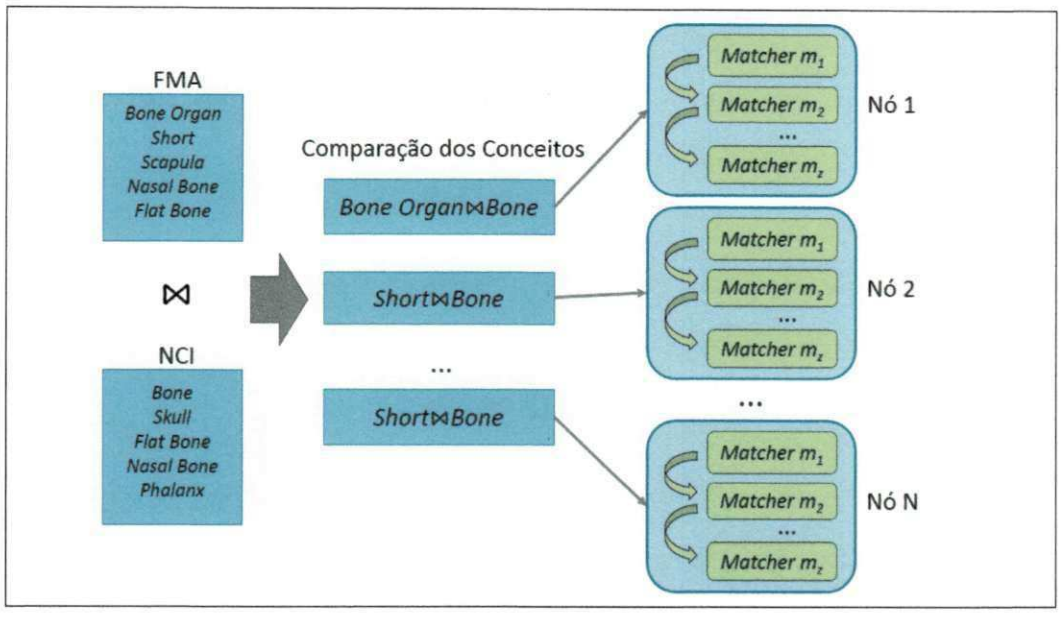

Figura 2.5: Modo *intra-matcher* de paralelização.

de replicação e o desbalanceamento de carga causado pelo desequilíbrio na quantidade de comparações realizadas em cada nó. No processo de MO em paralelo, para cada par de subontologias, os conceitos de uma subontologia são distribuídos entre os recursos computacionais. Por outro lado, todos os conceitos da outra subontologia devem ser replicados e enviados para os mesmo recursos computacionais, uma vez que todos os conceitos de uma subontologia devem ser comparados com todos os conceitos da outra subontologia. No pior caso, os conceitos de uma subontologia serão distribuídos entre todos os recursos computacionais, fazendo com que todos os conceitos da outra subontologia sejam replicados pela quantidade de recursos disponíveis e, posteriormente, enviados para todos os recursos computacionais.

Tomando como exemplo a Figura 2.5, os conceitos de *FMA* são distribuídos entre os N nós disponíveis. Consequentemente, todos os conceitos de *NCI* devem ser replicados N vezes e enviados para os N nós, possibilitando que todos os conceitos de *FMA* sejam comparados com todos os conceitos de *NCI* paralelamente.

Embora a replicação de conceitos seja necessária, a taxa de replicação deve ser minimizada, replicando os conceitos a menor quantidade de vezes possível. Uma vez que quanto maior a quantidade de conceitos replicados, maior a quantidade de dados transmitidos na

infraestrutura computacional e, consequentemente, menor a eficiência da utilização da infraestrutura.

Para abordagens de MO em paralelo que usam MapReduce como modelo de programação [10], o problema de desbalanceamento de carga pode acontecer tanto na fase de *map* quanto na de *reduce* (*map-skew* e *reduce-skew*, respectivamente) [17; 22; 30]. No entanto, uma vez que as comparações entre os conceitos são executados na fase de *reduce*, esta fase é a mais afetada. O problema de desbalanceamento de carga pode ocorrer devido a três razões [23; 22]:

1. O desequilíbrio causado por uma distribuição desigual de pares de conceitos enviados para as tarefas de *reduce*;
2. O desequilíbrio causado pelo fato de alguns pares de conceitos necessitarem de mais tempo para serem comparados do que outros;
3. A infraestrutura computacional possuir nós heterogêneos (configurações diferentes).

Para mitigar o problema de *reduce-skew*, as técnicas básica e refinada de balanceamento de carga propostas neste trabalho proveem soluções para minimizar os problemas 1 e 2, respectivamente.

2.5 Técnicas de Balanceamento de Carga

Como mencionado na Subseção 2.2.2, um dos principais desafios enfrentados pelas técnicas de paralelização do processo de MO é o desbalanceamento de carga. Esse problema ocorre quando alguns nós da infraestrutura computacional são executados por um longo tempo, enquanto os demais nós permanecem ociosos. No contexto de MO, existem poucas abordagens em paralelo e, dentre essas, nenhuma utiliza técnicas de balanceamento de carga. Porém, em algumas áreas como, por exemplo, resolução de entidades [21; 24] e *clustering* de entidades [14], já foram propostas técnicas de balanceamento de carga. Nessas áreas também são realizadas comparações entre dados, semelhante ao que é feito no processo de MO. Por essa razão, as técnicas de balanceamento de carga aplicadas nessas áreas podem servir como base para o desenvolvimento de técnicas de balanceamento de carga para o processo de MO em paralelo.

Existem dois métodos (estratégias) comumente utilizados pelas técnicas de balanceamento de carga: estático e dinâmico [23]. O método estático analisa as entradas antes de executar os *jobs* do MapReduce, a fim de identificar quais entradas podem causar desbalanceamento de carga. Normalmente, essa análise auxilia a abordagem orientando a distribuição uniforme das entradas entre as tarefas do MapReduce (*map* ou *reduce*).

No método dinâmico, o objetivo é identificar as tarefas (*map* ou *reduce*) que estão executando por um longo tempo (congestionadas) e quais estão ociosas. Após identificar as tarefas congestionadas, a execução dessas tarefas é interrompida e o seu contexto (por exemplo, valores, comparações realizadas, resultados) é armazenado. Posteriormente, a carga de trabalho é redistribuída dinamicamente entre os nós disponíveis, como descrito nas abordagens apresentadas em [23; 22].

A principal vantagem do método dinâmico é a independência da entrada do processo, ou seja, mesmo que não seja possível analisar e identificar as entradas que potencialmente podem causar o desbalanceamento (como é feito no método estático), o método dinâmico consegue minimizar o desbalanceamento de carga. No entanto, o método dinâmico é mais custoso no que diz respeito ao tempo de execução quando comparado ao método estático, tendo em vista que o método dinâmico precisa interromper a execução das tarefas para redistribuir a carga de trabalho.

2.6 Considerações Finais

Os tópicos apresentados neste capítulo têm o intuito de promover o embasamento teórico necessário para facilitar o entendimento dos próximos capítulos. Para tanto, foram descritos os principais aspectos relacionados ao processo de MO, sobretudo no que diz respeito ao desempenho do processo. Inicialmente, foram fornecidas informações relacionadas ao contexto de ontologias e as motivações que levam a realização do processo de MO. Posteriormente, as principais técnicas a serem aplicadas ao processo de MO, com o objetivo de obter um melhor desempenho do processo, foram descritas. Depois, foi descrito o funcionamento do *framework* MapReduce e como este *framework* pode ser utilizado para favorecer a execução do processo de MO em paralelo. Por fim, o problema relacionado ao desbalanceamento de carga, enfrentado por abordagens em paralelo, foi discutido e técnicas de balanceamento

de carga foram apresentadas. No capítulo seguinte, serão apresentados e discutidos vários trabalhos envolvendo o processo de MO em paralelo e técnicas de balanceamento de carga.

Capítulo 3

Trabalhos Relacionados

Neste capítulo será descrita uma visão geral dos trabalhos que abordam problemas relacionados ao MO e balanceamento de carga. Os trabalhos relacionados foram divididos em duas categorias: abordagens para *matching* de grandes ontologias em paralelo e técnicas de balanceamento de carga aplicadas ao processo de *matching* de grandes ontologias em paralelo.

3.1 Abordagens de Matching de Ontologias em Paralelo

O processo de MO em paralelo tem sido tratado em alguns trabalhos [2; 12; 35; 36; 40; 41], principalmente para minimizar o tempo de execução do processo de MO no contexto de grandes ontologias. Para isso, as abordagens distribuem os pares de conceitos a serem comparados entre os nós de um sistema distribuído.

Em [12; 2], os autores propuseram as suas próprias arquiteturas para executar o processo de MO em paralelo. A arquitetura proposta em [12] é dividida em três etapas: leitura dos dados (extrai as informações das ontologias), *workflow* (envia os pares de conceitos para os recursos disponíveis) e *matching* (compara os conceitos e determina as correspondências). Os autores não mencionam a utilização de estratégias de balanceamento de carga na etapa de *workflow*. A etapa de *matching* é realizada em paralelo, ou seja, cada nó da infraestrutura executa as comparações paralelamente aos demais nós.

No trabalho [2], a estratégia utilizada para paralelizar o processo de MO é semelhante ao *inter-matcher* (descrito no Capítulo 2), a diferença é que um nó pode executar mais de um *matcher*, se houver necessidade. A abordagem é dividida em etapas semelhantes à [12]:

leitura das ontologias de entrada, distribuição dos pares de conceitos entre nós da infraestrutura, comparação dos pares de conceitos e combinação dos alinhamentos parciais gerados por cada nó para formar o alinhamento final. Esse trabalho também não utiliza técnicas de balanceamento de carga na etapa de distribuição dos pares de conceitos entre os nós.

Em [35; 40; 36], os autores utilizam MR como solução programável para paralelizar o processo de MO. Em [35], cada nó executa uma ferramenta de MO dentre quatro disponíveis: Falcon-AO, Logmap, Optima+ e YAM++. Os autores de [40; 41] propõem uma abordagem que converte as ontologias de entrada em documentos virtuais, de maneira que, para cada conceito, é gerado um documento virtual responsável por armazenar as informações (valores, propriedades e instâncias). Após a criação dos documentos virtuais, a abordagem analisa de forma cíclica, por meio de vários *jobs* MapReduce, as semelhanças entre os documentos para definir as correspondências.

No trabalho [36], técnicas comumente utilizadas em MO são empregadas no contexto do *Linked Open Data* (LOD), que tem como objetivo determinar ligações (relações) entre dados abertos (públicos) na *web*. Essas técnicas são usadas para extrair informações dos dados, enquanto as métricas de similaridade auxiliam a definição das ligações entre os dados. Trabalhos relacionados a LOD utilizam as correspondências resultantes do processo de MO para selecionar entidades que potencialmente serão ligadas [26]. Assim, abordagens de MO em paralelo podem ser combinadas com trabalhos como [36], a fim de melhorar a eficiência do processo.

Ao contrário dos trabalhos discutidos anteriormente, a abordagem MOPP proposta neste trabalho considera como entrada pares de subontologias (produzidas por um algoritmo de particionamento), a fim de minimizar a quantidade de comparações entre conceitos. Com relação à transmissão de dados e tolerância a falhas, duas questões relevantes em sistemas distribuídos [10], os trabalhos [12; 2] não mencionam a forma como estas questões são tratadas. Uma vez que esses problemas já são tratados pela arquitetura do MR, os trabalhos que utilizam esse *framework* são beneficiados e não necessitam tratá-los.

Enquanto os trabalhos propostos em [40; 36] utilizam cinco e quatro *jobs* de MapReduce, respectivamente, para gerar o alinhamento das ontologias, a abordagem MOPP utiliza apenas um *job*. É importante destacar que, quanto maior o número de *jobs*, maior será o tempo de execução do processo de *matching*, uma vez que a configuração da infraestrutura

computacional para execução de cada *job* requer tempo.

A principal diferença entre os trabalhos acima mencionados e a abordagem MOPP é que uma técnica de balanceamento refinado de carga é aplicada à abordagem MOPP. Nessa técnica, a quantidade de comparações a serem executadas em cada par de conceitos é analisada, com base nas informações fornecidas pela técnica de particionamento de ontologias. Essa análise auxilia a técnica balanceamento de carga a distribuir uniformemente os pares de conceitos entre as tarefas de *reduce*. Nos trabalhos anteriormente citados, as técnicas de balanceamento de carga necessitam de um *job* (MapReduce) adicional para realizar a análise das entradas. Já na abordagem MOPP, a técnica de balanceamento refinado de carga não requer um *job* extra, pois leva apenas alguns segundos para ser executada e, por isso, é executada de forma sequencial.

A Tabela 3.1 apresenta um comparativo das abordagens em paralelo descritas e comentadas neste capítulo. O objetivo deste comparativo é dar uma visão geral sobre os trabalhos relacionados ao trabalho proposto. Para tanto, o comparativo é baseado em cinco aspectos importantes inseridos no contexto do trabalho proposto:

- Arquitetura Paralela: informa se a arquitetura em paralelo foi proposta pelo trabalho ou utilizou MapReduce como solução programável;
- Número de *Jobs*: informa a quantidade de *jobs* do MapReduce utilizados pela abordagem;
- Balanceamento de Carga: informa se a abordagem apresenta mecanismos para balancear a carga de trabalho na infraestrutura distribuída;
- Replicabilidade: informa se os autores disponibilizaram material (código fonte, pseudo-código, ontologias, resultados) suficiente para que os experimentos pudessem ser replicados;
- Domínio: informa o domínio (*Matching* de Ontologias, *Linked Open Data*) tratado no trabalho.

Trabalho	Arquitetura Paralela	Número de Jobs	Balanceamento de Carga	Replicabilidade	Domínio
Amin et al. 2015 [2]	Própria	-	Não	Não	MO
Gross et al. 2010 [12]	Própria	-	Não	Sim	MO
Thayasivam and Doshi 2013 [35]	MapReduce	1	Não	Não	MO
Zhang et al. 2012 [40; 41]	MapReduce	5	Não	Não	MO
Torre-Bastida et al. 2014 [36]	MapReduce	4	Não	Não	LOD
MOPP	**MapReduce**	**1**	**Sim**	**Sim**	**MO**

Tabela 3.1: Análise comparativa das abordagens de MO em paralelo.

3.2 Técnicas de Balanceamento de Carga para *Matching* de Ontologias em Paralelo

Alguns trabalhos [24; 21; 22; 14; 23], desenvolvidos em diferentes domínios, propõem técnicas de balanceamento de carga para melhorar a eficiência das suas abordagens. Como visto na Seção 2.4, há duas estratégias para atenuar o desbalanceamento de carga: métodos estáticos e dinâmicos [23]. Os trabalhos [24; 21; 14] utilizam métodos estáticos para analisar as entradas antes de executar os *jobs* do MapReduce, a fim de identificar quais entradas causam o desbalanceamento. Por outro lado, os trabalhos [22; 23] utilizam métodos dinâmicos.

Nos trabalhos [24; 21] são propostas abordagens para realizar Entidade Matching (EM) no contexto de grandes conjuntos de dados. Em uma etapa de pré-processamento, estas abordagens analisam a quantidade de comparações entre entidades a serem executadas. Como resultado, é gerada uma Matriz de Distribuição de Blocos (MDB) que armazena as informações referentes à quantidade de comparações. Portanto, a MDB orienta a distribuição uniforme das comparações entre as tarefas de *reduce*. Em [14], os autores propõem uma abordagem para *clustering* de entidades em paralelo. Nesse contexto, um *cluster* pode ser maior do que os outros e, assim, causar desbalanceamento de carga no processo. Para melhorar o balanceamento de carga, a abordagem estima o custo de processamento de cada *cluster*, com o intuito de gerar um histograma que armazena o custo de cada *cluster* e identifica qual deles é responsável pelo desbalanceamento. Tal como a MDB nas obras supracitadas, o histograma também auxilia a distribuição uniforme dos *clusters* entre as tarefas de *reduce*.

Os trabalhos [14; 23] utilizam métodos dinâmicos para melhorar o balanceamento de carga. Esses trabalhos não abordam um domínio específico (por exemplo, EM ou OM), eles propõem técnicas para identificar quais as tarefas (*map* ou *reduce*) estão executando por um longo tempo (acima de um limiar pré-definido) e quais as tarefas que já terminaram os seus trabalhos (ociosas). Quando as tarefas sobrecarregadas são identificadas, o contexto (por exemplo, valores, resultados, comparações já computadas) é armazenado e essas tarefas são interrompidas. A partir daí, a carga de trabalho (comparações) é redistribuída entre as tarefas disponíveis, proporcionando o balanceamento de carga.

Uma vez que no processo de MO é possível calcular a quantidade de comparações em cada par de conceitos, a abordagem MOPP utiliza um método estático para gerar as informações necessárias para auxiliar o balanceamento de carga da abordagem MOPP, proposta neste trabalho. Ao contrário das abordagens [24; 21; 14] que analisam a quantidade de pares de entidades a serem comparados, a abordagem MOPP realiza uma análise refinada que explora as comparações realizadas pelos *matchers* em cada par de conceitos. Essa análise tem como objetivo proporcionar um melhor balanceamento de carga. A abordagem MOPP calcula a quantidade de comparações a serem realizadas em cada par de conceitos em uma etapa de pré-processamento realizada em sequencial (que geralmente leva alguns poucos segundos para ser executada), enquanto as obras [24; 21] utilizam dois *jobs* de MapReduce, um para gerar a MDB e outro para realizar o processo de EM.

3.3 Considerações Finais

Neste capítulo, foram apresentados os principais trabalhos relacionados ao processo de MO em paralelo. Nesse contexto, foram abordados trabalhos que propuseram sua própria arquitetura e outros que utilizaram o *framework* MR como solução programável para paralelizar o processo de MO. Por fim, foram apontados alguns trabalhos que propuseram técnicas de balanceamento de carga, de maneira que essas técnicas serviram de base para as técnicas de balanceamento de carga propostas neste trabalho. No capítulo seguinte será apresentada a abordagem MOPP proposta neste trabalho, incluindo suas etapas e seu funcionamento.

Capítulo 4

Uma Abordagem para Matching de Ontologias Baseada em Particionamento e Paralelismo com Balanceamento de Carga

O objetivo deste trabalho é propor uma abordagem que melhore o desempenho do processo de MO no contexto de grandes ontologias. Para este fim, a abordagem MOPP utiliza técnicas de paralelização e particionamento de ontologias. Inicialmente é aplicada uma técnica de particionamento nas ontologias de entrada a fim de diminuir a quantidade de comparações a serem realizadas. Posteriormente, as comparações entre conceitos são executadas em paralelo nos nós de uma infraestrutura computacional, reduzindo o tempo de execução do processo de MO.

Com o intuito de melhorar ainda mais a eficiência do processo de MO, são propostas duas técnicas de balanceamento de carga: básica [4; 3] e refinada. A técnica básica distribui uniformemente a quantidade de pares de conceitos entre as tarefas de *reduce*. A técnica refinada analisa tanto a quantidade de pares de conceitos a serem comparados quanto a quantidade de comparações realizadas pelos *matchers* em cada par de conceitos. Isso permite que a técnica guie de forma eficaz os pares de conceitos para cada tarefa de *reduce*, ou seja, cada tarefa de *reduce* executa uma quantidade semelhante de comparações.

Neste capítulo é descrita a abordagem MOPP e as duas técnicas de balanceamento de

carga aplicadas à abordagem. Na seção 4.1 é apresentada a formalização da abordagem MOPP e dos problemas relacionados à abordagem. A seção 4.2 expõe uma visão geral da abordagem MOPP. Na seção 4.3 são descritas as etapas da abordagem MOPP e o funcionamento das técnicas de balanceamento de carga aplicadas à abordagem. Por fim, a seção 4.4 apresenta as limitações da abordagem.

4.1 Formalização do Problema

No processo de MO, as ontologias de entrada (O_1 e O_2) são representadas por um conjunto de conceitos $O_1 = \{c_1, c_2, ..., c_j\}$ e $O_2 = \{c_1, c_2, ..., c_k\}$. Um conceito $c \in O_i$ possui um conjunto de propriedades $P = \{P_1, P_2, ..., P_w\}$ como, por exemplo, rótulo, anotações e relacionamentos. O conjunto de propriedades P de c é dado pela notação $P(c)$. Cada propriedade $P_w \in P$ possui um conjunto de elementos $Pw = \{p_1, p_2, ..., p_y\}$, ou seja, as instâncias de P.

Na abordagem MOPP, cada ontologia de entrada (O_i) é submetida a um algoritmo de particionamento Cl_{alg} que gera um conjunto de subontologias, denotada por $O_i.S = \{S_1, S_2, ..., S_n\}$, de maneira que $\bigcap_{S \in O_i} = \emptyset$ e $\bigcup_{S \in O_i} = O_i$. O algoritmo Cl_{alg} também determina os pares de subontologias que devem ser comparados (pares similares), denotado por $Pares(O_1.S, O_2.S) = \{\langle S, S' \rangle_1, \langle S, S' \rangle_2, ..., \langle S, S' \rangle_b\}$, de modo que $S \in O_1.S$ e $S' \in O_2.S$.

Para cada par de subontologias $\langle S, S' \rangle \in Pares(O_1.S, O_2.S) : S \bowtie S'$ é computado, ou seja, $\forall c \in S$ e $\forall c' \in S' : \langle c \bowtie c', M \rangle$, onde $M = \{m_1, m_2, ..., m_z\}$ é o conjunto de *matchers* a ser utilizado no processo de MO. Uma vez que cada *matcher* $m \in M$ compara c e c' baseado em uma propriedade P comum a ambos os conceitos, cada par de conceitos ($c \bowtie c'$) pode ser entendido como um conjunto de comparações entre os elementos de $P(c)$ e $P(c')$, denotado por $COMP(c, c', P(c), P(c'), M) = \{comp(c, c', P_1, P'_1, m_1), comp(c, c', P_2, P'_2, m_2), ..., comp(c, c', P_w, P'_w, m_z)\}$. Para cada propriedade em $P(c)$ e $P(c')$, a quantidade de comparações entre os elementos de P e P' é mensurado por $|comp(c, c', P_w, P'_w, m_z)| = |P_w| * |P'_w|$. Dessa forma, para cada par de conceitos ($c \bowtie c'$), a quantidade de comparações realizadas é dada por $|COMP(c, c', P(c), P(c'), M)| = \sum_{i=1}^{z} |comp(c, c', P_i, P'_i, m_i)|$, onde $m_i \in M$. Para um par de subontologias, a quantidade de comparações a serem realizadas é dada por

$|COMP(\langle S, S'\rangle, M)| = \sum_{x=1}^{|S|} \sum_{y=1}^{|S'|} |COMP(c_x, c'_y, c_x(P), c'_y(P), M)|$ e para todos os pares de subontologias é dada por $A = \sum_{x=1}^{b} |COMP(\langle S, S'\rangle_x, M)|$.

Neste trabalho, o cálculo de $Pares(O_1.S, O_2.S)$ é realizado em paralelo utilizando uma infra-estrutura computacional distribuída, composta por vários nós com uma configuração homogênea. Neste contexto, assume-se que $I = \{n_1, n_2, ..., n_N\}$ é o conjunto dos nós utilizados para calcular $Pares(O_1.S, O_2.S)$, de tal modo que o tempo de execução usando um único nó $n \in I$ é denotado por $T^n_{exec}(Pares(O_1.S, O_2.S))$, enquanto que o tempo de execução utilizando toda a infraestrutura computacional I é indicado por $T^I_{exec}(Pares(O_1.S, O_2.S))$.

Para evitar o desbalanceamento de carga, os pares de conceitos $(c \bowtie c')$ deverão ser distribuídos uniformemente entre os nós de I, com base no $|COMP(c, c', P(c), P(c'), M)|$, de modo que cada nó n receba uma quantidade semelhante de comparações para serem executadas. Formalmente, assumindo que cp é a quantidade de comparações a serem executadas em um único nó $n \in I$, a quantidade de comparações a serem executadas em todos os nós de I é representada pelo conjunto $CP = \{cp_1, cp_2, ..., cp_N\}$ de maneira que é possível estender a notação para $\sum_{i=1}^{N} cp_i = A$. Para evitar o problema de desbalanceamento de carga, a diferença na quantidade de comparações entre os nós deve ser minimizada, ou seja, $\sum_{i=1}^{N-1} \sum_{k=i+1}^{N} (cp_i - cp_k)^2 \approx 0$. Como resultado, a diferença entre o tempo de execução dos nós, dada por $E = \{e_1, e_2, ...e_N\}$, tende a ser minimizada $\sum_{i=1}^{N-1} \sum_{k=i+1}^{N} (e_i - e_k)^2 \approx 0$.

No processo de MO em paralelo, para cada par de subontologias, é necessário distribuir os conceitos de uma subontologia entre os nós e replicar os conceitos da outra para que sejam enviados para os mesmos nós. Dessa forma, para cada par de subontologias $\langle S, S'\rangle \in Pares(O_1.S, O_2.S)$, os conceitos de S ou S' devem ser replicados como denotado por $rep(\langle S, S'\rangle) = \alpha * |S|$ (se $|S| \leq |S'|$) ou $rep(\langle S, S'\rangle) = \alpha * |S'|$, caso contrário. O valor α é determinado de acordo com a quantidade de nós a serem utilizados pelo par de subontologias, dependendo da forma como os pares de conceitos (do par de subontologias) são distribuídos entre os nós. Além disso, α representa a frequência de replicação dos conceitos da subontologia (S ou S'), de tal modo que $\alpha \in [1, n]$. Quanto mais alta a taxa de replicação α, mais baixa é a eficiência da utilização da infraestrutura, devido à elevada quantidade de conceitos sendo transmitidos entre os nós de I. Assim, a taxa de replicação α deve ser reduzida.

Para resumir os problemas abordados nesta seção, pode-se destacar os seguintes objetivos a serem atingidos por este trabalho:

$$\textbf{Maximizar: } T_{exec}(O_1 \bowtie O_2) - T_{exec}(Pares(O_1.S, O_2.S)) \tag{4.1}$$

$$\textbf{Minimizar: } N - \frac{T^n_{exec}(Pares(O_1.S, O_2.S))}{T^I_{exec}(Pares(O_1.S, O_2.S))} \tag{4.2}$$

$$\textbf{Minimizar: } \sum_{i=1}^{N}(cp_i - \frac{A}{N}) \tag{4.3}$$

4.2 Visão Geral da Abordagem MOPP

A abordagem MOPP, apresentada neste capítulo, utiliza o *framework* MapReduce para paralelizar a etapa de *matching* de subontologias (etapa 4 da Figura 2.2). As duas técnicas de balanceamento de carga, propostas neste trabalho, são aplicadas à abordagem MOPP com o objetivo de distribuir uniformemente a carga de trabalho (pares de conceitos ou comparações) entre as tarefas de *reduce*.

Resumidamente, a abordagem proposta funciona da seguinte forma: para cada par de subontologias a serem comparadas, os conceitos da subontologia menor (em termos de quantidade de conceitos) são replicados na fase de *map*. Os conceitos da subontologia maior são enviados juntamente com os conceitos replicados da subontologia menor para serem comparados na mesma tarefa de *reduce* (fase de *reduce*).

Para cada par de conceitos, os valores de similaridade (parciais) são computados pelos *matchers*. Os pares de conceitos que apresentam um valor de similaridade (agregado) acima de um limiar ϕ são considerados correspondências. A abordagem MOPP é dividida em quatro etapas: leitura e pré-processamento, *map*, *shuffle* e *reduce*. A seguir, cada etapa é descrita em mais detalhes. Para facilitar a compreensão, é utilizado o exemplo ilustrado na Figura 4.1 ao longo do capítulo.

4.3 Etapas da Abordagem MOPP

Nesta seção são descritas as etapas da abordagem para o Matching de Ontologias Baseadas em Particionamento e Paralelismo com Balanceamento de Carga.

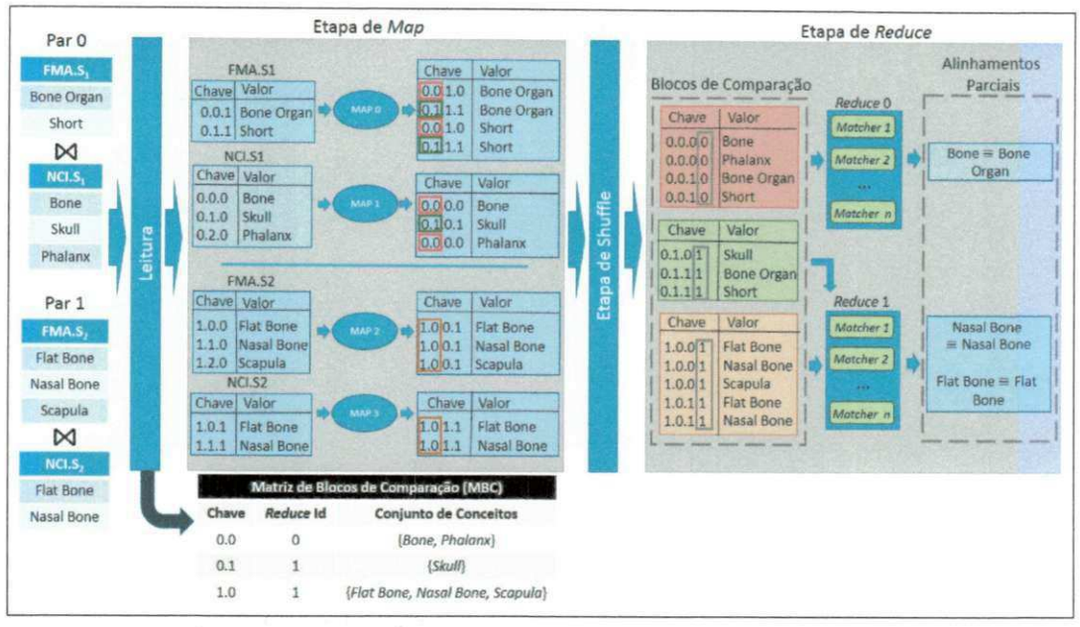

Figura 4.1: *Matching* dos pares de subontologias com balanceamento refinado de carga.

4.3.1 Etapa 1: Leitura de conceitos e Pré-processamento

Nesta etapa é realizada a leitura dos conceitos contidos nos pares de subontologias. Durante a leitura dos conceitos, é feito o cálculo da quantidade de pares de conceitos e comparações resultante de cada par de subontologias. Esta etapa é de fundamental importância para abordagem MOPP, pois tais informações são utilizadas pelas duas técnicas de balanceamento de carga (básica e refinada) propostas neste trabalho, com o intuito de distribuir uniformemente a carga de trabalho (pares de conceitos ou comparações) entre as tarefas de *reduce*. Nesta subseção, em primeiro lugar, é descrita a leitura dos conceitos e, posteriormente, é detalhado o pré-processamento, onde é descrito o funcionamento das técnicas básica e refinada de balanceamento de carga.

Leitura dos conceitos. Inicialmente, cada par de subontologias é associado a um identificador (*par_id*). Os conceitos são lidos e as propriedades de cada conceito (por exemplo, rótulo, anotações e relações) são extraídas para formarem tuplas. Cada conceito (tupla) recebe uma chave de identificação composta por: $\langle par\_id.conceito\_id.replica \rangle$, onde *conceito_id* é o identificador do conceito na subontologia correspondente e *replica* indica se o conceito deve ser replicado (*replica* = 1) ou não (0) na etapa de *map*. Na Figura 4.1, existem dois pares de subontologias: Par 0 = $\langle FMA.S_1, NCI.S_1 \rangle$ e Par 1 = $\langle FMA.S_2, NCI.S_2 \rangle$.

No Par 0, $FMA.S_1$ é a subontologia menor, uma vez que tem menos conceitos (2) do que a subontologia $NCI.S_1$ (3). A chave do conceito *Bone Organ* é "0.0.1", uma vez que o valor "0" representa o par 0, o segundo "0" é o identificador do conceito na subontologia $FMA.S_1$ e "1" indica que o conceito será replicado.

Pré-processamento. À medida em que os conceitos dos pares de subontologias são lidos, são calculadas a quantidade de pares de conceitos e de comparações resultantes de cada par de subontologias. A partir dessas informações, a técnica de balanceamento de carga gera uma matriz contendo informações a respeito das tarefas de *reduce* e dos pares de conceitos que devem ser enviados para cada uma das tarefas.

A matriz em questão é denominada Matriz de Blocos de Comparação (MBC). A MBC é formada a partir de um ou mais blocos de comparação resultantes de cada par de subontologias. Cada bloco de comparação de um par de subontologias contém um subconjunto dos conceitos que pertencem à subontologia maior (em termos de quantidade de conceitos) e todos os conceitos da subontologia menor. Cada linha da matriz MBC representa um bloco de comparação no formato: $\langle\langle par\_id, bloco\_comparação\_id\rangle; reduce\_id; conjunto\_de\_conceitos\rangle$, onde $\langle par\_id, bloco\_comparação\_id\rangle$ representa a chave bloco de comparação, par_id identifica o par de subontologias, $bloco\_comparação\_id$ é o identificador do bloco de comparação, $reduce\_id$ é o identificador da tarefa de *reduce* e $conjunto\_de\_conceitos$ é um conjunto composto pelos conceitos da subontologia maior que serão enviados para a mesma tarefa de *reduce* (com base no $reduce\_id$).

As técnicas de balanceamento de carga propostas neste trabalho são guiadas pelas informações contidas na MBC para realizar uma distribuição uniforme da carga de trabalho, ou seja, cada tarefa de *reduce* deve receber uma quantidade similar de pares de conceitos ou comparações. A MBC é utilizada na etapa de *map* (etapa seguinte) para auxiliar a alteração das chaves dos conceitos. Essas novas chaves são utilizadas na etapa de *shuffle* para guiar o envio dos conceitos para as tarefas de *reduce*, conforme determinado pela MBC.

Neste trabalho são propostas duas técnicas de balanceamento de carga: básica e refinada. Ambas realizam a leitura dos conceitos de maneira semelhante e geram uma MBC como resultado. Consequentemente, as demais etapas da abordagem MOPP (*map*, *shuffle* e *reduce*) independem da técnica de balanceamento de carga aplicada. Por outro lado, as duas técnicas

diferem na forma como realizam a extração das informações sobre os pares de subontologias, utilizadas na formação da MBC.

Na técnica básica de balanceamento de carga, a distribuição dos pares de conceitos entre as tarefas de *reduce* é baseada na quantidade de pares de conceitos. O cálculo da quantidade de pares de conceitos é feito a partir da quantidade de conceitos contidos em cada subontologia do par de subontologias. Na Figura 4.2, cada um dos pares de subontologias (Par 0 e Par 1) resultam em 6 pares de conceitos (produto Cartesiano), uma vez que uma subontologia possui 2 conceitos e a outra subontologia 3. Como o total de comparações a serem executadas é 12 (6 + 6) e existem duas tarefas de *reduce*, os pares de conceitos são distribuídos de maneira que cada uma das tarefas de *reduce* receba 6 pares de conceitos a serem comparados.

Após o cálculo da quantidade de pares de conceitos que cada tarefa de *reduce* deve receber, são definidos quais os pares de conceitos que devem ser enviados para cada tarefa de *reduce*. A distribuição dos pares de conceitos entre as tarefas de *reduce* é realizada a partir de uma estratégia gulosa que aloca os pares de conceitos para a tarefa de *reduce* com menos pares de conceitos alocados.

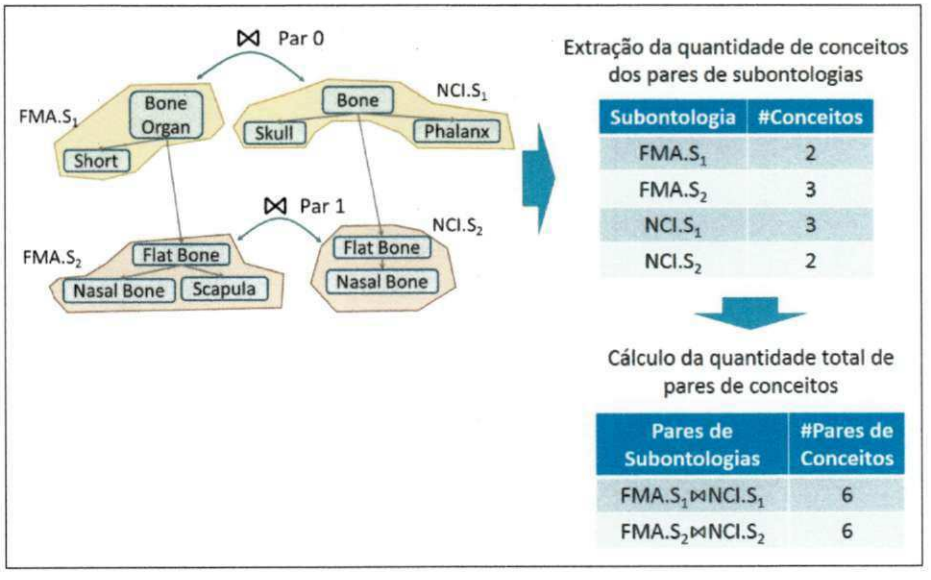

Figura 4.2: Quantidade de pares de conceitos para cada par de subontologias.

Na técnica básica de balanceamento de carga, os blocos de comparação são formados por um conceito da subontologia maior e todos os conceitos da subontologia menor. A

partir do exemplo ilustrado na Figura 4.3, os pares de conceitos são agrupados nos seguintes blocos de comparação: Par 0: *Bone*⋈{*Bone Organ, Short*}, *Skull*⋈{*Bone Organ, Short*} e *Phalanx*⋈{*Bone Organ, Short*}; Par 1: *Flat Bone*⋈{*Flat Bone, Nasal Bone*}, *Nasal Bone*⋈{*Flat Bone, Nasal Bone*} e *Scapula*⋈{*Flat Bone, Nasal Bone*}. Com base nesses blocos de comparação, a estratégia gulosa determina que os pares *Bone*⋈{*Bone Organ, Short*}, *Phalanx*⋈{*Bone Organ, Short*} e *Nasal Bone*⋈{*Flat Bone, Nasal Bone*} são alocados para a tarefa de *reduce* 0 e os pares *Skull*⋈{*Bone Organ, Short*}, *Flat Bone*⋈{*Flat Bone, Nasal Bone*} e *Scapula*⋈{*Flat Bone, Nasal Bone*} são alocados para a tarefa de *reduce* 1. Dessa forma, cada tarefa de *reduce* recebe a mesma quantidade (6) de pares de conceitos. Por fim, as informações a respeito da alocação dos pares de conceitos para as tarefas de *reduce* são armazenadas na MBC (Matriz de blocos de comparação).

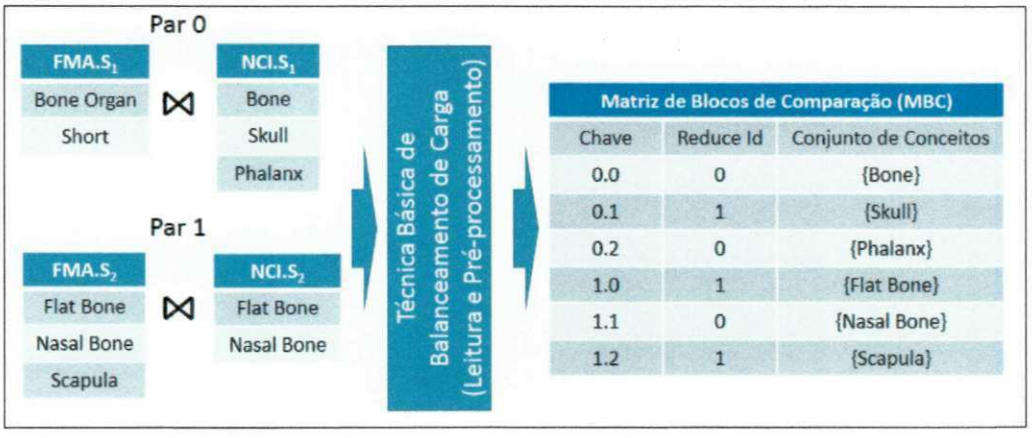

Figura 4.3: Formação da MBC segundo a técnica básica de balanceamento de carga na etapa de leitura e pré-prcessamento.

Na técnica refinada de balanceamento de carga, a análise dos pares de subontologias é realizada no sentido de obter a quantidade de comparações a serem realizadas nos pares de conceitos. Dessa forma, a partir da quantidade de propriedades de cada conceito de um par, é possível calcular a quantidade de comparações a serem realizadas para cada par de conceitos. Consequentemente, também é possível calcular a quantidade de comparações para cada par de subontologias.

A Figura 4.4 ilustra como as quantidades de pares de conceitos e comparações são calculadas para os pares de subontologias 0 e 1 da Figura 4.1. No exemplo, apenas as propriedades

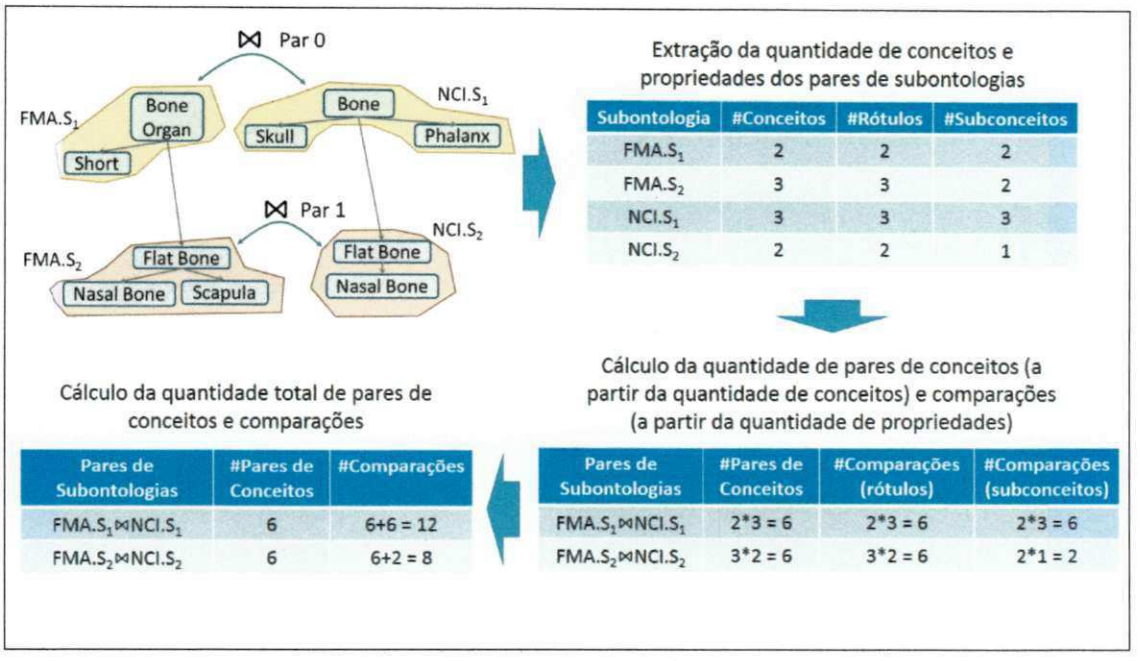

Figura 4.4: Quantidade de comparações para cada par de subontologias.

rótulo e relação *is-a* (subconceitos) são consideradas. A subontologia $FMA.S_1$ contém dois conceitos (rótulos: *Bone Organ* e *Short*). *Bone Organ* possui dois subconceitos e *Short* nenhum (*Bone Organ*: {*Short, Flat Bone*}, *Short*: {}). A subontologia $NCI.S_1$ contém três conceitos (rótulos: *Bone, Skull* e *Phalanx*). *Bone* possui três subconceitos enquanto os demais conceitos não possuem nenhum subconceito (*Bone*: {*Skull, Phalanx, Flat Bone*}, *Skull*: {}, *Phalanx*: {}). Uma vez que $FMA.S_1$ e $NCI.S_1$ formam um par de subontologias (Par 0), os dois conceitos de $FMA.S_1$ serão comparados com os três conceitos de $NCI.S_1$ resultando em seis pares de conceitos a serem comparados.

Para cada par de conceitos, os *matchers* (*Label* e *Children*) irão explorar as propriedades rótulo e relação *is-a* (subconceitos), respectivamente. Portanto, o par de subontologias $FMA.S_1 \bowtie NCI.S_1$ irá executar seis comparações explorando a propriedade rótulo e seis comparações explorando a propriedade relação *is-a*. Da mesma forma, a quantidade de comparações é calculada para o par de subontologias $FMA.S_2 \bowtie NCI.S_2$, o que resulta em seis comparações explorando a propriedade rótulo e duas comparações explorando a propriedade relação *is-a*. É importante destacar que, embora a quantidade de pares de conceitos seja a mesma para ambos os pares de subontologias (6), a quantidade de comparações é diferente

(12 e 8). Essa diferença é responsável pelo desbalanceamento de carga nas tarefas de *reduce*.

Baseando-se na quantidade de comparações de cada par de subontologias, é possível identificar quais são os pares (caso existam) responsáveis pelo desbalanceamento de carga. Essas informações são necessárias para orientar os pares de conceitos a serem enviados para cada tarefa de *reduce*, proporcionando o balanceamento de carga. Para este fim, é calculada a média de comparações por tarefa de *reduce*. Na Figura 4.4, a média de comparações é 10 comparações por tarefa de *reduce*, uma vez que existem 20 (12 + 8) comparações a serem realizadas no total e 2 tarefas de *reduce* para executarem as comparações.

Para os pares de subontologias cuja quantidade de comparações esteja acima da média de comparações por tarefa de *reduce*, as comparações pertencentes a estes pares serão fatiadas, onde o *fator de fatiamento* é dado por:

$$fator\ de\ fatiamento = \frac{quantidade\ de\ comparações}{média\ de\ comparações\ por\ tarefa\ de\ reduce} \tag{4.4}$$

O objetivo do *fator de fatiamento* (Equação 4.4) é determinar a quantidade de blocos de comparação para cada par de subontologias. Como cada bloco de comparação será enviado para uma tarefa de *reduce*, o *fator de fatiamento* também determina a quantidade mínima de tarefas de *reduce* necessárias para executar as comparações e a capacidade de cada tarefa de *reduce* a ser utilizada, minimizando a taxa de replicação de conceitos. Com relação aos pares de subontologias cuja quantidade de comparações esteja abaixo da média, as comparações resultantes de cada par serão enviadas para uma única tarefa de *reduce*.

Na Figura 4.4, o par de subontologias $\langle FMA.S_1, NCI.S_1 \rangle$ resultará em 12 comparações. Dado que a quantidade de comparações (12) é maior do que a média de comparações (10), as comparações desse par de subontologias devem ser fatiadas. Assumindo que o valor do *fator de fatiamento* é 1,2 (12/10), então uma tarefa de *reduce* será totalmente utilizada 1,0 (100%) e apenas 0,2 (20%) da capacidade da outra tarefa de *reduce* será necessária. Com relação ao par de subontologias $\langle FMA.S_2, NCI.S_2 \rangle$, uma vez que a quantidade de comparações (8) é menor do que a média de comparações por tarefa de *reduce* (10), todas as comparações serão enviadas para uma única tarefa de *reduce*.

Para os pares de subontologias a serem fatiados, é necessário identificar quais os pares de conceitos serão alocados para cada tarefa de *reduce*, com base na quantidade de comparações de cada par de conceitos. Para isto, os pares de conceitos são ordenados pela quantidade

de comparações realizadas em cada um deles. Uma estratégia gulosa é usada para alocar os pares de conceitos para cada tarefa de *reduce*, sem exceder a capacidade das tarefas (determinada pelo *fator de fatiamento*).

A Figura 4.5 ilustra a distribuição dos pares de conceitos pertencentes aos pares de subontologias 0 e 1. Os pares de conceitos do par $\langle FMA.S_1, NCI.S_1 \rangle$ são ordenados de forma decrescente pela quantidade de comparações. Em seguida, a estratégia gulosa aloca os pares de conceitos *Bone*⋈{*Bone Organ, Ósseo, Short*} para tarefa de *reduce* 0 e os pares de conceitos *Skull*⋈{*Bone Organ, Short*} para tarefa de *reduce* 1. Os pares de conceitos *Phalanx*⋈{*Bone Organ, Short*} são alocados para tarefa de *reduce* 0, mesmo esta tarefa tendo mais comparações alocadas do que a *reduce* 1. Isso ocorre em virtude de 20% (2 de 10 comparações) da capacidade da tarefa de *reduce* 1 já estar ocupada. Para o par $\langle FMA.S_2, NCI.S_2 \rangle$, todas as comparações serão alocadas para a tarefa de *reduce* 1, a mais vazia dentre as tarefas de *reduce*.

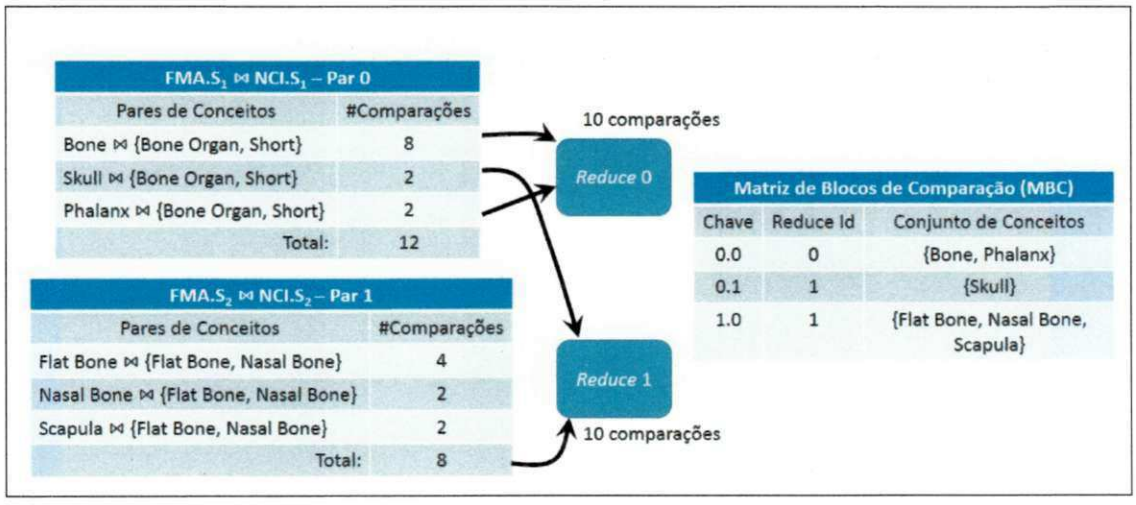

Figura 4.5: Subdivisão dos pares de conceitos entre os nós.

Por fim, as informações referentes à alocação dos blocos de comparação para as respectivas tarefas de *reduce* são armazenadas na MBC. Na Figura 4.5, o exemplo resulta em uma MBC = $\langle\langle 0.0\rangle;\ 0;\ \{Bone, Phalanx\}\ \rangle$, $\langle\langle 0.1\rangle;\ 1;\ \{Skull\}\rangle$, $\langle\langle 1.0\rangle;\ 1;\ \{FlatBone, NasalBone, Scapula\}\ \rangle$. O pseudocódigo referente a esta etapa está descrito no Apêndice A (A.1).

4.3.2 Etapa 2: Map

Esta etapa é responsável pela definição das chaves dos conceitos, com o objetivo de guiar a formação dos blocos de comparação. As tarefas de *map* recebem os pares chave-valor $\langle chave, conceito \rangle$ originados da etapa de leitura. Os conceitos com o valor *replica* igual a 1 são replicados. Para cada conceito replicado, o *conceito_id* da chave é substituído pelo *bloco_comparação_id* do bloco comparação com o mesmo *par_id* do conceito. Além disso, as tarefas de *map* acrescentam o *reduce_id* à chave do conceito. Essa parte da chave indica para qual tarefa de *reduce* o conceito será enviado. O valor do *reduce_id* é obtido a partir da MBC (gerada na etapa anterior).

Na Figura 4.1, o conceito *Bone Organ* (subontologia $FMA.S_1$) é replicado duas vezes, tendo em vista que será incluído em dois blocos de comparação. Em cada replicação de *Bone Organ*, o valor do *conceito_id* é substituído pelo valor *bloco_comparação_id* do bloco de comparação cujo *par_id* = 0 (Par 0). Além disso, é importante destacar que a tarefa de *map* acrescenta o valor 0 à *reduce_id* da chave de *Bone Organ* com *conceito_id* = 0 e o valor 1 à chave do conceito *Bone Organ* cujo *conceito_id* = 1.

Se o conceito não deve ser replicado (*replica* = 0), o *conceito_id* da chave é substituído pelo valor *bloco_comparação_id* do bloco de comparação (na MBC) cujo $conjunto\_de\_conceitos$ contém o conceito em questão. Além disso, o *reduce_id* é adicionado à chave conceito. Na Figura 4.1, a chave do conceito *Phalanx* é "0.2.0", onde o último "0"indica que o conceito não será replicado. Assim, o *bloco_comparação_id* do bloco de comparação que contém o conceito *Phalanx* é 0 e, consequentemente, o valor *conceito_id* é substituído por 0. Além disso, o valor "0"(*reduce_id* para a chave $\langle 0.0 \rangle$ na MBC) é adicionado à chave, resultando na chave "0.0.0.0". Por fim, os valores-chave $\langle chave, conceito \rangle$ são emitidos para a próxima etapa (*Shuffle*). O pseudocódigo das tarefas de *map* está descrito no Apêndice A (A.2).

4.3.3 Etapa 3: Shuffle

Nesta etapa, os pares chave-valor $\langle chave, conceito \rangle$ são particionados, ordenados e agrupados. Os pares $\langle chave, conceito \rangle$ são agrupados e ordenados pela chave, com o intuito de definir os blocos de comparação, tal como ilustrado na Figura 4.1. Os conceitos são agrupa-

dos baseados nos dois primeiros valores de suas chaves, ou seja, *par_id* e *conceito_id*. Os conceitos *Bone*, *Phalanx*, *Bone Organ* e *Short* são agrupados, uma vez que os dois primeiros valores de suas chaves são iguais ("0.0"). Para ordenar os conceitos em um bloco de comparação, todos os valores da chave são considerados. O último valor da chave (*reduce_id*) dos conceitos determina para qual tarefa de *reduce* os conceitos que compõem um bloco de comparação deverão ser enviados. Por fim, os pares $\langle chave, conceito \rangle$ são agrupados em cada bloco de comparação e enviados para a etapa de *reduce*. Na Figura 4.1, os conceitos *Bone*, *Phalanx*, *Bone Organ* e *Short* são alocados para tarefa de *reduce* 0, uma vez que o valor *reduce_id* da chave dos conceitos é 0.

4.3.4 Etapa 4: Reduce

O objetivo desta etapa é executar as comparações dos pares de conceitos contidos nos blocos de comparação. As tarefas de *reduce* recebem os blocos de comparação e, para cada bloco, realizam a comparação do subconjunto de conceitos da subontologia maior com todos os conceitos do subontologia menor. Os primeiros conceitos de um bloco de comparação sempre pertencem a maior subontologia, uma vez que o terceiro valor de suas chaves é igual a "0". Consequentemente, os demais conceitos de um bloco de comparação pertencem à subontologia menor (terceiro valor da chave é "1"). Os *matchers* executam as comparações entre os conceitos e um valor de similaridade (agregado) é retornado para cada par de conceitos. Os pares de conceitos com um valor acima do limiar de similaridade (ϕ) são considerados correspondências. Para concluir, a saída de cada tarefa de *reduce* é um conjunto de correspondências denominado alinhamento parcial.

Na Figura 4.1, o bloco de comparação formado pelos conceitos *Bone*, *Phalanx*, *Bone Organ* e *Short* são enviados para a tarefa de *reduce* 0 (*reduce_id* é igual a 0), onde *Bone* e *Phalanx* (os primeiros conceitos) pertencem à $NCI.S_1$ (subontologia maior) e os conceitos *Bone Organ* e *Short* pertencem à $FMA.S_1$ (subontologia menor). Assim, os conceitos *Bone* e *Phalanx* são comparados com os conceitos *Bone Organ* e *Short*. Respectivamente, cada comparação produz um valor de similaridade: *Bone* $\bowtie$ *Bone Organ* = 0,85, *Bone* $\bowtie$ *Short* = 0,1, *Phalanx* $\bowtie$ *Bone Organ*= 0,14 e Phalanx $\bowtie$ *Short* = 0,3. Uma vez que a comparação *Bone* $\bowtie$ *Bone Organ* tem um valor superior ao limiar de similaridade (ϕ = 0,7), este par de conceitos é considerado uma correspondência. Os outros pares de conceitos também

são comparados, resultando nas seguintes correspondências: $\langle FlatBone, FlatBone, 1.0 \rangle$ e $\langle NasalBone, NasalBone, 1.0 \rangle$. O pseudocódigo que descreve o funcionamento das tarefas de *reduce* está descrito no Apêndice A (A.3).

4.4 Limitações da Abordagem MOPP

A abordagem apresentada neste capítulo apresenta uma limitação em relação a sua aplicabilidade prática. Essa limitação é referente ao custo computacional dos *matchers*. Na técnica refinada de balanceamento de carga, assume-se que os custos computacionais das comparações realizadas nos *matchers* são os mesmos, independente dos *matchers* aplicados. Os custos das comparações podem ser considerados similares quando: apenas um *matcher* é aplicado ou as comparações realizadas nos *matchers* possuem o mesmo custo computacional. Por exemplo, embora os *matchers Label* e *Children* sejam diferentes, o custo computacional das comparações entre os rótulos dos conceitos/subconceitos pode ser considerado o mesmo, desde que seja utilizado o mesmo algoritmo de similaridade linguística.

Tipicamente, os *matchers* que exploram propriedades estruturais dos conceitos possuem um custo computacional superior aos demais *matchers* [11], uma vez que o *matcher* necessita explorar toda a vizinhança (estrutural) dos conceitos para determinar a similaridade entre os conceitos do par. Por essa razão, o custo computacional de um *matcher* linguístico dificilmente será o mesmo de um *matcher* estrutural [11].

Neste trabalho, os *matchers* aplicados ao processo de MO são *matchers* linguísticos cujas comparações possuem o mesmo custo computacional, uma vez que utilizam o mesmo algoritmo de similaridade linguística (por exemplo, distância de *Levenshtein*). Por essa razão, é possível assumir que os custos das comparações entre as propriedades dos conceitos são similares. Dessa forma, a técnica refinada de balanceamento de carga é capaz de distribuir uniformemente as comparações entre os nós da infraestrutura computacional, baseada na quantidade de comparações realizadas em cada par de conceitos.

4.5 Considerações Finais

Neste capítulo foi apresentada a abordagem para o Matching de Ontologias baseado em Particionamento e Paralelismo (MOPP). A abordagem MOPP utiliza o *framework* MR como solução programável para paralelizar o processo de MO e está dividida em quatro etapas: leitura, *map*, *shuffle* e *reduce*. A etapa de leitura tem como objetivo ler os conceitos contidos nos pares de subontologias e calcular a quantidade de comparações a serem realizadas para cada par de conceitos. Na etapa de *map*, as chaves dos conceitos são definidas a fim de guiar a formação dos blocos de comparação. A etapa de *shuffle* é responsável por particionar, ordenar e agrupar os pares chave-valor $\langle key, conceito \rangle$, oriundos da etapa de *map*, com o intuito de formar os blocos de comparação. Por fim, os conceitos são comparados na etapa de *reduce*. No próximo capítulo, serão avaliados o desempenho e a eficácia da abordagem MOPP em diferentes cenários experimentais.

Capítulo 5

Avaliação da Abordagem MOPP

Como parte da avaliação da abordagem proposta, foram realizados experimentos com o objetivo de analisar o desempenho e eficácia da abordagem proposta. A Seção 5.1 apresenta uma visão geral da avaliação da abordagem MOPP e as questões de pesquisa que auxiliaram a elaboração dos experimentos. Na Seção 5.2, é descrito um experimento comparativo envolvendo a abordagem MOPP (proposta neste trabalho) com a técnica básica de balanceamento de carga [4; 3], a abordagem MOPP com a técnica refinada de balanceamento de carga e uma abordagem estado da arte [12]. O experimento relatado na Seção 5.3 visa analisar o desempenho da abordagem proposta com a utilização das duas técnicas de balanceamento de carga em diferentes cenários de desbalanceamento de carga. A Seção 5.4 descreve um experimento que avalia o impacto da utilização de técnicas de particionamento de ontologias no processo de OM, no que diz respeito à eficácia e ao desempenho do processo. No experimento abordado na Seção 5.5 é analisada a taxa de replicação de conceitos na abordagem proposta com a utilização das duas técnicas de balanceamento de carga. Por fim, o experimento detalhado na Seção 5.6 analisa o comportamento da abordagem MOPP em um cenário no qual são realizadas bilhões de comparações e empregados múltiplos *matchers* no processo de MO.

5.1 Visão Geral da Avaliação

Neste capítulo, a implementação[1] da abordagem MOPP proposta neste trabalho é avaliada em uma infraestrutura computacional. Os cinco experimentos abordam respectivamente as seguintes questões de pesquisa:

- Com relação ao desempenho, a abordagem MOPP é comparável às abordagens estado da arte existentes? (Objetivo de pesquisa definido na Equação 4.2)
- A técnica refinada de balanceamento de carga é capaz de melhorar a eficiência da abordagem MOPP? (Equação 4.3)
- Os algoritmos de particionamento reduzem o tempo de execução, sem comprometer a eficácia? (Equação 4.1)
- A técnica refinada de balanceamento de carga pode reduzir a quantidade de conceitos replicados na abordagem MOPP? (Equação 4.2)
- A abordagem proposta é escalável em um contexto onde são utilizados múltiplos matchers? (Equação 4.2)

Para responder a essas perguntas, os seguintes experimentos foram realizados.

5.2 Análise de Desempenho de Abordagens de *Matching* de Ontologias

Esta seção descreve o experimento comparativo envolvendo a abordagem MOPP com balanceamento de carga básico, a abordagem MOPP com balanceamento de carga refinado e a abordagem para MO em paralelo proposta por Gross [12]. O experimento realiza uma análise comparativa relacionada ao desempenho das abordagens.

[1]Disponível em: https://sites.google.com/site/dqgroupufcg/mapreduce-approach-for-matching-large-scale-ontologies

5.2.1 Planejamento do Experimento

Neste experimento o objetivo é avaliar comparativamente, no que diz respeito ao desempenho, três abordagens: a abordagem MOPP com balanceamento refinado de carga, a abordagem MOPP com balanceamento básico de carga e a abordagem proposta por Gross [12]. A abordagem de Gross possui arquitetura própria para paralelizar processo de OM, composta por módulos responsáveis pela leitura das ontologias, distribuição dos pares de conceitos entre os nós e comparação dos pares de conceitos.

Por outro lado, a abordagem MOPP proposta neste trabalho utiliza o *framework* MapReduce como solução programável para paralelização do processo de MO, como descrito no Capítulo 4. As duas técnicas de balanceamento de carga (básica e refinada) aplicadas à abordagem diferem na forma como analisam os pares de conceitos, com o intuito de auxiliar a distribuição uniforme dos pares de conceitos entre os nós. A técnica básica de balanceamento de carga considera apenas a quantidade de pares de conceitos a serem comparados, com o intuito de orientar a distribuição dos pares de conceitos entre os nós, de forma que cada nó receba uma quantidade similar de pares de conceitos a serem comparados. Por outro lado, a técnica refinada de balanceamento de carga analisa as múltiplas comparações realizadas para cada par de conceitos para orientar a distribuição dos pares de conceitos entre os nós, dessa forma cada nó realiza um número similar de comparações.

Como ontologias de entrada, foi utilizado o mesmo par de ontologias do experimento de Gross [12]: *Molecular Functions* (MF) e *Biological Processes* (BP), que possuem 9.395 e 17.104 conceitos, respectivamente. Ambas as ontologias foram extraídas da ontologia GO (Subseção 2.1).

Para uma comparação justa, as três abordagens devem executar a mesma quantidade de comparações, por isso, uma vez que a abordagem de Gross realiza as comparações entre conceitos guiadas pelo produto Cartesiano, ignoramos a etapa de particionamento da abordagem MOPP neste experimento.

Os *matchers* linguísticos aplicados neste experimento são: *Label* (compara os rótulos de conceitos), *Children* (compara os rótulos dos subconceitos), e *NamePath* (compara os rótulos dos conceitos ancestrais). Nos *matchers* linguísticos, a semelhança entre os rótulos foi determinada pelo algoritmo Trigrama.

Os experimentos foram executados em uma infraestrutura computacional com quatro

nós, cada um com 4 núcleos (totalizando 16 núcleos na infraestrutura). Cada nó possui um processador Intel (R) Xeon (R) CPU 4x1.0GHz, 4GB de memória e executa o sistema operacional Debian GNU/Linux OS de 64 bits com uma JVM de 64 bits.

Com relação ao MapReduce, foi utilizada a implementação Hadoop[2] 2.6.0 configurada com duas tarefas de *map* e quatro tarefas de *reduce* por nó (seguindo a configuração fornecida pela documentação do Hadoop).

5.2.2 Variáveis

Neste experimento foram avaliadas duas variáveis: tempo de execução e *speedup*. O tempo de execução é refente ao tempo, em segundos, que as abordagens levaram para executar o processo de MO em paralelo. No que diz respeito ao *speedup*, essa variável mede o quão mais rápido um processo é executado em paralelo quando comparado com o tempo do mesmo processo em sequencial. Em outras palavras, o *speedup* é dado pela razão entre o tempo necessário para executar um processo em um único nó (ou núcleo) e o tempo necessário para executar o mesmo processo em n nós, onde n é a quantidade de nós. Quando o valor do *speedup* obtido é igual ao número (n) de nós, o *speedup* é considerado ideal. Porém, o *speedup* ideal raramente ocorre em cenários reais, uma vez que, para paralelizar um processo, existem acréscimos relacionados ao tempo de execução devido ao tempo necessário para configurar os nós e distribuir as comparações entre eles.

5.2.3 Resultados e Discussão

A Figura 5.1 ilustra os resultados da análise comparativa de desempenho das três abordagens. Quanto ao tempo de execução, a abordagem MOPP com balanceamento refinado de carga superou as abordagens MOPP com balanceamento básico e Gross para todas as variações no número de núcleos. Além disso, permitiu uma redução no tempo de execução de 38.650 segundos (com um núcleo) para 2.539 segundos (16 núcleos). A abordagem MOPP com balanceamento refinado chegou a reduzir o tempo de execução em até 24% em relação à abordagem de Gross e 14% em relação à abordagem MOPP com balanceamento básico.

No que diz respeito ao *speedup*, ilustrado na Figura 5.3, inicialmente (entre 1 e 4 núcleos)

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

os valores das três abordagens permaneceram bem próximos. Porém, à medida em que o número de núcleos computacionais aumentaram, a diferença de *speedup* das duas abordagens MOPP em relação à abordagem de Gross aumentou consideravelmente. No cenário com 12 núcleos computacionais, as duas abordagens MOPP atingiram um *speedup* de aproximadamente 11, bem próximo ao *speedup* ideal (12), enquanto a abordagem de Gross obteve um *speedup* de apenas 9. Podemos ainda ressaltar o bom desempenho obtido pela abordagem MOPP com balanceamento refinado de carga, que obteve valores de *speedup* bem próximos ao *speedup* ideal para todas as variações no número de núcleos.

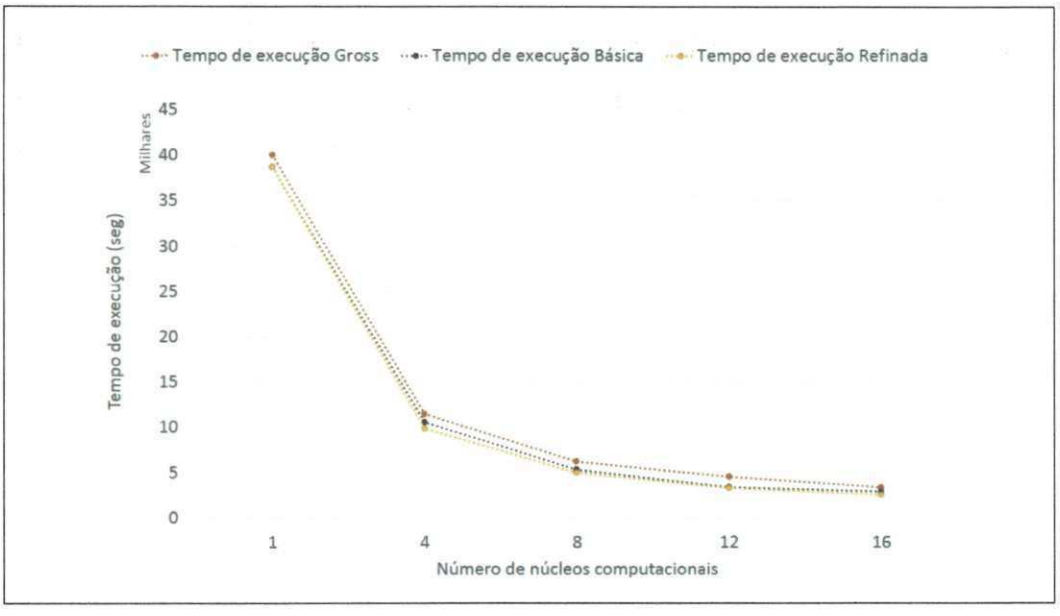

Figura 5.1: Tempo de execução da abordagem MOPP com balanceamento refinado de carga, MOPP com balanceamento básico de carga e Gross

Com o intuito de realizar uma análise mais aprofundada com relação ao tempo de execução das três abordagens, utilizamos a Figura 5.2 para ilustrar os tempos de execução obtidos pelas abordagens no intervalo entre 4 e 16 núcleos. Analisando comparativamente a abordagem de Gross com a abordagem proposta neste trabalho (MOPP), é possível perceber uma diferença considerável entre os tempos de execução. No cenário onde são utilizados 12 núcleos computacionais, o tempo de execução da abordagem MOPP (com balanceamento refinado de carga) chega a ser aproximadamente 20 minutos inferior ao tempo da abordagem proposta por Gross.

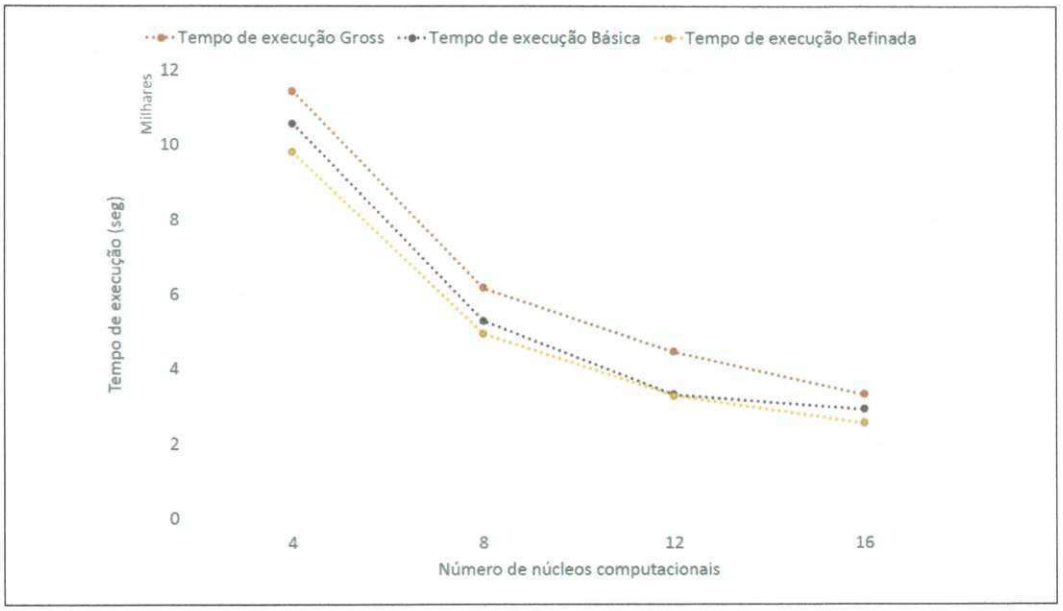

Figura 5.2: Tempo de execução da abordagem MOPP com balanceamento refinado de carga, MOPP com balanceamento básico de carga e Gross para o intervalo de 4 a 16 núcleos.

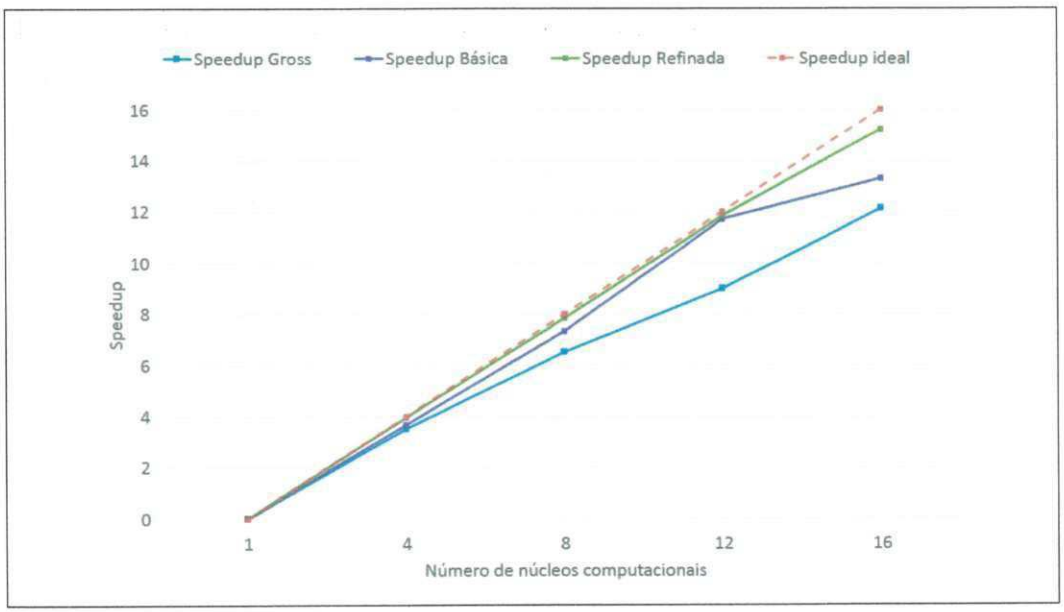

Figura 5.3: *Speedup* da abordagem MOPP com balanceamento refinado de carga, MOPP com balanceamento básico de carga e Gross

A diferença entre os tempos de execução pode ser justificada por dois fatores: a utilização do MapReduce como solução programável e técnicas de balanceamento de carga. Como as abordagens MOPP são baseadas em MapReduce, problemas relacionados à tolerância a falhas e transmissão dos dados já são tratados na própria arquitetura MapReduce. Por outro lado, a arquitetura proposta por Gross não mencionou como esses problemas são sanados. No que diz respeito às técnicas de balanceamento de carga, o fato da abordagem de Gross não utilizar uma técnica de balanceamento de carga certamente impactou no tempo de execução da abordagem. Dessa forma, as abordagens MOPP foram beneficiadas e obtiveram tempos de execução inferiores à abordagem de Gross.

Ao comparar os tempos de execução das abordagens MOPP com balanceamento refinado de carga e MOPP com balanceamento básico de carga, é possível perceber que, com exceção do cenário no qual são utilizados 16 núcleos, o aumento no número de núcleos diminui a diferença entre as duas abordagens MOPP. Isso ocorre porque, à medida em que o número de núcleos é aumentado, é possível distribuir os pares de conceitos entre mais tarefas de *reduce*. Uma vez que os pares de conceitos estão melhor distribuídos, as chances são menores de que uma tarefa de *reduce* execute uma quantidade muito alta de comparações quando comparadas com as demais, beneficiando a abordagem com o balanceamento básico de carga. Quando são utilizados 12 núcleos, temos o melhor cenário (com relação ao desempenho) para abordagem MOPP com balanceamento básico de carga. Isso é motivado por uma distribuição mais uniforme das comparações entre os nós, mesmo que a técnica de balanceamento de carga aplicada nessa abordagem não leve em consideração a quantidade de comparações executadas em cada par de conceitos.

Com base nos resultados experimentais, conclui-se que as duas abordagens MOPP obtiveram um desempenho, tanto no tempo de execução como no *speedup*, superior à abordagem de Gross para todos os números de núcleos computacionais. Com relação as técnicas de balanceamento de carga aplicadas na abordagem MOPP, é possível concluir que a técnica refinada promoveu um desempenho superior à técnica básica, para todas as variações no número de núcleos computacionais.

5.2.4 Dificuldades para Realização do Experimento

A principal dificuldade enfrentada na realização deste experimento está relacionada à falta de replicabilidade dos experimentos realizados em outros trabalhos relacionados. Embora vários trabalhos [12; 2; 35; 40; 36] tenham sido abordados nos trabalhos relacionados (Capítulo 3), foi possível comparar apenas a abordagem de Gross [12] com a abordagem proposta neste trabalho. Os demais trabalhos não forneceram informações (por exemplo, configurações dos ambientes distribuídos, códigos-fonte e pseudocódigos) suficientes ou omitiram partes importantes da implementação das abordagens, impossibilitando uma análise comparativa com a abordagem MOPP.

5.3 Estudo do Impacto do Balanceamento de Carga no Desempenho do MOPP

O objetivo desta seção é descrever o experimento relacionado ao impacto, no que diz respeito ao desempenho, das duas técnicas de balanceamento de carga aplicadas à abordagem MOPP.

5.3.1 Planejamento da Investigação Experimental

Este experimento avalia a abordagem MOPP com a utilização das duas técnicas de balanceamento de carga propostas neste trabalho: básica e refinada. O objetivo principal deste experimento é avaliar o desempenho da abordagem em diferentes cenários de desbalanceamento das comparações.

Para variar a quantidade de comparações de cada par de conceitos, a quantidade de propriedades em cada conceito foi modificada. Para este fim, as ontologias de entrada foram criadas sinteticamente, de tal maneira que a quantidade de subconceitos (relação *is-a*) contidos em cada conceito da ontologia pudesse ser alterada. Foram utilizados rótulos de conceitos de ontologias reais para nomear os rótulos dos conceitos das ontologias sintéticas.

Para gerar as ontologias de entrada, foi fixada a quantidade de conceitos de ambas as ontologias e, em apenas uma ontologia, foi variada a quantidade de subconceitos (relação *is-a*) em cada conceito. A *taxa de concentração* é a variável responsável por determinar a quantidade de conceitos que possuem subconceitos. Essa taxa varia entre 0 e 1, onde 0 indica

que um único conceito contém todos os subconceitos e 1 indica que todos os conceitos da ontologia possuem subconceitos, de maneira que cada conceito possui a mesma quantidade de subconceitos.

Para realizar este experimento, uma ontologia com 30.000 conceitos e outra com 60.000 conceitos foram geradas. Na primeira ontologia, foi fixada a quantidade de subconceitos em 20.000, uniformemente distribuídos entre 10.000 conceitos da ontologia (2 subconceitos por conceito). Para a segunda ontologia, foi fixada a quantidade de subconceitos em 50.000, para serem distribuídos entre os outros 10.000 conceitos da ontologia. A variável *taxa de concentração* influencia a quantidade de subconceitos em cada conceito da ontologia. Por exemplo, para uma *taxa de concentração* de 0,5 (50%), metade dos conceitos (5.000) deve possuir subconceitos. Dessa forma, os 50.000 subconceitos são distribuídos uniformemente entre os 5.000 conceitos, ou seja, cada conceito deve possuir 10 subconceitos.

Os *matchers* linguísticos aplicados neste experimento são *Label* e *Children*. Os experimentos foram realizados utilizando a mesma infraestrutura computacional do experimento anterior (Seção 5.2).

5.3.2 Variáveis

Com o objetivo de avaliar o desempenho da abordagem, foi utilizada a variável tempo de execução que calcula o tempo (em segundos) necessário para abordagem concluir o processo. Para medir o desbalanceamento de carga entre as tarefas de *reduce* para as técnicas refinada e básica de balanceamento de carga, foi calculado o desvio padrão da quantidade de comparações realizadas em cada tarefa de *reduce*.

5.3.3 Resultados e Discussão

Os resultados da análise comparativa entre as duas técnicas de balanceamento de carga são apresentados na Figura 5.4. Para o cenário onde a *taxa de concentração* = 0 (totalmente desbalanceado), o tempo de execução e o desvio-padrão são os mesmos para ambas as técnicas. No entanto, este cenário dificilmente ocorreria com o uso de ontologias reais, uma vez que dificilmente existirá uma ontologia na qual um único conceito possui como subconceitos todos os demais conceitos da ontologia. Mesmo em um cenário extremamente desbalanceado

(por exemplo, *taxa de concentração* = 0,1), a técnica refinada reduziu o desvio-padrão de dezenas de milhões para centenas de milhares de comparações. Além disso, a técnica refinada obteve um desvio padrão quase constante, mesmo aumentando a *taxa de concentração*. Por outro lado, a técnica básica reduziu o desvio padrão gradualmente, à medida em que a *taxa de concentração* era aumentada.

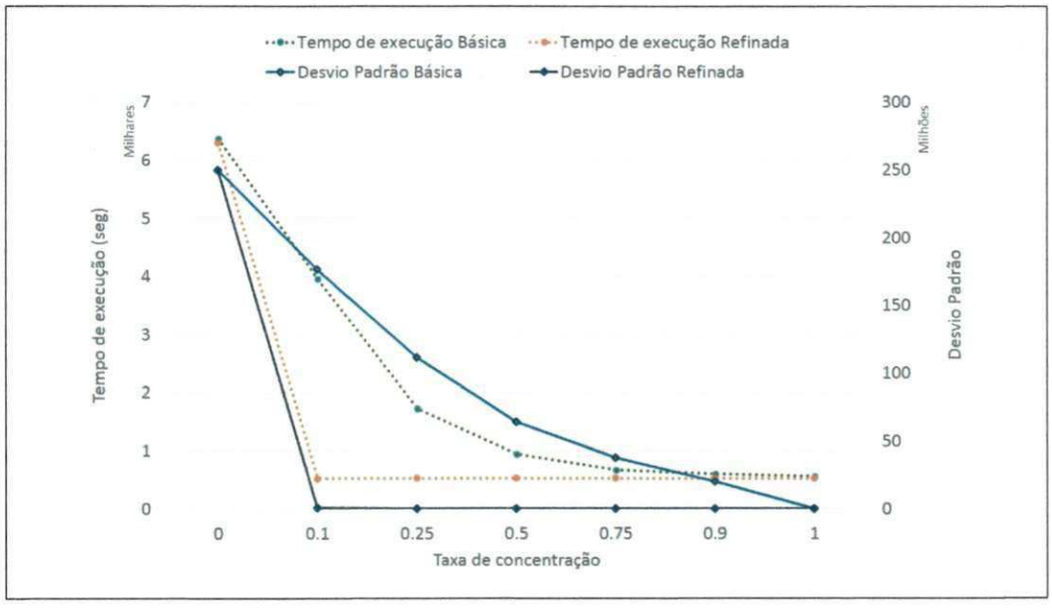

Figura 5.4: Tempo de execução e desvio padrão da abordagem com a aplicação da técnica básica e da técnica refinada de balanceamento de carga

Tanto em relação ao tempo de execução como ao desvio-padrão, a técnica refinada apresentou resultados praticamente contantes a partir da *taxa de concentração* = 0.1. Isso é reflexo do balanceamento de carga promovido pela técnica refinada, uma vez que mesmo em cenários onde alguns pares de conceitos potencialmente causariam um elevado grau de desbalanceamento de carga, a técnica consegue balancear a quantidade de comparações realizadas em cada tarefa de *reduce*.

Para a *taxa de concentração* = 1 (totalmente balanceado), o tempo de execução e o desvio padrão foram os mesmos para ambas as técnicas de balanceamento de carga. Isso ocorre devido ao fato de todos os pares de conceitos executarem a mesma quantidade de comparações. Consequentemente, isso beneficia a técnica básica de balanceamento de carga, que analisa a quantidade de pares de conceitos a serem comparados. É importante destacar, neste

experimento, os resultados significativos obtidos pela técnica refinada, que chegou a reduzir o tempo de execução em até 87% em relação à técnica básica. Com o aumento da *taxa de concentração*, os tempos de execução de ambas as técnicas tendem a se aproximar, uma vez que a quantidade de comparações realizadas em cada par de conceitos também se aproxima. Com base nos resultados obtidos, é possível concluir que a abordagem com técnica refinada de balanceamento de carga atingiu um desempenho superior ou igual à abordagem com a técnica básica de balanceamento de carga.

5.4 Análise de Desempenho e Eficácia da Abordagem com (e sem) Particionamento

Nesta seção serão abordados os experimentos relacionados ao impacto da utilização (ou não) de técnicas de particionamento no desempenho e eficácia da abordagem proposta neste trabalho.

5.4.1 Planejamento da Investigação Experimental

Neste experimento, é avaliado se a etapa de particionamento pode melhorar o tempo de execução, mantendo a eficácia dos alinhamentos gerados. Para isto, a abordagem MOPP foi executada com e sem a etapa de particionamento das ontologias. No caso da abordagem sem particionamento, as comparações foram guiadas pelo produto Cartesiano. As ontologias de entrada utilizadas neste experimento são *Adult Mouse Anatomy* e *NCI Thesaurus* que possuem 2.744 e 3.304 conceitos, respectivamente. Ambas as ontologias são fornecidas pela OAEI[3], uma vez que essas ontologias são utilizadas em uma das *tracks* da competição. Para determinar a similaridade entre conceitos, são aplicados três *matchers* linguísticos: *Label*, *Children* e *Annotation* (compara as anotações contidas nos conceitos). As semelhanças linguísticas dos rótulos e das anotações foram determinadas calculando a média dos resultados obtidos a partir dos algoritmos de distância de *Levenshtein* e Trigrama.

O algoritmo PAP, descrito na Subseção 2.2.1, foi usado para particionar as ontologias de entrada, de maneira que foram gerados dois pares de subontologias. As subontologias

[3]http://oaei.ontologymatching.org/2015/anatomy/index.html

resultantes da ontologia *NCI Thesaurus* continham 1.256 e 1.379 conceitos. Por sua vez, os subontologias geradas a partir da ontologia *Adult Mouse Anatomy* possuíam 1.440 e 1.228 conceitos. No algoritmo PAP, os conceitos que não foram incluídos em nenhuma das subontologias a serem comparadas, não foram considerados no processo de *matching*.

Os experimentos foram realizados em uma infraestrutura computacional com 32 nós (cada um com um núcleo computacional), onde a configuração de cada nó era a mesma do experimento descrito na Seção 5.3. No que diz respeito ao MapReduce, foi utilizada a mesma implementação dos experimentos anteriores, no entanto, configurado com uma tarefa de *map* para cada dois nós e uma tarefa de *reduce* por nó, tendo em vista que cada nó possuía apenas um núcleo computacional.

5.4.2 Variáveis

Para medir o desempenho da abordagem com (e sem) a etapa de particionamento, é utilizada a variável tempo de execução que calcula o tempo (em segundos) gasto pela abordagem para concluir o processo. Para a abordagem com particionamento, o tempo de execução inclui tanto o tempo do particionamento quanto o tempo para executar o *matching*.

No que diz respeito à eficácia dos alinhamentos, é utilizada a métrica *F-measure*. Para entender melhor essa métrica, é preciso compreender outras duas métricas: precisão e cobertura. Precisão é a fração de correspondências verdadeiras que foram identificadas com relação a quantidade total de correspondências identificadas entre as ontologias. Cobertura é a fração de correspondências verdadeiras identificadas com relação ao total de correspondências verdadeiramente existentes entre as ontologias. A avaliação das métricas precisão e cobertura é feita comparando o alinhamento resultante do *matching* das ontologias com o alinhamento de referência, uma espécie de "gabarito" com as correspondências entre conceitos verdadeiramente similares. Por fim, a métrica *F-measure* é utilizada para mensurar a qualidade dos alinhamentos neste experimento, dada pela Equação 5.1.

$$F - measure = 2 * \frac{precisão * cobertura}{precisão + cobertura} \tag{5.1}$$

5.4.3 Resultados e Discussão

De acordo com os resultados mostrados na Figura 5.5, é possível identificar uma redução considerável no tempo de execução e uma pequena diminuição na eficácia (*F-measure*) quando o algoritmo de particionamento é utilizado. As ontologias de entrada foram particionadas em sequencial (usando apenas um nó), levando aproximadamente 2 segundos para concluir o particionamento.

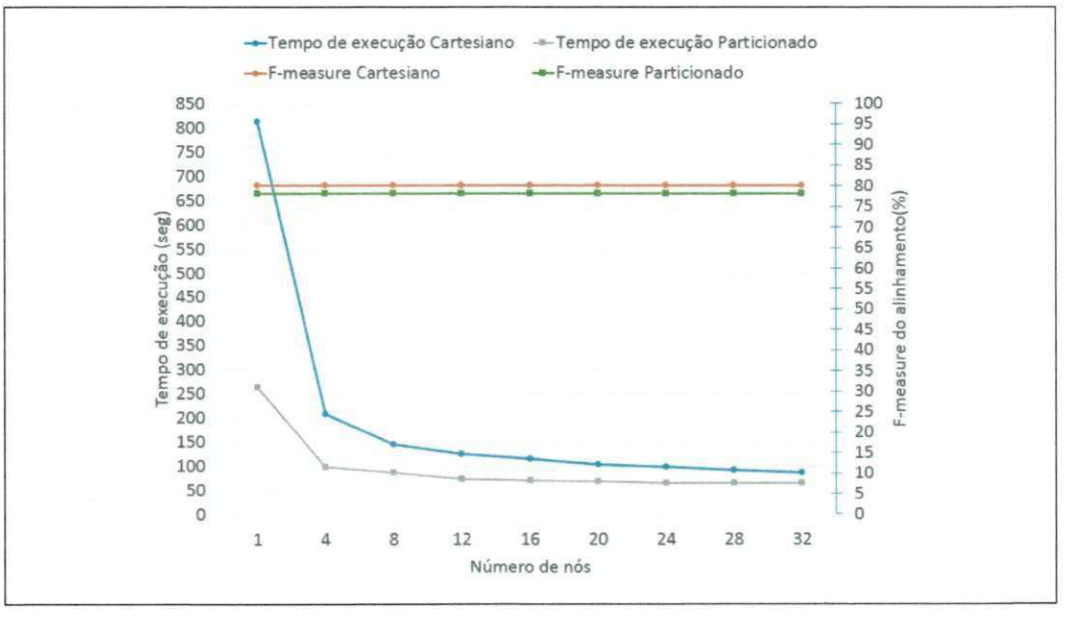

Figura 5.5: Tempo de execução e *F-measure* da abordagem MOPP com (e sem) a etapa de particionamento.

Para o cenário com 4 nós, o tempo de execução da abordagem sem o algoritmo de particionamento é o dobro em relação ao tempo de execução da abordagem com o algoritmo de particionamento. Essa significativa redução no tempo de execução é resultado da diminuição na quantidade de pares de conceitos a serem comparados, uma vez que a abordagem sem particionamento compara 9.066.176 pares de conceitos enquanto a abordagem com particionamento realiza a comparação de apenas 3.502.052 pares de conceitos. Nos cenários de 4 à 12 nós, o tempo de execução continuou diminuindo, porém de uma forma mais modesta, com um decréscimo de 70 segundos, em média. Após a execução com 12 nós, o tempo de execução de ambas as abordagens tenderam a se estabilizar, uma vez que o tempo necessário para configurar os nós sobrepôs o tempo de execução do *matching*. A partir desses dados, é

possível inferir que a utilização de mais de 12 nós é desnecessária, pois não existe ganho no desempenho para ambas as abordagens.

Em relação à eficácia do alinhamento gerado (neste experimento, utilizamos um limiar de similaridade $\phi = 0{,}8$), a abordagem sem particionamento (produto Cartesiano) obteve uma *F-measure* de 80% enquanto a abordagem com particionamento obteve uma *F-measure* de 78%. Assim, pode-se concluir que a utilização de um algoritmo de partição pode reduzir consideravelmente o tempo de execução, sem necessariamente comprometer a eficácia dos alinhamentos gerados.

5.5 Estudo do Impacto do Balanceamento de Carga na Taxa de Replicação de Conceitos

Esta seção descreve o experimento relacionado ao impacto, no que diz respeito à taxa de replicação de conceitos, das duas técnicas de balanceamento de carga aplicadas à abordagem MOPP.

5.5.1 Planejamento do Experimento

O objetivo deste experimento é avaliar a influência da utilização das duas técnicas (básica e refinada) de balanceamento de carga na quantidade de conceitos replicados na abordagem MOPP.

Em um cenário onde apenas um par de subontologias é submetido ao processo de *matching*, os conceitos da subontologia menor são replicados e enviados para todas as tarefas de *reduce*. Por essa razão, independente da técnica de balanceamento de carga aplicada, a quantidade de conceitos replicados é a mesma. Quando mais de um par de subontologias é submetido ao processo de *matching*, a taxa de replicação de conceitos é influenciada diretamente pela quantidade de blocos de comparação, uma vez que cada bloco de comparação é enviado para uma tarefa de *reduce*.

Os pares de subontologias utilizados neste experimento são os mesmos do experimento anterior: as subontologias resultantes da ontologia *NCI Thesaurus* (com 1.256 e 1.379 conceitos) e as subontologias geradas a partir da ontologia *Adult Mouse Anatomy* (com 1.440 e

1.228 conceitos). Neste experimento, foram empregadas as mesmas configurações da infra-estrutura computacional e do MapReduce aplicadas no experimento anterior (Seção 5.4).

5.5.2 Variáveis

Para avaliar a taxa de replicação de conceitos da abordagem MOPP com a aplicação das técnicas básica e refinada de balanceamento de carga, é calculada a quantidade de conceitos replicados.

5.5.3 Resultados e Discussão

A partir dos resultados deste experimento, apresentados na Figura 5.6, é possível perceber uma redução significativa na quantidade de conceitos replicados com a utilização da técnica refinada de balanceamento de carga.

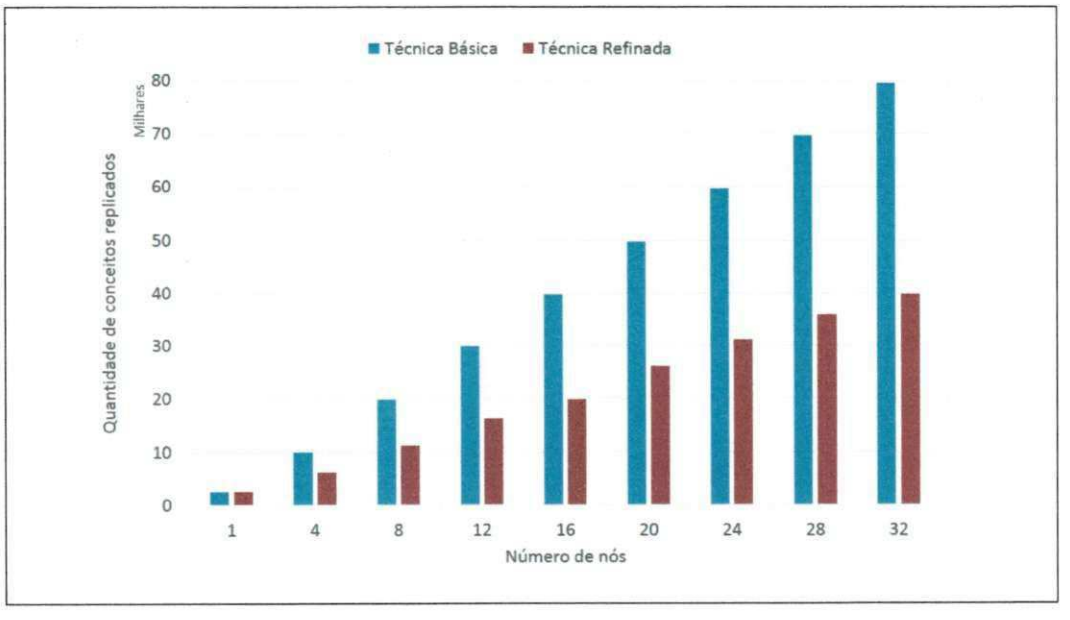

Figura 5.6: Taxa de replicação de conceitos da abordagem MOPP com balanceamento básico de carga e MOPP com balanceamento refinado de carga

Considerando que na técnica refinada os blocos de comparação resultantes de um par de subontologias são enviados para a menor quantidade possível de tarefas de *reduce*, a quantidade de conceitos replicados tende a reduzir.

No cenário com apenas um nó, a quantidade de conceitos replicados é a mesma para ambas as técnicas de balanceamento de carga, uma vez que não existe necessidade de replicação dos conceitos. Em todos os cenários do experimento (variando o número de nós), a abordagem MOPP com a técnica refinada replica menos conceitos do que a abordagem MOPP com a técnica básica. Ao utilizar 32 nós, a abordagem MOPP com a técnica refinada reduz pela metade a quantidade de conceitos replicados em relação à abordagem MOPP com a técnica básica.

Baseado nos resultados obtidos, é possível concluir que a técnica refinada de balanceamento de carga promove uma redução na quantidade de conceitos replicados quando comparada com a técnica básica.

5.6 Análise de Desempenho e Eficácia da Abordagem no Contexto de Múltiplos *Matchers*

Nesta seção será detalhado o experimento que visa analisar o desempenho e a eficácia da abordagem MOPP no contexto da utilização de múltiplos *matchers*.

5.6.1 Planejamento da Investigação Experimental

Neste experimento, foi avaliado o impacto de combinar múltiplos *matchers* na eficácia e no desempenho da abordagem MOPP. Com o intuito de analisar um cenário em que bilhões de pares de conceitos são comparados (elevado tempo de execução e consumo de recursos), todos os conceitos de uma ontologia foram comparados com todos os conceitos da outra ontologia (produto Cartesiano). O seguinte par de ontologias foi usado no experimento: *Foundational Model of Anatomy* (FMA) e National Cancer Institute Thesaurus (NCI), que possuem 66.724 e 78.989 conceitos, respectivamente. Essas ontologias também são fornecidas pela OAEI e são utilizadas pela competição na *track* de grandes ontologias.

Foram avaliados o tempo de execução e a eficácia da abordagem MOPP usando o *matcher Label* e a combinação dos *matchers Label* e *Annotation*. A semelhança dos rótulos e das anotações foi determinada pelo algoritmo de distância de *Levenshtein*. A combinação dos *matchers* foi realizada executando em primeiro lugar o *matcher Label*, caso o valor de se-

melhança fosse igual ou maior do que o limiar de similaridade (ϕ = 0,8), os conceitos eram considerados correspondentes e o *matcher Annotation* não era executado. Caso contrário, o *matcher Annotation* era executado, verificando se alguma das anotações (dos conceitos) possuía um valor de similaridade igual ou maior do que limiar de similaridade. Neste experimento, foram empregadas as mesmas configurações da infraestrutura computacional e do MapReduce aplicadas no experimento da Seção 5.4.

5.6.2 Variáveis

Neste experimento, foi utilizada a variável tempo de execução para mensurar o tempo (em segundos) gasto pela abordagem para executar o processo. No que diz respeito à qualidade dos alinhamentos gerados, foi empregada a métrica *F-measure* (descrita na Subseção 5.4.2) para avaliar os resultados.

5.6.3 Resultados e Discussão

De acordo com a Figura 5.7, a diferença no tempo de execução entre o *matcher Label* e a combinação dos *matcher Label* e *Annotation* tende a diminuir à medida em que o número de nós aumenta. Com apenas um nó, a diferença de tempo de execução é 3.960 segundos, com 32 nós essa diferença é reduzida para apenas 163 segundos.

Quanto à eficácia dos alinhamentos, a combinação dos *matchers Label* e *Annotation* produz um aumento de 6% na *F-measure*, saindo de 65% para 71%. Claramente, a combinação de múltiplos *matchers* tende a melhorar a eficácia dos alinhamentos gerados pelo processo de OM. No entanto, a sua utilização aumenta significativamente o tempo de execução do processo. É importante destacar que a combinação dos *matchers Label* e *Annotation* possui um custo computacional superior a utilização apenas do *matcher Label*. Mesmo assim, a abordagem MOPP possibilitou a aproximação do tempo de execução em ambos os cenários, deste modo, a abordagem MOPP surge como uma solução escalável que beneficia a utilização de múltiplos *matchers* mantendo tempos de execução razoáveis.

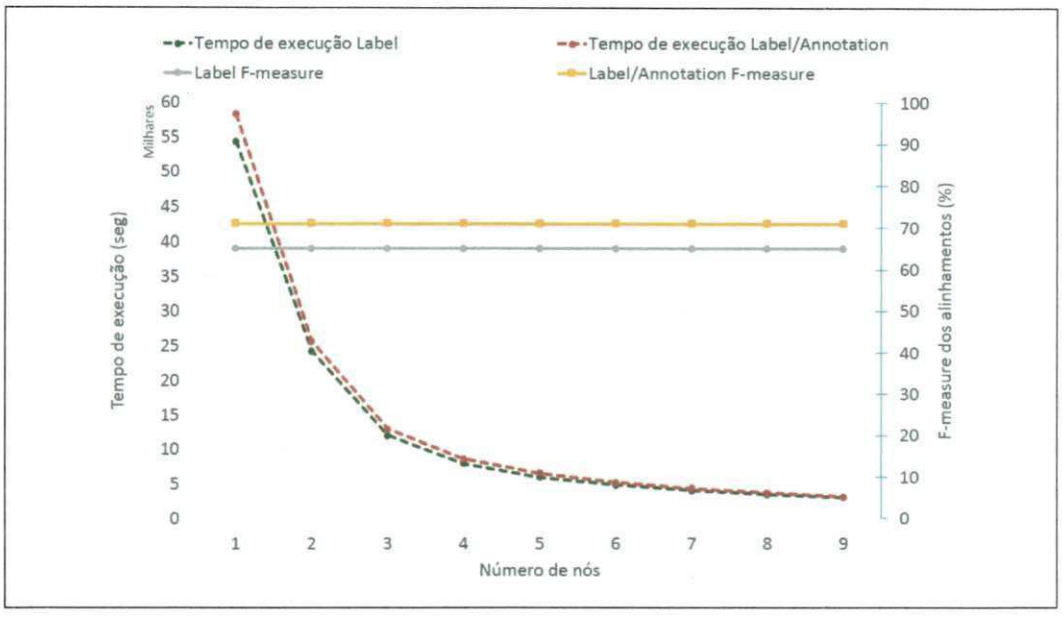

Figura 5.7: Desempenho e eficácia da abordagem MOPP usando os *matchers Label* (apenas) e *Label/Annotation* (combinados).

5.7 Considerações Finais

Os experimentos apresentados neste capítulo tiveram o intuito de avaliar a abordagem MOPP em diferentes cenários experimentais. Inicialmente, foi realizado uma análise comparativa da abordagem MOPP com a abordagem estado da arte proposta por Gross. Posteriormente, a abordagem MOPP foi submetida a um experimento que avaliou seu desempenho, utilizando as técnicas de balanceamento de carga propostas neste trabalho, em cenários que variavam o grau de desbalanceamento das comparações nos pares de conceitos. O terceiro experimento teve o objetivo de analisar o impacto da utilização (ou não) de técnicas de particionamento no desempenho e eficácia da abordagem MOPP. O objetivo do quarto experimento foi avaliar a taxa de replicação de conceitos na abordagem MOPP com a utilização das duas técnicas de balanceamento de carga. Por fim, o último experimento avaliou o desempenho e a eficácia da abordagem MOPP no contexto da utilização de múltiplos *matchers*. Com base nos resultados apresentados neste capítulo, foi possível chegar a algumas conclusões referentes à abordagem MOPP. Essas conclusões serão apresentadas e discutidas no capítulo seguinte.

Capítulo 6

Conclusões

As ontologias vêm sendo cada vez mais utilizadas como forma de modelar o conhecimento em diferentes domínios (Saúde, Agricultura, Astronomia). Com o crescimento da quantidade de informações nesses domínios, cresce também a quantidade de conceitos existentes nessas ontologias, dando origem às grandes ontologias. Dessa forma, é importante reiterar a relevância do processo de *Matching* de Ontologias (MO), bem como as dificuldades e a complexidade atreladas ao processo, sobretudo no que diz respeito ao *matching* de grandes ontologias, no qual existe um elevado tempo de execução e consumo de memória.

Este trabalho propõe uma abordagem para o processo de *Matching* de Ontologias baseado em Particionamento e Paralelismo (MOPP), com o intuito de prover eficiência ao processo de MO. A abordagem utiliza MapReduce como solução programável para paralelizar o processo de MO, mais precisamente a etapa de *matching* das subontologias (Figura 2.2).

Inicialmente, as ontologias de entrada são submetidas à etapa de particionamento, utilizando o algoritmo de particionamento PAP [15]. Como resultado da etapa de particionamento, são gerados pares de subontologias a serem comparadas, que serão utilizados como entrada da abordagem MOPP. A utilização de algoritmos de particionamento promove ganhos no que diz respeito ao desempenho da abordagem, uma vez que esses algoritmos reduzem a quantidade de pares de conceitos a serem comparados, como foi descrito no experimento da Seção 5.4.

Com o intuito de otimizar o consumo dos recursos da infraestrutura computacional e melhorar o desempenho da abordagem MOPP, este trabalho ainda propõe a utilização de duas técnicas de balanceamento de carga (básica [4; 3] e refinada). Essas técnicas analisam

os pares de subontologias (entradas da abordagem) e calculam a quantidade de pares de conceitos a serem comparados e a quantidade de comparações que serão executadas para cada par de conceitos. Tal análise auxilia a distribuição dos pares de conceitos entre os nós da infraestrutura, minimizando o problema de desbalanceamento de carga.

Outro ponto que merece destaque no presente trabalho é seu caráter inovador. Primeiramente, como apresentado no Capítulo 3, existem poucos trabalhos que abordam o processo de MO em paralelo [12; 2; 35; 40], dentre eles, alguns ainda estão em andamento [2; 4]. Além disso, a abordagem MOPP não apenas é executada em paralelo, como também aplica técnicas de particionamento de ontologias com o intuito de melhorar ainda mais o desempenho do processo de MO. Em segundo lugar, dentre as abordagens de MO em paralelo, a abordagem MOPP é uma das poucas que utiliza o modelo programático de computação distribuída MapReduce, o qual desponta como um recurso poderoso para potencializar o desempenho do processo de MO.

Por fim, este trabalho ainda propõe duas técnicas de balanceamento de carga para abordagens em paralelo, sobretudo as que utilizam MapReduce. Dentre as técnicas de balanceamento de carga, podemos ressaltar a técnica refinada de balanceamento, capaz de analisar a quantidade de comparações a serem executadas para cada par de conceitos com o objetivo de nivelar a quantidade de comparações executadas em cada nó da infraestrutura computacional. Essas técnicas de balanceamento de carga conseguiram aumentar a eficiência da abordagem MOPP, como descrito no experimento da Seção 5.3.

A avaliação de desempenho e eficácia, realizada em ambiente distribuído real e utilizando ontologias reais e sintéticas, mostrou que a abordagem MOPP é escalável com o aumento do número de nós disponíveis, sobretudo quando são utilizadas grandes ontologias. A abordagem MOPP com a utilização da técnica refinada de balanceamento de carga também se mostrou robusta na presença de ontologias que causam um elevado grau de desbalanceamento. Além disso, a abordagem MOPP foi comparada com abordagem de Gross (estado da arte) e verificou-se que a abordagem MOPP supera a abordagem proposta por Gross em termos de desempenho.

6.1 Trabalhos Futuros

No que tange aos trabalhos futuros, é possível estender a abordagem MOPP nos seguintes aspectos: propor abordagens para executar o particionamento de ontologias em paralelo, propor técnicas de balanceamento de carga baseadas no custo computacional dos *matchers* e propor estratégias para estimar (semi-)automaticamente a quantidade ideal de nós para execução do processo de MO em paralelo.

Propor abordagens para executar o particionamento de ontologias em paralelo. Após o levantamento referente aos algoritmos estado da arte para particionamento de ontologias, com o objetivo de selecionar um dentre esses para gerar as entradas (pares de subontologias) da abordagem MOPP, não foi encontrado nenhum trabalho em paralelo capaz de gerar pares de subontologias. Embora essa oportunidade de pesquisa já tenha sido relatada nos próprios trabalhos envolvendo particionamento de ontologias [15], esse problema tornou-se real no contexto dessa pesquisa, no qual são utilizadas grandes ontologias. Para grandes ontologias, as abordagens de particionamento existentes são ineficazes no ponto de vista de desempenho, chegando a levar muitas horas ou dias para terminar. Em muitos casos, o particionamento sequer foi concluído, pois o consumo de memória ultrapassou o limite disponível. Dessa forma, um possível trabalho futuro é propor uma abordagem que realize o particionamento de ontologias em paralelo. Além disso, essa técnica de particionamento poderia ser acoplada à abordagem MOPP proposta neste trabalho com o objetivo de propor uma nova abordagem MOPP na qual mais etapas seriam realizadas em paralelo.

Propor técnicas de balanceamento de carga baseadas no custo computacional dos *matchers*. Tendo em vista que comumente o processo de MO utiliza diferentes tipos de *matchers* como, por exemplo, *matchers* semânticos e estruturais. Quando isso acontece, dificilmente é possível assumir que o custo de comparação de um *matcher* linguístico é o mesmo de um *matcher* estrutural [11]. Dessa forma, estratégias para mensurar o custo computacional das comparações realizadas por diferentes tipos de *matchers* são úteis às técnicas de balanceamento de carga, no ponto de vista prático. As técnicas de balanceamento de carga podem utilizar os custos computacionais das comparações para auxiliar na distribuição das comparações entre os nós. As técnicas estáticas de balanceamento de carga analisam os custos das comparações com o intuito de distribuir de forma igualitária as comparações (carga

de trabalho) entre os nós da infraestrutura. Como resultado, é possível estender a técnica refinada de balanceamento de carga de maneira que o balanceamento igualitário entre os nós seja feito independentemente dos tipos de *matchers* utilizados.

Propor estratégias para estimar a quantidade ideal de nós para execução do processo de MO em paralelo. Em alguns experimentos realizados neste trabalho, foi possível verificar que mesmo aumentando a quantidade de nós na infraestrutura, não eram obtidos ganhos consideráveis de desempenho. Por essa razão, um outro trabalho futuro é propor uma abordagem com o objetivo de analisar diferentes variáveis (por exemplo, configuração dos nós disponíveis, tamanho das ontologias e *matchers* a serem aplicados) com o intuito de determinar (semi-)automaticamente a melhor configuração da infraestrutura para o cenário em questão. Estratégias como essas tem o objetivo de otimizar o consumo de recursos disponíveis na infraestrutura, evitando o desperdício dos mesmos. Se levarmos em consideração infraestruturas pagas (por exemplo, Azure, Amazon, Google Cloud, entre outras), essas estratégias ganham mais destaque tendo em vista que impactam diretamente no custo financeiro da execução do processo, como descrito em [25].

Referências Bibliográficas

[1] Alsayed Algergawy, Sabine Massmann, and Erhard Rahm. A clustering-based approach for large-scale ontology matching. In *Advances in Databases and Information Systems*, pages 415–428. Springer, 2011.

[2] Muhammad Bilal Amin, Wajahat Ali Khan, Sungyoung Lee, and Byeong Ho Kang. Performance-based ontology matching. *Applied Intelligence*, pages 1–30, 2015.

[3] Tiago Brasileiro Araújo and Carlos Eduardo Pires. Uma abordagem para matching de grandes ontologias usando mapreduce. In *Workshop de Teses e Dissertações em Banco de Dados (WTDBD-2015), SBBD-2015, Petrópolis, Brasil*, pages 260–266, 2015.

[4] Tiago Brasileiro Araújo, Carlos Eduardo Pires, Thiago Pereira da Nobrega, and Dimas C Nascimento. A parallel approach for matching large-scale ontologies. *Journal of Information and Data Management*, 6(1):18, 2015.

[5] Samira Babalou, Mohammad Javad Kargar, and Seyyed Hashem Davarpana. A Comprehensive Review Of The Ontology Matching Systems By A Focus On Large Ontologies. In *International Congress of Chemical and Process Engineering*, pages 357–373, Prague, Czech Republic, 2014.

[6] Kenneth Baclawski and Todd Schneider. The open ontology repository initiative: Requirements and research challenges. In *Proceedings of workshop on collaborative construction, management and linking of structured knowledge at the ISWC*, page 18, 2009.

[7] Michelle Cheatham, Zlatan Dragisic, Jérôme Euzenat, Daniel Faria, Alfio Ferrara, Giorgos Flouris, Irini Fundulaki, Roger Granada, Valentina Ivanova, Ernesto Jiménez-Ruiz, et al. Results of the ontology alignment evaluation initiative 2015. In *10th ISWC workshop on ontology matching (OM)*, pages 60–115. No commercial editor., 2015.

[8] Michelle Cheatham and Pascal Hitzler. String similarity metrics for ontology alignment. In *The Semantic Web–ISWC 2013*, pages 294–309. Springer, 2013.

[9] Peter Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media, 2012.

[10] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[11] Jérôme Euzenat, Pavel Shvaiko, et al. *Ontology matching*, volume 333. Springer, 2007.

[12] Anika Gross, Michael Hartung, Toralf Kirsten, and Erhard Rahm. On matching large life science ontologies in parallel. In *Data Integration in the Life Sciences*, pages 35–49. Springer, 2010.

[13] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[14] Benjamin Gufler, Nikolaus Augsten, Angelika Reiser, and Alfons Kemper. Load balancing in mapreduce based on scalable cardinality estimates. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 522–533. IEEE, 2012.

[15] Fayçal Hamdi, Brigitte Safar, Chantal Reynaud, and Haifa Zargayouna. Alignment-based partitioning of large-scale ontologies. In *Advances in knowledge discovery and management*, pages 251–269. Springer, 2010.

[16] Michael Hartung, Lars Kolb, Anika Groß, and Erhard Rahm. Optimizing similarity computations for ontology matching-experiences from gomma. In *Data Integration in the Life Sciences*, pages 81–89. Springer, 2013.

[17] Hesham A Hefny, Mohamed Helmy Khafagy, and M Wahdan Ahmed. Comparative study load balance algorithms for map reduce environment. *International Journal of Computer Applications*, 106(18):41–50, 2014.

[18] Wei Hu, Ningsheng Jian, Yuzhong Qu, and Yanbing Wang. Gmo: A graph matching for ontologies. In *Proceedings of K-CAP Workshop on Integrating Ontologies*, pages 41–48, 2005.

[19] Wei Hu, Yuzhong Qu, and Gong Cheng. Matching large ontologies: A divide-and-conquer approach. *Data & Knowledge Engineering*, 67(1):140–160, 2008.

[20] Wei Hu, Yuanyuan Zhao, and Yuzhong Qu. Partition-based block matching of large class hierarchies. *Lecture Notes in Computer Science*, 4185:72, 2006.

[21] Lars Kolb, Andreas Thor, and Erhard Rahm. Load balancing for mapreduce-based entity resolution. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 618–629. IEEE, 2012.

[22] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. Skewtune in action: mitigating skew in mapreduce applications. *Proceedings of the VLDB Endowment*, 5(12):1934–1937, 2012.

[23] YongChul Kwon, Kai Ren, Magdalena Balazinska, Bill Howe, and Jerome Rolia. Managing skew in hadoop. *IEEE Data Eng. Bull.*, 36(1):24–33, 2013.

[24] Demetrio Gomes Mestre and Carlos Eduardo Pires. Efficient entity matching over multiple data sources with mapreduce. *Journal of Information and Data Management*, 5(1):40, 2014.

[25] Dimas C Nascimento, Carlos Eduardo Pires, and Demetrio Gomes Mestre. A data quality-aware cloud service based on metaheuristic and machine learning provisioning algorithms. 2015.

[26] Markus Nentwig, Michael Hartung, Axel-Cyrille Ngonga Ngomo, and Erhard Rahm. A survey of current link discovery frameworks. *Semantic Web Journal*, 2015.

[27] DuyHoa Ngo, Zohra Bellahsene, and Konstantin Todorov. Opening the black box of ontology matching. In *The Semantic Web: Semantics and Big Data*, pages 16–30. Springer, 2013.

[28] Lorena Otero-Cerdeira, Francisco J Rodríguez-Martínez, and Alma Gómez-Rodríguez. Ontology matching: A literature review. *Expert Systems with Applications*, 42(2):949–971, 2015.

[29] Erhard Rahm. Towards large-scale schema and ontology matching. In *Schema matching and mapping*, pages 3–27. Springer, 2011.

[30] Smriti R Ramakrishnan, Garret Swart, and Aleksey Urmanov. Balancing reducer skew in mapreduce workloads using progressive sampling. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 16. ACM, 2012.

[31] Anny Kartika Sari and Wenny Rahayu. Ontology-based change propagation in shareable health information applications.

[32] K Saruladha, G Aghila, and B Sathiya. A comparative analysis of ontology and schema matching systems. *International Journal of Computer Applications*, 34(8):14–21, 2011.

[33] Nico Schlitter, Tanja Falkowski, and Jörg Lässig. Dengraph-ho: a density-based hierarchical graph clustering algorithm. *Expert Systems*, 31(5):469–479, 2014.

[34] Pavel Shvaiko and Jérôme Euzenat. Ontology matching: state of the art and future challenges. *Knowledge and Data Engineering, IEEE Transactions on*, 25(1):158–176, 2013.

[35] Uthayasanker Thayasivam and Prashant Doshi. Speeding up batch alignment of large ontologies using mapreduce. In *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on*, pages 110–113. IEEE, 2013.

[36] Ana I Torre-Bastida, Esther Villar-Rodriguez, Javier Del Ser, David Camacho, and Marta Gonzalez-Rodriguez. On interlinking linked data sources by using ontology matching techniques and the map-reduce framework. In *Intelligent Data Engineering and Automated Learning–IDEAL 2014*, pages 53–60. Springer, 2014.

[37] Rares Vernica, Michael J Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 495–506. ACM, 2010.

[38] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas AJ Schweiger. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD*

international conference on Knowledge discovery and data mining, pages 824–833. ACM, 2007.

[39] Miaomiao Yu, Junli Wang, and Xiaodong Zhao. A pam-based ontology concept and hierarchy learning method. *Journal of Information Science*, 2013.

[40] Hang Zhang, Wei Hu, and Yu-zhong Qu. Vdoc+: a virtual document based approach for matching large ontologies using mapreduce. *Journal of Zhejiang University SCIENCE C*, 13(4):257–267, 2012.

[41] Hang Zhang, Wei Hu, and Yuzhong Qu. Constructing virtual documents for ontology matching using mapreduce. In *The Semantic Web*, pages 48–63. Springer, 2012.

Apêndice A

Pseudocódigos

A.1 Leitura

Conforme descrito no Algoritmo A.1, a Matriz de blocos de Comparação (MBC) é criada, de maneira que o seu tamanho é definido a partir da quantidade de tarefas de *reduce* disponíveis (linha 2). Cada linha da MBC representa um bloco de comparação no formato: $\langle\langle par\_id, bloco\_comparação\_id\rangle; reduce\_id; conjunto\_de\_conceitos\rangle$, onde $\langle par\_id, bloco\_comparação\_id\rangle$ representa a chave bloco de comparação, par_id identifica o par de subontologias, $bloco\_comparação\_id$ é o identificador do bloco de comparação com base no *fator de fatiamento*, $reduce\_id$ é o identificador da tarefa de *reduce* e $conjunto\_de\_conceitos$ é um conjunto composto pelos conceitos da subontologia maior que serão enviados para a mesma tarefa de *reduce* (com base no $reduce\_id$).

A variável *quantidadeDeComparações*, que representa a quantidade total de comparações para todos os pares de subontologias, é definida como 0 (linha 3). A lista que representa os pares de subontologias é iterada (linha 4) e a quantidade de comparações realizadas em cada par é somada à variável *quantidadeDeComparações* (linha 5). Após a definição da quantidade total de comparações a serem realizadas, é possível calcular a média de comparações por tarefa de *reduce* (linha 7). A variável *chavePar* é utilizada para guiar a formação da chave par_id na MBC (linha 8) e a variável *alocacaoDasComparacoes* representa uma matriz que armazena a distribuição das comparações para cada par de subontologias (linha 9).

A lista dos pares de subontologias é iterada novamente, dessa vez, com o intuito de dis-

tribuir as comparações a serem realizadas em cada par de conceitos (linhas 10 a 17). Para isso, as subontologias são extraídas do par (linhas 11 e 12) e armazenadas em variáveis. Posteriormente é calculada a quantidade de comparações a ser realizada neste par (linha 13). O *fator de fatiamento* é calculado a partir da divisão da quantidade de comparações a ser realizada no par de subontologias pela média de comparações por tarefa de *reduce* (linha 14). O objetivo do *fator de fatiamento* é guiar a distribuição das comparações entre as tarefas de *reduce* baseada na quantidade de comparações. Dessa forma, o método *DistribuirComparações()* (Algoritmo A.2) distribui as comparações entres as tarefas de *reduce* que serão utilizadas pelo par de subontologias em questão (linha 15). Como resultado do método *DistribuirComparações()*, é gerada uma matriz contendo apenas os blocos de comparação do par de subontologias em questão, que é armazenada na matriz *alocacaoDasComparacoes*. Ao fim da iteração, a variável *chavePar* é incrementada (linha 16).

Por fim, a MBC é gerada a partir da matriz *alocacaoDasComparacoes*. Inicialmente, a matriz *alocacaoDasComparacoes* é ordenada (linha 18) pela quantidade de comparações. Posteriormente, a matriz *alocacaoDasComparacoes* é iterada (linhas 19 a 22). Durante a iteração, é identificado o índice da tarefa de *reduce* (representada na MBC) com a menor quantidade de comparações alocadas (linha 20) para que o bloco de comparação contido na matriz *alocacaoDasComparacoes* seja alocado para a tarefa de *reduce* em questão (linha 21).

Código Fonte A.1: Algoritmo Leitura

```
1  Leitura(paresDeSubontologias, numReduces){
2    MBC = criarMatriz(numReduces)
3    quantidadeComparacoes = 0
4    for par in paresDeSubontologias do{
5      quantidadeComparacoes = quantidadeComparacoes +
           calculaQuantidadeDeComparacoes(subOntologia1, subOntologia2)
6    }
7    mediaPorReduce = quantidadeComparacoes/numReduces
8    chavePar = 0
9    alocacaoDasComparacoes = {}
10   for par in paresDeSubontologias do{
11     subOntologia1 = par[0]
12     subOntologia2 = par[1]
13     comparacoesNoPar = calculaQuantidadeDeComparacoes(subOntologia1,
```

```
        subOntologia2)
14      fatorDeFatiamento = comparacoesNoPar/mediaPorReduce
15      alocacaoDasComparacoes.adiciona(DistribuirComparacoes(subOntologia1;
            subOntologia2;fatorDeFatiamento;chavePar))
16      chavePar = chavePar + 1
17    }
18    ordenar(alocacaoDasComparacoes)
19    for blocoComparacao in alocacaoDasComparacoes do{
20      indice = recuperarReduceMaisVazia(MBC)
21      MBC.recuperar(indice).adicionar(blocoComparacao)
22    }
23  }
```

No Algoritmo A.2, a distribuição das comparações entre as tarefas de *reduce* é ilustrada. Inicialmente, os conceitos, a quantidade de conceitos e a quantidade de propriedades contidas nas subontologias são extraídos (linhas 2 a 7) e armazenados em variáveis. A variável *mapaDeConceitos* (linha 8) armazena a quantidade de comparações a serem realizadas que envolvem um determinado conceito, denotado pelo par $\langle conceito, quantidade\ de\ comparações \rangle$. A *matrizAlocacaoPorPar* (linha 9) armazena os blocos de comparação do par de subontologias em questão.

Caso a primeira subontologia do par seja maior (em quantidade de conceitos) do que a segunda (linha 10), os conceitos contidos na primeira subontologia são iterados (linhas 11). Para cada conceito da primeira subontologia, é calculada a quantidade total de comparações que envolvem o conceito em questão e todos os demais da segunda subontologia (linha 12). Em seguida, o par $\langle conceito, quantidade\ de\ comparações \rangle$ é adicionado ao *mapaDeConceitos* (linha 13). O *mapaDeConceitos* é ordenado para ser submetido ao algoritmo responsável pela alocação das comparações para as tarefas de *reduce*, ilustrado pelo algoritmo *AlocarComparacoes()* (Algoritmo A.3). Caso contrário, os conceitos da segunda subontologia são iterados (linhas 16). Semelhante ao processo anterior, para cada conceito da segunda subontologia, também é calculada a quantidade total de comparações (linha 17) e o par $\langle conceito, quantidade\ de\ comparações \rangle$ é adicionado ao *mapaDeConceitos* (linha 18). Os pares contidos no *mapaDeConceitos* são ordenados de forma decrescente pela quantidade de comparações (linha 21). Então, as informações referentes à alocação das comparações (li-

nha 22) são armazenadas na *matrizAlocacaoPorPar*, que será retornada (linha 23) no fim do algoritmo.

Código Fonte A.2: Algoritmo de Distribuição das Comparações

```
1  DistribuirComparacoes(subOntologia1, subOntologia2, fatorDeFatiamento,
       chavePar){
2    conceitosSub1 = extrairConceitos(subOntologia1)
3    quantidadeConceitosSub1 = recuperarQuantidadeConceitos(subOntologia1)
4    quantidadePropriedadesSub1 = recuperarQuantidadePropriedades(
         subOntologia1)
5    conceitosSub2 = extrairConceitos(subOntologia2)
6    quantidadeConceitosSub2 = recuperarQuantidadeConceitos(subOntologia2)
7    quantidadePropriedadesSub2 = recuperarQuantidadePropriedades(
         subOntologia2)
8    mapaDeConceitos = {}
9    matrizAlocacaoPorPar = {}
10   if quantidadeConceitosSub1 > quantidadeConceitosSub2 then{
11     for conceitoSub1 in conceitosSub1 do{
12       comparacoes = calculaComparacoes(recuperarQuantidadePropriedades(
             conceitoSub1),quantidadePropriedadesSub2)
13       mapaDeConceitos.adicionar(conceitoSub1, comparacoes)
14     }
15   } else {
16     for conceitoSub2 in conceitosSub2 do{
17       comparacoes = calculaComparacoes(recuperarQuantidadePropriedades(
             conceitoSub2), quantidadePropriedadesSub1)
18       mapaDeConceitos.adicionar(conceitoSub2, comparacoes)
19     }
20   }
21   ordenar(mapaDeConceitos)
22   AlocarComparacoes(mapaDeConceitos, chavePar, matrizAlocacaoPorPar)
23   return matrizAlocacaoPorPar
24 }
```

No Algoritmo A.3, é realizada a alocação das comparações entre as tarefas de *reduce*. A quantidade de tarefas de *reduce* que serão utilizadas pelo par de subontologias é determinado pelo *fatorDeFatiamento*. A *matrizComparacoesPorReduce* é responsável por armazenar a

quantidade de comparações já alocadas para cada tarefa de *reduce* (linha 2). A *matrizConceitosNasReduces* é responsável por armazenar os conceitos oriundos da subontologia maior que irão compor o bloco de comparação a ser enviado para a tarefa de *reduce* (linha 3). É realizada uma iteração que vai de 0 até a quantidade de tarefas de *reduce* a serem utilizadas pelo par de subontologias, dado pelo valor do *fatorDeFatiamento*. Essa iteração tem o objetivo de armazenar uma linha vazia, que representa cada tarefa de *reduce*, nas variáveis *matrizComparacoesPorReduce* e *matrizConceitosNasReduces* (linhas 4 a 7).

Posteriormente, é realizada uma iteração no *mapaDeConceitos*, que contém todos os conceitos da subontologia maior e a quantidade de comparações que envolvem cada um dos conceitos (linhas 8 a 12). O método *recuperarReduceMaisVazia()* é responsável por identificar o índice da linha da matriz com menos comparações alocadas, a partir da *matrizComparacoesPorReduce* (linha 9). Com base nesse índice, o conceito é inserido na linha de mesmo índice da *matrizConceitosNasReduces* (linha 10) e a quantidade de comparações é somada ao valor contido na linha da *matrizComparacoesPorReduce* de mesmo índice (linha 11).

Uma nova iteração é realizada, iniciando do valor 0 até a quantidade de tarefas de *reduce*, determinada pela variável *fatorDeFatiamento* (linhas 13 a 15). O objetivo dessa iteração é compor a *matrizAlocacaoPorPar*, que irá armazena os blocos de comparação do par de subontologias. Dessa forma, para cada iteração, é inserida uma nova linha na *matrizAlocacaoPorPar* (linha 14), que representa um bloco de comparação no mesmo formato das linhas da MBC ($\langle\langle par\_id, bloco\_comparação\_id\rangle; reduce\_id; conjunto\_de\_conceitos\rangle$).

Código Fonte A.3: Algoritmo de Alocação das Comparações entre as Tarefas de *Reduce*

```
1  AlocarComparacoes(mapaDeConceitos, fatorDeFatiamento, chavePar,
       matrizAlocacaoPorPar){
2      matrizComparacoesPorReduce = {}
3      matrizConceitosNasReduces = {}
4      for i=0 to fatorDeFatiamento do{
5          matrizComparacoesPorReduce.adicionar({})
6          matrizConceitosNasReduces.adicionar({})
7      }
8      for conceito in mapaDeConceitos do{
9          indice = recuperarReduceMaisVazia(matrizComparacoesPorReduce)
10         matrizConceitosNasReduces.recuperar(indice).adicionar(conceito
```

```
            [0])
11          matrizComparacoesPorReduce.recuperar(indice).alterar(
                matrizComparacoesPorReduce.recuperar(indice) + conceito[1])
12      }
13      for i=0 to fatorDeFatiamento do{
14          matrizAlocacaoPorPar.adicionar(chavePar.i, i,
                matrizConceitosNasReduces.recuperar(i))
15      }
16  }
```

A.2 Map

Como ilustrado no Algoritmo A.4, a chave do conceito é inicialmente subdividida nas variáveis: par_id, conceito_id e replica (linhas 2 a 4). Se o conceito não deve ser replicado (linha 5), o algoritmo irá percorrer todos os blocos de comparação do par de subontologias (*par_id)* (linha 6) até identificar o bloco de comparação que contém o conceito no *conjunto de conceitos* (linha 7). A partir do bloco de comparação, é possível recuperar o identificador da tarefa de *reduce* (linha 8), uma vez que cada bloco de comparação armazena informações no seguinte formato: $\langle\langle par\_id$,*bloco_comparação_id*$\rangle; reduce\_id; conjunto\_conceitos\rangle$ onde $\langle par\_id$,*bloco_comparação_id*$\rangle$ representa a chave bloco de comparação, par_id identifica o par de subontologias, *bloco_comparação_id* é o identificador do bloco de comparação, $reduce\_id$ é o identificador da tarefa de *reduce* e $conjunto\_conceitos$ é um conjunto composto pelos conceitos da subontologia maior que serão enviados para a mesma tarefa de *reduce* (com base no $reduce\_id$). A variável *reduce* é anexada à chave (linha 9) e o par $\langle chave, conceito\rangle$ é emitido para a fase seguinte (linha 10). Caso contrário, o conceito é replicado (linha 13) pela quantidade de blocos de comparações (linha 14) e o *conceito_id* da chave é alterado (linha 15). O identificador da tarefa de *reduce* é recuperado a partir do bloco de comparação (em MBC) (linha 16) e anexado à chave (linha 17). Por fim, o par $\langle chave, conceito\rangle$ é emitido (linha 18).

Código Fonte A.4: Algoritmo *Map*

```
1  Map(chave, conceito){
2      par_id = chave[0]
```

```
3       conceito_id = chave[1]
4       replica = chave[2]
5       if replica == 0 then{
6           for i= 0 to (MBC.recuperar(par_id).tamanho()) do{
7               if MBC.recuperar(par_id.i).recuperarConjuntoConceitos().
                    contem(conceito)then{
8                   reduce = MBC.recuperar(par_id.i).recuperarReduceId()
9                   chave = chave.anexar(reduce)
10                  emitir(chave, conceito)
11              }
12          }
13      } else {
14          for i=0 to MBC.recuperar(par_id).tamanho() do{
15              chave = par_id.i.replica
16              reduce = MBC.recuperar(par_id.i).recuperarReduceId()
17              chave = chave.anexar(reduce)
18              emitir(chave, conceito)
19          }
20      }
21  }
```

A.3 Reduce

Conforme detalhado no Algoritmo A.5, o valor do limiar é recuperado a partir da variável global (*configuracao*) (linha 2) e as demais variáveis são inicializadas (linhas 3 e 4). A lista de conceitos é iterada (linhas 5 a 22) e os conceitos que pertencem à ontologia maior são separados dos conceitos pertencentes à ontologia menor (linhas 6 a 9). A variável *maxSimilaridade* (linha 10) indica o maior valor de similaridade entre os conceitos que já foram comparados. Para cada conceito da ontologia maior (linha 11), é calculado o valor de similaridade com os conceitos da ontologia menor (linha 12). Se a similaridade for maior do que *limiar* e a *maxSimilaridade*, a variável *maxSimilaridade* recebe o novo valor de similaridade e o *resultado* assume o novo par de conceitos com o valor de similaridade entre eles (linhas 13 a 16). Se existe um par de conceitos com o valor de similaridade acima do *limiar* (linha 18), o *resultado* é emitido na saída da tarefa de *reduce* (linha 19).

Código Fonte A.5: Algoritmo *Reduce*

```
1  Reduce(chave, lista(conceitos)){
2      limiar = configuracao.recuperar(valorLimiar)
3      conceitosDaOntologiaMaior = {}
4      resultado = null
5      for conceito in lista(conceitos) do{
6          if conceito.naoReplicado() then{
7              conceitosDaOntologiaMaior.adicionar(conceito)
8          }else{
9              conceitoDaOntologiaMenor = conceito
10             maxSimilaridade = 0
11             for conceitoMaior in conceitosDaOntologiaMaior do{
12                 similaridade = computaMatchers(conceitoMaior,
                       conceitoDaOntologiaMenor)
13                 if (similaridade > limiar) and (similaridade >
                       maxSimilaridade) then{
14                     maxSimilaridade = similaridade
15                     resultado = conceitoMaior x conceitoDaOntologiaMenor:
                           maxSimilaridade
16                 }
17             }
18             if resultado != null then{
19                 emitir(chave, resultado)
20             }
21         }
22     }
23 }
```