

UNIVERSIDADE FEDERAL DO PARANÁ

AILTON SERGIO BONIFACIO

UMA ABORDAGEM SEMÂNTICA BASEADA EM ONTOLOGIA PARA SINTONIA DE
PARÂMETROS: UM ESTUDO DE CASO PARA MAPREDUCE

CURITIBA PR

2014

AILTON SERGIO BONIFACIO

UMA ABORDAGEM SEMÂNTICA BASEADA EM ONTOLOGIA PARA SINTONIA DE
PARÂMETROS: UM ESTUDO DE CASO PARA MAPREDUCE

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Ciência da Computação no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Prof. Dr. Fabiano Silva.

CURITIBA PR

2014

Catálogo na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

B715a

Bonifacio, Ailton Sergio

Uma abordagem semântica baseada em ontologia para sintonia de parâmetros: um estudo de caso para MapReduce [recurso eletrônico] / Ailton Sergio Bonifacio. – Curitiba, 2014.

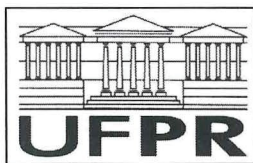
Tese - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2014.

Orientador: Fabiano Silva .

1. Framework (Programa de computador). 2. Processamento eletrônico de dados. 3. Hadoop. 4. MapReduce. I. Universidade Federal do Paraná. II. Silva, Fabiano. III. Título.

CDD: 005.36

Bibliotecário: Elias Barbosa da Silva CRB-9/1894



Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa do aluno de Doutorado em Ciência da Computação, Ailton Sergio Bonifacio, avaliamos a tese de doutorado intitulada “*Uma abordagem semântica baseada em ontologia para sintonia de parâmetros: um estudo de caso para MapReduce*”, cuja defesa pública foi realizada no dia 15 de outubro de 2014, às 09:00 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após avaliação, decidimos pela:

☒ **aprovação** do candidato. () **reprovação** do candidato.

Curitiba, 15 de outubro de 2014

Prof. Dr. Fabiano Silva
DINF/UFPR – Orientador

Prof. Dr. André Luis Andrade Menolli
UENP – Membro Externo

Prof. Dr. Clodis Boscarioli
UNIOESTE – Membro Externo

Prof. Dr. Luis Carlos Erpen de Bona
DINF/UFPR – Membro Interno

Prof. Dr. Marcos Sfair Sunyé
DINF/UFPR – Membro Interno



Dedico este trabalho a minha família. Meus filhos: razão de todo meu esforço. Minha esposa: minha razão. Meus pais: razão da minha dedicação aos estudos. Em especial, dedico a maior incentivadora dos meus estudos. A minha primeira professora do primário: minha amada Mãe!

AGRADECIMENTOS

Uma vez mais, meu agradecimento maior é direcionado ao Senhor Deus Todo Poderoso. Foi Ele que me proporcionou a sabedoria necessária, o discernimento no momento correto e as forças mais profundas quando se fez necessário menter-se na caminhada. A Ti Deus, meu maior agradecimento pela conclusão deste trabalho. Todos os parentes, amigos, professores que participaram de uma forma ou de outra da feitura deste trabalho fica todo a minha gratidão. Aos meus pais, sempre! Por tudo e pelo sempre! A minha companheira Taís por suportar a distância e sempre me apoiar! Aos meus filhos, Caio e Amanda, meu enteado Kenzo, amores da minha vida e razão da minha luta diária! Aos meus amigos mais próximos: Ricardo, André, Luizão, Guilherme, Clévio, Everaldo, entre tantos outros. vocês fazem parte disso tudo! Em especial, agradeço a minha amiga e parceira Cristiane Yanase Hirabara por estar por perto sempre, e mesmo agora, tendo se ido tão cedo e tão precocemente, ainda permanece presente. Aos meus colegas e parceiros do LaRSis Ao meu orientador Fabiano Silva um agradecimento especial por confiar em mim e me deixar livre para alçar meu próprio vôo. Sem me esquecer de agradecer pela parceria de sempre! Aos professores e funcionários do Departamento de Informática da UFPR, pelo apoio, pelas orientações, pelas conversas, pelas trocas de ideia. Em especial aos professores Bona, Sunye, Didonet, Aldri e Eduardo Cunha, o meu muito obrigado! Por fim, meu agradecimento a Universidade Estadual de Londrina e aos meus amigos da PRORH pelo apoio e incentivo a mais esta capacitação.

RESUMO

MapReduce é um *framework* para processamento de grandes volumes de dados; e Hadoop é a sua implementação de código aberto mais conhecida e utilizada. Hadoop possui mais de uma centena de parâmetros de configuração e, destes, cerca de 29 parâmetros devem ser sintonizados, a fim de obter um melhor desempenho das aplicações a ele submetidas. No entanto, a melhor sintonização dos parâmetros não é uma tarefa muito simples de ser realizada. Além disso, o conhecimento para se saber quais os parâmetros que devem ser sintonizados e quais os melhores valores para eles só estão nos trabalhos científicos e livros que tratam especificamente deste assunto. Portanto, é necessária a criação de ferramentas ou ambientes que promovam e motivem o compartilhamento de informação e disseminação do conhecimento a respeito deste assunto. Dessa forma, é importante que todo o conhecimento adquirido seja organizado, com o objetivo de ser reutilizado o mais rápido, fácil e eficientemente possível, quando necessário. Sendo assim, este trabalho propõe uma abordagem semântica baseada em ontologia para a sintonia dos parâmetros de configuração do Hadoop MapReduce, com o objetivo de melhorar o desempenho das aplicações. Trata-se de um estudo que explora e visa compreender o *framework* Hadoop, seus parâmetros de configuração e as relações destes parâmetros com os padrões das cargas de trabalho que o utilizam. Para isso, foi elaborada uma revisão sistemática sobre o assunto e, baseada no conhecimento adquirido nesta revisão, foi modelada e implementada uma ontologia chamada OntoHadoop. O foco da ontologia é a representação do conhecimento semântico necessário para se obter a melhor sintonia dos parâmetros de configuração do Hadoop que têm influência sobre o desempenho das aplicações. O conhecimento semântico explora as características da carga de trabalho, bem como a taxonomia dos conceitos, propriedades, relações e axiomas da ontologia. Baseada na ontologia, foram definidas a arquitetura da abordagem e dos processos que criam as unidades de sintonia dos parâmetros. Por fim, a ontologia foi avaliada por consultas e inferências nas regras implementadas; e a abordagem semântica simulada como um ambiente computacional implementado. Os resultados obtidos ratificam o propósito de uma base de conhecimento sobre o assunto e se mostram promissores no sentido de indicarem sintonias relevantes para se alcançar um melhor desempenho das aplicações submetidas ao Hadoop. Isto é possível, principalmente, através de regras de caracterização de padrões das cargas de trabalho e regras de sintonia de parâmetros implementadas na ontologia. Além disso, a abordagem proposta direciona a tarefa de obtenção de sintonia de parâmetros rumo a Web Semântica.

Palavras-chave: Hadoop. MapReduce. Parâmetros de Configuração. Ontologia. Desempenho do Hadoop. Sintonia de Parâmetros.

ABSTRACT

MapReduce is a framework for processing large volumes of data; and Hadoop is the most know and used open source implementation. Hadoop has over a hundred configuration parameters and around 29 parameters should be tuned in order to obtain a better application performance. However, the best parameters tuning is not very simple task to be performed. Furthermore, the knowledge to know which parameters should be tuned and what the best values for them remain just in scientific papers and books which specifically dealing with this subject. Therefore, it is necessary to create tools or environments that will promote and motivate the information sharing and knowledge dissemination on this issue. Therefore, it is important that all knowledge acquired should be organized in order to be reused quickly, easily and efficiently as possible, when necessary. Thus, this work proposes an ontology-based semantic approach to tune the Hadoop MapReduce configuration parameters, aiming to improve application performance. This is a study that explores and aims to understand the Hadoop framework, its configuration parameters and the relationships of those parameters with the workload patterns that use it. For this, we developed a systematic review about it and, based on the knowledge acquired in this review, was modeled and implemented an ontology called OntoHadoop. The focus of ontology is the semantic knowledge representation needed to obtain the best Hadoop configuration parameters tuning that influence the performance of applications. Semantic knowledge explores the workload characteristics, as well as the taxonomy of concepts, properties, relations and axioms of the ontology. The approach architecture and the processes that create the units tuning parameters were defined based on ontology. Finally, the ontology was evaluated by queries and inferences in the implemented rules; and the simulated semantic approach as a computational environment implemented. The results confirm the purpose of a knowledge base about it and show promise towards indicate relevant tunings to achieve a better application performance. This is possible mainly through workload patterns rules and parameters tuning rules implemented in the ontology. Furthermore, the proposed approach directs the task of obtaining tuning parameters towards the Semantic Web.

Keywords: Hadoop. MapReduce. Configuration Parameters. Ontology. Hadoop Performance. Tuning Parameters.

LISTA DE FIGURAS

2.1	Arquitetura de Execução do MapReduce. Extraído de (22)	20
2.2	Arquitetura do HDFS. Extraído de (6)	21
4.1	Número de artigos científicos por parâmetro.. . . .	45
4.2	Número de artigos científicos por método de validação.	51
5.1	Mapa Conceitual para representação dos conceitos sobre os parâmetros de configuração do Hadoop MapReduce.	56
5.2	OntoHadoop.	60
5.3	Definição da propriedade de objeto <i>hasWorkloadPattern</i> no Protégé.	62
5.4	Definição da propriedade de dados <i>sizeInputJobKb</i> no Protégé.	63
5.5	Classe <i>dfs_block_size</i> e Hierarquia parcial das classes no Protégé.	64
5.6	Instâncias da classe <i>dfs_block_size</i> vistas no Protégé.	65
5.7	View para criação de regras SWRL no Protégé.	67
6.1	Arquitetura proposta para a Abordagem Semântica.	71
6.2	Esquema de Implementação da Abordagem Semântica.	75
6.3	Fluxograma demonstrando o fluxo dos processos para criação de Unidades de Sintonia.. . . .	77
7.1	Resultado de consulta SPARQL no ambiente Protégé.	84
7.2	Resultado de inferência via Pellet para o "Job2".. . . .	84
7.3	Informações de entrada do "Job3".	86
7.4	Inferências para o "Job3".. . . .	88
7.5	Inferências de configurações similares para o "Job3".	88
7.6	Tempos Total de execução do "Job3"em segundos por configuração.	91

LISTA DE TABELAS

4.1	Termos para busca	43
4.2	Bases de Dados pesquisadas na revisão sistemática	43
4.3	Parâmetros de Configuração.	44
4.4	Regras para composição de axiomas e propriedades da ontologia.	48
4.5	Principal Finalidade e Abordagem de Experimentação.	50
4.6	Métodos de Validação.	51
5.1	Principais conceitos e propriedades dos conceitos	58
5.2	Principais Relações da ontologia.. . . .	61
7.1	Parâmetros de Configuração para Simulação.	89

SUMÁRIO

1	INTRODUÇÃO	12
1.1	MOTIVAÇÃO	13
1.2	DESCRIÇÃO DOS PROBLEMAS	14
1.3	OBJETIVOS	14
1.4	MÉTODO DE PESQUISA E ABORDAGEM	16
1.5	ORGANIZAÇÃO DA TESE	16
2	HADOOP MAPREDUCE	18
2.1	MODELO DE PROGRAMAÇÃO	18
2.2	MODELO DE EXECUÇÃO	19
2.3	SISTEMA DE ARQUIVOS DISTRIBUÍDO DO HADOOP - HDFS	20
2.4	PARÂMETROS DE EXECUÇÃO DO HADOOP	21
2.5	MAPREDUCE E SEMÂNTICA	23
2.6	CONSIDERAÇÕES FINAIS	24
3	ONTOLOGIAS	25
3.1	METADADOS SEMÂNTICOS	25
3.2	ONTOLOGIAS E WEB SEMÂNTICA	26
3.2.1	Ontologias e Representação do Conhecimento	27
3.2.2	Tipos de Ontologias	28
3.2.3	Metodologias de Construção de Ontologias	29
3.3	LINGUAGENS E FERRAMENTAS PARA MANIPULAÇÃO DE ONTOLOGIAS	29
3.3.1	XML	30
3.3.2	RDF e RDF Schema	30
3.3.3	OWL - <i>Web Ontology Language</i>	31
3.3.4	Regras em OWL - SWRL	35
3.3.5	Ferramenta para Construção de Ontologias - Protégé	36
3.3.6	Raciocinadores Lógicos e Inferências	36
3.3.7	Linguagem de Consulta - SPARQL	38
3.4	CONSIDERAÇÕES FINAIS	39
4	REVISÃO SISTEMÁTICA DA LITERATURA - PARÂMETROS X DESEMPENHO	41
4.1	MÉTODO DE PESQUISA	41
4.1.1	Planejando a Revisão	42
4.1.2	Conduzindo a Revisão	42

4.2	RESULTADOS	44
4.2.1	Parâmetros de Configuração	44
4.2.2	Fases do Hadoop x Parâmetros	46
4.2.3	Características das Cargas de Trabalho x Parâmetros	46
4.2.4	Regras para Sintonia de Parâmetros	46
4.3	ANÁLISE DOS RESULTADOS	49
4.3.1	Considerações sobre as Questões	49
4.3.2	Principal Finalidade e Abordagem de Experimentação.	49
4.3.3	Métodos de Validação.	51
4.3.4	Classificação dos <i>Jobs</i>	52
4.4	SIMULADOR MRSIM.	53
4.5	CONSIDERAÇÕES FINAIS	53
5	ONTOHADOOP.	54
5.1	ESPECIFICAÇÃO	54
5.2	AQUISIÇÃO DO CONHECIMENTO	55
5.3	CONCEITUALIZAÇÃO	55
5.4	FORMALIZAÇÃO	59
5.5	AXIOMAS E RELAÇÕES	61
5.6	IMPLEMENTAÇÃO	62
5.6.1	Implementação das Propriedades de Objetos.	62
5.6.2	Implementação das Propriedades de Dados	63
5.6.3	Implementação das Classes	64
5.6.4	Implementação de Instâncias	65
5.6.5	Implementação de Regras SWRL.	66
5.7	CONSIDERAÇÕES FINAIS	69
6	ABORDAGEM SEMÂNTICA PARA SINTONIA DE PARÂMETROS . . .	70
6.1	ARQUITETURA DA ABORDAGEM	70
6.1.1	Camada de Aplicação.	70
6.1.2	Camada Mediadora	73
6.1.3	Camada de Sintonia.	73
6.1.4	Camada de Conhecimento	74
6.2	IMPLEMENTAÇÃO DA ABORDAGEM	74
6.2.1	Pré-Processamento	75
6.2.2	Gerador de Sintonia de Parâmetros	76
6.2.3	Módulo de Buscas	76
6.3	FLUXO DOS PROCESSOS PARA GERAÇÃO DE UNIDADES DE SINTONIA	77
6.4	CONSIDERAÇÕES FINAIS	80

7	AVALIAÇÃO EXPERIMENTAL.	81
7.1	AVALIAÇÃO DA ONTOHADOOP	81
7.1.1	Questões de Competência.	82
7.1.2	Regras e Inferências Obtidas	83
7.1.3	Análise dos Resultados	85
7.2	SIMULAÇÕES	85
7.3	CONSIDERAÇÕES FINAIS	92
8	CONSIDERAÇÕES FINAIS	93
8.1	RELEVÂNCIA	93
8.2	CONTRIBUIÇÕES DA PESQUISA	94
8.3	LIMITAÇÕES E TRABALHOS FUTUROS.	95
	REFERÊNCIAS	97
	APÊNDICE A – CÓDIGOS PARA IMPLEMENTAÇÃO DO PROTÓTIPO DA ABORDAGEM SEMÂNTICA	105
A.1	VALIDAÇÃO DOS PARÂMETROS DE ENTRADA	105
A.2	LEITURA DOS <i>LOGS</i>	105
A.3	SIMULAÇÃO DOS <i>JOBS</i>	106
A.3.1	Simulador MRSim	111
A.4	INSTANCIADOR DE UNIDADES DE SINTONIA - ONTOHADOOP	112
	APÊNDICE B – PUBLICAÇÕES	114
B.1	TRABALHOS PUBLICADOS	114

1 INTRODUÇÃO

MapReduce é um *framework* relativamente recente de um modelo de programação, com seu associado sistema de execução para processamento de grandes volumes de dados (22); e Hadoop (5) é a sua implementação de código aberto mais popular. O processamento das aplicações se dá em três fases: a fase *Map*, a fase intermediária (*Shuffle*, *Merge* e *Sort*) e a fase *Reduce*. Na primeira fase são processados os dados de entrada. Na fase intermediária os dados são recolhidos, classificados e agrupados. Em seguida, os resultados da fase intermediária são enviados para a fase *Reduce* como entradas. Por fim, a fase *Reduce* é processada e o resultado final é gerado.

Um grande número de empresas, centros de pesquisa e pesquisadores têm utilizado Hadoop MapReduce nas implementações de aplicações tais como mineração de dados, produção de relatórios, indexação de páginas *Web*, análises de arquivos de *log*, aprendizado de máquina, análises financeiras, simulação científica, estatísticas, pesquisas em bioinformática, etc (39).

Hadoop possui mais de 200 parâmetros de configuração que devem ser ajustados a fim de se obter um melhor desempenho do sistema (22). Segundo (8), alguns parâmetros são de extrema importância para o desempenho das tarefas no Hadoop MapReduce, e devem ser analisados com bastante cuidado e detalhes. Sendo assim, a melhor sintonia dos parâmetros não é facilmente obtida por usuários inexperientes (8). Além disso, a maioria dos estudos que abordam este tema abrangem apenas poucos parâmetros, o que torna difícil a pesquisa e entendimento global do conhecimento inerente ao tema.

A compreensão global de todos os aspectos do *framework* Hadoop MapReduce é importante para o entendimento do desempenho de cada aplicação em execução nesse ambiente e da eficiência geral do sistema. Alguns trabalhos recentes (22; 39; 86) mostram que o bom desempenho das aplicações no Hadoop MapReduce depende da configuração do *cluster* que ele é submetido, do tipo de dados de entrada e da sintonia de determinados parâmetros do Hadoop. Além disso, alguns estudos apontam que a maioria das aplicações que são executadas no Hadoop mostram uma grande diferença de comportamento em cada fase de execução do *framework* (86).

Um outro problema é que muitos usuários destas aplicações não são especialistas na área da computação e, portanto, não sabem de antemão quais parâmetros devem ser configurados e quais os valores devem ser atribuídos a estes parâmetros, a fim de se obter o melhor desempenho de suas aplicações. Da mesma forma, muitos usuários da área de computação também não possuem conhecimento detalhado sobre o assunto. Assim, a falta de conhecimento dos detalhes de sintonia desses parâmetros torna-se um grande desafio.

Na maioria dos casos, os usuários de Hadoop MapReduce devem executar *benchmarks* para avaliar o desempenho do sistema. Como a quantidade e diversificação dos sistemas se tornam cada vez maiores, torna-se cada vez mais difícil a avaliação de cada configuração e melhor sintonização para uma solução próxima da ótima. Da mesma forma, mesmo os pesquisadores e usuários mais experientes do MapReduce podem ter o mesmo problema, pois nem todos possuem tempo e recursos necessários para se fazer estas avaliações empíricas. Além disso, mesmo que estes pesquisadores tivessem os recursos necessários, os experimentos em escala real demandariam grande quantidade de tempo e custos elevados, pois as aplicações devem ser executadas com e sem otimização e os resultados devem ser comparados.

Alguns dos trabalhos mais recentes, em sua maioria, apontam para métodos de experimentações empíricas, onde uma hipótese é proposta, e esta é validada por um método de análise estatística. Outros trabalhos usam a abordagem analítica, onde é desenvolvida uma teoria formal

para um problema e os resultados derivados dessa teoria são comparados com observações empíricas. Outrossim, a maioria desses trabalhos utilizam análise dinâmica (*benchmark*) e simulação para validar suas pesquisas (12). Entre as abordagens de sintonia que visam a descoberta dos melhores valores dos parâmetros, podem-se destacar os sistemas baseados em regras (94), sistemas baseados em simulação (76; 33; 92; 90; 87; 61), sistemas baseados em análise dos *logs* de execução (91; 48; 47; 37) e os sistemas baseados em perfilização das aplicações (56; 54; 38; 37; 102).

Pode-se perceber, portanto, que muitos trabalhos se voltam para pesquisas na otimização e sintonia das configurações do Hadoop MapReduce, a fim de que propiciem alternativas nos métodos de avaliação das melhores configurações para a melhoria do desempenho. Contudo, a maioria destes trabalhos não indicam globalmente todos os parâmetros que influenciam no desempenho e nem apontam a melhor sintonia para todos. Além disso, a maioria destes trabalhos não classificam os parâmetros de configuração em relação as fases críticas para cada um deles, e não compartilham o conhecimento embutido em suas soluções. Isto mantém o conhecimento da sintonia em uma caixa preta dentro desses sistemas e não permitem aos usuários mais inexperientes o entendimento das relações dos parâmetros com o desempenho. Bem como não propicia aos usuários mais experientes colaborar com estes sistemas de forma que se busque a melhor sintonia de configuração e o conhecimento compartilhado.

1.1 MOTIVAÇÃO

O conhecimento pode ser entendido como fatos, informações e competências adquiridas através da experiência ou educação por meio da percepção, descoberta ou aprendizagem, ou seja, a compreensão teórica (explícito) ou prática (implícito) de um determinado assunto (20).

A representação formal do conhecimento baseia-se na conceitualização dos objetos, conceitos e outras entidades, em alguma área específica de interesse, bem como as relações entre elas (30). Esta conceitualização pode ser entendida como uma visão abstrata e sintetizada do domínio que desejamos representar para algum propósito. Sendo assim, uma base de conhecimento ou um sistema baseado em conhecimento compromete-se com alguma conceitualização, explícita ou implicitamente (29).

Trabalhos na área de Inteligência Artificial têm desenvolvido, ao longo dos anos, técnicas para a representação do conhecimento de forma que o mesmo possa ser explorado por processos computacionais e heurísticas (43). Na área de engenharia de software, por exemplo, têm sido produzidas técnicas para capturar o conhecimento que relaciona os requerimentos, decisões de projeto e raciocínio para os sistemas de informação. Sendo assim, a área de sistemas de informação tem se beneficiado, direta ou indiretamente destes desenvolvimentos (31).

O termo "Ontologia" é definido como uma explícita especificação de uma conceitualização, onde a conceitualização é uma visão abstrata e simplificada do contexto que se quer representar para propósitos específicos (29). Portanto, uma ontologia descreve formalmente os conceitos e relações que podem existir em um domínio específico de problema, com a descrição do significado dos conceitos e dos axiomas formais que restringem a interpretação e uso destes termos. Formalmente, uma ontologia é a declaração de uma teoria lógica (21). Ao representar o conhecimento de domínio específico com vocabulários baseados em objetos (classes) e suas relações, os motores de inferência e outros sistemas de inteligência são capazes de compreender e reutilizar o conhecimento de outros sistemas.

Além disso, as ontologias podem ser aplicadas no contexto da Web Semântica. Isto permite que os conteúdos da Web sejam representados de forma estrutural com uma lista finita de vocabulários e seus relacionamentos. Desta forma, programas de computador e agentes

de software são capazes de entender a semântica do conteúdo da Web e, assim, processar estes conteúdos (10; 23; 21). As ontologias tem sido aplicadas nas áreas de Gerenciamento de Conhecimento (KM), comércio eletrônico, computação em nuvens, integração de sistemas empresariais, desenvolvimento de software, aprendizagem organizacional, entre outras (70; 65; 96; 99; 19; 80; 62; 98).

Sendo assim, é de grande importância a criação de ambientes que facilitem a seleção e a sintonia dos mais adequados parâmetros de configuração com o objetivo de melhorar o desempenho das aplicações Hadoop MapReduce, seguindo a tendência a Web Semântica. Além disso, o ambiente deve promover e motivar o compartilhamento das informações e disseminação do conhecimento em relação a este assunto. Portanto, todo conhecimento adquirido deve ser classificado e organizado a fim de ser utilizado e/ou reutilizado quando necessário, de maneira fácil e eficiente.

1.2 DESCRIÇÃO DOS PROBLEMAS

As soluções mais recentemente apresentadas na comunidade científica buscam de alguma forma a sintonia dos parâmetros de configuração do Hadoop MapReduce. No entanto, o conhecimento e a expertise utilizadas em algumas dessas soluções estão embutidas na implementação. Sem um estudo detalhado desses trabalhos, não se pode inferir este conhecimento. O usuário torna-se refém de soluções proprietárias com o conhecimento oculto a respeito da sintonia dos parâmetros.

Uma das formas de prever quais recursos são os mais adequados e necessários para a execução destas aplicações seria a determinação de um perfil comportamental para cada uma delas (25; 66). Em se tratando de aplicações Hadoop MapReduce, os parâmetros de configuração são de fundamental importância na mudança de comportamento e desempenho desta mesma aplicação com os mesmos recursos alocados. No entanto, apenas a determinação do perfil da aplicação não é o suficiente para se obter a mais condizente sintonia de parâmetros, nem a caracterização de alocação de recursos necessária para um melhor desempenho. Deve-se vincular a um prévio conhecimento para que se possa fazer a melhor seleção de parâmetros e a sintonia dos mesmos.

Alguns trabalhos provêm vantagens em alguns pontos e desvantagens em outros. Trabalhos baseados em simulação, por exemplo, cumprem o papel da melhor sintonia através da execução de simulações. Porém, o usuário tem a necessidade de escolher quais valores de parâmetros aplicar e, assim, testá-los. Outros adquirem a perfilização e predição através do histórico de execuções anteriores (*traces*), tal como Starfish (37; 38). Neste caso, o usuário não tem acesso ao conhecimento utilizado na solução de sintonia. Além disso, os trabalhos dirigidos por *traces* de execuções possuem um grande obstáculo na obtenção de *traces* realísticos. Apenas empresas ou instituições que executam *clusters* Hadoop em grandes proporções os possui, bem como somente seus funcionários ou colaboradores possuem acesso a estes *traces*, pois os mesmos não são públicos.

1.3 OBJETIVOS

No tocante a sintonia dos parâmetros de configuração do Hadoop MapReduce, com o objetivo de melhorar o desempenho das aplicações, é fundamental o conhecimento dos usuários experientes e dos trabalhos científicos já publicados a respeito deste assunto. Sendo assim, tanto um quanto o outro tornam-se fontes imprescindíveis de conhecimento.

Partindo do pressuposto que o parágrafo anterior relata uma realidade, faz-se necessário tomar medidas e criar ferramentas que minimizem esse problema na qual os usuários Hadoop MapReduce se deparam, experientes ou não. Dessa forma, prover um ambiente, abordagem ou ferramenta que, mesmo que parcialmente, modele, prolifere e compartilhe o conhecimento seria de grande valia.

Portanto, alguns questionamentos iniciais para este trabalho devem ser levantados: Quais parâmetros de configuração do Hadoop têm influência e impactam no desempenho do sistema? Estes parâmetros são influenciados pelas fases de execução do Hadoop (*map*, intermediária e *reduce*)? Quais parâmetros são influenciados pelas características das cargas de trabalho? Quais tecnologias estão sendo aplicadas no contexto de sintonia de parâmetros do Hadoop? Como representar e compartilhar o conhecimento adquirido nas pesquisas e através de usuários experientes? Como caracterizar as cargas de trabalho submetidas ao Hadoop? Como relacionar estas cargas de trabalho aos parâmetros e quais os melhores valores para estes parâmetros?

Todas estas questões iniciais e suas respostas permitiram delimitar o objetivo geral deste trabalho, que é propor uma abordagem semântica baseada em ontologia para sintonia de parâmetros do Hadoop MapReduce, com o objetivo de melhorar o desempenho das aplicações.

A fim de que se possa atingir o objetivo principal da pesquisa, são apresentados os seguintes objetivos específicos:

1. Investigar, na literatura científica mais recente, os trabalhos que abordam a sintonia dos parâmetros do Hadoop MapReduce, a fim de identificar quais parâmetros são os mais relevantes em relação ao desempenho do sistema, bem como as abordagens utilizadas nestes trabalhos;
2. Propor uma classificação dos parâmetros relevantes, de forma que respondam as questões em relação a sua influência nas fases de execução do Hadoop e suas relações com as características da carga de trabalho;
3. Desenvolver uma ontologia como base para o conhecimento adquirido, bem como as relações e axiomas entre as classes e objetos investigados;
4. Propor uma arquitetura base, a fim de dar suporte a abordagem semântica proposta;
5. Propor uma abordagem semântica baseada na ontologia, para gerar unidades de sintonia para os parâmetros de configuração do Hadoop;
6. Simular um ambiente computacional de inferência semântica baseado na abordagem proposta;
7. Avaliar a abordagem proposta;

Identificados os objetivos específicos, o trabalho final deve responder a duas questões fundamentais:

1. A abordagem proposta, baseada em uma ontologia, para gerar unidades de sintonia dos parâmetros de configuração do Hadoop, pode auxiliar na melhora do desempenho das aplicações?
2. A abordagem semântica contribui para o compartilhamento e reuso do conhecimento sobre a sintonia de parâmetros do Hadoop?

1.4 MÉTODO DE PESQUISA E ABORDAGEM

A pesquisa foi conduzida como enumeramos a seguir:

1. Uma revisão sistemática da literatura foi conduzida para investigar os parâmetros de configuração do Hadoop MapReduce que influenciam no desempenho das aplicações, bem como identificar experiências na sintonia dos parâmetros e a classificação dos mesmos de acordo com seus tipos, níveis, fases e as características da carga de trabalho que os influenciam. Além disso, a revisão sistemática permite a classificação dos trabalhos revisados segundo seus métodos de experimentação e avaliação;
2. Um modelo conceitual da ontologia foi proposto para modelar o conhecimento adquirido;
3. Foi proposta uma arquitetura para a abordagem semântica;
4. Um fluxo de processos foi proposto para visualizar globalmente a abordagem semântica para sintonia de parâmetros;
5. De acordo com a abordagem foi proposto um ambiente computacional de inferência semântica simulado;
6. Uma avaliação da abordagem foi proposta através da simulação de resultados e comparação com os valores padrão dos parâmetros de configuração.

1.5 ORGANIZAÇÃO DA TESE

O restante da tese é organizado como vemos a seguir.

O Capítulo 2 faz uma revisão sobre o *framework* MapReduce, e foca na implementação mais difundida deste framework: o Hadoop. Hadoop é apresentado com o objetivo de se obter uma boa base de entendimento sobre sua arquitetura, modelo de programação e execução, bem como seu sistema de arquivos, entre outros detalhes. Entre outras coisas, o capítulo também apresenta alguns estudos que utilizam ontologias em suas soluções.

O Capítulo 3 faz uma revisão bibliográfica sobre Ontologias, suas características e tipos, metodologia para construção, linguagens, ferramentas, assim como outros assuntos essenciais para este estudo.

O Capítulo 4 apresenta uma revisão sistemática para a identificação de artigos de pesquisas recentes, que abordam a correlação entre a sintonia dos parâmetros de configuração do Hadoop e o desempenho das aplicações. Neste sentido, esta revisão sistemática identificou 743 artigos em cinco bases de dados pesquisadas. Aplicando as etapas de seleção e alguns critérios de exclusão, o número de artigos foi reduzido para 40. Estes artigos foram analisados e classificados de acordo com os parâmetros de configuração do Hadoop e os objetivos dos estudos. Esta revisão é parte fundamental para a aquisição de conhecimento para a construção da ontologia OntoHadoop.

O Capítulo 5 apresenta a ontologia OntoHadoop. São apresentadas as etapas de modelagem, formalização e definição das classes, relações e axiomas da ontologia. Além disso, é apresentada a construção da ontologia através da ferramenta Protégé na versão 4.3 (81) e suas etapas de implementação.

O Capítulo 6 faz a proposição da arquitetura da abordagem semântica, baseada na ontologia proposta, para sintonia dos parâmetros de configuração do Hadoop. Ainda neste

capítulo, são apresentados os processos de simulação de um ambiente computacional de inferência semântica e os módulos deste ambiente.

O Capítulo 7 apresenta os experimentos e os resultados obtidos através da ontologia e da abordagem semântica proposta.

O Capítulo 8 apresenta as considerações finais deste trabalho, e descreve a relevância e as contribuições desta pesquisa. Além disso, são expostos alguns comentários sobre trabalhos futuros em relação a pesquisa apresentada.

Por fim, o Apêndice A detalha os códigos de implementação utilizados neste trabalho. O Apêndice B apresenta as publicações geradas por esta pesquisa.

2 HADOOP MAPREDUCE

MapReduce é um *framework* relativamente recente que é um modelo de programação com seu associado sistema de execução para processamento de dados em larga escala (22); e Hadoop (5) é a sua implementação de código aberto mais popular.

Um grande número de empresas, centros de pesquisa e pesquisadores têm utilizado Hadoop MapReduce nas implementações de aplicações tais como mineração de dados, produção de relatórios, indexação de páginas Web, análises de arquivos de log, aprendizado de máquina, análises financeiras, simulação científica, estatísticas, pesquisas em bioinformática, etc (39).

MapReduce é um modelo de programação e uma implementação associados para processamento e geração de grandes conjuntos de dados. Os usuários especificam uma função *map* que processa um par chave/valor para gerar um conjunto de chaves intermediárias de pares chave/valor pares, e uma função *reduce* que combina todos os valores intermediários associados com a mesma chave intermediária. Muitas tarefas do mundo real podem ser expressas neste modelo. Os programas escritos neste estilo funcional são automaticamente paralelizados e executados em um *cluster* de máquinas, ou mais recentemente, em uma nuvem computacional (22). O sistema de execução cuida dos detalhes do particionamento dos dados de entrada, agendando a execução do programa em um conjunto de máquinas, gerencia as falhas das máquinas, e gerencia a comunicação entre as máquinas. Isso permite que programadores sem grandes experiências com sistemas paralelos e distribuídos, facilmente utilizem os recursos de um grande sistema distribuído (22).

2.1 MODELO DE PROGRAMAÇÃO

O modelo de programação MapReduce consiste de duas funções principais: $map(k_1, v_1)$ e $reduce(k_2, list(v_2))$. Os usuários podem implementar sua própria lógica de processamento através da customização das funções *map* e *reduce* através de linguagens de programação conhecidas tal como Java.

A função *map()*, tem como entrada um par chave/valor e produz um conjunto de pares intermediários chave/valor. As bibliotecas MapReduce agrupam todos os valores intermediários associados a mesma chave intermediária *I* e passa para a função *reduce()*. A função *reduce()*, que também é escrita pelo usuário/programador, recebe uma chave intermediária *I* e um conjunto de valores para esta chave. A função *reduce()* faz um *merge* deste conjunto de valores de modo a formar um conjunto menor de valores. Normalmente apenas zero ou um valor de saída é produzido pela chamada do *Reduce*. Os valores intermediários são fornecidos para a função *reduce()* através de um iterador. Isto permite a manipulação de listas de valores que são muito grandes para caber na memória.

Dentre vários programas que podem ser expressados através do MapReduce, pode-se citar alguns exemplos clássicos, tais como: Grep distribuído (*Grep*), Contador de frequência de acesso a URLs ou palavras (*WordCount*), Classificação distribuída (*TeraSort*), etc.

WordCount é um exemplo típico de aplicação Hadoop MapReduce. Esta aplicação. Esta aplicação tem como objetivo contar o número de ocorrências de uma palavra em uma grande coleção de documentos. Um exemplo do código do programa WordCount em Java é apresentado no código 2.1:

```

1 public class WordCount {
2     public static class Map extends MapReduceBase implements
3 Mapper<LongWritable, Text, Text, IntWritable> {
4         private final static IntWritable one = new IntWritable(1);
5         private Text word = new Text();
6         public void map(LongWritable key, Text value, OutputCollector<Text,
7 IntWritable> output, Reporter reporter) throws IOException {
8             String line = value.toString();
9             StringTokenizer tokenizer = new StringTokenizer(line);
10            while (tokenizer.hasMoreTokens()) {
11                word.set(tokenizer.nextToken());
12                output.collect(word, one); } } }
13
14    public static class Reduce extends MapReduceBase implements Reducer<Text,
15 IntWritable, Text, IntWritable> {
16        public void reduce(Text key, Iterator<IntWritable> values,
17 OutputCollector<Text, IntWritable> output, Reporter reporter) throws
18 IOException
19 {
20     int sum = 0;
21     while (values.hasNext()) {
22         sum += values.next().get(); }
23     output.collect(key, new IntWritable(sum)); } }
24
25    public static void main(String[] args) throws Exception {
26        JobConf conf = new JobConf(WordCount.class);
27        conf.setJobName("wordcount"); } }

```

Na fase *map*, cada chamada da função *map()* recebe uma linha de texto. Para cada palavra encontrada na linha recebida, a função emite (*output.collect*) um par chave/valor, onde a chave é a palavra em si, e o valor é a constante *1*. A função *reduce()*, por sua vez, recebe como entrada uma palavra (chave), e um iterador para todos os valores emitidos pela função *map()*, associados com a palavra em questão. Todos os valores são, então, somados (*sum* em um par chave/valor contendo a palavra; e seu total de ocorrências é emitido (*output.collect*).

2.2 MODELO DE EXECUÇÃO

As chamadas a função *map()* são distribuídas em múltiplas máquinas através do particionamento dos dados de entrada em um conjunto *M* de pedaços. Estas partes podem ser processadas em paralelo por diferentes máquinas. Por outro lado, as chamadas a função *reduce()* são distribuídas através do particionamento das chaves intermediárias em *R* pedaços através de uma função *Hash(key) mod R*. Este número de partições (*R*) e a função de particionamento são definidos pelo usuário.

A Figura 2.1 apresenta o fluxo das atividades que ocorrem durante o processo execução do MapReduce. São elas: (1) A biblioteca do MapReduce particiona os dados de entrada em *M* partes e inicia várias cópias do programa em um *cluster* de máquinas; (2) Uma destas cópias do programa é o *Master* e as demais são os *Workers*. Estes últimos recebem o trabalho escalonados pelo *Master*. São atribuídos *M* tarefas de *map* e *R* tarefas de *reduce*. Além disso o *Master* é responsável por selecionar *Workers* ociosos e atribuir tarefas a eles; (3) Um *Worker* que recebe uma tarefa *map*, faz a leitura do conteúdo da parte dos dados que lhe foi atribuída. Em seguida, o par chave/valor é interpretado e passado como parâmetro a função *map* que produz os pares intermediários de chave/valor. Estes pares são, então, armazenados em memória; (4) Os pares chave/valor armazenados em memória são, periodicamente, escritos no disco local e

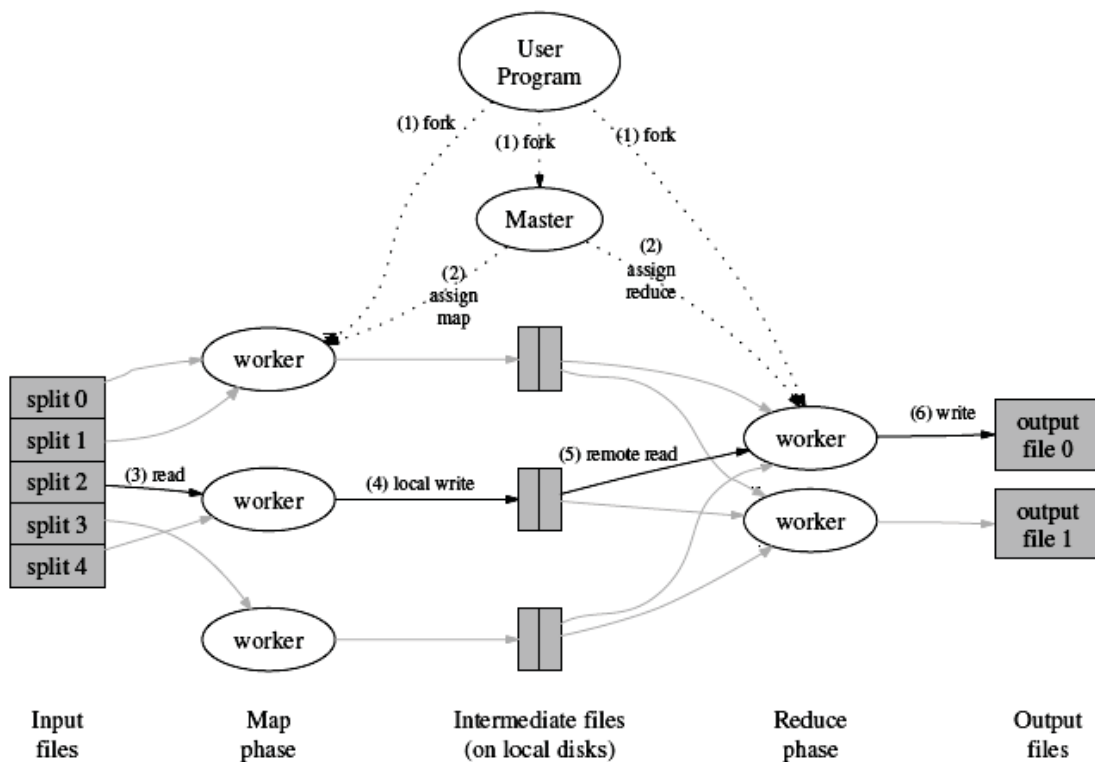


Figura 2.1: Arquitetura de Execução do MapReduce. Extraído de (22)

particionados em R regiões através da função de particionamento. A localização destes pares de dados é passada ao *Master* que fica com a responsabilidade de repassar aos *Workers* de Reduce; (5) O *worker* então utiliza chamadas remotas de procedimentos para ler os dados armazenados no disco local. Assim que o *Worker* de Reduce faz a leitura dos pares intermediários, ele ordena por chave a fim de que as mesmas chaves sejam agrupadas. Isto é necessário visto que muitas chaves diferente são mapeadas para a mesma tarefa de Reduce; (6) Em seguida, os dados intermediários ordenados são percorridos pelo *Worker* de Reduce que passa cada uma das chaves encontradas para a função *reduce()*. A saída de cada uma das funções é adicionada ao arquivo de saída final para a referida partição; (7) Assim que todas as tarefas de *map* e *reduce* forem concluídas, retorna-se ao ponto de chamada do MapReduce no programa do usuário (22).

2.3 SISTEMA DE ARQUIVOS DISTRIBUÍDO DO HADOOP - HDFS

O HDFS (*Hadoop Distributed File System*) é um sistemas de arquivos distribuído que foi projetado para o framework Hadoop MapReduce. O HDFS fornece alto *throughput* para os dados das aplicações MapReduce, sendo adequado para aplicações que tratam grandes conjuntos de dados.

O HDFS possui uma arquitetura mestre-escravo. Seu *cluster* consiste de um único *NameNode*, de um servidor mestre que gerencia *namespace* do sistema de arquivos e regula o acesso dos clientes aos arquivos. Além disso, existe um número de *DataNodes* (normalmente um por nó do *cluster*) que gerencia o armazenamento nos nós. Internamente, um arquivo é dividido em um ou mais blocos. Estes blocos são armazenados em um conjunto de *DataNodes*. Os *DataNodes* são responsáveis por servirem a requisições de leitura e escrita dos clientes do

sistema de arquivos. Além disso, os *DataNodes* executam a criação, exclusão e replicação dos blocos mediante instruções do *NameNode*.

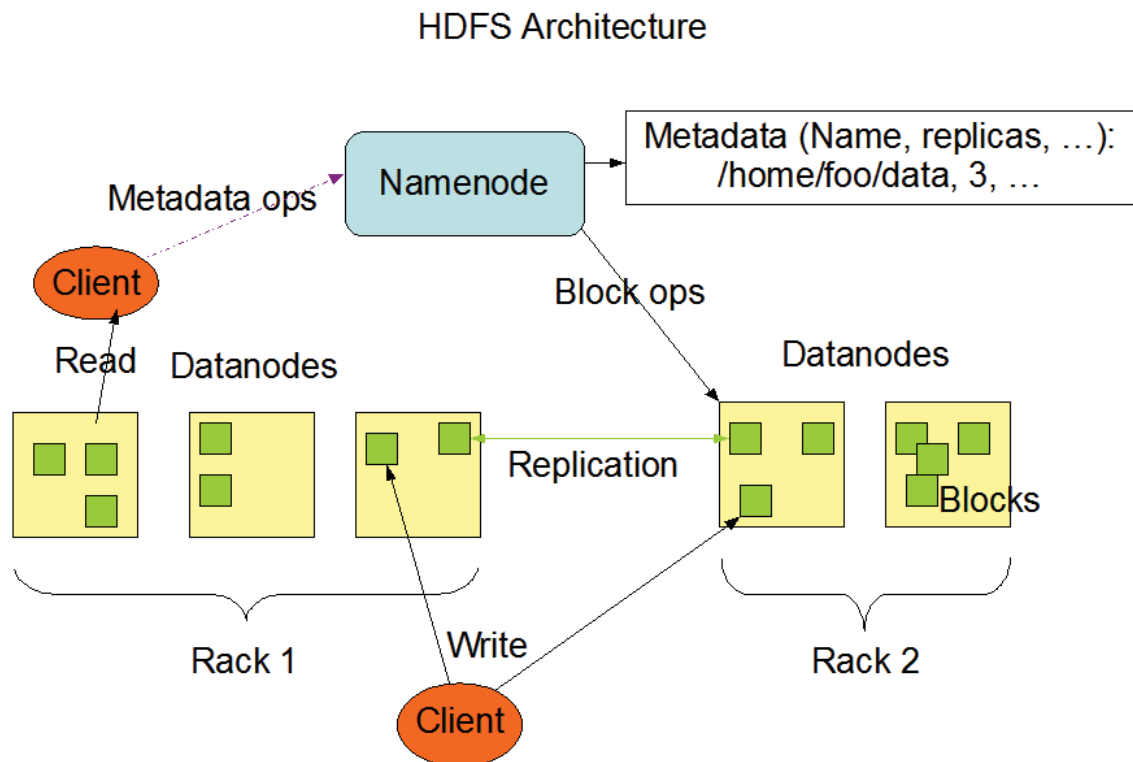


Figura 2.2: Arquitetura do HDFS. Extraído de (6)

Na Figura 2.2, que apresenta a arquitetura do sistema de arquivos distribuído do Hadoop MapReduce, pode-se verificar que o *NameNode* e o *DataNode* são partes de um programa projetado para executar em *clusters* de máquinas. Essas máquinas tipicamente executam em um sistema operacional Linux (OS). O HDFS foi construído utilizando a linguagem Java. Sendo assim, qualquer máquina que suporte Java pode executar o *NameNode* ou o *DataNode*. Uma implementação típica do HDFS tem uma única máquina dedicada para executar o *NameNode*. Cada uma das outras máquinas do *cluster* executam uma instância do *DataNode*. No entanto, a arquitetura não impede execução *DataNodes* múltiplos na mesma máquina. O *NameNode* é o mediador e o repositório para todos os metadados do HDFS. O sistema é concebido de uma forma que os dados do usuário não passem pelo *NameNode*.

2.4 PARÂMETROS DE EXECUÇÃO DO HADOOP

Para que se execute um programa tal como um *job* no Hadoop, um objeto de configuração do *job* é criado e os parâmetros deste *job* devem ser especificados. Ou seja, uma vez que o *job* fora criado para o Mapreduce, é necessário hpositionabilá-lo para execução em escala em um *cluster* (86). Sendo assim, a fim de que se evite alguns problemas, tal como desempenho, por exemplo, é necessário configurar alguns parâmetros.

MapReduce fornece muitos parâmetros configuráveis que o levam a melhores ou piores desempenhos de execução em um *cluster*. A maioria destes parâmetros geram efeito sobre a execução do *job* e outros sobre o *cluster*. Em se tratando de grandes *clusters* de máquinas, estes parâmetros tornam-se bastante importantes e fornecem uma indicação do desempenho deste *cluster* em relação ao *job* submetido.

O Hadoop MapReduce foi projetado para executar aplicações que possuem um grande conjunto de dados de entrada e saída de dados e podem ser executadas sobre um grande conjunto de máquinas não necessariamente similares. Sendo assim, o *framework* deve ser mais otimizado quanto possível for.

Dentre as otimizações possíveis, também inclui-se a otimização do HDFS (*Hadoop File System*). O sistema de arquivos HDFS é otimizado para um pequeno número de arquivos muito grandes que são acessados sequencialmente. Os dados armazenados no HDFS não são, geralmente, considerados valiosos ou insubstituíveis.

O HDFS pode gerar algum tipo de problema durante a entrada e saída dos grandes volumes de dados, principalmente quando um grande número de arquivos estão sendo criados ou existem nos *DataNodes*. A verdade é que Hadoop MapReduce não fornece alta disponibilidade para o HDFS ou para a submissão de *jobs*. Sendo assim, alguns cuidados devem ser tomados para garantir que os dados requisitados ao HDFS possam ser recuperados em caso de uma falha crítica do *NameNode*.

São mais de 200 parâmetros a serem especificados para o controle do comportamento de uma tarefa MapReduce no Hadoop. Destes, mais de 25 parâmetros podem impactar no desempenho da tarefa. Alguns parâmetros de configuração utilizados pelo Hadoop tem o objetivo de controlar os vários aspectos do comportamento da tarefa durante a execução. Alguns destes aspectos incluem a alocação e uso da memória, concorrência, otimização de I/O e o uso da banda de rede. O solicitante da tarefa Hadoop tem a opção de configurar estes parâmetros usando uma interface em nível de programação ou através de arquivos de configuração em XML. Outras linguagens de mais alto nível para o *framework* MapReduce, como HiveQL¹ e Pig², desenvolveram suas próprias sintaxes para especificação dos parâmetros. Caso os parâmetros não sejam especificados, valores default são assumidos.

Segundo (8), um subconjunto relativamente grande destes parâmetros apresentam fortes interações de desempenho com um ou mais outros parâmetros. Sendo assim, uma interação pode existir entre os parâmetros P1 e P2 quando a magnitude do impacto que a variação de P1 tem sobre o desempenho da tarefa depende da configuração específica de P2. Ou seja, o impacto sobre o desempenho pela variação de P1 é diferente através das diferentes configurações de P2.

Muitas escolhas devem ser feitas a fim de se especificar completamente como um *job* deve ser executado. As configurações destes parâmetros incluem, dentre outros (37):

- o número de tarefas *map* dentro do *job*;
- o número de tarefas *reduce* dentro do *job*;
- a quantidade de memória alocada a cada tarefa de *map* ou *reduce* armazenar os dados de entrada e saída;
- as configurações para a ordenação externa multi-fase usada pela maioria dos *frameworks* MapReduce a fim de agrupar os valores de saída do *map* por chave;
- os parâmetros que indicam se os dados de saída do *maps* e *reduces* devem ser comprimidos antes de serem escritos, e como comprimí-los;
- os parâmetros que indicam se a função *combine* deve ser utilizada para pré-agregar as saídas dos *maps* antes de serem transferidas para os *reduces*.

¹<http://hive.apache.org/>

²<http://pig.apache.org/>

Portanto, pode-se agrupar os parâmetros configuráveis do Hadoop MapReduce de acordo com os módulos de sua arquitetura, tais como (86):

- Parâmetros do HDFS;
- Parâmetros do *JobTracker* e *TaskTracker*;
- Parâmetros relacionados as tarefas e ao sistema de arquivos por *job*;
- Parâmetros que diretamente controlam as funções centrais do MapReduce;
- Parâmetros que melhoram o desempenho do *job*;
- Parâmetros específicos para otimização da fase *map*;
- Parâmetros específicos para otimização da fase *reduce*.

(61) considera 3 importantes agrupamentos de parâmetros que influenciam no desempenho das aplicações:

- Parâmetros dos nós: dizem respeito a CPU, disco (I/O), memória, rede, instância do *map* e instância do *reduce*;
- Parâmetros do *Cluster*: dizem respeito ao número de nós, topologia, instalações da rede, fila de serviço e escalonador;
- Parâmetros do sistema Hadoop: dizem respeito as especificações do *job* e parâmetros do Hadoop.

(13) cita que Predicting Execution Bottlenecks in Map-Reduce Clusters

2.5 MAPREDUCE E SEMÂNTICA

Segundo (22), existe uma semântica simples para raciocinar sobre o comportamento do programa. Isto é possível, pelo fato do MapReduce possuir uma semântica equivalente a de uma execução sequencial. Desta forma, através da semântica, outros modos de *input* podem ser implementados por uma interface simples de um *reader*. Algumas informações referentes a execução podem ser extraídas através do *master*. O *master* executa um servidor HTTP interno que exporta informações do status da execução, tais como, tarefas completadas, quantidade de tarefas em progresso, quantidade de bytes de *input*, intermediários e *output*, bem como algumas taxas de processamento. Além disso, fornece informações de erros padrão e arquivos de saída gerados por cada tarefa. Estas informações, se mapeadas para um ontologia e aplicadas inferências, podem ser usadas para predição da quantidade de tempo de processamento que será necessário, e se mais recursos devem ser adicionados ou não. Estas informações podem ser utilizadas, também, para determinar quando a computação é muito mais lenta do que o esperado.

Em relação a tolerância a falhas, estas informações de estado de alto nível mostram quais os *workers* estão em falha e quais tarefas *map* e *reduce* estavam em processamento quando eles falharam. Esta informação é útil ao tentar diagnosticar *bugs* no código do usuário, por exemplo.

Para análise do comportamento da execução e dos programas, existem algumas informações extras do contador, tais como o número de pares de entrada processados e o número de pares de saída produzidos. Dependendo da aplicação, mais I/O ou mais processamento são necessários no que diz respeito as funções executadas pelas mesmas, por exemplo.

2.6 CONSIDERAÇÕES FINAIS

Este capítulo apresentou uma breve pesquisa bibliográfica sobre o modelo MapReduce, com mais ênfase em sua implementação de código aberto mais popular, o Hadoop. MapReduce é tema central na proposta deste trabalho. Como visto neste capítulo, são mais de 190 parâmetros a serem especificados para o controle do comportamento de uma tarefa Hadoop MapReduce. Destes, mais de 25 parâmetros podem impactar no desempenho da tarefa. A análise dos parâmetros, a fim de identificar os que mais influenciam no desempenho das aplicações Hadoop MapReduce será um dos objetivos deste trabalho.

A abordagem semântica proposta é em direção a sintonia de parâmetros do Hadoop MapReduce, com o objetivo de se obter melhora no desempenho das aplicações. Esta abordagem pretende coletar e manter as informações sobre os parâmetros de configuração e suas relações com outros parâmetros, bem como promover a sintonia dos valores mais adequados de acordo as características das cargas de trabalho e a relação das mesmas com os parâmetros.

O grande desafio é que os usuários de Hadoop MapReduce, possam saber, antes mesmo de submeterem suas cargas de trabalho e aplicações reais ao *framework* Hadoop, possam de antemão possuírem o conhecimento necessário para a sintonia dos melhores valores dos parâmetros de configuração para um melhor desempenho.

Uma das formas mais difundidas e utilizadas atualmente para representação e gerenciamento do conhecimento e semântica, são as ontologias. Portanto, o próximo capítulo faz uma revisão literária sobre Ontologias.

3 ONTOLOGIAS

O termo "ontologia" é emprestado da filosofia, onde uma Ontologia é uma descrição da Existência. Na Ciência da Computação vem sendo aplicado desde o início da década de 90 na área de inteligência artificial para representação computacional de conhecimento em áreas como engenharia de conhecimento e processamento de linguagem natural. Para sistemas de Inteligência Artificial, o que "existe" é o que pode ser representado. Neste contexto, podem ser entendidas como uma especificação formal e explícita de uma conceitualização consensual (29).

A noção de conceitualização exige, entretanto, uma adequada formalização, visto que ela pode gerar algumas confusões. Em (29), a conceitualização é definida com uma estrutura $\langle D, R \rangle$, onde D é um domínio e R é um conjunto de relações sobre D .

O corpo de um conhecimento representado formalmente é baseado numa conceitualização: os objetos, conceitos e outras entidades que são assumidas para existirem em alguma área de interesse e os relacionamentos que são organizados dentro delas (21). Uma conceitualização é uma visão abstrata e simplificada do mundo que se quer representar para algum propósito. Toda base de conhecimento, sistema baseado em conhecimento que representa conhecimento está comprometido com alguma conceitualização, explícita ou implícita. Uma ontologia é uma especificação explícita de uma conceitualização.

Usam-se ontologias comuns para descrever comprometimentos ontológicos para um conjunto de agentes. Desta forma eles podem passar informação sobre o domínio do discurso, sem necessariamente operar sobre uma teoria globalmente compartilhada. O Conhecimento é atribuído aos agentes através da observação de suas ações. Um agente "sabe" algo se ele age como se ele tivesse a informação e está agindo racionalmente para alcançar seus objetivos (1). As "ações" dos agentes - incluindo servidores de base de conhecimento e sistemas baseados em conhecimento - podem ser vistas através de uma ordem e perguntas da interface funcional. Nesta interface, um cliente interage com um agente fazendo afirmações lógicas e propondo perguntas. Pragmaticamente, uma ontologia comum define o vocabulário com a qual as pesquisas e afirmações são trocadas entre os agentes.

Para representar uma conceitualização é necessário uma linguagem de representação. Muitas linguagens e sistemas de representação são definidos. Entretanto, para aplicações sobre a Web, é importante ter uma linguagem com uma sintaxe padronizada (XML) (14). Este requerimento leva novamente às linguagens baseadas em XML, definindo-as sobre o topo do XML. Um exemplo é RDFS (*Resource Description Framework Schema*), entre outras. Todas elas usam a sintaxe do XML, mas com leves diferenças nas *tag names*.

3.1 METADADOS SEMÂNTICOS

A mais breve definição de metadado é "dado sobre dado". Neste contexto, metadado refere-se a alguma estrutura descritiva da informação de outro dado; que é usado para ajudar na identificação, descrição, localização e gerenciamento de recursos da Web. Entretanto, eles podem ser aplicados em qualquer meio (10).

Metadados podem possuir a classificação de estrutural ou semântico. Metadado estrutural representa a informação que descreve a organização e estrutura dos dados gravados. Por exemplo, informações sobre o formato, os tipos de dados usados e os relacionamentos sintáticos entre eles. Em contraste, metadados semânticos fornecem informações sobre o significado dos dados disponíveis e seus relacionamentos semânticos. Por exemplo, dados que descrevem o

conteúdo semântico de um valor de dado (como unidades de medida e escala), ou dados que fornecem informações adicionais sobre sua criação (algoritmo de cálculo ou derivação da fórmula usada), linhagem dos dados (proveniência) e qualidade (atualidade e precisão). Neste sentido, é desejável uma conceitualização de um domínio específico de problema, ou ontologias que forneçam um acordo comum de vocabulários, para que os dados sejam referenciados. Assim, uma ontologia serve como uma base comum para a representação de dados e metadados (10).

Um objeto semântico representa um item de dado junto com sua base de contexto semântico que consiste de um conjunto flexível de meta-atributos que explicitamente descrevem a compreensão implícita sobre o significado do item de dado. Adicionalmente, cada objeto semântico possui um rótulo de conceito associado a ele, que especifica o relacionamento entre o objeto e os aspectos do mundo real que ele descreve. Estes rótulos são adquiridos de uma Ontologia (42).

Um domínio específico de ontologias pode ser utilizado para assegurar a correta interpretação dos metadados disponíveis. Uma ontologia fornece um entendimento sobre uma conceitualização compartilhada de um determinado domínio de aplicação. Os conceitos específicos numa ontologia fornecem um vocabulário comum para que nenhuma negociação adicional seja necessária.

Metadados semânticos representam um significativo papel no contexto da Web Semântica (*Semantic Web*) (42; 23; 10) e, por consequência, em qualquer outro assunto relacionado a disponibilidade e acesso de dados e recursos na Web. Dentre outros, comércio eletrônico e bibliotecas digitais. Os agentes necessitam de metadados que descrevem o contexto de recursos, a fim de executarem operações, tais como consultas sobre estes recursos. Além disso, se a semântica de um domínio for bem declarada e explicitada, estes agentes podem então aplicar raciocínio lógico (*reasoning*) sobre os metadados, melhorando seu poder de processamento.

3.2 ONTOLOGIAS E WEB SEMÂNTICA

A visão da Web Semântica (10) é a de uma Web no qual os recursos disponíveis são acessíveis não somente por seres humanos, mas também por processos automatizados. Por exemplo, agentes automatizados percorrendo a Web e executando tarefas, tais como aumentar as buscas em termos de precisão e descoberta de recursos, recuperação e filtragem de informações. A automação de tarefas depende da elevação do status da Web de *machine-readable* (lida automaticamente) para algo que é chamado de *machine-understandable* (entendida automaticamente). A ideia chave é ter dados sobre a Web. Estes dados devem ser definidos e ligados de tal forma que seus significados sejam, preferencialmente, interpretados explicitamente por processos de software ao invés de interpretados implicitamente por seres humanos.

Para atingir este objetivo, torna-se necessária a anotação de recursos sobre a Web através de metadados. Na verdade, seria ideal, se houvesse a possibilidade de se anotar ou criar estes metadados com semântica, que proveria alguma indicação dos conteúdos de um recurso. Faz-se necessário, então, o uso de linguagens que suportem a representação de metadados semânticos. Muitas propostas de padronização de linguagens para metadados já são reconhecidas na W3C ¹. Dentre elas, Resource Description Framework (RDF) (52) e RDF Schema (RDFS) (16). Entretanto, a menos que elas compartilhem um entendimento comum em relação a seu significado, estas linguagens possuem valor limitado para processos automatizados. As ontologias podem colaborar muito para o preenchimento deste requisito. Elas fornecem uma representação de uma conceitualização compartilhada de um domínio particular, além de um vocabulário controlado compartilhado que pode ser comunicado através das pessoas e aplicações (9). Assim,

¹<http://www.w3.org/>

são possíveis as perguntas de frases cujas respostas deveriam exigir de uma máquina uma busca por toda a Web e uma quantia inimaginável de tempo para solucioná-la. Isto não impede que seja feita uma linguagem completa. Cada aplicação mecânica de RDF usará um esquema para restringir seu uso de RDF para uma linguagem deliberadamente limitada. Porém, quando são feitos vínculos entre os nodos de RDF, o resultado é uma expressão de uma quantia enorme de informação. Está claro que, pelo fato da Web Semântica poder incluir todos os tipos de dados para representar o mundo real, a própria linguagem deve ser completamente expressiva.

3.2.1 Ontologias e Representação do Conhecimento

Segundo (57), conhecimento é identificado como a informação que é relevante, acionáveis e baseada pelo menos em parte na experiência. Conhecimento é descrito como crença verdadeira justificada que é racional, dinâmica, humanista e em contexto específico (69). Assim, a informação somente se tornaria conhecimento se a interpretação pessoal de experiências, crenças e compromissos forem adicionadas.

Ontologia é uma parte fundamental de uma ampla gama de tecnologias baseadas em semântica, que incluem as áreas de representação do conhecimento (KR) e inferência automatizada que vem da área de Inteligência Artificial. Muitos formalismos de representação diferentes foram exploradas, e motores de raciocínio desenvolvidos. Um resultado importante é a existência comprovada de um equilíbrio entre poder de representação de uma língua (ou seja, a capacidade de representar/expressar muitos tipos diferentes de conhecimento) e a eficiência dos motores de raciocínio (84). Dessa forma, ontologias são uma sub-área dentro de representação do conhecimento. Linguagens tradicionais KR são usados para representar ontologias formais, e motores de inferência são usados para o raciocínio sobre ontologias. Além disso, uma boa base de conhecimento terá frequentemente uma ontologia como sua espinha dorsal. A principal diferença é que, ao contrário de KR em geral, o foco principal das ontologias é o compartilhamento de conhecimento.

O objetivo das ontologias é, de modo geral, obter um entendimento compartilhado e comum de um domínio de conhecimento que pode ser comunicado entre pessoas e sistemas de aplicações (21). Dentre as principais categorias de cenários de aplicação de ontologias (84), pode-se citar as duas mais relevantes para este trabalho. São elas:

1. Especificação baseada em Ontologia: é quando uma ontologia de um determinado domínio é criado e usado como base para a especificação e desenvolvimento de algumas aplicações. A ideia é criar uma ontologia que caracteriza e especifica as coisas que o sistema de software deve resolver, e depois usar essa ontologia como um conjunto (mesmo que parcial) de requisitos para a construção do software. A ontologia é usada como base para o desenvolvimento de software. Isso garantiria uma interoperabilidade mais fácil entre os sistemas de software cujas relações são normalmente apenas implícitas. Os benefícios da especificação baseada em ontologias são melhor vistos quando há uma ligação formal entre a ontologia e o software;
2. Busca baseada em Ontologia: com o objetivo de facilitar uma busca, uma ontologia é usada como um dispositivo de estruturação de um repositório de informações (por exemplo, documentos, páginas web, proveniência, recursos, etc), que apoia a organização e classificação de repositórios de informações em um nível maior de abstração do que é usado nas abordagens tradicionais. Ontologias são adequadas para a busca semântica intrínseca nos relacionamentos das informações. Embora este seja adequado para usuários humanos, torna-se impossível para uma máquina para

fazer raciocínio automatizado com resultados previsíveis. O uso de ontologias para estruturar repositórios de informação também implica na utilização de técnicas de indexação semântica, ou a adição de anotações semânticas para os "recursos". Se repositórios diferentes são indexados em ontologias diferentes, então um sistema de acesso a informações semanticamente integradas poderia implantar mapeamentos entre as ontologias diferentes e recuperar respostas de múltiplos repositórios.

Neste sentido, o uso da linguagem OWL (89; McGuinness e van Harmelen) é de grande importância. Assim, vários assuntos que dizem respeito a linguagem OWL devem ser analisados a fim de se atingir o objetivo de se obter metadados semânticos. A seção 3.3.3 apresenta mais detalhes sobre a linguagem OWL.

3.2.2 Tipos de Ontologias

Diversas são as classificações para os tipos de ontologia registradas na literatura. As classificações presentes na literatura definem as ontologias quanto a sua função, ao grau de formalismo, à aplicação, a estrutura e ao conteúdo. (3), em sua revisão de literatura, apresenta estas classificações sintetizando-as de forma mais precisa estas. De forma sintética, podemos classificar as ontologias e dois critérios principais: quanto a natureza e quanto ao grau de formalismo.

Os principais tipos de ontologias, classificados quanto a natureza são:

- **Ontologias de Domínio:** descrevem conceitos e vocabulários relacionados a domínios particulares. É o tipo mais comum de ontologia e, geralmente, é construída para representar um domínio bastante específico. Além disso, fornece vocabulários específicos sobre os conceitos, seus relacionamentos e suas regras.
- **Ontologias Genéricas:** São ontologias que definem conceitos genéricos e aplicáveis a qualquer domínio, e com caráter bastante abstrato.
- **Ontologias de Tarefa:** São utilizadas para descrever tarefas ou atividades genéricas. Dessa forma, contribuem para a resolução de problemas, independente do domínio que ocorrem.
- **Ontologias de Aplicação:** Define conceitos necessários para se modelar o conhecimento em uma aplicação. No entanto, dependem tanto de um domínio particular (ontologia de domínio) quanto de uma tarefa específica. São especializações dos termos das ontologias de domínio e de tarefas correspondentes. Os conceitos normalmente correspondem a regras aplicadas a entidades de domínio enquanto executam determinada tarefa.

Quanto ao grau de formalismo, as ontologias podem ser:

- **Altamente informais:** são expressas em linguagem natural, isentas de qualquer tipo de controle;
- **Informais:** Os termos que serão utilizados na representação do conhecimento do domínio devem ser especificados;
- **Semi-informais:** são expressas em linguagem natural de forma restrita e estruturada.
- **Semiformais:** são expressas em linguagem artificial definida formalmente;

- Formais: São as que utilizam de lógica de primeira ordem, onde os termos do vocabulário aparecem como predicados unários ou binários, chamados respectivamente de conceitos e relações;
- Rigorosamente formais: onde os termos são definidos com semântica formal, teoremas e provas.

3.2.3 Metodologias de Construção de Ontologias

Para construir ontologias de alta qualidade, os desenvolvedores de ontologias devem escolher e seguir uma metodologia adequada que contenha uma série de passos, atividades e diretrizes (incluindo a articulação dos propósitos, a aquisição de conhecimento e a formalização e avaliação) de uma maneira organizada e sistemática. Cada metodologia pode ter suas próprias características. No entanto, a maioria das metodologias concordam que o reuso de ontologias é essencial no processo de desenvolvimento. Assim é a *Methontology* (28). Dentre todas as metodologias propostas, a *Methontology* se destaca na cobertura de todas as etapas do desenvolvimento de uma ontologia. Além disso, sugerir um ciclo de vida evolutivo, de maneira que as regras sejam geradas e testadas evolutivamente.

O ciclo de vida proposto pela *Methontology* segue as seguintes atividades:

- Especificação: é a fase da produção de documentos de especificação informal, semi-formal ou formal da ontologia. Utiliza-se de questionamentos de competência e um conjunto de representações intermediárias, tais como propósito, nível de formalidade e escopo da ontologia;
- Aquisição de conhecimento: atividade independente no processo de desenvolvimento. A maior parte da aquisição deve ser feita simultaneamente com a fase de especificação de requisitos, e diminui à medida que o processo de desenvolvimento de ontologias avança;
- Conceitualização: fase de estruturação do domínio do conhecimento. O domínio será estruturado em um modelo conceitual que descreve o problema e as soluções em termos do domínio do vocabulário identificado na atividade de especificação;
- Formalização: formalização do modelo conceitual;
- Integração: fase que possui o objetivo de acelerar a construção de sua ontologia. Pode-se considerar a reutilização de definições já construídos em outras ontologias;
- Implementação: fase de codificação da ontologia em uma linguagem formal, como por exemplo OWL (89). Para isso, pode ser utilizada ferramentas de construção como o Protégé (81).

3.3 LINGUAGENS E FERRAMENTAS PARA MANIPULAÇÃO DE ONTOLOGIAS

A necessidade de linguagens e ferramentas para construção, anotação, consulta e integração de ontologias é incontestável. Entretanto, somente a representação de conhecimento e informação não é suficiente. Pessoas e/ou agentes da Web que buscam informações necessitam usar e consultar ontologias e os recursos inerentes a elas. Nesta seção, o propósito é apresentar brevemente as linguagens e ferramentas que serão utilizadas neste trabalho. Dentre elas, linguagens para construção de ontologias, linguagem para definição de regras e linguagens para

consultas; além das ferramentas associadas que objetivam o armazenamento, manutenção e anotação em ontologias. Esta lista de linguagens e ferramentas obviamente não é completa. Portanto, são apenas indicativos da tendência para se fornecer suporte total ao armazenamento e consultas para os padrões de metadados/ontologias baseados na Web, tal como XML (*eXtensible Markup Language*)² (88; 14), RDF (*Resource Description Framework*)³ (16), RDFS (*Resource Description Framework*)⁴ (16) e OWL (89).

Todas estas linguagens fazem parte do advento da Web Semântica proposto por (10).

3.3.1 XML

A proposição da arquitetura da Web Semântica é baseada em XML (9). XML é uma linguagem de marcação e teve por objetivo inicial a criação de uma linguagem que pudesse ser lida por software. Atualmente é adotada como recomendação da W3C como a linguagem de marcação que facilita a integração na troca de documentos eletrônicos pela Web, de uma maneira simples, estruturada e extensível.

Entretanto, com o advento da Web Semântica, XML mostra algumas limitações. Segundo (23), a maior limitação do XML é que ela somente descreve gramática. Tem-se a liberdade para definir e usar *tags*, atributos e outras primitivas da linguagem de um modo arbitrário, designando diferentes semânticas para descrever o modelo do domínio conceitual que se deseja. No entanto, visto que XML não impõe regras para tal descrição e que há muitos meios de denotar "coisas" semanticamente equivalentes, torna-se difícil a reconstrução do significado semântico de um documento XML.

3.3.2 RDF e RDF Schema

A RDF é uma linguagem declarativa que utiliza a sintaxe XML para ser escrita. RDF tornou-se um padrão recomendado pelo W3C para representar metadados ou qualquer tipo de informações no formato de sentenças sobre propriedades e relacionamentos entre recursos (texto, vídeo, figuras, música, etc) na Web. Os recursos são representados através de suas URI's⁵ (*Uniform Resource Identifier*).

Os arquivos RDF são representados por triplas, que denominam-se *triplas RDF*. Estas triplas possuem os componentes: recurso, propriedade e indicação.

- Sujeito (*Subject*): tem o objetivo de descrever o recurso em questão, ou seja, qualquer recurso na Web que contenha uma URI.
- Predicato (*Predicate*): descreve a propriedade de um recurso.
- Objeto (*Object*): é a relação entre um recurso e uma propriedade. Pode conter valores ou outros recursos.

RDF é independente de domínio e possui a finalidade de embutir semântica nos documentos que descreve. Assim pode fornecer a interoperabilidade e a semântica para metadados de modo a facilitar a busca por recursos na Web. A semântica oferecida por RDF é simples e suficiente para a representação de metadados. Porém, não fornece subsídios necessários para uma linguagem de ontologias.

²<http://www.w3.org/XML/>

³<http://www.w3.org/RDF/>

⁴<http://www.w3.org/TR/rdf-schema/>

⁵<http://www.w3.org/Addressing/>

Assim, o RDF Schema surge como uma linguagem para a definição de vocabulários, estruturas (classes) e restrições (*constraints*) para expressar metadados sobre os recursos da Web. Entretanto a semântica formal para as primitivas definidas em RDF Schema não são fornecidas. Sendo assim, a expressividade destas primitivas não são suficientes para a completa modelagem e raciocínio ontológico. Para a execução destas tarefas, uma camada adicional no topo do RDF Schema é necessária. Até no mais baixo nível da Web Semântica, um mecanismo genérico para expressar a semântica dos dados processáveis automaticamente é requerida (10). O RDF é a base para este processamento de metadados, fornecendo um simples modelo de dados e uma sintaxe padronizada para metadados. Basicamente, ele provê a linguagem para a escrita das declarações fatuais. Em uma camada superior, está a camada de esquema (*Schema*), fornecida pela especificação RDF Schema (23; 21).

Um exemplo de um descrição em RDF é apresentada a seguir:

```

1 <?xml version="1.0"?>
2 <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
3   <ex:editor>
4     <rdf:Description>
5       <ex:homePage>
6         <rdf:Description rdf:about="http://purl.org/net/dajobe/">
7         </rdf:Description>
8       </ex:homePage>
9     </rdf:Description>
10   </ex:editor>
11 </rdf:Description>

```

RDF Schema possuem primitivas que vão além das primitivas de modelagem de RDF. Exemplos são classes, subclasses, restrições de domínio (*domain*) e alcance (*range*) para propriedades e subpropriedades. Com estas extensões, RDF Schema chega perto das linguagens para ontologias, tal como OWL, que veremos a seguir.

3.3.3 OWL - Web Ontology Language

Um pré-requisito para difundir o uso de ontologias é a junção de padrões para sua descrição e troca (40). RDF Schema⁶ (RDFS) já é reconhecida como uma linguagem para ontologia e representação de conhecimento: ela trata de classes e propriedades (relações binárias), restrições de alcance (*range*) e domínio (*domain*) sobre as propriedades, além de relações subclasses (*subClassOf*) e subpropriedades (*subPropertyOf*). RDFS é uma linguagem relativamente primitiva, e mais poder de expressividade deveria ser necessário e desejável para a descrição de recursos em suficiente detalhamento. Além disso, estas descrições deveriam ser disciplinadas para raciocínio automático se elas forem utilizadas efetivamente para processos automáticos. Como dito anteriormente, estas considerações levam ao desenvolvimento da linguagem OWL (89) que estende RDFS com um conjunto mais rico de primitivas de modelagem. As aplicações mais sofisticadas de busca e especificação baseadas em ontologias exigem uma especificação mais formal através de um padrão de linguagem ontológica tal como OWL.

OWL é uma linguagem ontológica, e como tal, foi definida para descrever a estrutura de um domínio. OWL assume uma abordagem orientada a objetos, com a estrutura do domínio sendo descrita em termos de classes e propriedades. Uma ontologia consiste de um conjunto de axiomas que são declarados; por exemplo, classificação de relacionamentos entre classes e propriedades.

⁶<http://www.w3.org/TR/rdf-schema/>

OWL foi projetada para ser usada por aplicações que necessitam processar o conteúdo da informação ao invés de apenas apresentar as informações para os seres humanos. OWL facilita uma maior interpretabilidade pela máquina de conteúdo da Web que vai além do suportado por XML⁷, RDF⁸ e RDF Schema (RDF-S)⁹, fornecendo vocabulário adicional juntamente com uma semântica formal. Além disso, OWL é uma linguagem de marcação semântica para publicação e compartilhamento de ontologias na *World Wide Web*. Esta linguagem foi desenvolvida como uma extensão do vocabulário de RDF e é derivado do DAML+OIL¹⁰.

As ontologias OWL podem ser classificadas em três espécies de acordo com a sublinguagem utilizada: OWL LITE, OWL DL e OWL FULL¹¹. A característica principal de cada sublinguagem é a sua expressividade. A OWL LITE é a menos expressiva; a OWL FULL é a mais expressiva; a expressividade da OWL DL está entre as duas, entre a OWL LITE e a OWL FULL. A OWL DL baseia-se em lógica descritiva (*Descriptions Logics* - DL) (68), um fragmento de Lógica de Primeira Ordem, passível portanto de raciocínio automático. É possível assim computar automaticamente a hierarquia de classes e verificar inconsistências na ontologia.

3.3.3.1 Componentes da Ontologia OWL

As ontologias OWL possuem quatro componentes básicos:

- Indivíduos (*Individuals*):

Representam objetos no domínio de interesse (ou domínio do discurso). Os indivíduos são também conhecidos como instâncias. Os indivíduos podem ser referenciados como Instâncias de Classes, ou seja, são as "coisas" ("*things*") representadas por uma classe.

- Relações e Propriedades (*Relation and Properties*):

Assim como as classes, relações podem ser organizadas em taxonomias. As relações representam a associação entre os indivíduos.

As relações possuem propriedades que capturam o conhecimento sobre os relacionamentos entre as classes. As propriedades são relações binárias entre indivíduos, ou seja, as propriedades ligam dois indivíduos. As propriedades são conhecidas como papéis em lógica descritiva, e relações em UML-Unified Modeling Language e em outras abordagens de orientação a objeto. Em alguns formalismos são denominadas de atributos.

As funções são eventos que podem ocorrer no contexto da ontologia. Por exemplo, para um indivíduo, haverá no máximo um outro indivíduo relacionado a ele por meio da função.

- Classes (*Classes*):

São conjuntos que contêm os indivíduos. Elas são descritas formalmente (descrições matemáticas) de forma que sejam apresentados os requisitos para a participação na classe. As classes podem ser organizadas em hierarquias superclasse-subclasse, também conhecidas como taxonomias. Subclasses são especializações de suas superclasses. O termo conceito é às vezes usado no lugar de classe. As classes são representações

⁷<http://www.w3.org/XML/>

⁸<http://www.w3.org/RDF/>

⁹<http://www.w3.org/TR/rdf-schema/>

¹⁰<http://www.w3.org/TR/daml+oil-reference>

¹¹<http://www.w3.org/TR/owl-ref/>

concretas de conceitos. Em OWL, as classes são construídas a partir de descrições, as quais especificam as condições que devem ser satisfeitas por um indivíduo para que ele possa ser um membro da classe.

- Axiomas (*Axioms*)

São as afirmações descritas de forma lógica, juntamente com as regras, e que compõem a idéia geral do que a ontologia descreve em seu domínio de aplicação. Axiomas devem ser fornecidas de forma a fixar a semântica dos termos, e estabelecer restrições de domínio (27). Geralmente os axiomas são especificados em linguagem natural e podem ser especificados através de lógica de primeira ordem, como sentenças de primeira ordem usando os predicados da ontologia.

As restrições são descrições formalmente declaradas como verdadeiras para alguma afirmação que deve ser aceita como entrada.

As regras são declarações na forma de uma sentença *if-then* (antecedente-consequente) que descrevem as inferências lógicas que podem ser extraídas a partir de uma afirmação de uma forma particular.

3.3.3.2 Definição de Classes e Propriedades

Em OWL, as definições de classes são providas através do uso de afirmações (declarações) e axiomas. Adicionalmente, podem ser usadas expressões de restrições (*constraints*) que representam o papel de restringir ou quantificar.

As classes têm a finalidade de agruparem recursos que possuam características similares. Cada classe OWL é associada a um conjunto de indivíduos. Estes indivíduos são chamados de instâncias da classe. As classes OWL são descritas a através de classes que descrevem um conjunto de indivíduos. Estas classes podem ser combinadas com classes de axiomas (89). A construção das classes em OWL é diretamente relacionadas ao formalismo utilizado pelas construções RDF e RDS. Abaixo são listadas as principais construções para definição das classes e axiomas para as classes em OWL¹²:

- *owl:Class*: Define o nome da classe que agrupa um conjunto de indivíduos com propriedades similares. É um identificador de classe (uma referência URI).
- *rdfs:subClassOf*: Define uma subclasse de uma classe, sendo um subconjunto de uma categoria de classe.
- *owl:equivalentClass*: Define que uma classe possui o mesmo conjunto de indivíduos que outra classe.
- *owl:disjointWith*: Define que uma classe não tem membros em comum com a outra classe.
- *owl:intersectionOf*: Define uma classe com a intersecção dos indivíduos de duas ou mais classes.
- *owl:unionOf*: Define uma classe com a união de indivíduos que ocorrem em duas ou mais classes.

¹²Para uma completa referência sobre as construções de classes e propriedades, vide <http://www.w3.org/TR/owl-ref/>

- *owl:complementOf*: Define uma classe com indivíduos que não pertencem as outras classes declaradas.
- *owl:allValuesFrom*: Define uma classe de todos os indivíduos, para os quais todos os valores da propriedade em questão são instâncias da classe ou valores de dados dentro do intervalo de dados especificado.
- *owl:someValuesFrom*: Define uma classe de todos os indivíduos, para os quais pelo menos um valor da propriedade em questão é uma instância da classe ou um valor de dados no intervalo de dados especificado.
- *owl:cardinality*: Define a classe de todos indivíduos que tem exatamente a cardinalidade de N valores distintos.
- *owl:minCardinality*: Define a classe de todos indivíduos que tem pelo menos a cardinalidade de N valores distintos.
- *owl:maxCardinality*: Define a classe de todos indivíduos que tem no máximo a cardinalidade de N valores distintos.

A construção de propriedades em OWL são classificadas em duas categorias de propriedades que podem ser definidas: *Object properties* e *Datatype properties*. A primeira relaciona indivíduos a indivíduos, ou seja, define uma propriedade com a restrição de possuir valores que devem ser indivíduos. A segunda relaciona indivíduos a valores de dados, ou seja, define uma propriedade que relaciona um indivíduo a um tipo de dado.

Além disso, pode-se ter construções de axiomas de propriedades. Um axioma de propriedade define as características de uma propriedade. Abaixo segue a lista dos principais axiomas de propriedades que podem ser definidos em OWL:

- Construções RDF Schema: *rdfs:subPropertyOf* (define que uma propriedade é subpropriedade de uma outra propriedade), *rdfs:domain* (vincula uma propriedade a uma classe) e *rdfs:range* (vincula uma propriedade aos valores de uma classe ou a um intervalo de dados);
- Relações para outras propriedades: *owl:equivalentProperty* (usada para declarar propriedade sinônimas) e *owl:inverseOf* (define relações em ambas direções, é usado para definir uma relação inversa entre as propriedades).
- Restrições de Cardinalidade: *owl:FunctionalProperty* (define que uma propriedade só pode ter um único valor x para uma instância y) e *owl:InverseFunctionalProperty* (define que o objeto de uma declaração de propriedade determina exclusivamente o sujeito).
- Características das propriedades lógicas: *owl:SymmetricProperty* (afirma que se o par (x,y) é uma instância de P , então o par (y,x) também é uma instância de P) e *owl:TransitiveProperty* (afirma que se um par (x,y) é uma instância de P , e o par (y,z) é também instância de P ; então podemos inferir que o par (x,z) também é uma instância de P).

3.3.4 Regras em OWL - SWRL

Como foi visto, OWL DL é baseada em descrições lógicas (*Description logics*-DLs) (7). DL é uma linguagem de representação de conhecimento utilizada para a representação de um domínio de aplicação de uma forma estruturada e formal. A noção de domínio é descrita por descrições de conceitos. Sendo assim, a OWL DL acrescenta poder expressivo considerável à Web Semântica. Porém, embora OWL DL inclua um conjunto relativamente poderoso de construtores de classes, ela é relativamente deficiente no tratamento das propriedades, com limitações de expressividade. Isto se deve a algumas razões, incluindo a de manter a decidibilidade dos mecanismos de inferência na OWL DL. Sendo assim, essas limitações podem ser onerosas em alguma aplicação de domínio onde seja necessário relacionar as entradas e saídas de algum processamento às entradas e saídas de outros. Por exemplo, no caso deste trabalho, as regras devem ser aplicadas para se obter, através das entradas de um *Job*, a classificação de uma carga de trabalho.

Por outro lado, RuleML¹³ (*Rule Markup Language*) também é uma linguagem desenvolvida para expressar regras em XML, com o objetivo de se obter dedução, reescrita e tarefas de inferência. RuleML é caracterizada pelas seguintes dimensões: Linguagens naturais e formais, regras de reação e deliberação, serialização XML, apresentação de sintaxes, semântica, tradutores internosexternos e motores de referência, além de expressar regras horizontal e verticalmente (11).

Uma recente linguagem para a Web Semântica, com o objetivo de expressar regras e lógica, é a SWRL¹⁴ (*Semantic Web Rule Language*). SWRL é baseada na combinação de OWL DL com a sublinguagem *Unary/Binary Datalog* (11) da RuleML.

Através de uma sintaxe abstrata de alto nível, a SWRL adiciona um maior poder de expressividade às descrições lógicas da OWL DL. Assim, SWRL estende o conjunto de axiomas OWL para que se possa incluir regras baseadas em lógica de primeira ordem (41).

A definição de uma regra se dá na forma de uma implicação entre um antecedente (corpo) e um conseqüente (cabeça), ou seja, se as condições especificadas no antecedente acontecerem, então as condições especificadas no conseqüente também devem acontecer. A definição das cláusulas (antecedente ou conseqüente) devem consistir de nenhum ou vários átomos e seguem a especificações abaixo:

- Se o átomo for vazio no antecedente, ele é considerado verdadeiro e, assim, o conseqüente também deve ser satisfeito;
- Se o átomo for vazio no conseqüente, ele é considerado falso e, assim, o antecedente também não deve ser satisfeito;
- Átomos múltiplos são tratados como uma conjunção;
- Regras com conseqüentes conjuntivos podem ser transformados em múltiplas regras, cada uma com um conseqüente atômico.

Abaixo é apresentado as formas de construção:

- $C(x)$: C é uma descrição em OWL (uma classe, por exemplo);
- $P(x,y)$: P é uma propriedade OWL e x,y podem ser variáveis ou instâncias OWL ou valores de dados OWL;

¹³http://wiki.ruleml.org/index.php/RuleML_Home

¹⁴<http://www.w3.org/Submission/SWRL/>

Pode-se observar que, através da construção e extensão de regras em SWRL, a OWL DL torna-se indecidível quando se simula a função de mapeamento de valores (79). Assim, uma desvantagem da SWRL é sua complexidade computacional. Mesmo assim, SWRL torna-se bastante útil na descrição de regras e suporta a maioria das características da DL. Portanto, para estender a ontologia proposta através de regras que possam ser adequadas ao raciocínio lógico dos raciocinadores e obter-se respostas mais claras a respeito do domínio modelado, será utilizada a linguagem SWRL.

3.3.5 Ferramenta para Construção de Ontologias - Protégé

Os editores de ontologias são aplicações projetadas para auxiliar na criação e manipulação de ontologias. A maioria deles são capazes de expressar ontologias em muitas linguagens ontológicas.

Protégé (81) é um editor de ontologias de código aberto e um sistema de aquisição de conhecimento. A ferramenta Protégé surgiu de um projeto da Universidade de Stanford ¹⁵ com a colaboração da Universidade de Manchester ¹⁶. A ferramenta fornece uma interface gráfica para definir as ontologias e inclui raciocinadores para classificar e validar os modelos. Dessa forma, verifica-se a consistência e faz-se inferências de novas informações baseadas na análise da ontologia. Além disso, por ser escrito em JAVA, o *framework* pode incorporar vários outros projetos como *plugins*, geralmente escritos em JAVA. Um exemplo disso é o *plugin* OWLViz, que é um editor visual da linguagem OWL.

A plataforma do Protégé suporta dois tipos de modelagem para ontologias: Protégé-Frames e Protégé-OWL. As ontologias criados por este ambiente podem ser exportadas para outros formatos de linguagem semântica, tais como RDF, RDF-Schema, OWL, XML-Schema, etc.

Por ser o mais difundido e consolidado editor de ontologias de código aberto, Protégé-OWL será o editor utilizado por este trabalho.

3.3.6 Raciocinadores Lógicos e Inferências

Uma das principais características das ontologias escritas em OWL DL é que elas podem ser processadas por raciocinadores. Um raciocinador é um programa que infere consequências lógicas de um conjunto de fatos afirmados explicitamente ou axiomas, e normalmente fornece suporte automatizado para tarefas de raciocínio, tais como classificação, depuração e consultas (1).

A ferramenta Protégé, permite diferentes raciocinadores OWL através de *plugins* dos mesmos. Dentre o grande número de raciocinadores disponíveis, podemos citar o Fact++ ¹⁷, Pellet ¹⁸, RACER ¹⁹, Hermit ²⁰, entre outros. Estes raciocinadores possuem vários atributos importantes (características) como algoritmos de inferência, suporte a lógica, grau de amplitude do raciocínio, linguagem de implementação, etc (24).

Os serviços de raciocínio oferecidos pelas descrições lógicas (DL) suportam o desenvolvimento e a manutenção incremental de uma ontologia. O conceito fundamental de modelagem de uma DL é o axioma. Axioma é uma afirmação lógica que relaciona papéis e/ou conceitos. DL

¹⁵<http://www.stanford.edu/>

¹⁶<http://www.manchester.ac.uk/>

¹⁷<http://owl.cs.manchester.ac.uk/tools/fact/>

¹⁸<http://clarkparsia.com/pellet/>

¹⁹<http://racer.sts.tuhh.de/>

²⁰<http://hermit-reasoner.com/>

é utilizada na inteligência artificial para raciocínio formal sobre os conceitos de um domínio de aplicação. Assim, uma ontologia expressada utilizando-se OWL pode ser verificada usando-se um raciocinador lógico (24). A chave dos serviços de raciocínio são:

- Checagem de classificação entre duas descrições de conceitos, C e D; C inclui D quando o conjunto de objetos que são instâncias de D são sempre um subconjunto de objetos que são instâncias de C.
- A classificação organiza uma coleção de expressões de conceitos dentro de uma ordem parcial baseada na checagem de classificação. Isto provê uma rede simétrica de definições, dispondo-se do geral para o específico. As definições compostas têm sua posição implicitamente determinada automaticamente. Assim, a classificação é um processo dinâmico que pode ser adicionado a uma hierarquia existente.
- A satisfação (*satisfiability*) dos conceitos checa se uma descrição de conceito nunca pode ter instâncias por causa das inconsistências ou contradições do modelo.

Uma importante característica das ontologias descritas com o OWL DL é a possibilidade de processamento através de um raciocinador (mecanismo de inferência) ou classificador. Isso é um relevante serviço oferecido pelos mecanismos de inferência. Assim, as descrições das classes (condições) são utilizadas para determinar se existe dentre elas uma relação superclasse/subclasse. Através dos testes em todas as classes de uma ontologia, é possível inferir automaticamente a hierarquia de classes da ontologia. Outro serviço padrão oferecido pelo mecanismo de inferência é o de *consistency checking* (verificação da consistência). Baseado na descrição (condições) de uma classe, o mecanismo de inferência pode verificar se é possível, ou não, que uma classe possua instâncias. Uma classe é considerada inconsistente se não for possível que ela contenha instâncias.

Quando classifica-se uma ontologia, um número de novos relacionamentos de classificação pode ser descoberto (devido as definições da classe no modelo). Isto pode ser útil durante a fase de construção da ontologia. OWL DL é equivalente a uma descrição lógica muito expressiva. OWL DL pode explorar uma parte considerável das pesquisas em descrições lógicas. Por exemplo:

- Para definir a semântica da linguagem e o entendimento formal das suas propriedades, particularmente a decidibilidade e complexidade dos problemas chaves de inferência;
- Como fonte de algoritmos sólidos e completos, e técnicas de implementação otimizada para determinantes problemas chaves de inferência;
- Para usar sistemas de descrições lógicas implementados para fornecer suporte (parcial) a raciocínio (42).

Uma consideração importante no objetivo de OWL é que os problemas chaves de inferência na linguagem, particularmente consistência e classificação de classes, deveriam ter capacidade de decisão, a medida que isto facilitaria o fornecimento dos serviços de raciocínio. A manutenção da decidibilidade da linguagem requer certas restrições sobre seu poder de expressividade, que podem não ser aceitas por todas as aplicações.

Raciocínio pode ser útil em muitos estágios durante o projeto, manutenção e desenvolvimento das ontologias.

- Raciocínio pode ser usado para dar suporte ao projeto da ontologia e para melhorar a qualidade dos resultados da ontologia. Por exemplo, raciocínio da consistência e classificação das classes pode ser usado para verificar classes logicamente inconsistentes e relacionamentos de classificação implícitas. Este tipo de suporte tem sido muito importante em grandes ontologias, nas quais são frequentemente construídas e mantidas por um longo período e por muitos autores;
- Como integração de informação (70), a integração de ontologias pode também ser suportada pelo "raciocínio". Por exemplo, a integração pode ser executada usando afirmações inter-ontologias especificando relacionamentos entre classes e propriedades, com raciocínio sendo usado para computar a hierarquia integrada e destacando-se problemas ou inconsistências;
- Raciocínio, no que diz respeito ao desenvolvimento de ontologias aumentará o poder dos "agentes inteligentes", permitindo-os determinar se um conjunto de fatos esta consistente em relação a ontologia, identificando objetos que são membros implícitos de uma dada classe, etc. Um adequado serviço de ontologia deve, por exemplo, permitir um agente procurar serviços seguros para identificar um serviço requerendo um usuário e senha como um possível candidato.

3.3.7 Linguagem de Consulta - SPARQL

SPARQL (*SPARQL Protocol and RDF Query Language*) (74) é uma linguagem de consulta projetada para recuperar e manipular dados no formato RDF. RDF define a sintaxe e semântica para linguagem SPARQL. Portanto, SPARQL compreende, além da sintaxe, o modelo de dados do RDF e a semântica dos vocabulários das ontologias representadas em OWL.

A linguagem de consulta SPARQL se apóia no reconhecimento de padrões de grafos, representados por uma tripla, tal como a tripla RDF. Porém, variáveis podem ser acrescentadas além do sujeito, predicado e objeto das triplas RDF.

A seguir é apresentado um exemplo da sintaxe SPARQL:

```

1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX owl: <http://www.w3.org/2002/07/owl#>
3 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 SELECT ?subject ?object
6     WHERE { ?subject rdfs:subClassOf ?object }
7     ORDER BY ?subject ?object

```

A primeira característica que se nota é que, assim como no RDF, pode-se adotar abreviações para representar os *namespaces* (URI's) muito extensos. Isso tem a finalidade de se obter maior facilidade na leitura e compreensão da construção da consulta.

A consulta acima retorna todos os grafos com triplas que possuam o predicado *rdfs:subClassOf* relacionados a qualquer recurso (*?subject*), apresentados através da variável *?x* que representa um *object*.

Sendo assim, uma consulta SPARQL deve ser composta por:

- Declarações dos prefixos com o objetivo de abreviar os *namespaces* (URI's). Por exemplo: *PREFIX foo: <http://example.com/resources/>;*
- Definição do conjunto de dados, informando quais grafos RDF estão sendo consultados. Por exemplo: *FROM;*

- A cláusula de resultado que identifica quais informações devem ser recuperadas através da consulta. Por exemplo: *SELECT*;
- O padrão de consulta, especificando o que consultar dentro do conjunto de dados. Por exemplo: *WHERE*;
- Modificadores de consulta, limites, ordenação, e outros que podem modificar o resultado final. Por exemplo: *ORDER BY*.

São quatro os tipo de consultas da SPARQL:

- *SELECT*: cláusula utilizada para extrair resultados da consulta, apresentados em forma de tabela;
- *CONSTRUCT*: retorna valores construídos a partir de variáveis e as transforma em um conjunto de triplas RDF válidas;
- *ASK*: retorna valores booleanos para uma consulta;
- *DESCRIBE*: retorna um grafo que descreve os recursos encontrados.

Por fim, vale ressaltar que SPARQL é orientada a dados. Sendo assim, ela não suporta a semântica das informações, ou seja, não é capaz de realizar inferências sobre a ontologia. Esta função, como já foi dito, fica sobre responsabilidade dos raciocinadores lógicos, que classificam e geram as triplas RDF.

3.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou o conceito e as características das ontologias. Destacou, também, as ontologias como uma potencial ferramenta semântica para representação do conhecimento. Ontologias têm sido utilizadas em diversas comunidades de pesquisa, e aplicadas em diversos domínios, por ser uma forma de representação do conhecimento poderosa, que permite, por exemplo, criar restrições gerais e formais (ou axiomas) entre conceitos, atributos e relações. Um dos objetivos de apresentar este capítulo é que a abordagem que este trabalho pretende apresentar utilizará técnicas semânticas, baseadas em ontologias, para se obter a melhor sintonia dos parâmetros de configuração do *framework* Hadoop MapReduce. Esta ontologia será modelada e construída para os fins citados. Sendo assim, neste capítulo foram apresentadas o conceito de metadados semânticos, uma breve explanação sobre a Web Semântica, a definição das ontologias, assim como seus tipos, metodologia de construção, componentes e linguagens. Além disso, foi apresentada a relação entre ontologias e representação do conhecimento, bem como a relação com raciocínio lógico e inferências. Além disso, algumas linguagens base para OWL, tais como XML, RDF e RDFS foram introduzidas, bem como a linguagem de consulta SPARQL.

Após análise do contexto geral sobre ontologias, definiu-se a utilização, para este trabalho, da linguagem ontológica OWL DL. A linguagem OWL DL possui uma grande expressividade e ainda mantém a integridade computacional que garante que todas as conclusões são computáveis. Além disso, mantém a decidibilidade, que garante que todas inferências terminem em tempo finito. Isto é possível porque a OWL DL baseia-se na lógica descritiva e, assim, permite a utilização de raciocínio lógico. Algumas regras da ontologia serão baseadas na linguagem SWRL.

Além de OWL DL e SWRL, far-se-á uso da ferramenta para construção de ontologias Protégé, e da metodologia *Methontology* para a definição da ontologia que esta tese propõe.

Além disso, de acordo com a natureza da abordagem semântica e do propósito desta tese, definiu-se, por ser a mais adaptada ao contexto, a construção de uma ontologia de aplicação, com um grau de construção formal.

No entanto, o primeiro passo para a construção de uma ontologia, mesmo que seja considerada uma atividade independente (28), é o processo de aquisição de conhecimento. Sendo assim, o próximo capítulo apresenta uma revisão sistemática neste sentido.

4 REVISÃO SISTEMÁTICA DA LITERATURA - PARÂMETROS X DESEMPENHO

Este capítulo apresenta uma revisão sistemática (12), identificando trabalhos de pesquisa atuais, que abordam a correlação entre as definições de configuração do Hadoop e o desempenho. O capítulo explora como a sintonia dos parâmetros de configuração do Hadoop impactam no desempenho geral do sistema. Mais especificamente, esta pesquisa deve responder as seguintes questões:

- (1) Quais parâmetros de configuração têm influência e impactam no desempenho do sistema?
- (2) Quais parâmetros são influenciados pelas fases do Hadoop?
- (3) Quais parâmetros são influenciados pelas características das cargas de trabalho?

Neste sentido, esta revisão sistemática identificou 743 artigos em cinco bases de dados pesquisadas. Aplicando as etapas para a seleção e alguns critérios de exclusão, o número de artigos final foi reduzido para 40. Estes documentos foram analisados e classificados de acordo com os parâmetros de configuração do Hadoop e os objetivos dos estudos.

Este capítulo está organizado como a seguir. Seção 4.1 descreve o método de pesquisa aplicado a revisão sistemática. A seguir, os resultados obtidos são apresentados na Seção 4.2 e discutidos na Seção 4.3. A Seção 4.5 apresenta as considerações finais do capítulo.

4.1 MÉTODO DE PESQUISA

Um método que ganhou popularidade significativa nas várias áreas de pesquisa foi a Revisão Sistemática de Literatura (ou revisão sistemática). Especificamente em *Big Data*, são poucas as revisões sistemáticas encontrados. Portanto, é um campo a ser explorado. Mais especificamente, não se encontrou nenhuma revisão sistemática sobre o desempenho de sistemas Hadoop relacionados aos seus parâmetros de configuração e sintonia. Neste sentido, um comentário sobre este assunto de uma forma sistemática e rigorosa, será valiosa para a pesquisa e a prática.

Revisão sistemática é uma revisão metodológica rigorosa dos resultados da investigação. Além de adicionar a evidência existente sobre um tema de pesquisa específico, uma revisão também se destina a apoiar o desenvolvimento de diretrizes e consensos para estudos futuros, com base em evidências de vários estudos anteriores. Portanto, uma revisão sistemática tem como objetivo identificar, avaliar e interpretar a pesquisa relevante sobre um determinado assunto, uma área temática ou fenômenos de interesse usando um método explícito e rigoroso (50).

Uma revisão sistemática consiste de especificar um tópico de pesquisa, definir e documentar uma estratégia de pesquisa, bem como realizar a avaliação e alcançar conclusões (50; 51). Dentre os vários motivos para fazer uma revisão sistemática (50), destacamos para este trabalho:

- Identificar as direções que as pesquisas recentes sobre o assunto em questão estão tomando;
- Fornecer uma base para os novos pesquisadores da área, a fim de posicioná-los adequadamente sobre o que foi feito até o momento, e do consenso da comunidade.

Uma revisão sistemática envolve várias atividades distintas que, de acordo com o (51), resumem os estágios da revisão sistemática em três fases principais: planejamento, condução

e relato da revisão. As próximas subseções detalham as medidas tomadas para produzir este estudo.

4.1.1 Planejando a Revisão

O planejamento da revisão sistemática começou por desenvolver um protocolo. Este protocolo especifica o processo e os meios que devem ser aplicados. O protocolo especifica as questões de pesquisa, a estratégia de pesquisa e os critérios de inclusão e exclusão. Esta revisão sistemática tem como objetivo fornecer uma visão geral dos estudos que se aplicam ao desempenho dos sistemas Hadoop no que diz respeito aos seus parâmetros de configuração, respondendo às perguntas de pesquisa mencionados na introdução deste capítulo.

4.1.2 Conduzindo a Revisão

Assim que o protocolo foi acordado, as etapas para a realização da revisão são apresentadas abaixo.

4.1.2.1 Identificação de Pesquisa

Uma revisão sistemática tem como objetivo encontrar o maior número possível de estudos primários relacionados ao tema de pesquisa. Assim, o primeiro passo é a identificação de palavras-chave e os termos de pesquisa. Neste estudo, as palavras-chave e termos têm sido utilizados a fim de se obter os trabalhos relevantes de uma forma mais ampla, em relação ao universo do estudo (64). Todas as possíveis permutações das palavras "*Hadoop*" e "*parameters*" foram consideradas. Além disso, variamos os termos relacionados a "*performance*" (ver Tabela 4.1). Por exemplo, a primeira seqüência de pesquisa foi: "*Hadoop*", "*Performance Tuning*" e "*parameters*". Todas os termos de busca foram pesquisados por sentença completa, título e palavras-chave em todos os bancos de dados listados na Tabela 4.2. Estas bases de dados foram escolhidas devido a sua relevância na comunidade científica (50), e pela indexação das conferências, *workshops* e revistas mais importantes de Processamento Paralelo e Distribuído. Como o objetivo deste estudo é entender o desempenho do *framework* Hadoop, através da definição e sintonia dos seus parâmetros de configuração; e este é um tema de pesquisa relativamente novo, apenas estudos a partir do ano 2008 foram pesquisados. A pesquisa foi realizada em outubro de 2013, o que significa que as publicações do final de 2013 até agora podem não ter sido indexadas pelos bancos de dados. O processo de identificação produziu 1181 artigos. Isto constituiu a base para o próximo passo.

4.1.2.2 Seleção dos Principais Trabalhos

O primeiro passo, após a identificação preliminar dos artigos foi a eliminação dos títulos duplicados. A execução desta etapa reduziu o conjunto para 743 artigos. O próximo passo foi remover os títulos claramente não relacionados com a revisão. Esta etapa reduziu o conjunto para 296 artigos. Em seguida, os resumos foram lidos e avaliados com os seguintes critérios de exclusão:

- excluir artigos que claramente não eram da área do sistema Hadoop, ou não aplicados ao desempenho do Hadoop ou ao desempenho da aplicação;
- excluir artigos que não eram relacionados com os termos de sintonia de desempenho, análise, predição, modelo ou de auto-ajuste;

Tabela 4.1: Termos para busca

Termos sobre Desempenho
Performance Tuning
Performance Analysis
Performance Prediction
Performance of MapReduce
Performance Model
Hadoop Performance
Job Performance
Self-Tuning

Tabela 4.2: Bases de Dados pesquisadas na revisão sistemática

Bases de Dados Pesquisadas
ACM Digital Library
IEEE Xplore Digital Library
Science Direct
Springer Link
Scopus (Elsevier)

- excluir artigos que não eram relacionados a sintonia de parâmetros ou configuração, bem como as suas implicações sobre o desempenho;
- excluir artigos que eram apenas de revisão da literatura,
- excluir artigos sem método de validação, caso estes não fossem claramente um estudo prático.

Após a execução deste processo, o conjunto de artigos foi reduzido para 124. Assim, o texto completo para todos os 124 artigos foi obtido e lido, sendo aplicados os mesmos critérios de exclusão. O número final de trabalhos selecionados para a revisão foi de 40 artigos, nos quais estavam claramente ajustados aos critérios definidos para realização da pesquisa sistemática, que foram então analisados.

4.1.2.3 Estudo de avaliação da qualidade e classificação

Após a leitura e análise dos trabalhos selecionados, o próximo passo foi a classificação dos mesmos de acordo com os parâmetros estudados em cada artigo. Devido à falta de literatura específica sobre o tema discutido nesta revisão sistemática, a classificação dos parâmetros baseou-se nas definições e classificações adotadas no livro *Pro Hadoop* (86), que é uma das referências mais citadas nos artigos analisados; e com base em nossa própria compreensão. Isto se deu ao fato de não existirem classificações ratificadas pela comunidade científica. Nesta revisão sistemática, o propósito da classificação dos parâmetros, é responder às perguntas elaboradas no início do capítulo, além de apresentar quais parâmetros foram identificados e estudados nas obras que lidam com o desempenho do Hadoop.

4.1.2.4 Síntese dos Dados

Para a síntese, foram extraídos os parâmetros de configuração do Hadoop que tem influência sobre o desempenho, os objetivos centrais e os métodos de experimentação.

4.2 RESULTADOS

Os resultados deste estudo foram divididos em três seções principais, com o objetivo de responder às perguntas iniciais. Seção 4.2.1 foca na identificação dos parâmetros que foram utilizados nos estudos detectados, Seção 4.2.2 foca sobre quais parâmetros são afetados por cada fase Hadoop e Seção 4.2.3 se concentra na identificação de quais parâmetros estão relacionados com as características cargas de trabalho.

A Tabela 4.3 resume os parâmetros encontrados. A classificação utilizada na tabela responde às três primeiras perguntas desta revisão sistemática. As discussões sobre o assunto seguem nas próximas seções.

4.2.1 Parâmetros de Configuração

De acordo com a classificação adotada, foram identificados todos os parâmetros que influenciam de alguma forma no desempenho do sistema e que foram estudados nos artigos identificados por esta revisão.

Tabela 4.3: Parâmetros de Configuração.

Categoria	Nível	Fase	Característica da Carga de Trabalho	Parâmetro	Artigos Científicos
Hadoop	Cluster Level	Merge/ Shuffle	IO	dfs.replication	(90; 92; 33; 8; 54; 58; 44; 91; 59; 95; 2; 34; 61; 72)
				dfs.replication.interval	(90; 92; 33; 44; 91; 59)
				mapred.min.split.size	(35; 60)
			Number of maps	dfs.block.size	(90; 92; 33; 8; 4; 54; 97; 58; 44; 91; 59; 53; 75; 45; 94; 83; 78; 48; 95; 2; 34; 102; 61; 73; 35; 71)
	Job Level	Core Job	CPU	mapred.compress.map.output	(54; 94; 95; 35; 37; 36; 39; 60)
				mapred.job.reuse.jvm.num.tasks	(61; 35; 60)
				mapred.output.compression.type	(95; 35; 60)
				mapred.reduce.slowstart.completed.maps	(2; 72; 37; 36; 39; 60)
				mapred.child.java.opts	(8; 58; 59; 95; 35; 60)
			Memory	mapred.reduce.parallel.copies	(95; 35; 56; 34)
			Network		
		Map	CPU	mapred.map.tasks speculative.execution	(2; 60)
				mapred.tasktracker.map.tasks.maximum	(90; 92; 33; 54; 58; 44; 91; 59; 94; 95; 2; 73; 35; 71; 60; 93; 46)
				mapred.map.output.compression.codec	(95; 35; 60)
		Merge/ Shuffle	IO	io.file.buffer.size	(90; 92; 33; 8; 44; 91; 59; 94; 61)
				io.sort.factor	(90; 92; 33; 8; 58; 44; 91; 59; 48; 95; 2; 34; 61; 37; 36; 39; 56; 60)
				io.sort.mb	(90; 92; 33; 8; 58; 44; 91; 59; 95; 35; 37; 36; 39; 56; 60; 55)
				io.sort.record.percent	(90; 92; 33; 8; 58; 44; 91; 59; 35; 37; 36; 39; 56; 60; 55)
				io.sort.spill.percent	(90; 92; 33; 44; 91; 59; 94; 61; 73; 35; 37; 36; 39; 56; 60)
			Memory	mapred.inmem.merge.threshold	(8; 59; 2; 35; 37; 36; 39; 60)
				mapred.job.reduce.input.buffer.percent	(8; 59; 35; 37; 36; 39; 56; 60)

Continua na próxima página

Tabela 4.3 – Continuação da página anterior

Categoria	Nível	Fase	Característica da Carga de Trabalho	Parâmetro	Artigos Científicos
		Reduce	CPU	mapred.job.shuffle.put.buffer.percent	(8; 59; 94; 35; 37; 36; 39; 56; 60)
				mapred.job.shuffle.merge.percent	(8; 59; 2; 37; 36; 39; 56; 60)
				mapred.reduce.tasks.speculative.execution	(60)
				mapred.tasktracker.reduce.tasks.maximum	(90; 92; 33; 54; 58; 44; 91; 59; 94; 95; 2; 73; 35; 71; 60; 93; 46)
				mapred.tasktracker.tasks.maxmemory	(38)
Workload	Job Level	Map	CPU	mapred.map.tasks	(90; 92; 33; 4; 54; 44; 91; 59; 53; 75; 83; 78; 34; 102; 61; 73; 35; 71; 49; 26; 38; 32; 77)
				mapreduce.combine.class	(94; 37; 36; 39)
			IO	min.num.spills.for.combine	(37; 36; 39; 60)
		Reduce	CPU; IO	mapred.reduce.tasks	(90; 92; 33; 8; 4; 54; 97; 44; 91; 59; 53; 75; 45; 83; 78; 48; 34; 102; 61; 73; 35; 71; 37; 36; 39; 56; 60; 49; 26; 38; 77)

Foram identificados cerca de 29 parâmetros de configuração que, de acordo com os trabalhos analisados, impactam no desempenho do sistema. A Figura 4.1 apresenta um gráfico que indica o número de artigos por parâmetros. Mais detalhes são apresentados na Seção 4.3.

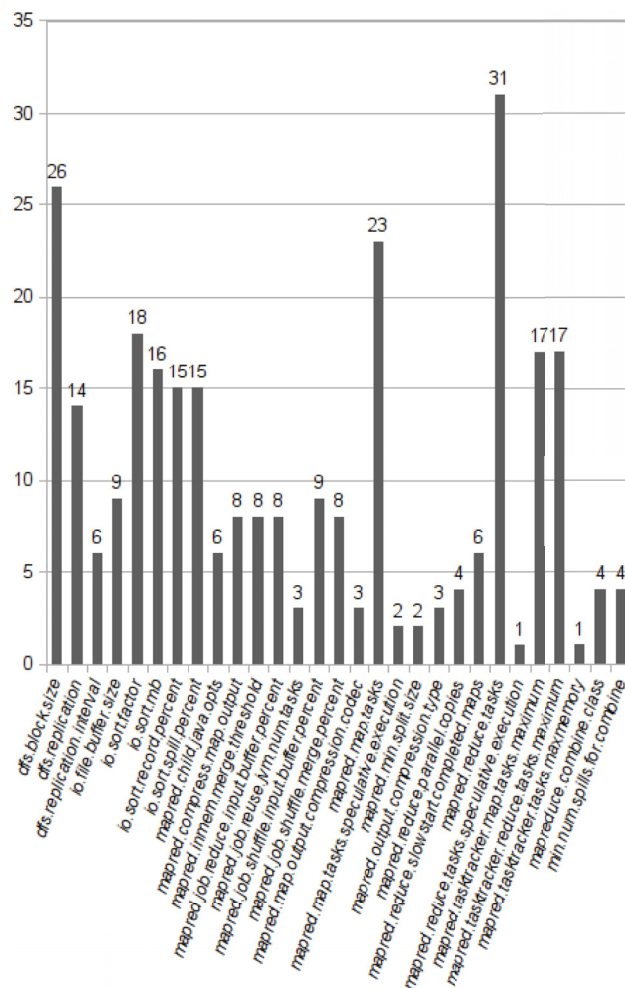


Figura 4.1: Número de artigos científicos por parâmetro.

4.2.2 Fases do Hadoop x Parâmetros

A execução de um *job* Hadoop MapReduce é dividida em uma fase *map* e uma fase *reduce*. Os dados de saída para cada tarefa *map* são escritos em um *buffer* de memória circular. Quando este *buffer* atinge um limite, o seu conteúdo é classificado por chave e liberados para um arquivo temporário. Esses arquivos são, então, enviados via HTTP para as máquinas que rodam as tarefas *reduce*. Estas tarefas são divididas em três sub-fases: *Shuffle*, *Sort* e *Reduce*. A sub-fase *Shuffle* copia os dados de saída a partir dos nós *map* para os nós *reduce*. A sub-fase *sort* ordena os dados intermediários por chave. Por fim, a sub-fase *reduce* executa as funções *reduce()* e o resultado final é escrito no sistema de arquivos distribuídos (82).

Muitas aplicações em execução no Hadoop mostram uma grande diferença de comportamento da aplicação nas fases *map* e *reduce*. Portanto, é muito importante saber quais os parâmetros que devem ser sintonizados de acordo com as fases de processamento de um aplicativo no Hadoop.

Na Tabela 4.3, na coluna *Fases*, classificamos os parâmetros que influenciam as fases do Hadoop: *Map*, *Reduce* e sub-fases intermediárias da fase *Reduce*. As sub-fases são classificadas como a fase *Merge/Shuffle*. Além disso, *Core Job* foi listada como uma fase que representa os parâmetros que controlam diretamente as funções essenciais do aplicação. O parâmetro *dfs.block.size* foi classificado pela fase *Number of maps* apenas para demonstrar que o valor deste parâmetro é o que define o número de tarefas *map*.

4.2.3 Características das Cargas de Trabalho x Parâmetros

A otimização do desempenho do Hadoop através das características das cargas de trabalho submetidas são importantes no sentido de se fazer pleno uso dos recursos do *cluster* (CPU, memória, IO e rede - ver Tabela 4.3, na coluna *Características da Carga de Trabalho*).

Entre outros exemplos, a sintonia de determinados parâmetros pode reduzir o custo de IO e tráfego rede, mas podem provocar uma sobrecarga da CPU. Isto acontece, por exemplo, se configurarmos o parâmetro *mapred.compress.map.output* para *true* (o padrão é *false*). Esta configuração irá diminuir o tamanho dos dados transferidos dos *Mappers* para os *Reducers*, porém, irá adicionar mais processamento no processo de compressão e descompressão dos dados (86). Além disso, alguns parâmetros estão correlacionados uns com os outros. A relação entre o parâmetro *io.sort.mb* (a quantidade em Mb de espaço de *buffer* utilizados para ordenar os fluxos de dados) e o parâmetro *mapred.child.java.opts* (opções de controle de Java para todas tarefas *map* e *reduce*) é um exemplo. Ou seja, o limite superior do primeiro é menor do que o tamanho do segundo. Entre outros exemplos, cita-se a configuração suficiente de *slots map* e *textitslots reduce* a fim de maximizar a utilização da CPU e a configuração do tamanho do *Java Heap* (espaço de memória utilizados pelos processos de JVM da tarefas *map* e *reduce*) para que uma ampla memória ainda esteja disponível para o kernel do sistema operacional, buffers, e subsistemas de cache (60).

4.2.4 Regras para Sintonia de Parâmetros

A Tabela 4.4 apresenta algumas regras importantes para a composição dos axiomas que devem ser implementados.

As regras são baseadas em cada parâmetro de configuração do Hadoop. Para cada um deles é apresentado seu valor padrão, o valor desejado e as características que devem ser analisadas para cada um deles, a fim de que sejam encontrados os melhores valores para os mesmos.

Pode-se dizer que o valor desejável vem do conhecimento adquirido da revisão sistemática realizada e aos experimentos que todos estes parâmetros foram submetidos, a fim de serem obtidos os melhores valores para eles.

As características apresentadas na tabela são tratadas como regras de grande importância para que sejam implementadas na ontologia, a fim de que sejam feitas as inferências necessárias para se obter a melhor sintonia dos parâmetros de configuração do Hadoop.

Tabela 4.4: Regras para composição de axiomas e propriedades da ontologia.

Parameter	Default Value	Desirable Value	Depends of
dfs.block.size	64Mb	128Mb a 256Mb	Cluster Config: Input Size
dfs.replication	3	1	Cluster Config
dfs.replication.interval	3	(empty)	Cluster Config: dfs.replication
io.file.buffer.size	4k	4k a 32k	Cluster Config
io.sort.factor	10	Until 100	Cluster Config: Number map output; Counter spilled files
io.sort.mb	100	Until 1Gb	Cluster Config: Number map output; Counter spilled files
io.sort.record.percent	0,05	io.sort.mb/4	Cluster Config: Number map output; Counter spilled files
io.sort.spill.percent	0,8	Until 1	Cluster Config: Number map output; Counter spilled files
mapred.child.java.opts	-Xmx200m	As large as possible	Cluster Config: Number processors CPU
mapred.compress.map.output	false	True	Cluster Config: CPU
mapred.inmem.merge.threshold	1000		Cluster Config: Memory of reduce function; If value = 0 Then mapred.job.shuffle.merge.percent commands
mapred.job.reduce.input.buffer.percent	0	1	Cluster Config: Memory of reduce function; If value = 1 Then mapred.inmem.merge.threshold = 0
mapred.job.reuse.jvm.num.tasks	1	-1	Workload CPU bound;
mapred.job.shuffle.input.buffer.percent	0,7	0,7	Cluster Config
mapred.job.shuffle.merge.percent	0,66	0,9	Cluster Config: Memory of reduce function; Percent of mapred.job.shuffle.input.buffer.percent; If mapred.inmem.merge.threshold = 0 Then commands
mapred.map.output.compression.codec	DEFLATE	LZOCCodec	Cluster Config: CPU
mapred.map.tasks	2	Input Data Size / dfs.block.size	dfs.block.size; number of maps >workers; Resources used by task tracker and data nodes; WorkLoad characteristic: CPU, memory, and/or IO bound; mapred.child.java.opts
mapred.map.tasks.speculative.execution	True	False	Cluster Configuration; Task is running slower than expected; On a busy cluster, Speculative Execution can reduce overall throughput
mapred.min.split.size	1	min.split=block.size(data locality)	File type; Jobs require that the input files not be splited
mapred.output.compression.type	RECORD	Block	Cluster Config: CPU; Random access to the output is not desired
mapred.reduce.parallel.copies	5	(empty)	Bandwidth
mapred.reduce.slowstart.completed.maps	0,05	(empty)	Workload characteristics; IO, CPU or Memory bound
mapred.reduce.tasks	1	0,9 Num reducers slots; NumSlots=NumNodes X NumberSlots per node	If workload = map only job Then mapred.reduce.tasks = zero
mapred.reduce.tasks.speculative.execution	True	False	Cluster Configuration; Task is running slower than expected; On a busy cluster, Speculative Execution can reduce overall throughput
mapred.tasktracker.map.tasks.maximum	2	number processors - 1	CPU Utilization of jobs
mapred.tasktracker.reduce.tasks.maximum	2	number processors - 1	CPU Utilization of jobs
mapred.tasktracker.tasks.maxmemory	-1	-1	Cluster Config: Memory
mapreduce.combine.class	null	Function	Limited to bandwidth of cluster; map output; WorkLoad Characteristics
min.num.spills.for.combine	3	3	If mapreduce.combiner.class is not null

4.3 ANÁLISE DOS RESULTADOS

Esta seção discute os resultados desta revisão sistemática com base nas respostas das três questões iniciais. Além disso, os resultados obtidos a partir da revisão identificaram os principais objetivos de cada um dos trabalhos analisados; bem como os métodos de experimentação e validação utilizados em cada estudo.

4.3.1 Considerações sobre as Questões

Respondendo a primeira pergunta, fica claro que cerca de 29 parâmetros de configuração do Hadoop são os que mais impactam no desempenho do sistema. A partir desses 29 parâmetros, 10 parâmetros foram abordados em mais de 65% dos artigos analisados. Observou-se que os parâmetros mais discutidos nestes trabalhos são *mapred.reduce.tasks* (o número sugerido de tarefas *reduce* de um *job*) com 31 artigos dos 40 pesquisados, *mapred.map.tasks* (o número sugerido de tarefas *map* para um *job*), com 23 artigos, *dfs.block.size* (o tamanho do bloco básico para o sistema de arquivos), com 26 artigos e *io.sort.factor* (o número de partições de saída *map* para *merge*) com 18 artigos. Estes parâmetros são aqueles que permitem que ao *framework* tentar fornecer dados localmente para a tarefa que processa o (*split* dos dados. Apesar de alguns parâmetros não terem sido amplamente explorados pela grande maioria dos artigos, alguns estudos mostraram a suas importâncias em relação ao desempenho.

Quanto a segunda pergunta, foram identificados os parâmetros que são afetados por cada fase do Hadoop. Isto também irá permitir o redirecionamento de sintonia dos parâmetros identificados em cada fase do Hadoop se isso for necessário. Esta abordagem seria bastante útil, por exemplo, caso se saiba as características da aplicação e em qual fase esta aplicação é mais intensiva (CPU, IO ou rede). Deste modo, a sintonia dos parâmetros seria dirigida por este conhecimento prévio.

Quanto a terceira questão, pode-se observar que os parâmetros são afetados de acordo com características cargas de trabalho. Assim, é possível, em trabalhos futuros, observar as características das cargas de trabalho e trabalhar com a sintonia dos parâmetros do Hadoop dirigida a eles.

4.3.2 Principal Finalidade e Abordagem de Experimentação

Os trabalhos relacionados nesta revisão sistemática tiveram, em comum, a exploração dos parâmetros de configuração e suas relações com o desempenho do Hadoop. No entanto, nota-se que os estudos têm diferentes finalidades centrais, bem como diferentes métodos de experimentação. Assim, para melhor entendimento, classificou-se os artigos de acordo com o objetivo principal e os métodos de experimentação aplicados.

Para a classificação do método de experimentação, foi utilizada a taxonomia proposta por Zelkowitz e Wallace (100). Nesta taxonomia há quatro abordagens para a experimentação: (1) método científico, (2) método da Engenharia, (3) método empírico e (4) método analítico.

A Tabela 4.5 mostra o objetivo principal de cada obra e os principais métodos de experimentação encontrados nos artigos analisados. É importante ressaltar que a identificação do método utilizado nos estudos analisados nem sempre foram tão claros de se identificar. Alguns estudos mostram uma interpretação dupla. Assim, mesmo com características duvidosas, em alguns casos, podemos classificar de acordo com o método predominante.

Observou-se que muitos estudos focaram entre a finalidade de se obter modelos de desempenho (cerca de 22,5%) e predição de desempenho (37,5%). Outros foram divididos em análise de desempenho e sintonia de desempenho, e apenas um trabalho para predição de

Tabela 4.5: Principal Finalidade e Abordagem de Experimentação

Finalidade	Abordagem	Artigos Científicos
Predição de Energia	Analítico	(58)
Análise de Desempenho	Empírico	(48; 49; 54; 53; 72; 91)
Modelo de Desempenho	Empírico	(2; 46; 102)
	Analítico	(26; 34; 59; 71; 83; 97)
Penalidades de Desempenho	Empírico	(44)
Predição de Desempenho	Empírico	(4; 33; 38; 36; 39; 45; 56; 90; 92; 94)
	Analítico	(37; 55; 77; 78; 95)
Sintonia de Desempenho	Empírico	(8; 35; 60; 61; 73; 75; 93)
	Analítico	(32)

energia e penalidades de desempenho. Além disso, observou-se que 67,5% dos artigos analisados utilizaram a abordagem empírica como método de experimentação, e 32,5% usaram o método analítico.

A maioria dos estudos adotaram uma abordagem empírica em métodos experimentais. Isto significa que uma hipótese é proposta e esta é validada por um método de análise estatística. Em outras palavras, os dados foram recolhidos e, depois, foi estatisticamente verificada a hipótese. Por outro lado, alguns estudos têm utilizado o método analítico. Estes trabalhos desenvolveram uma teoria formal para um problema, e os resultados derivados dessa teoria foram comparados com observações empíricas.

Para se tentar obter uma perfilização dinâmica, antes mesmo de iniciar a execução da tarefa, deve-se coletar quaisquer informações que sejam necessárias para otimização dos parâmetros de configuração do *job*. Um exemplo seria coletar informações de uma amostra dos dados de entrada de um TeraSort, a fim de se descobrir como está a distribuição dos mesmos. As saídas das tarefas *map* seriam particionadas igualmente para os *reduces* e isto permitiria balancear o trabalho igualmente através das tarefas *reduce*, assumindo-se a alocação de recursos homogêneos.

No entanto, a ideia básica da perfilização dinâmica baseada em amostragem dos dados, proposta por (8), possui alguns desafios, tais como:

- Uma grande quantidade de programas MapReduce não tem as mesmas características que o programa TeraSort, no que diz respeito a distribuição das chaves de entrada e as saídas serem as mesmas;
- A maioria das implementações prévias por tipo de amostra e divisões probabilísticas podem ser montadas randomicamente. Entretanto, dado um programa arbitrário MapReduce, existem restrições sobre como os dados de entrada podem ser acessados. Usualmente, os dados somente podem ser acessados de uma maneira sequencial do primeiro ao último (*splits* em Hadoop).

Um dos trabalhos e projetos mais completos na literatura atual, e que trata do problema de predição de recursos para *jobs* Hadoop MapReduce, bem como provê todo um ambiente de

execução para o Hadoop MapReduce, é o Starfish (39), que é um sistema de auto configuração para análises de grande volume de dados através do Hadoop MapReduce. Starfish foi contruído sobre Hadoop, de forma que o usuário possa adaptar suas necessidades e sistemas de cargas de trabalho a fim de fornecer, automaticamente, um bom desempenho, sem qualquer necessidade dos usuários entenderem e/ou manipularem os diversos parâmetros de configuração do Hadoop. Um sistema que permite ao usuário este entendimento automático se faz bastante necessário para o Hadoop. O uso da maioria dos mecanismos do Hadoop tem que ser gerenciados manualmente. Um dos exemplos disto, diz respeito a elasticidade. O Hadoop suporta adição dinâmica de nós, tanto quanto a desativação dos nós que sofrem alguma falha ou que estejam excedentes. No entanto, estes mecanismos não fazem o Hadoop ser elástico automaticamente devido a falta de módulos que decidem quando há a necessidade de adicionar ou desativar nós, e quando e como rebalancear os dados neste processo (39).

4.3.3 Métodos de Validação

Uma análise adicional foi alcançada através do levantamento das informações adquiridas através da revisão sistemática. Diz respeito aos métodos de validação utilizados pelos trabalhos pesquisados. Os métodos de validação foram analisados da mesma forma como os métodos de experimentação, e foram classificados de acordo com a taxonomia para a experimentação da engenharia de software (101).

Os estudos podem ser classificados em três dos doze métodos considerados na taxonomia. Vale ressaltar que, mais uma vez, aqui, foram encontrados artigos com interpretações dúbias. Assim, os métodos de validação com características predominantes foram classificados. A tabela 4.6 apresenta a classificação do métodos de validação encontrados e as respectivas referências aos artigos científicos que os utiliza em seus trabalhos.

Tabela 4.6: Métodos de Validação

Tipo de Validação	Método de Validação	Artigos Científicos
Controlado	Análise Dinâmica (Benchmark)	(2; 4; 8; 32; 35; 37; 38; 36; 39; 34; 45; 46; 48; 49; 54; 53; 55; 56; 58; 60; 72; 73; 75; 77; 78; 83; 93; 97; 94; 95; 102)
	Simulação	(33; 44; 61; 71; 90; 92; 91)
Formal	Análise Teórica	(26; 59)

A figura 4.2 mostra o número de artigos científicos encontrados classificados por métodos de validação.

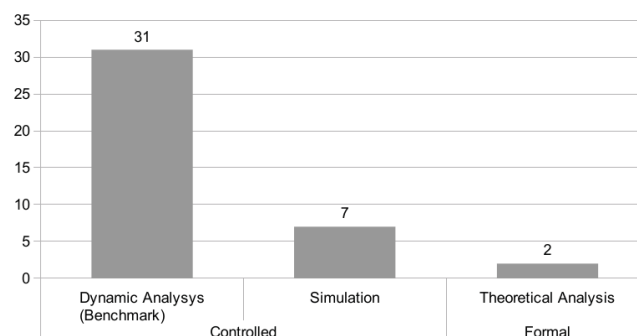


Figura 4.2: Número de artigos científicos por método de validação.

É possível observar na figura 4.2 que a maioria dos trabalhos utilizaram a análise dinâmica (benchmark) e a simulação para validar suas pesquisas. Como se pode ver, o método de análise dinâmica controlada é o método mais utilizado nos projetos de pesquisa e o método de análise teórica formal é o menos utilizado.

4.3.4 Classificação dos *Jobs*

No trabalho de (18) é apresentada uma metodologia para se compreender as características das cargas de trabalho do MapReduce usando os *traces* de execução de *jobs* de produção do Facebook e Yahoo. Foi desenvolvido um modelo de carga de trabalho empírico na qual foi utilizado para gerar e produzir cargas de trabalho sintéticas. Assim, foi demonstrado o uso da metodologia para responder questões, entre outras, sobre a intensidade dos dados e a classificação das cargas de trabalho.

Uma grande contribuição do trabalho de (18) e (17) foi descrever e classificar as cargas de trabalho quanto as características dos dados, tempo de execução e padrão. Como já foi apresentado anteriormente, os padrões de processamento do Hadoop MapReduce operam em pares chave-valor nas fases *input*, *shuffle* (intermediária) e *output*. Sendo assim, a classificação dos padrões de cargas de trabalho utilizou como referência os tamanhos de dados em cada uma destas fases, além do tempo de execução das tarefas *map* e *reduce*. Para tanto, foi aplicado o algoritmo *k-means* para agrupamento das informações.

A classificação abaixo segue as características extraídas do trabalho de (18) e é de suma importância para se obter os padrões das cargas de trabalho.

- Small jobs: Pequenos valores para todas as dimensões;
- Load data, fast: Input « output, sem shuffle, curta duração;
- Load data, slow: Input « output, sem shuffle, longa duração;
- Load data, large: Pequeno input, sem shuffle, output muito grande;
- Load data, huge: Pequeno input, sem shuffle, output enorme;
- Aggregate, fast: Input » shuffle » output, curta duração;
- Aggregate and expand: Input » shuffle « output;
- Expand and aggregate: Input « shuffle » output;
- Data transform: Input ? shuffle ? output;
- Data summary: Input enorme, Output muito pequeno;
- Data summary, large: Input ainda maior, Output muito pequeno;
- Transform and expand: Input ? shuffle « output.

4.4 SIMULADOR MRSIM

MRSim (33) é um simulador para o *framework* Hadoop Mapreduce. Ele simula o comportamento de aplicações no ambiente Hadoop a fim de se encontrar os melhores valores dos parâmetros para se obter melhor desempenho através da sintonia dos parâmetros de configuração do Hadoop. Além disso, o MRSim simula o ambiente do Hadoop.

A avaliação do simulador conclui que ele atinge um alto nível de precisão dos resultados se comparados a um ambiente de execução real.

O usuário do simulador entra com dois tipos de arquivos baseados no formato *Json*: o primeiro com a topologia do *cluster*, que consiste de um grupo de máquinas conectadas a um roteador; o segundo arquivo de entrada contém a especificação do *job* e os parâmetros e valores que serão executados.

O arquivo de configuração do *job* contém o número de tarefas *map* e *reduce*, o layout dos dados, a descrição do algoritmo e os parâmetros de configuração do *job*. O layout dos dados oferece a localização e as replicações dos *splits* dos dados nos nós simulados. O algoritmo fornece a informação sobre a aplicação, tal como o número de instruções por registro e média do tamanho do registro nas tarefas Mapreduce. Outras especificações fornecem o controle do tamanho do *buffer* de memória e limites para as tarefas *map* e *reduce*, além de outros parâmetros de configuração.

MRSim é capaz de simular diferentes tipos de aplicações, com a capacidade de se estudar a precisão dos efeitos de vários parâmetros de configuração da aplicação em relação ao desempenho. Desta forma, o usuário é capaz de estimar as melhores configurações para se obter o melhor desempenho possível para certas aplicações. Sendo assim, o simulador captura os efeitos de diferentes configurações dos parâmetros em termos de tempo de execução das aplicações e utilização dos recursos do *cluster*.

4.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou uma revisão sistemática sobre a relação entre os parâmetros de configuração do Hadoop e o desempenho do sistema. O objetivo foi um estudo para analisar as pesquisas mais recentes dentro deste contexto e responder às seguintes perguntas: (1) Quais parâmetros de configuração do Hadoop têm influências e impactos no desempenho do sistema? (2) Quais parâmetros são influenciados pelas fases do Hadoop? (3) Quais parâmetros são influenciados pelas características das cargas de trabalho?

Os resultados alcançados foram capazes de responder a essas questões e fornecer referências para trabalhos futuros voltados a este assunto, bem como apresentar as principais regras para a melhor sintonia de cada um dos parâmetros identificados. Além disso, a análise dos resultados identificou o principal objetivo e os métodos de experimentação e validação abordados pelos artigos examinados. Mais uma vez, com grande importância para a referência em trabalhos futuros.

Como complemento ao estudo, foi apresentado um estudo sobre a classificação dos padrões das cargas de trabalho e um trabalho sobre um simulador do ambiente Hadoop MapReduce chamado MRSim.

Por fim, esta revisão sistemática tem contribuição fundamental para a elaboração de uma base de conhecimento no que diz respeito a sintonia de parâmetros de configuração do Hadoop que influenciam no desempenho.

5 ONTOHADOOP

Este capítulo apresenta OntoHadoop: uma ontologia para representar o conhecimento sobre a sintonia dos parâmetros de configuração do Hadoop, com o objetivo de melhorar o desempenho das aplicações submetidas a ele. O foco é a representação do conhecimento semântico necessário para se obter o melhor ajuste de parâmetros de configuração do Hadoop que têm influência sobre o desempenho das aplicações. O conhecimento semântico explora as características das cargas de trabalho e do *cluster*, bem como a taxonomia dos conceitos, propriedades, relações e axiomas da ontologia.

As próximas seções apresentam as fases de atividades efetuadas de acordo com o ciclo de vida proposto pela metodologia para construção de ontologias *Methontology*, já descrita no Capítulo 3.

5.1 ESPECIFICAÇÃO

As declarações T-Box da ontologia descrevem um conjunto de conceitos e propriedades para esses conceitos, obtidos a partir da revisão sistemática apresentada no Capítulo 4 que apresenta um estudo que avaliou 40 trabalhos que abordam especificamente os parâmetros de configuração que afetam o desempenho do Hadoop. A base conceitual tem como referência a tabela 4.3 no referido capítulo.

As informações necessárias para a sua especificação da ontologia, segundo a metodologia *Methontology*, seguem abaixo:

1. Controle de versão:

Versão	Data	Autor	Descrição
1	08/2014	Ailton Sergio Bonifacio	Criação

2. Identificação:

Domínio: Parâmetros de Configuração do *framework* Hadoop MapReduce.

Conceitualizado por: Ailton Sergio Bonifacio

3. Propósito:

Propor uma ontologia para classificação e sintonia dos parâmetros de configuração do Hadoop MapReduce que influenciam no desempenho das aplicações.

4. Escopo:

- Definir os conceitos básicos para compor a classificação dos parâmetros de configuração quanto a categoria, nível, fase de execução e características das cargas de trabalho;
- Definir os conceitos dos padrões das cargas de trabalho, estabelecidos através das suas características;
- Definir as classes, subclasses e instâncias dos conceitos;

- Definir as propriedades das classes, bem como as relações e restrições entre classes; e relações e restrições entre classes e objetos de classes.

5. Nível de formalidade: Formal.

6. Fonte de conhecimento: Revisão Sistemática de Literatura sobre a sintonia dos parâmetros de configuração do Hadoop MapReduce.

5.2 AQUISIÇÃO DO CONHECIMENTO

Nesta tese, propõe-se organizar todo conhecimento adquirido em uma ontologia para o Hadoop: a *OntoHadoop*. Esta ontologia foi desenvolvida através da criação de uma taxonomia de conceitos que descreve as relações entre as cargas de trabalho, fases do Hadoop, parâmetros de configuração e um conjunto de axiomas com regras para as inferências, que podem determinar as melhores sintonias para os parâmetros.

Na coluna *Fases*, da Tabela 4.3, classificamos os parâmetros que influenciam as fases do Hadoop: *Map*, *Reduce* e sub-fases intermediárias da fase *Reduce*. As sub-fases provêm das fases *Merge/Shuffle*. Além disso, *Core Job* foi listada como uma fase que representa os parâmetros que controlam diretamente as funções essenciais da aplicação. O parâmetro *dfs.block.size* foi classificado pela fase *Number of maps* apenas para demonstrar que o valor deste parâmetro é o que define o número de tarefas *map*.

O conhecimento gerado pela inserção de conteúdo é classificado e organizado de acordo com as características das cargas de trabalho. A caracterização das cargas de trabalho baseia-se em (18). Este trabalho apresenta uma metodologia para a compreensão do desempenho do Hadoop MapReduce usando *logs* do Facebook¹ e Yahoo². Essa pesquisa tenta responder questões *what-if* que estão relacionadas com o tamanho do sistema, o tamanho dos dados de entrada e a configuração de hardware/software.

A caracterização das cargas de trabalho de acordo com a classificação proposta por (18), juntamente com a caracterização delas através do uso da CPU, memória, IO e rede, que é feita através da análise dos dados de *logs* de execução, são de grande importância para a sintonia dos parâmetros. Esta caracterização permite inferir e identificar, através dos axiomas estabelecidos, os parâmetros que estão relacionados com as características daquele tipo de carga de trabalho.

5.3 CONCEITUALIZAÇÃO

Através da aquisição prévia do conhecimento, verificou-se os principais conceitos e propriedades da ontologia. Estes conceitos representam a taxonomia para os parâmetros de configuração do Hadoop MapReduce, bem como as propriedades inerentes a cada um deles.

A conceitualização da ontologia descreve o problema e as soluções em termos do domínio do vocabulário especificado. Sendo assim, foi utilizado um mapa conceitual (67) para representar este conjunto de conceitos. Dessa maneira, torna-se mais clara a identificação dos conceitos e as relações entre os conceitos do domínio de conhecimento modelado.

A Figura 5.1 apresenta o mapa conceitual que descreve o domínio do problema e as relações dos conceitos.

Pode-se observar que a hierarquia dos conceitos segue a classificação dos parâmetros identificada no 4, na seção 4.2.1. Os parâmetros são representados pelo conceito *Parameters* e

¹<https://www.facebook.com/>

²<https://www.yahoo.com/>

classificados de acordo com a categoria a que ele pertence, o nível, a fase de execução do Hadoop em que este parâmetro é mais significativo e as características da carga de trabalho que eles são mais intensivos.

Assim, a identificação dos parâmetros que são afetados por cada fase Hadoop permite identificar os parâmetros que devem ser sintonizados em cada fase crítica da aplicação submetida ao Hadoop. Esta abordagem torna-se bastante útil quando se conhece as características da aplicação, e em que fase ela é intensa e se a sua característica principal é ser intensiva em CPU, IO ou rede, por exemplo. Deste modo, a sintonia dos parâmetros deverá ser dirigida por este conhecimento prévio.

Além disso, verifica-se, no mapa conceitual, que a ontologia poderá conter os dados extraídos dos *logs* de execução do Hadoop. Esta classificação é representada através do conceito *JobData*. Este conceito contém os dados que dizem respeito ao tempo de execução do *job* e o tamanho dos dados de entrada, saída e intermediários. Outra classificação importante, contextualizada através do mapa conceitual, é a representação dos padrões das cargas de trabalho de acordo com a classificação proposta por (18). O conceito *WorkLoadPattern* representa estes padrões.

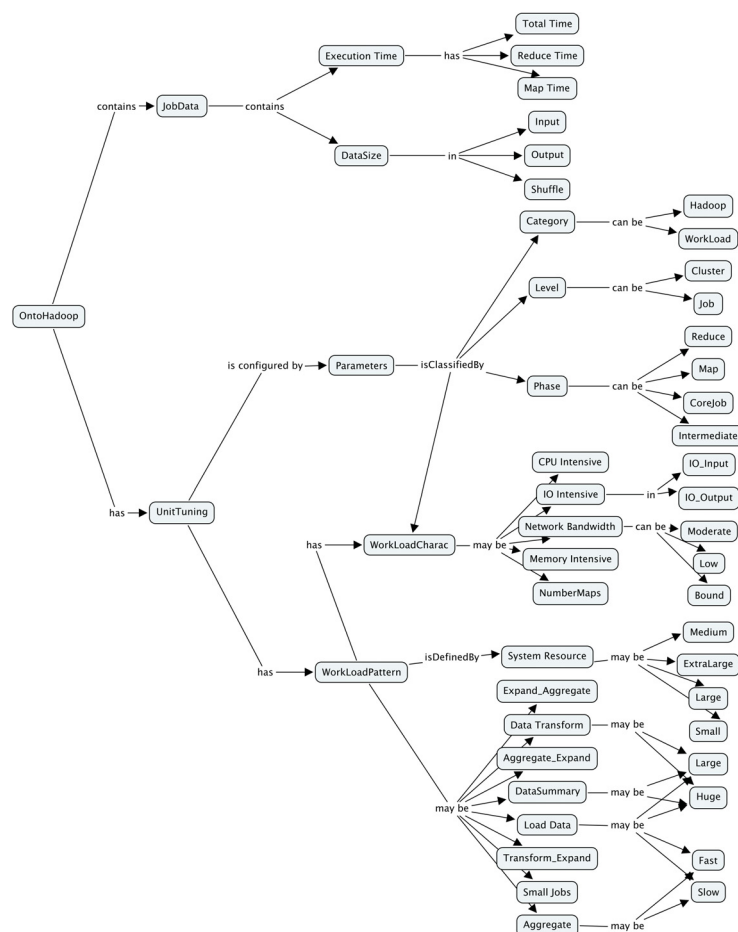


Figura 5.1: Mapa Conceitual para representação dos conceitos sobre os parâmetros de configuração do Hadoop MapReduce.

Por fim, é modelado o conceito *UnitTuning* para representar as Unidades de Sintonia para os parâmetros de configuração. Este conceito possui a relação *isConfiguredBy* que relaciona

uma unidade de sintonia a um ou vários parâmetros de configuração; e a relação *has* para denotar que uma unidade de sintonia possui, associada a ela, um padrão de carga de trabalho (*WorkLoadPattern*).

Nesta fase de conceitualização, todos os conceitos e suas propriedades devem ser definidas. Sendo assim, a tabela 5.1 apresenta os conceitos e propriedades para a sintonia de parâmetros de configuração do Hadoop. A coluna *Conceitos* apresenta cada um dos conceitos que devem ser formalizados. A coluna *Descrição dos Conceitos* faz uma breve explanação sobre o significado do conceito apresentado. Através das coluna *Propriedades dos Conceitos* são apresentadas as propriedades particulares de cada um dos conceitos. Por fim, a coluna *Tipo da Propriedade* classifica as propriedades por dois tipos: propriedades do objeto e propriedades dos dados. As propriedades do objeto denotam propriedades que relacionam um conceito a um ou mais outros conceitos. As propriedades de dados são as relações entre os conceitos e os indivíduos daquele conceito.

Tabela 5.1: Principais conceitos e propriedades dos conceitos

Conceitos	Descrição	Propriedades dos Conceitos	Tipo da Propriedade	
			Objeto	Dados
ParameterCategory	Conceito para classificar os parâmetros por Categoria	nameCategory		x
ParameterLevel	Conceito para classificar os parâmetros por Nível	nameLevel		x
ParameterPhase	Conceito para classificar os parâmetros por Fase de execução do Hadoop	namePhase		x
WorkLoadCharacteristic	Conceito para classificar os parâmetros de acordo com as influências das características da carga de trabalho	typeWorkLoadCharacteristic		x
Parameters	Conceito para modelar os parâmetros que têm influência no desempenho	nameParameter		x
		descriptionParameter		x
		valueParameter		x
		hasParameterCategory	x	
		hasParameterLevel	x	
WorkLoadPattern	Conceito para representar as cargas de trabalho	hasParameterPhase	x	
		hasWorkLoadCharac	x	
		hasSpeedJob	x	
		hasSizeInput	x	
		hasSystemResource	x	
SpeedJob	Conceito para classificar a velocidade do Job (Fast ou Slow)	nameSpeed		x
SizeInput	Conceito para classificar o tamanho do input do Job (Huge, Large e Small)	nameSize		x
		sizeInputMinKb		x
		sizeInputMaxKb		x
JobData	Conceito para classificar os dados de execução do Job	sizeInputJobKb		x
		sizeOutputJobkb		x
		sizeShuffleJobkb		x
		durationJobSec		x
		maptimeJobSec		x
		reducetimeJobSec		x
SystemResource	Conceito para classificar os recursos de cluster	nameInstance		x
		numberCPU		x
		memGB		x
		storageGB		x
UnitTuning	Conceito para representar as Unidades de Sintonia	hasParameter (n)	x	
		hasWorkloadPattern (1)	x	

Estes conceitos irão compor a ontologia e formarão a base de conhecimento necessária para que seja utilizada no desenvolvimento da abordagem semântica pretendida.

A próxima seção apresenta a formalização desta conceitualização.

5.4 FORMALIZAÇÃO

Através da conceitualização do domínio de problema, foi criado um modelo formal do mapa conceitual. Este modelo formal é representado por um diagrama de classes. Assim, o diagrama de classes permite o direcionamento da implementação da ontologia.

A *OntoHadoop* completa foi modelada e apresentada sob a forma de diagrama de classes, como mostrado pela Figura 5.2. Através do diagrama de classes modelado, pode-se observar a classificação das classes e das relações entre elas (axiomas de cobertura).

A classe *ParameterCategory*, *ParameterLevel*, *ParameterPhase* contém, pela ordem, as categorias, níveis e fases em que os parâmetros são classificados. Cada parâmetro é relacionado a uma instância de cada uma dessas classes.

Além disso, foi especificada a classe *WorkLoadCharacteristic* para relacionar cada parâmetro à sua principal influência pela característica da carga de trabalho. Por exemplo, uma carga de trabalho que é intensiva no uso de CPU deve ser relacionada a um parâmetro que trate da sintonia em relação a esta característica específica.

Todos os 29 parâmetros foram modelados como classes especializadas da classe *Parameters*. Isto é importante porque cada parâmetro pode assumir múltiplas instâncias de acordo com as características e os padrões de cada Unidade de Sintonia.

Fazendo uma leitura do diagrama de classes, fica claramente identificadas as relações dos parâmetros com as classes supracitadas. Por exemplo, um parâmetro possui uma relação *hasParameterCategory* com a classe que identifica as categorias de parâmetros; uma relação *hasParameterLevel* com classe que identifica o nível do parâmetro; uma relação *hasParameterPhase* que identifica a fase crítica daquele parâmetro; e, por fim, uma relação *hasWorkLoadCharac* para identificar em qual característica aquele parâmetro é mais intensivo. Vale a pena ressaltar, e observa-se isso através das cardinalidades modeladas, que um parâmetro só pode pertencer a uma única categoria, nível e fase. No entanto, eventualmente, este parâmetro pode se caracterizar por uma ou mais características de carga de trabalho.

A classe *WorkLoadPattern* contempla os padrões de carga de trabalho classificados pelo trabalho de (18). Estes padrões possuem características específicas sobre os dados de entrada, intermediários e de saída das cargas de trabalho submetidas ao Hadoop MapReduce. Sendo assim, os padrões indicam os parâmetros que teriam relação com elas, sendo associados as características da classe *WorkLoadCharacteristic* (IO, CPU, Memória, etc).

Uma Unidade de Sintonia é composta por n instâncias de parâmetros através da propriedade de objeto *hasParameter* e um Padrão de Carga de Trabalho, através da propriedade de objeto *hasWorkLoadPattern*. Cada parâmetro de uma Unidade de Sintonia possui valores que indicam a melhor sintonia para o padrão de carga de trabalho daquela Unidade de Sintonia.

A classe *Parameters* deve contemplar todos os parâmetros de configuração do Hadoop MapReduce que têm influência no desempenho das aplicações e que foram encontrados através da revisão sistemática apresentada no Capítulo 4 na Seção 4.2.1.

5.5 AXIOMAS E RELAÇÕES

O Capítulo 4 proporcionou uma enorme gama de conhecimentos fundamentais para a descrição das regras e axiomas da ontologia.

A tabela 5.2 apresenta as principais relações que devem ser implementados na ontologia.

Tabela 5.2: Principais Relações da ontologia.

Domain	Range	Functional	Transitive	Inverse Functional	Properties	Inverse Relation
Parameters	ParameterCategory	X			hasParameterCategory	isParameterCategoryOf
	ParameterLevel	X			hasParameterLevel	isParameterLevelOf
	ParameterPhase	X			hasParameterPhase	isParameterPhaseOf
	WorkLoadCharac				hasWorkLoadCharac	isWorkLoadCharacOf
WorkLoadPattern	SpeedJob		X		hasSpeedJob	isSpeedJobOf
	SizeInput		X		hasSizeInput	isSizeInputOf
	SystemResource		X		hasSystemResource	isSystemResourceOf
	WorkLoadCharac				hasworkLoadCharac	isworkLoadCharacOf
UnitTuning	Parameters			X	hasParameter	isParameterOf
	WorkLoadPattern			X	hasWorkLoadPattern	isWorkLoadPatternOf

A classe *Parameters* é uma generalização das classes especializadas. As classes especializadas são representadas pelos 29 parâmetros de configuração responsáveis pelo desempenho do Hadoop MapReduce como sub-classes. Isto se faz necessário porque cada subclasse representará os indivíduos apenas daquela subclasse específica.

Por exemplo, a subclasse *dfs_block_size*, da classe *Parameters*, conterá instâncias com os vários valores que este parâmetro pode assumir. Outro motivo para se representar cada parâmetro como subclasse é que cada um dos parâmetros podem possuir n indivíduos com valores distintos.

Uma Unidade de Sintonia poderá assumir apenas alguns parâmetros na sua sintonia e não obrigatoriamente todos. Assim, uma Unidade de Sintonia pode utilizar uma instância da subclasse *dfs_block_size* e uma outra Unidade de Sintonia assumir outra instância da mesma subclasse.

Além disso, a classe *Parameters* é composta por indivíduos das classes *ParameterCategory*, *ParameterLevel*, *ParameterPhase* e *WorkLoadCharac*. Cada uma destas classes representam, respectivamente, a categoria que cada parâmetro está classificado, o nível, a fase de execução do Hadoop MapReduce que ele é mais importante e influente, e as características da carga de trabalho que são mais influentes no parâmetro.

A classe *WorkLoadPattern* é composta por indivíduos das classes *SpeedJob*, *SizeInput*, *SystemResource* e *WorkLoadCharac*. Um padrão de carga de trabalho é, portanto, composta por características tais como, velocidade do *Job*, tamanho do dados de entrada, recursos utilizados pela aplicação e características da carga de trabalho, tais como CPU, IO, memória e rede.

Através destas relações e composições, uma Unidade de Sintonia é composta por um padrão de carga de trabalho (representado pelas instâncias da classe *WorkLoadPattern* e um ou vários parâmetros de configuração (representados pelas instâncias das subclasses da classe *Parameters*) que são relacionados ao padrão padrão de carga de trabalho em questão.

5.6 IMPLEMENTAÇÃO

Esta etapa apresenta a construção e implementação da ontologia OntoHadoop. Para isso, foi utilizada a linguagem OWL (Mcguinness e van Harmelen; 89) e o editor de ontologias Protégé versão 4.3 (81).

A descrição completa da ontologia OntoHadoop, especificada em OWL, pode ser acessada em: <http://www.inf.ufpr.br/ailtons/OntoHadoop.owl>.

5.6.1 Implementação das Propriedades de Objetos

As propriedades de objetos (*Object Properties*) são aquelas que relacionam indivíduos a indivíduos, ou seja, definem propriedades com a restrição de possuir valores que devem ser indivíduos. Podem, também, ser chamadas de propriedades de classes. A figura 5.3 apresenta a definição da propriedade *hasWorkloadPattern* na ferramenta Protégé.

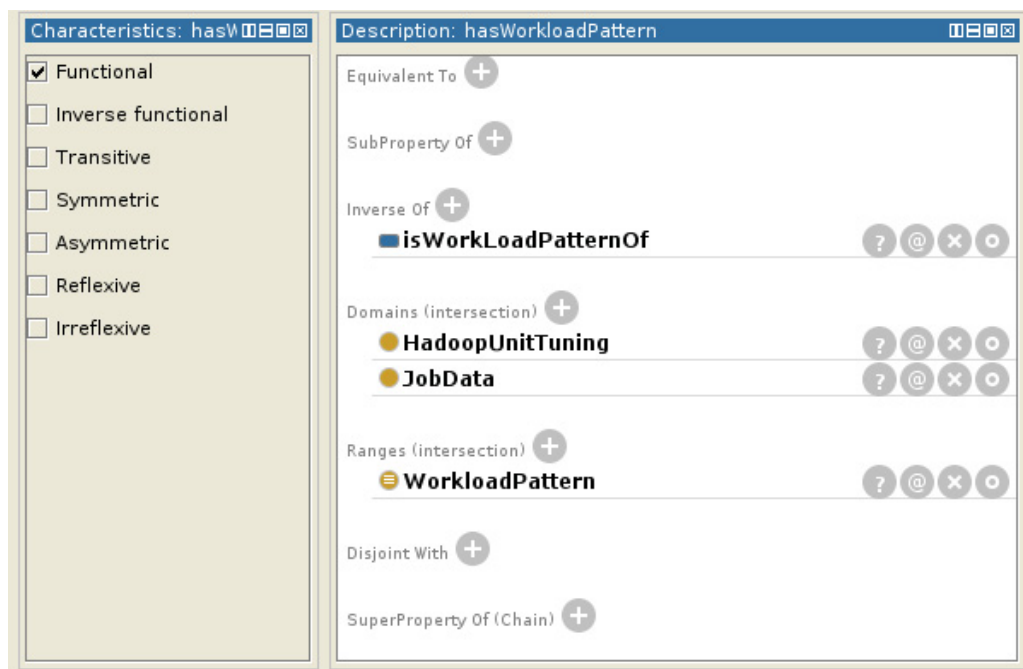


Figura 5.3: Definição da propriedade de objeto *hasWorkloadPattern* no Protégé.

A definição desta mesma propriedade definida em OWL é apresentada abaixo:

```

1 <owl:ObjectProperty rdf:about="http://www.owl-ontologies.com/
2   OntoHadoop.owl#hasWorkloadPattern">
3   <rdf:type rdf:resource="FunctionalProperty"/>
4   <rdfs:domain rdf:resource="&OntoHadoop;JobData"/>
5   <rdfs:domain rdf:resource="http://www.owl-ontologies.com/
6     OntoHadoop.owl#HadoopUnitTuning"/>
7   <rdfs:range rdf:resource="http://www.owl-ontologies.com/
8     OntoHadoop.owl#WorkloadPattern"/>
9 </owl:ObjectProperty>

```

Primeiramente, a propriedade é declarada como sendo do tipo *owl:ObjectProperty*. Em seguida, observa-se que a propriedade *hasWorkLoadPattern* possui uma restrição e é definida

como sendo do tipo "funcional"(<rdf:type rdf:resource="FunctionalProperty"/>), ou seja, só poderá assumir um único valor para cada relação de indivíduo para outro indivíduo.

Além disso, ela é declarada como pertencente ao dois domínios (classes): *JobData* e *HadoopUnitTuning*. Isso significa que esta propriedade poderá ser utilizada por estas duas classes. Finalmente é declarada o alcance (*range*) que esta propriedade pode assumir. No caso da propriedade *hasWorkloadPattern*, o *range* são as instâncias da classe *WorkloadPattern*.

5.6.2 Implementação das Propriedades de Dados

As propriedades de dados (*Data Properties*) são aquelas que relacionam indivíduos a valores de dados, ou seja, definem propriedades que relaciona um indivíduo a um tipo de dado. A figura 5.4 apresenta a definição da propriedade *hasWorkloadPattern* na ferramenta Protégé.



Figura 5.4: Definição da propriedade de dados *sizeInputJobKb* no Protégé.

A definição da propriedade de dados *sizeInputJobKb* definida em OWL é apresentada abaixo:

```

1 <owl:DatatypeProperty rdf:about="&OntoHadoop;sizeInputJobKb">
2   <rdf:type rdf:resource="&owl;FunctionalProperty"/>
3   <rdfs:domain rdf:resource="&OntoHadoop;JobData"/>
4   <rdfs:range rdf:resource="&xsd;double"/>
5 </owl:DatatypeProperty>

```

Essa propriedade é declarada como sendo do tipo *owl:DatatypeProperty*. Assim como o exemplo apresentado na subseção anterior, para a propriedade de objetos *hasWorkloadPattern*, a propriedade *sizeInputJobKb* também é definida como sendo do tipo "funcional". Além disso, também são definidos os domínios (*domain*) e alcance (*range*) que a propriedade pode assumir. Neste exemplo, o domínio onde será utilizada esta propriedade é a classe *JobData* e o *range* do tipo *double*.

Vale ressaltar que o *range* pode ser, inclusive, instâncias de outras classes definidas na ontologia.

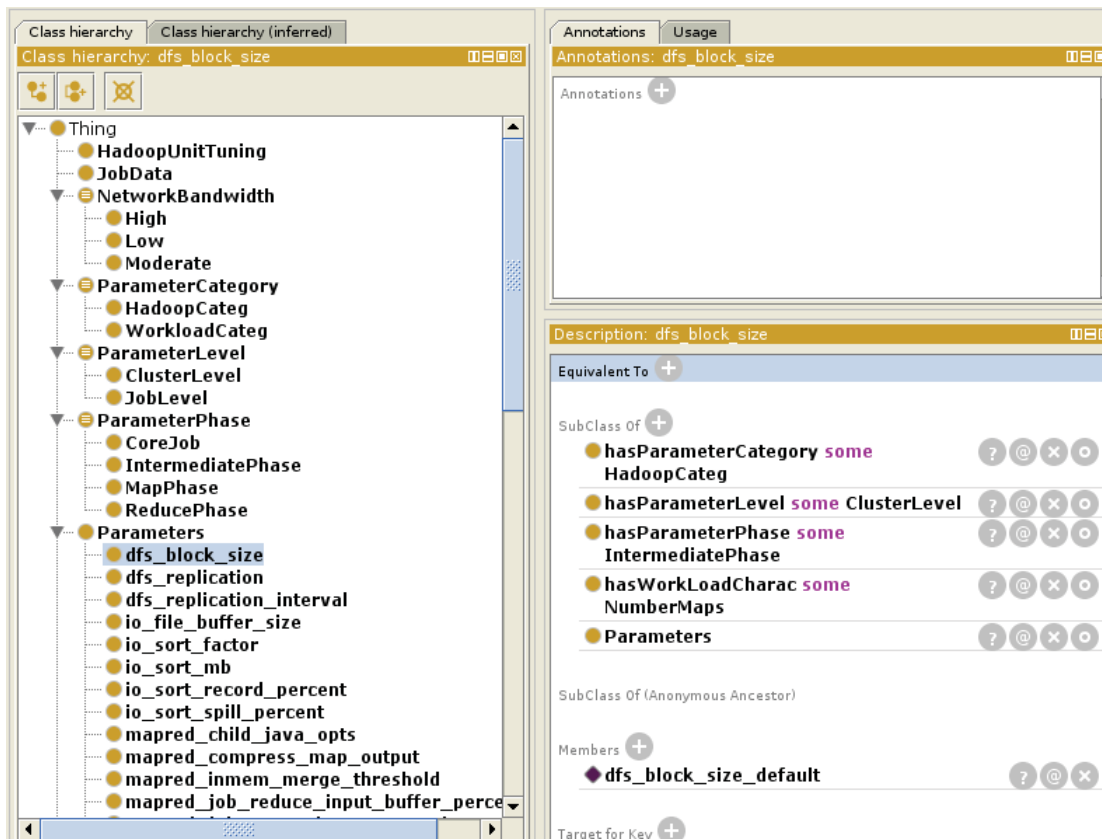


Figura 5.5: Classe *dfs_block_size* e Hierarquia parcial das classes no Protégé.

5.6.3 Implementação das Classes

A hierarquia de classes da ontologia pode ser vista parcialmente através da figura 5.5.

Abaixo é apresentado um trecho parcial, em OWL, para a implementação da classe *dfs_block_size* como uma subclasse da classe *Parameters*:

```

1  <owl:Class rdf:about="http://www.owl-ontologies.com/OnToHadoop.owl#
   dfs_block_size">
2  <rdfs:subClassOf rdf:resource="http://www.owl-ontologies.com/
   OnToHadoop.owl#Parameters"/>
3  <rdfs:subClassOf>
4  <owl:Restriction>
5  <owl:onProperty rdf:resource="&OnToHadoop;hasParameterCategory"/>
6  <owl:someValuesFrom rdf:resource="http://www.owl-ontologies.com/
   OnToHadoop.owl
7  #HadoopCateg"/>
8  </owl:Restriction>
9  </rdfs:subClassOf>
10 <rdfs:subClassOf>
11 <owl:Restriction>
12 <owl:onProperty rdf:resource="&OnToHadoop;hasParameterPhase"/>
13 <owl:someValuesFrom rdf:resource="http://www.owl-ontologies.com/
   OnToHadoop.owl
14 #IntermediatePhase"/>
15 </owl:Restriction>

```

```

16     </rdfs:subClassOf>
17     <rdfs:subClassOf>
18         <owl:Restriction>
19             <owl:onProperty rdf:resource="&OntoHadoop;valueParameter"/>
20             <owl:qualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger"
21                 ">1
22                                     </owl:qualifiedCardinality>
23             <owl:onDataRange rdf:resource="&xsd;string"/>
24         </owl:Restriction>
25     </rdfs:subClassOf>
</owl:Class>

```

Observa-se, no código OWL que implementa a classe *dfs_block_size* que todas as restrições em relação a classe são codificadas. Um exemplo é restrição `<rdfs:subClassOf>` que define esta classe como subclasse da classe *Parameters*.

Além disso, são declaradas as restrições de propriedades do objeto, tais como `<owl:Restriction>` para a propriedade *hasParameterPhase*, na qual restringe que a classe *dfs_block_size* possui algum valor dentro do *range* de instâncias da classe *IntermediatePhase*, por exemplo.

A definição das propriedades de dados para a classe usa a propriedade *owl:onProperty*. No exemplo acima a propriedade de dados associada a classe é a *valueParameter*. Para esta propriedade de dados é, também, declarada a restrição *owl:qualifiedCardinality* para restringir a entrada de apenas um valor para esta propriedade.

Assim, segue toda a implementação das classes da ontologia, de acordo com a especificação na formalização apresentada na Seção 5.4.

5.6.4 Implementação de Instâncias

A figura 5.6 mostra parcialmente as classes da ontologia e os axiomas da instância *unitDefault* pertencente a classe *UnitTuning*.

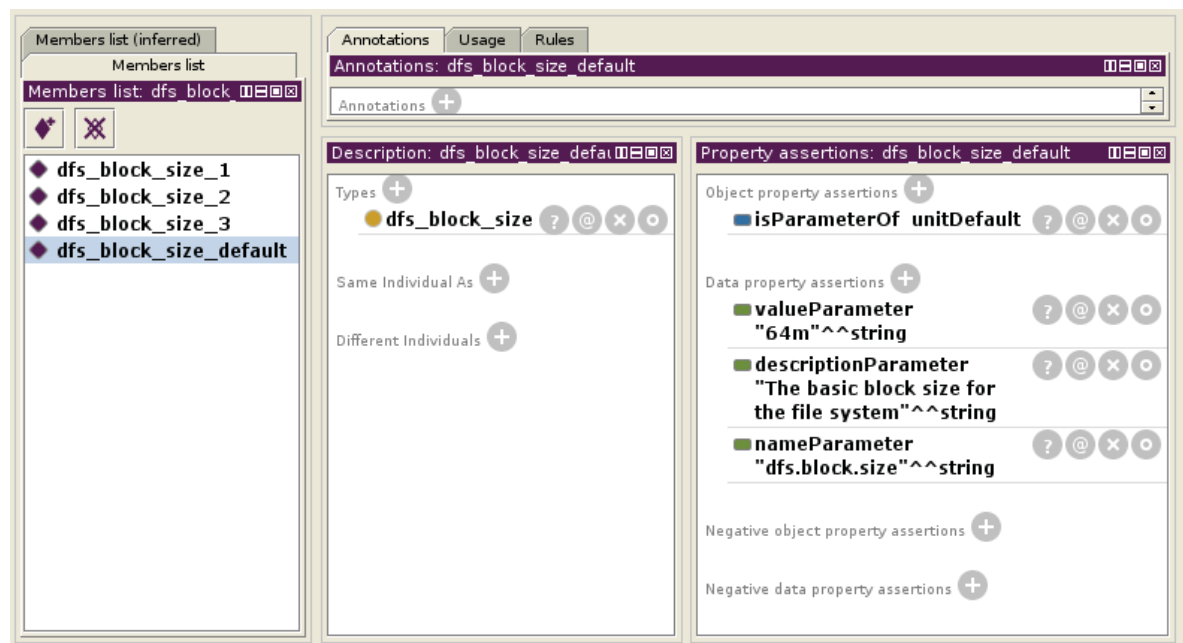


Figura 5.6: Instâncias da classe *dfs_block_size* vistas no Protégé.

Pode-se observar quatro instâncias (indivíduos) para a classe *dfs_block_size*, suas propriedades de dados e propriedades de objeto relacionadas ao indivíduo.

Abaixo é apresentado o trecho, em OWL, para a criação de uma instância de classe *dfs_block_size*:

```

1 <owl:NamedIndividual rdf:about="&OntoHadoop;dfs_block_size_default">
2   <rdf:type rdf:resource="http://www.owl-ontologies.com/OntoHadoop.owl
3     #dfs_block_size"/>
4   <OntoHadoop:valueParameter rdf:datatype="&xsd:string">64m</
     OntoHadoop:valueParameter>
5   <OntoHadoop:descriptionParameter rdf:datatype="&xsd:string">The basic
     block size for the file system</OntoHadoop:descriptionParameter>
6   <OntoHadoop:nameParameter rdf:datatype="&xsd:string">dfs.block.size</
     OntoHadoop:nameParameter>
7   <OntoHadoop:isParameterOf rdf:resource="&OntoHadoop;unitDefault"/>
8 </owl:NamedIndividual>

```

Para se criar uma instância em OWL, deve-se utilizar a propriedade *owl: NamedIndividual*, seguido do nome da instância (*dfs_block_size_default*) e a classe a que este indivíduo pertence (*dfs_block_size*). A seguir, cada uma das propriedades de dados recebem os valores de acordo com o *range* que foi definido para cada uma delas. Por exemplo, para a propriedade *OntoHadoop:valueParameter*, para a instância *dfs_block_size_default*, é atribuído o valor de "64m", e assim para todas as demais propriedades.

5.6.5 Implementação de Regras SWRL

Esta seção apresenta um exemplo de como as regras SWRL são implementadas na ontologia e integradas ao código OWL. As regras SWRL foram implementadas de acordo com duas categorias de regras:

- Regras para Padrão de Carga de trabalho: este conjunto de regras inferem os padrões de carga de trabalho para cada *Job* que é instanciado na OntoHadoop. As regras são baseadas na caracterização de cargas de trabalho apresentada pelo trabalho de (18) e discutidas no Capítulo 4, na Seção 4.3.4.
- Regras para Sintonia de Parâmetros: este conjunto de regras tem o objetivo de inferir os, teoricamente, mais adequados valores para os parâmetros de configuração do Hadoop. A definição dos melhores valores segue o estudo realizado no Capítulo 4, na Seção 4.2.4, bem como nas referências dos estudos apresentados naquele capítulo.

A seguir, é apresentada um exemplo de implementação de regras SWRL na OntoHadoop. A figura 5.7 mostra a *view* para criação e manipulação de regras SWRL na ferramenta Protégé.

A regra apresentada nesta figura representa uma das regras para a definição dos padrões de carga de trabalho. Esta regra, especificamente, define quais instâncias da classe *JobData* são do padrão de carga de trabalho *Aggregate_Fast*. Através da definição da regra em SWRL, é possível, através de um raciocinador, inferir o padrão da carga de trabalho de acordo com o tamanho dos dados de entrada, intermediário e saída do *Job*.

O trecho desta regra na linguagem SWRL é apresentado abaixo:

```

1 JobData(?x), sizeInputJobKb(?x, ?i), sizeOutputJobKb(?x, ?o),
   sizeShuffleJobKb(?x, ?s), greaterThan(?i, ?s), greaterThan(?s, ?o) ->
   hasWorkloadPattern(?x, Aggregate_Fast)

```

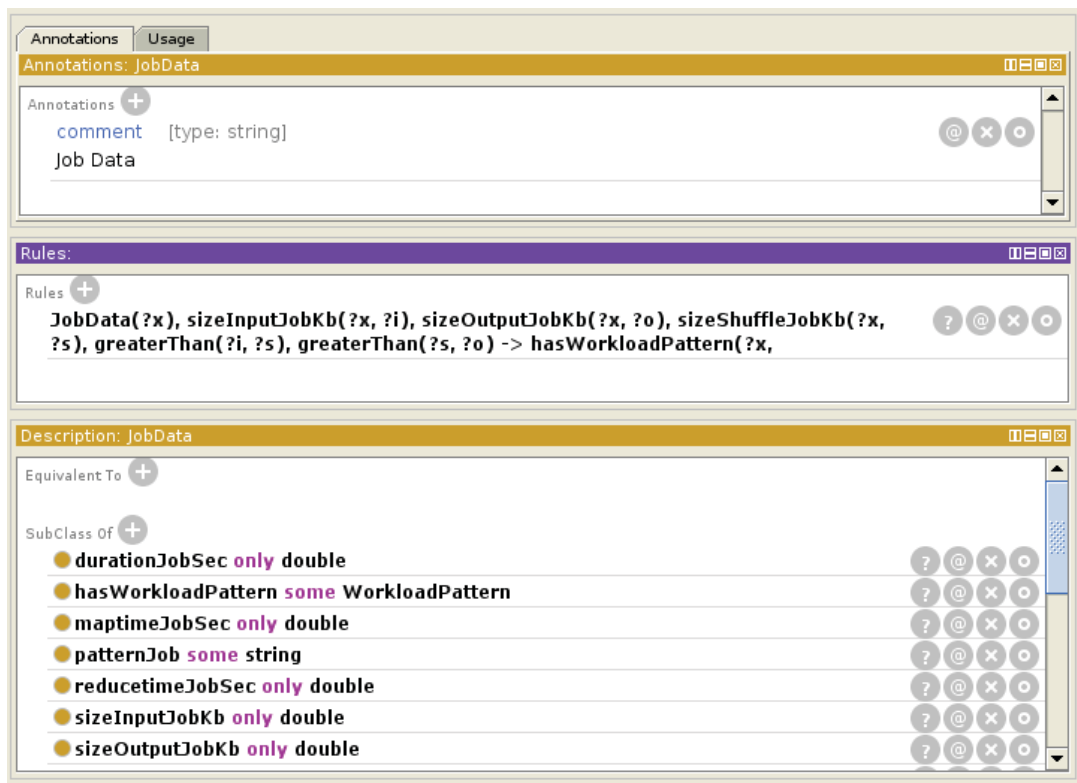


Figura 5.7: View para criação de regras SWRL no Protégé.

A regra é dividida em duas cláusulas: a antecedente e a consequente. Caso as condições (átomos) especificadas no antecedente forem verdadeiras, então as condições especificadas no consequente também devem ser. Neste exemplo, a cláusula antecedente possui seis átomos, separados por uma vírgula um do outro. A cláusula consequente possui apenas um átomo. A descrição de cada um dos átomos da cláusula antecedente são descritas a seguir:

- **JobData(?x):** é um átomo de classe (unário) que define a variável *?x* como uma indivíduo (instância) da classe *JobData*;
- **sizeInputJobKb(?x, ?i):** é um átomo de propriedades de dados (binário), e define *sizeInputJobKb* como uma propriedade de dados que relaciona a variável *?x* (indivíduo de *JobData*) a variável *?i* (valor da propriedade *sizeInputJobKb*). Ou seja, a primeira variável define o indivíduo e a segunda o seu valor;
- **sizeOutputJobKb(?x, ?o):** é um átomo de propriedades de dados (binário), e define *sizeOutputJobKb* como uma propriedade de dados que relaciona a variável *?x* (indivíduo de *JobData*) a variável *?o* (valor da propriedade *sizeOutputJobKb*);
- **sizeShuffleJobKb(?x, ?s)** como uma propriedade de dados que relaciona a variável *?x* (indivíduo de *JobData*) a variável *?s* (valor da propriedade *sizeShuffleJobKb*);
- **greaterThan(?i, ?s):** define a construção de uma propriedade SWRL embutida (*built-ins*) para avaliar se o átomo é verdadeiro através dos argumentos. Neste exemplo, verifica-se se a variável *?i* (*sizeInputJobKb*) é maior do que a variável *?s* (*sizeShuffleJobKb*);
- **greaterThan(?s, ?o):** este átomo, assim como o anterior, verifica-se se a variável *?s* (*sizeShuffleJobKb*) é maior do que a variável *?o* (*sizeOutputJobKb*).

A cláusula consequente é definida pelo átomo **hasWorkloadPattern(?x, Aggregate_Fast)**. Esta cláusula define, através de um átomo de propriedade de indivíduo, que se a cláusula antecedente for verdadeira, então, o indivíduo ?x (um indivíduo da classe *JobData*, como foi declarado) tem o padrão de carga de trabalho (definido pela propriedade de objeto *hasWorkloadPattern*) igual a *Aggregate_Fast*.

Em linguagem natural, a regra SWRL exemplificada acima, define: *Se o tamanho da carga de trabalho do Job for maior que o tamanho dos dados intermediários e, este, maior que o tamanho dos dados de saída, então o Job é do tipo de padrão de carga de trabalho Aggregate_Fast.*

Abaixo é apresentado um trecho parcial da definição da regra SWRL apresentada anteriormente, na linguagem OWL:

```

1 <!-- Definição das variáveis -->
2 <rdf:Description rdf:about="urn:swrl#x">
3   <rdf:type rdf:resource="&swrl;Variable"/>
4 </rdf:Description>
5 <rdf:Description rdf:about="urn:swrl#i">
6   <rdf:type rdf:resource="&swrl;Variable"/>
7 </rdf:Description>
8 <rdf:Description rdf:about="urn:swrl#o">
9   <rdf:type rdf:resource="&swrl;Variable"/>
10 </rdf:Description>
11 <rdf:Description rdf:about="urn:swrl#s">
12   <rdf:type rdf:resource="&swrl;Variable"/>
13 </rdf:Description>
14 <rdf:Description>
15   <rdf:type rdf:resource="&swrl;Imp"/>
16 <!-- Definição dos átomos pertencentes ao "corpo" (antecedente) -->
17   <swrl:body>
18     <rdf:Description>
19       <rdf:type rdf:resource="&swrl;AtomList"/>
20 <!-- Definição do primeiro átomo do "corpo" - JobData(?x) -->
21       <rdf:first>
22         <rdf:Description>
23           <rdf:type rdf:resource="&swrl;ClassAtom"/>
24           <swrl:classPredicate rdf:resource="&OntoHadoop;JobData"/>
25           <swrl:argument1 rdf:resource="urn:swrl#x"/>
26         </rdf:Description>
27       </rdf:first>
28 <!-- Definição do restante do "corpo" -->
29       <rdf:rest>
30         <rdf:Description>
31           <rdf:type rdf:resource="&swrl;AtomList"/>
32           <rdf:first>
33             <rdf:Description>
34               <rdf:type rdf:resource="&swrl;
35                 DatavaluedPropertyAtom"/>
36               <swrl:propertyPredicate rdf:resource="&OntoHadoop;
37                 sizeInputJobKb"/>
38               <swrl:argument2 rdf:resource="urn:swrl#i"/>
39               <swrl:argument1 rdf:resource="urn:swrl#x"/>
40             </rdf:Description>
41           </rdf:first>
42           <!-- Denificação dos demais átomos do "corpo" -->
43           </rdf:Description>
44         </rdf:rest>
45       </rdf:Description>

```

```

44         </swrl:body>
45 <!-- Final do "corpo" e início da "cabeça" (consequente) -->
46         <swrl:head>
47             <rdf:Description>
48                 <rdf:type rdf:resource="&swrl;AtomList"/>
49                 <rdf:rest rdf:resource="&rdf:nil"/>
50                 <rdf:first>
51                     <rdf:Description>
52                         <rdf:type rdf:resource="&swrl;IndividualPropertyAtom"/>
53                         <swrl:argument2 rdf:resource="&OntoHadoop;Aggregate_Fast"/>
54                         <swrl:propertyPredicate rdf:resource="http://www.owl-
55                             ontologies.com/
56                             OntoHadoop.owl#hasWorkloadPattern"/>
57                         <swrl:argument1 rdf:resource="urn:swrl#x"/>
58                     </rdf:Description>
59                 </rdf:first>
60             </rdf:Description>
61         </swrl:head>
62     </rdf:Description>
63 </rdf:RDF>

```

Como se pode observar, o código de implementação da regra em OWL é muito extenso. Sendo assim, a implementação das regras em OWL constam no arquivo de implementação da ontologia OntoHadoop (OntoHadoop.owl).

5.7 CONSIDERAÇÕES FINAIS

O capítulo apresentou OntoHadoop: uma ontologia para classificação e sintonia dos parâmetros de configuração do Hadoop MapReduce. Esta ontologia visa fornecer suporte semântico para a construção de um ambiente para alcançar a sintonia de parâmetros do Hadoop e melhorar o desempenho das aplicações Hadoop.

A maioria dos usuários Hadoop MapReduce não sabem exatamente quais os parâmetros de configuração são apropriados ou quais devem ser sintonizados para se obter o melhor desempenho das suas aplicações. Dessa forma, pode-se evitar desperdício de tempo e custos na busca de melhores sintonias, ao proporcionar para o usuário, indicativos dos parâmetros de configuração que podem melhorar o desempenho das aplicações.

Além disso, o ambiente Hadoop MapReduce precisa de ambientes que discutam sobre técnicas para prever a melhor sintonia dos parâmetros de configuração associados as cargas de trabalho e suas características específicas semanticamente modeladas. Além disso, para o nosso conhecimento, não há trabalho já realizado neste sentido, abordando a semântica com base ontológica para essa finalidade.

Por fim, foi implementada uma ontologia que será base de conhecimento para a abordagem semântica baseada em ontologia para a sintonia de parâmetros de configuração do Hadoop MapReduce a ser apresentada no próximo capítulo.

6 ABORDAGEM SEMÂNTICA PARA SINTONIA DE PARÂMETROS

Como foi visto nos capítulos anteriores, a melhor sintonia dos parâmetros do Hadoop não é uma tarefa fácil de ser obtida. Sendo assim, é de grande importância a criação de ambientes que promovam e motivem o compartilhamento de informação e disseminação do conhecimento em relação a este assunto. Além disso, é importante que todo conhecimento adquirido seja organizado para ser utilizado o mais rápido possível, de maneira fácil e eficiente quando for necessário.

Este capítulo propõe uma abordagem em direção a um ambiente semântico, baseado em uma ontologia, para melhorar o desempenho do Hadoop. O objetivo é apresentar uma abordagem semântica para a sintonia dos parâmetros de configuração do Hadoop com base no conhecimento embutido na ontologia proposta. Assim, os usuários de Hadoop MapReduce poderiam obter um melhor desempenho de seus aplicativos Hadoop. A abordagem foca nos parâmetros de configuração do Hadoop que têm influência sobre o desempenho da aplicação. A abordagem explora as características da carga de trabalho e do *cluster*, e analisa o histórico de registros de execuções anteriores. Em seguida, com base nos conceitos, propriedades, axiomas e regras implementadas na base de conhecimento ontológica, indicam-se os parâmetros de configuração sintonizáveis e procura-se a melhor sintonia para eles, a fim de melhorar o desempenho de aplicativos Hadoop em relação aos valores padrão dos parâmetros, criando uma Unidade de Sintonia para um determinado padrão de carga de trabalho.

Vale ressaltar que os códigos que implementam o protótipo desta abordagem, aqui proposta, são apresentados no Apêndice A.

6.1 ARQUITETURA DA ABORDAGEM

Esta seção apresenta uma arquitetura que reúne e organiza os componentes da abordagem semântica proposta para a sintonia de parâmetros de configuração do Hadoop MapReduce.

Como foi visto, depende-se de vários fatores para se alcançar uma configuração ideal dos parâmetros do Hadoop a fim de se obter o melhor desempenho possível de suas aplicações. Este fatores incluem as características da aplicação, das cargas de trabalho e do *cluster*, entre outros. O grande desafio é que os usuários do Hadoop MapReduce obtenham um indicativo dos parâmetros de configuração e da sintonia mais adequada para eles, de acordo com as características das cargas de trabalho que estão submetendo. Com isso, pretende-se obter o melhor desempenho possível do sistema como um todo. Esta arquitetura propõe uma abordagem que combina recursos semânticos, simulação, aprendizado de máquina e conhecimento sobre a sintonia dos parâmetros do Hadoop.

Como se pode observar na Figura 6.1, a arquitetura é dividida em quatro camadas principais: Aplicação, Mediadora, Sintonia e Conhecimento.

6.1.1 Camada de Aplicação

A Camada de Aplicação (*Application Tier*) é responsável pela interação com o usuário. O objetivo é a inserção de recursos para a inserção de novas unidades de sintonia e buscas das melhores sintonias através de busca simples ou através de *Logs* de histórico de execuções passadas.

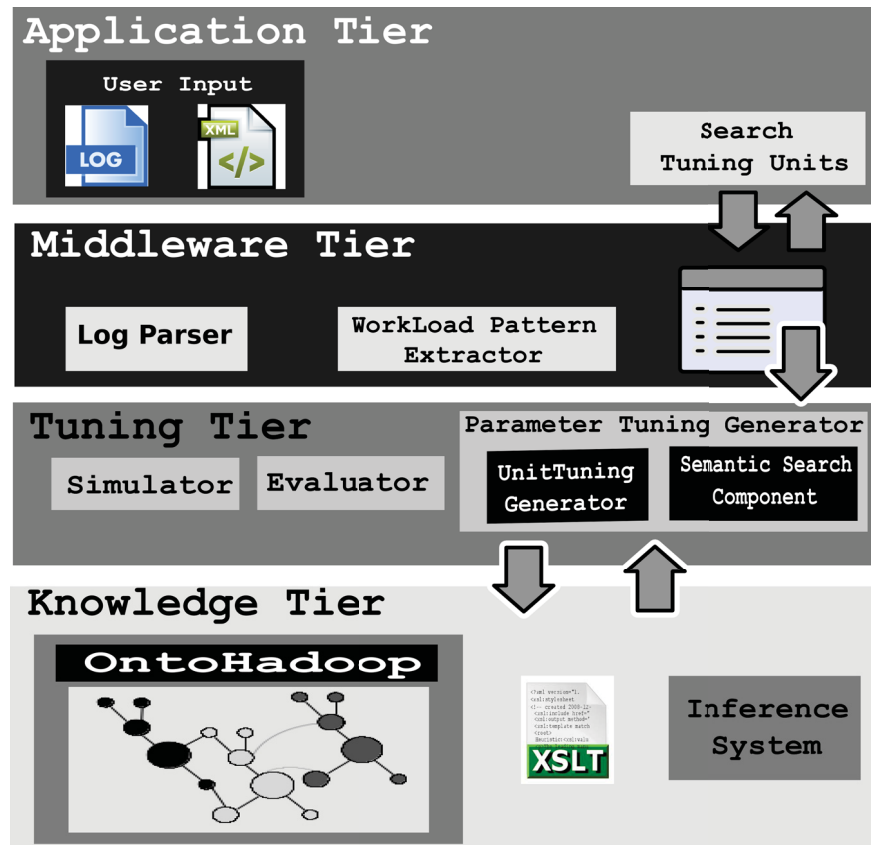


Figura 6.1: Arquitetura proposta para a Abordagem Semântica.

1. **Log:** entrada por *Log* de histórico de execução de aplicações do Hadoop. O usuário opta por este tipo de entrada quando ele já executou sua aplicação no ambiente Hadoop e, este, retornou dois *logs* para cada execução. O primeiro com informações sobre os valores dos parâmetros utilizadas na execução; o segundo com os valores de saída da execução, tais como: tamanho da carga de trabalho, quantidade de dados lidos e escritos nas fases *map*, *shuffle* e *reduce*, tempos de execução total e por fases, números de tarefas *map*, números de tarefas *reduce*, etc. O exemplo abaixo apresenta estes o *log* de configuração de parâmetros parcialmente:

```

1  <?xml version="1.0" encoding="UTF-8" standalone="no"?><configuration
2  >
3  <property>
4    <name>fs.s3n.impl</name>
5    <value>org.apache.hadoop.fs.s3native.NativeS3FileSystem</value>
6  </property>
7  <property>
8    <name>io.sort.factor</name>
9    <value>10</value>
10 </property>
11 <property>
12   <name>dfs.datanode.data.dir.perm</name>
13   <value>755</value>
14 </property>
15 <property>
16   <name>dfs.block.size</name>
   <value>67108864</value>

```

```

17 </property>
18 <property>
19     <name>dfs.replication.max</name>
20     <value>512</value>
21 </property>
22 <property>
23     <name>io.sort.spill.percent</name>
24     <value>0.80</value>
25 </property>
26 <property>
27     <name>mapred.job.shuffle.input.buffer.percent</name>
28     <value>0.70</value>
29 </property>
30 </configuration>

```

Observe que algumas informações são desnecessárias para o nosso objetivo, pois tratam de parâmetros de configuração de ambiente, permissões, etc. Por exemplo:

```

1 <property>
2     <name>dfs.datanode.data.dir.perm</name>
3     <value>755</value>
4 </property>

```

Outras informações são fundamentais, pois referem-se aos valores dos parâmetros que tem influência no desempenho. Por exemplo:

```

1 <property>
2     <name>dfs.block.size</name>
3     <value>67108864</value>
4 </property>

```

O exemplo abaixo apresenta um trecho do segundo arquivo de *log* gerado pela execução de uma aplicação Hadoop Mapreduce:

```

1 Meta VERSION="1" .
2 Job JOBID="job_201305152048_0001" JOBNAME="grep-search"
3 USER="ailton" SUBMIT_TIME="1368661899273"
4 JOBCONF="hdfs://localhost:54310/tmp/hadoop-ailton/mapred/staging/
  ailton/
5 \.staging/job_201305152048_0001/job\.xml"
6 VIEW_JOB="*" MODIFY_JOB="*" JOB_QUEUE="default" .
7 Job JOBID="job_201305152048_0001" JOB_PRIORITY="NORMAL".
8 Job JOBID="job_201305152048_0001" LAUNCH_TIME="1368661899912"
9 TOTAL_MAPS="20" TOTAL_REDUCES="1" JOB_STATUS="PREP" .
10 Task TASKID="task_201305152048_0001_m_000021" TASK_TYPE="SETUP"
11 START_TIME="1368661899916" SPLITS="" .
12 MapAttempt TASK_TYPE="SETUP" TASKID="task_201305152048_0001_m_000021"
  "
13 TASK_ATTEMPT_ID="attempt_201305152048_0001_m_000021_0" START_TIME="
  1368661900708"
14 TRACKER_NAME="tracker_ailton-note:localhost/127\0\0\1:45298"
  HTTP_PORT="50060" .
15 MapAttempt TASK_TYPE="SETUP" TASKID="task_201305152048_0001_m_000021"
  "
16 TASK_ATTEMPT_ID="attempt_201305152048_0001_m_000021_0"
17 TASK_STATUS="SUCCESS" FINISH_TIME="1368661902789"
18 HOSTNAME="/default-rack/ailton-note" STATE_STRING="setup"
19 COUNTERS="{(FileSystemCounters)(FileSystemCounters)}

```

```

20 [ (FILE_BYTES_WRITTEN) (FILE_BYTES_WRITTEN) (52025) ] }
21 { (org\.apache\.hadoop\.mapred\.Task

```

2. **XML**: entrada por arquivo XML contendo informações sobre o padrão da carga de trabalho, parâmetros de configuração e valores para os parâmetros de configuração. Estas informações são geradas por um especialista e, teoricamente já é um esquema de Unidade de Sintonia ratificada.
3. **Search Tuning Units**: interface que permite a pesquisa de Unidades de Sintonia através da entradas de *Log* ou arquivo XML. Permite realizar consultas na base de conhecimento através da linguagem SPARQL, utilizando-se de triplas RDF geradas pelo modelo ontológico.

6.1.2 Camada Mediadora

A camada de mediação (*Middleware Tier*) fornece mecanismos entre a camada de aplicação e as demais camadas da abordagem. Ela é responsável por mover, transportar e transformar os dados de entrada em dados que integram as camadas inferiores. Esta camada apresenta dois componentes principais:

1. **Log Parser**: este componente é responsável por fazer o *parser* dos arquivos de *log* do histórico de execução de uma aplicação. O *parser* extrai apenas as informações relevantes no que diz respeito aos parâmetros de configuração e seus valores, bem como as informações (já citadas) sobre as cargas de trabalho e tempos de execução. Além da extração, o *parser* gera arquivos em um formato que deverá ser aceito pelo simulador (*JobConf*) e aceito como entrada para se instanciar a ontologia com os dados do *Job*, na classe *JobData*.
2. **WorkLoad Pattern Extractor**: este componente é responsável pela extração das características da carga de trabalho através dos dados de entrada do *Job*. Ele analisa as informações de entrada e aplica as regras para caracterização da carga de trabalho, já citadas na Seção 4.3.4 do Capítulo 4.

6.1.3 Camada de Sintonia

A camada de sintonia (*Tuning Tier*) é responsável pela manipulação dos dados de entrada que foram transformados pela camada mediadora. Nesta camada ocorrem as principais funções da abordagem em termos de se gerenciar o conteúdo semântico e gerar as Unidades de Sintonia. Esta camada possui três componentes principais:

1. **Simulator**: este componente é responsável por aceitar um *JobConf* como entrada e fazer a simulação desta configuração. O resultado da simulação é transferido ao componente *Evaluator* para a análise dos resultados em relação a uma execução com uma configuração anterior.
2. **Evaluator**: este componente é responsável pela análise dos resultados obtidos através da simulação. O componente faz a comparação de uma configuração anterior com a nova configuração, a fim de se observar os tempos de execução e verificar se houve melhora no desempenho. Além disso, este componente, pode criar novas populações de configurações, baseadas em regras implementadas na base de conhecimento, para novas simulações.

3. **Parameter Tuning Generator:** esse componente é subdividido em dois subcomponentes:

- **UnitTuning Generator:** este componente é responsável por criar as instâncias de Unidades de Sintonia (*Tuning Units*) a partir da simulação e avaliação de um *JobConf* ou, por meio da entrada de um esquema XLM contendo Unidades de Sintonia criadas por um especialista.
- **Semantic Search Component:** este componente organiza as consultas para o motor de inferência e controla o conhecimento inferido, classificando e organizando os resultados.

6.1.4 Camada de Conhecimento

A camada de conhecimento (*Knowledge Tier*) é basicamente composta pela ontologia OntoHadoop. Esta camada é fundamental para a arquitetura proposta. Toda abordagem deste trabalho é baseada no conhecimento embutido na ontologia e no sistema de inferência que gerencia o conhecimento compartilhado entre os usuários, aplicações e comportamentos do processo como um todo.

Nessa camada são organizados o conhecimento a respeito da classificação dos parâmetros, suas relações com as cargas de trabalho, relações com as características das aplicações, valores que os parâmetros podem assumir, etc (todos os detalhes são apresentados no Capítulo 5. Todo esse conhecimento vem da experiência dos usuários, da experiência de especialistas e das pesquisas realizadas na revisão sistemática apresentada no Capítulo 4, tendo como principal objeto a geração de Unidades de Sintonia.

Além disso, a camada apresenta um sistema de inferência para classificar e organizar todos os conceitos, propriedades, valores de dados, axiomas e regras da OntoHadoop. Assim, o conhecimento inferido pode servir de base de buscas de informações a respeito dos parâmetros de várias maneiras.

6.2 IMPLEMENTAÇÃO DA ABORDAGEM

Através da utilização da arquitetura proposta na seção anterior, pode-se apresentar um esquema de implementação da abordagem semântica baseada em ontologias para sintonia de parâmetros do Hadoop MapReduce.

A Figura 6.2 fornece uma visão geral do esquema de implementação da abordagem semântica proposta. A abordagem baseia-se na ontologia para o Hadoop proposta neste trabalho: a OntoHadoop. O esquema de implementação divide-se em três módulos principais: pré-processamento (*Pre-Processing*), o gerador de sintonia de parâmetros (*Parameter Tuning Generator*) e o módulo de busca (*Search*) (12).

Sendo uma abordagem semântica, propõe-se que toda a representação do conhecimento, gerado pelas ferramentas de colaboração e pela inserção de conhecimentos externos, deve ser organizada em uma ontologia. A descrição e os detalhes de cada componente do ambiente são apresentados nas próximas subseções.

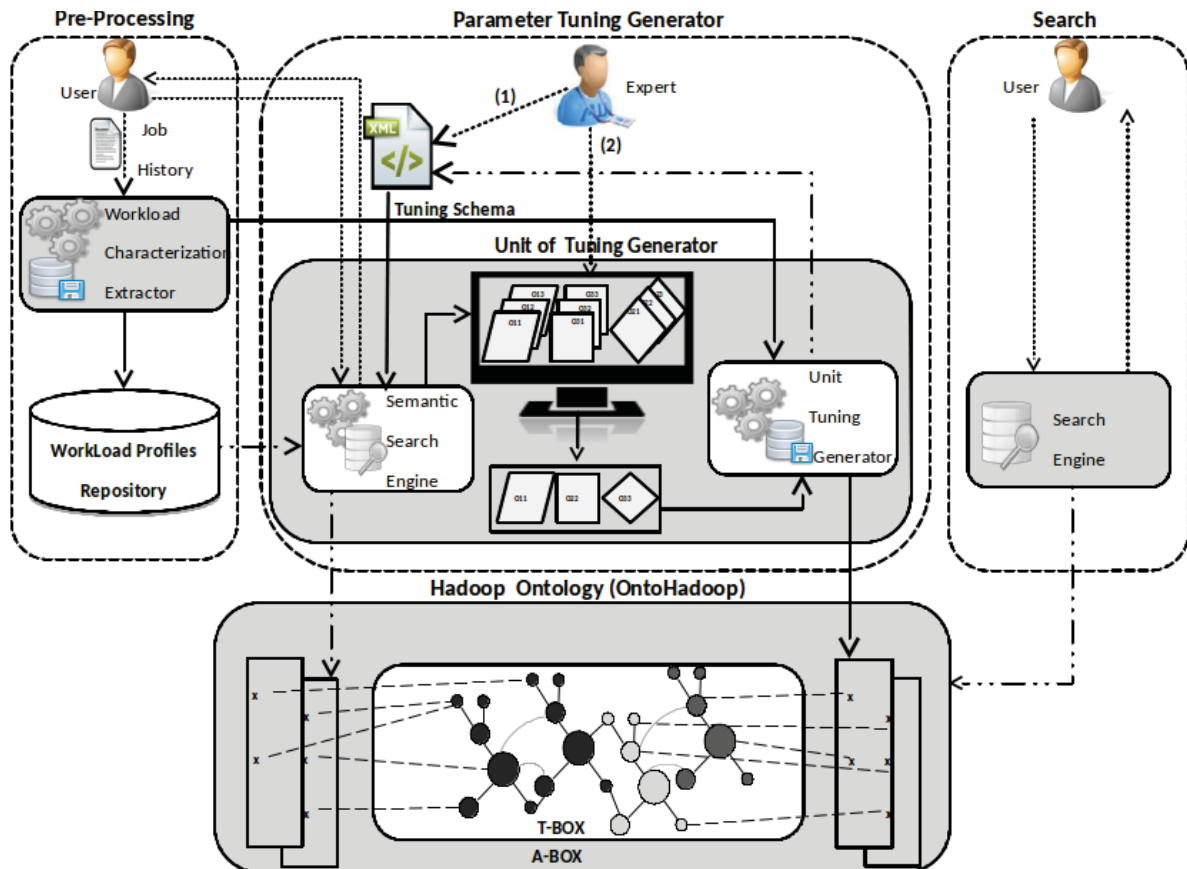


Figura 6.2: Esquema de Implementação da Abordagem Semântica.

6.2.1 Pré-Processamento

O módulo pré-processamento é caracterizado principalmente pelo Extrator de Características da Carga de Trabalho (WCE - *Workload Characterization Extractor*). O usuário envia o histórico de execução (*log*) de um *job* no Hadoop. Este registro é analisado pelo WCE e armazenado no repositório de perfis das cargas de trabalho. Este processo serve para dois propósitos: o primeiro é a caracterização da própria carga de trabalho e a segunda é instanciar a ontologia com essas características.

6.2.1.1 Extrator de Padrão da Carga de Trabalho

Quando os *logs* são apresentados, um *parser* extrai as informações das taxas de dados, como o número de bytes lidos e escritos nas fases de *input*, *shuffle* e *output*; além da duração total de execução da aplicação e da duração das fases *map* e *reduce*. Assim, as métricas de desempenho representam cada *job* de acordo com seu tamanho, tempo de execução, ou tempo das tarefas *map* e *reduce*.

Em seguida, o módulo utiliza-se do motor de inferência e aplica as regras SWRL embutidas na ontologia para estimar o tipo ou padrão da carga de trabalho a partir das variáveis já mencionadas. Outra forma de extração do padrão da carga de trabalho seria alguma abordagem baseada em aprendizado de máquina. Pode-se utilizar abordagens tais como, regressão linear (58; 73), dentre outras, onde os dados são modelados usando funções de predição linear, e os modelos de parâmetros desconhecidos são estimados a partir dos dados.

Desta forma, o módulo tem o papel de extrair a principal característica da carga de trabalho em relação à classificação sugerida por (18), que classifica as cargas de trabalho em:

small jobs, *load data*, *aggregate*, *aggregate and expand*, *expand and aggregate*, *data transform*, *data summary* and *transform and expand*. Todas as informações extraídas são armazenadas em um repositório de perfil para uso posterior em uma nova submissão de uma configuração de *job*. A intenção é que quanto mais configurações de *jobs* sejam submetidas, melhor as caracterizações no futuro.

6.2.2 Gerador de Sintonia de Parâmetros

O gerador de unidades de sintonia (*Parameter Tuning Generator*) é dividido em dois subcomponentes: o motor de busca semântico (*Semantic Search Engine*) e o gerador de unidade de sintonia (*Unit Tuning Generator*).

6.2.2.1 Motor de Busca Semântico

O motor de busca semântico tem três propósitos:

- Pesquisar unidades de sintonia existentes: Depois que o esquema de sintonia foi definido, o motor de busca semântico verifica se já existe alguma unidade de sintonia semelhante e apresenta-o ao especialista;
- Fazer buscas por um esquema completo de configuração de *job*: o registro é analisado e classificado no pré-processamento (ver Seção 6.2.1) e, depois de armazenado no repositório, é lançado para o motor de busca semântico para encontrar uma unidade de sintonia adequada dentro da ontologia;
- Busca por declarações em SPARQL (74) diretamente na ontologia por declarações específicas sobre o tipo, tamanho e outras características da carga de trabalho.

6.2.2.2 Gerador de Unidades de Sintonia

O Gerador de Unidades de Sintonia é responsável pela geração de unidades de sintonia que compõem as instâncias da ontologia. Em primeiro lugar, um *expert* é responsável pela criação de um esquema de sintonia (ver figura 6.2 (1)), o qual define um esquema de sintonia de parâmetros teoricamente ideal para uma dada carga de trabalho. Este esquema de sintonia irá conter informações sobre a carga de trabalho, tais como o perfil da carga de trabalho, quais as fases do Hadoop são críticos para esta carga e se a aplicação é influenciada pelo uso intensivo de CPU, IO ou memória, entre outros. Assim, na figura 6.2 (2), o especialista avalia, seleciona, e se quiser, altera as unidades de sintonia. Então, esta unidade vai para o módulo Gerador de Unidade de Sintonia para ser instanciado na ontologia OntoHadoop;

Outra forma de criação de Unidades de Sintonia é através da inferência de parâmetros e valores de acordo com o padrão da carga de trabalho. Este processo partiria das inferências sobre as regras SWRL definidas na OntoHadoop. Assim, seria criadas instâncias de Unidades de Sintonia (*Tuning Units*) a partir da simulação e avaliação de um *JobConf*.

6.2.3 Módulo de Buscas

Após a ontologia ser populada, os usuários podem realizar buscas por cargas de trabalho ou parâmetros de configuração do Hadoop. Este componente não tem apoio completo de busca semântica. As buscas podem ser feitas por declarações simples de SPARQL. O componente de pesquisa busca a ontologia, tanto por tipo de carga de trabalho e por unidades de sintonia, de acordo com as palavras-chave definidas pelo usuário. Os resultados são apresentados para

o usuário, a fim de saber quais informações sobre as Unidades de Sintonia estão contidas na ontologia. As buscas podem ser efetuadas por parâmetros, por fases do Hadoop, por padrão da carga de trabalho, entre outras. Além disso, outras consultas com caráter de informação e conhecimento sobre os parâmetros podem ser efetivadas, tais como a classificação dos parâmetros, descrição, valores padrão, entre outras.

6.3 FLUXO DOS PROCESSOS PARA GERAÇÃO DE UNIDADES DE SINTONIA

A Figura 6.3 apresenta um fluxograma dos processos envolvidos na criação, simulação e avaliação das Unidades de Sintonia que irão compor a base de conhecimento da abordagem semântica.

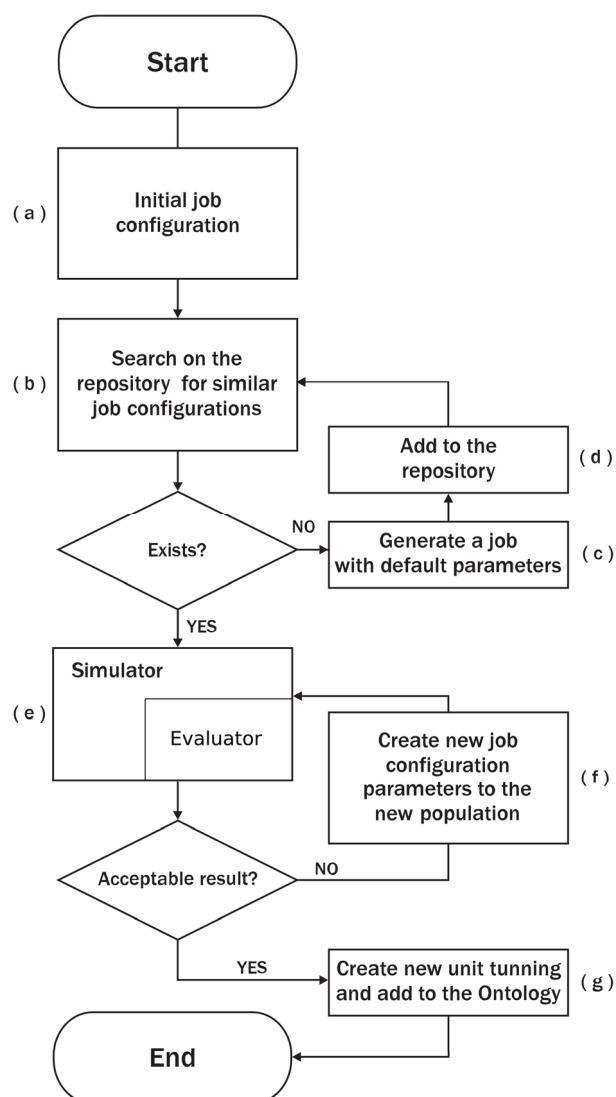


Figura 6.3: Fluxograma demonstrando o fluxo dos processos para criação de Unidades de Sintonia.

Abaixo, são descritos os módulos do fluxograma:

1. Após a entrada do *log* histórico ou arquivo XML com configurações do *Job* e análise do *parser* e/ou extrator de padrão de carga de trabalho, recebe um *Job Configuration*, com

os parâmetros de configuração do Hadoop que sejam compatíveis com os parâmetros do simulador (neste trabalho utiliza-se o simulador MRSim (18)).

Para visualizar os códigos de implementação, vide o Apêndice A, Seção A.1 e Seção A.2.

2. Responsável por buscar no repositório de configurações de *job* (pode ser um banco de dados, um repositório de arquivos ou, ainda, na própria ontologia), as configurações de similares com as da entrada.
3. Caso não exista uma configuração inicial similar no repositório de configurações, gera uma configuração de *job* com valores padrão;
4. Caso exista uma configuração similar, avança para o módulo (*e*), senão, adiciona uma configuração padrão do *job* para ser novamente procurada e chegar a resposta positiva na busca por configurações similares.
5. Utiliza o *Job configuration* semelhante ou exato ao de entrada para a simulação utilizando o MRSim. A saída desse bloco indica uma execução simulada do Hadoop MapReduce, por exemplo:

```

1 Map Counters:
2 DURATION      1,681.253
3 FILE_BYTES_READ 1,134,000,000.
4 FILE_BYTES_WRITTEN 2,268,000,000.
5 HDFS_BYTES_READ 478,627,929.
6 MAP_INPUT_RECORDS 1,620,000.
7 MAP_INPUT_SIZE 478,627,929.
8 MAP_OUTPUT_BYTES 1,134,000,000.
9 MAP_OUTPUT_RECORDS 81,000,000.
10 SPILLED_RECORDS 162,000,000.
11 START_TIME    64,620.
12 STOP_TIME     66,301.253
13
14 Reduce Counters:
15 DURATION      237.225
16 FILE_BYTES_READ 1,549,800,000.
17 FILE_BYTES_WRITTEN 1,644,300,000.
18 HDFS_BYTES_WRITTEN 989,236.112
19 REDUCE_INPUT_RECORDS 81,000,000.
20 REDUCE_OUTPUT_RECORDS 100,035.
21 REDUCE_SHUFFLE_BYTES 1,134,000,000.
22 SHUFFLE 1,134,000,000.
23 SPILLED_RECORDS 117,450,000.
24 START_TIME    1,091.
25 STOP_TIME     1,328.225
26 av mappers: 28.02088912378694
27 av reducers: 237.2251833172577
28
29 Job Counters:
30 DURATION      1,918.479
31 FILE_BYTES_READ 2,683,800,000.
32 FILE_BYTES_WRITTEN 3,912,300,000.
33 HDFS_BYTES_READ 478,627,929.
34 HDFS_BYTES_WRITTEN 989,236.112
35 JOB_START_TIME 1,063.
36 JOB_STOP_TIME 1,338.

```

```

37 | JOB_TOTAL_TIME 275.
38 | MAP_INPUT_RECORDS 1,620,000.
39 | MAP_INPUT_SIZE 478,627,929.
40 | MAP_OUTPUT_BYTES 1,134,000,000.
41 | MAP_OUTPUT_RECORDS 81,000,000.
42 | REDUCE_INPUT_RECORDS 81,000,000.
43 | REDUCE_OUTPUT_RECORDS 100,035.
44 | REDUCE_SHUFFLE_BYTES 1,134,000,000.
45 | SHUFFLE 1,134,000,000.
46 | SPILLED_RECORDS 279,450,000.
47 | START_TIME 65,711.
48 | STOP_TIME 67,629.479
49 | avMappersTime 28.021
50 | avReducersTime 237.225

```

O módulo de avaliação analisa as saídas da simulação e apresenta essas saídas ao usuário. Isto é feito através da geração de um arquivo CSV e gráficos gerados a partir desses dados. Dessa forma pode-se avaliar se os atributos pertinentes a sua execução, e os tempos de execução gerados são aceitáveis. Caso não sejam o bloco (f) é executado.

Para visualizar os códigos de implementação para os arquivos de entrada e simulação, vide o Apêndice A, Seção A.3.

6. Neste bloco os parâmetros de configurações são modificados, utilizando qualquer processo de *machine learning*, algoritmos genéticos, etc, ou ainda, através da inferência sobre as regras SWRL embutidas na base de conhecimento. Assim, uma nova população de configurações de *jobs* (*Job configuration*) é criada. Esta população então é novamente simulada e avaliada (*bloco (e)*). A saída pode ser tanto a lista de todas as simulações executadas ou, caso o usuário tenha informado os parâmetros que o interessam, somente o *Job configuration* que tenha melhor resultado em relação ao parâmetro(s) de interesse. A avaliação dos resultados é feita através da comparação com a simulação dos valores de entrada iniciais, sejam eles por uma configuração inicial do usuário ou a configuração dos valores padrão dos parâmetros do Hadoop.
7. Os valores da simulação pertinentes a última execução aprovada pelo usuário, são instanciados na ontologia juntamente com o *Job configuration* aprovado. Os valores são armazenados no formato de Unidades de Sintonia, sendo que a configuração aprovada pelo usuário tem a *flag* de "*approved by expert*" ("aprovada por especialista"), e as outras de valores similares são adicionadas ao repositório de ontologia sem esta *flag*. Então, é criada uma nova Unidade de Sintonia e instanciada na OntoHadoop.

Para visualizar os códigos da OWL API, utilizados para instanciar a ontologia, vide o Apêndice A, Seção A.4.

A partir do momento que a ontologia é populada pelas Unidades de Sintonia, o processo de seleção de *Job configuration* é feito com base na classe *HadoopUnitTuning*, ou seja, o *Job configuration*, para simulações futuras, é criado a partir dos atributos presentes nas Unidade de Sintonia.

Por fim, verificamos que o simulador, não somente é parte do processo de avaliação das Unidades de Sintonia, mas também parte do processo de criação das Unidades de Sintonia.

6.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou a arquitetura de uma abordagem semântica baseada em ontologia. Esta abordagem visa a construção de um ambiente para alcançar os melhores parâmetros de sintonia do Hadoop e melhorar o desempenho das aplicações neste ambiente.

Saber de antemão quais os melhores parâmetros de sintonia e quais os melhores valores de sintonia para eles antes da aplicação ser executada pode evitar, entre outras coisas, desperdício de tempo e economia nos custos de execução.

Além disso, o ambiente Hadoop MapReduce carece de ambientes que discutam sobre técnicas para prever a melhor definição dos parâmetros de configuração associados às cargas de trabalho e suas características específicas, bem como de uma base de conhecimento modelada com este objetivo. Outrossim, até onde se sabe, não há um trabalho já realizado neste sentido, que aborde o conhecimento semântico com base ontológica para esta finalidade.

7 AVALIAÇÃO EXPERIMENTAL

Neste capítulo será apresentada uma avaliação experimental que pretende responder as duas questões iniciais propostas neste trabalho:

1. A abordagem proposta, baseada em uma ontologia, para gerar unidades de sintonia dos parâmetros de configuração do Hadoop, pode auxiliar na melhora do desempenho das aplicações?
2. A abordagem semântica contribui para o compartilhamento e reuso do conhecimento sobre a sintonia de parâmetros do Hadoop?

A avaliação é dividida em duas etapas: a primeira avaliando a ontologia, seus conceitos, relações, axiomas e regras; na segunda será apresentada uma simulação da produção de uma Unidade de Sintonia com seu ciclo completo.

7.1 AVALIAÇÃO DA ONTOHADOOP

Segundo (85), a avaliação de uma ontologia consiste em verificar se a mesma satisfaz os requisitos definidos em sua construção. A sugestão proposta é que a etapa da avaliação seja realizada em paralelo com as fases de captura e formalização, definindo-se um conjunto de critérios com o objetivo de guiar tanto o desenvolvimento, quanto a avaliação da qualidade das ontologias construídas. Dentre estes critérios estão: clareza, coerência, extensibilidade e compromissos ontológicos (classes, propriedades, axiomas). Além disso, as questões de competência devem ser utilizadas para avaliação dos axiomas.

De fato, a avaliação de ontologias e taxonomias vem sendo bastante discutida e ainda não existem abordagens abrangentes e gerais para o problema. Como afirmam (15), existem problemas inerentes na tentativa de avaliar uma ontologia caso não esteja claro o que se deseja avaliar. Porém os trabalhos que falam sobre avaliação de ontologias concordam, de uma maneira geral, que boas ontologias são aquelas que servem aos seus propósitos.

De um modo ou de outro, é consenso da comunidade científica que a tarefa de avaliação é manual, trabalhosa e subjetiva (28) e dependente da intuição do especialista, sendo o rigor da avaliação de difícil verificação. Assim, alguns critérios que focalizam-se sobre os conceitos e definições que compõem a OntoHadoop foram utilizados:

- Verificação da estrutura ou arquitetura da ontologia: o objetivo é verificar se as definições foram construídas seguindo os critérios de projeto;
- Verificação da sintaxe das definições: o objetivo é verificar se existe estruturas ou palavras-chave sintaticamente incorretas nas definições;
- Verificação dos conteúdos das definições: verificar resultados das definições, se estão corretos ou não e o que pode ou não ser inferido.

A avaliação da ontologia consistiu da verificação de características, tais como sua expressividade e consistência, delimitadas através das questões de competência e inferências obtidas após a execução do motor de inferência Pellet¹, que compõe a ferramenta Protégé.

¹<http://clarkparsia.com/pellet/>

7.1.1 Questões de Competência

As questões de competência dizem respeito a verificação da construção da ontologia desde sua fase de modelagem até a obtenção de resultados através de um motor de inferência. Assim verificam-se se as respostas obtidas respondem as questões de competência inicialmente formuladas durante a definição do escopo da ontologia. As respostas obtidas foram baseadas no resultado recuperado por consultas efetivadas através da linguagem SPARQL.

As questões devem responder questões de funcionalidade, tais como: A ontologia descreve todos os parâmetros de configuração do Hadoop que influenciam no desempenho do sistema, bem como suas descrições, valores, classificação por categoria, nível, fases críticas dentro da execução e características da carga de trabalho associadas?

Algumas questões exemplos e suas respostas são apresentadas a seguir. Quaisquer questões podem ser respondidas de acordo com a declaração SPARQL construída. As duas primeiras consultas foram executadas em SPARQL via linha de comando. A consulta 3 foi executada por SPARQL dentro do ambiente Protégé:

1. Consulta 1: Quais os parâmetros de configuração do Hadoop influenciam no desempenho?

Query em SPARQL:

```

1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX owl: <http://www.w3.org/2002/07/owl#>
3 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 PREFIX onto: <file:/home/ailton/Dados_Linux/Tese/Texto_Tese/
    OntoHadoop.owl#>
6 SELECT ?AllParameters
7     WHERE { ?AllParameters rdfs:subClassOf <http://www.owl-
    ontologies.com/OntoHadoop.owl#Parameters> }
8     ORDER BY ?x

```

Resposta:

```

-----
| AllParameters
-----
| <http://www.owl-ontologies.com/OntoHadoop.owl#dfs_block_size>
| <http://www.owl-ontologies.com/OntoHadoop.owl#dfs_replication>
| <http://www.owl-ontologies.com/OntoHadoop.owl#dfs_replication_interval>
| <http://www.owl-ontologies.com/OntoHadoop.owl#io_file_buffer_size>
| <http://www.owl-ontologies.com/OntoHadoop.owl#io_sort_factor>
| <http://www.owl-ontologies.com/OntoHadoop.owl#io_sort_mb>
| <http://www.owl-ontologies.com/OntoHadoop.owl#io_sort_record_percent>
| <http://www.owl-ontologies.com/OntoHadoop.owl#io_sort_spill_percent>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_child_java_opts>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_compress_map_output>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_inmem_merge_threshold>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_job_reduce_input_buffer_percent>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_job_reuse_jvm_num_tasks>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_job_shuffle_input_buffer_percent>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_job_shuffle_merge_percent>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_map_output_compression_codec>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_map_tasks>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_map_tasks_speculative_execution>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_min_split_size>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_output_compression_type>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_reduce_parallel_copies>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_reduce_slowstart_completed_maps>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_reduce_tasks>
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_reduce_tasks_speculative_execution>

```

```
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_tasktracker_map_tasks_maximum> |
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_tasktracker_reduce_tasks_maximum> |
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapred_tasktracker_tasks_maxmemory> |
| <http://www.owl-ontologies.com/OntoHadoop.owl#mapreduce_combine_class> |
| <http://www.owl-ontologies.com/OntoHadoop.owl#min_num_spills_for_combine> |
-----
```

2. Consulta 2: Quais as Unidades de Sintonia instanciadas na ontologia?

Query em SPARQL:

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX owl: <http://www.w3.org/2002/07/owl#>
3 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5
6 SELECT ?UnitsTuning
7     WHERE { ?UnitsTuning rdf:type <http://www.owl-ontologies.com/
8         OntoHadoop.owl#HadoopUnitTuning> }
9     ORDER BY ?UnitsTuning
```

Resposta:

```
-----
| UnitsTuning |
|-----|
| <file:/home/ailton/Dados_Linux/Tese/Texto_Tese/OnoHadoop.owl#unit1> |
| <file:/home/ailton/Dados_Linux/Tese/Texto_Tese/OnoHadoop.owl#unit3> |
| <file:/home/ailton/Dados_Linux/Tese/Texto_Tese/OnoHadoop.owl#unitDefault> |
|-----|
```

3. Consulta 3: Quais propriedades compõe a Unidade de Sintonia "unit1"?

Query em SPARQL:

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX owl: <http://www.w3.org/2002/07/owl#>
3 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
4 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5 PREFIX onto: <file:/home/ailton/Dados_Linux/Tese/Texto_Tese/
6     OntoHadoop.owl#>
7 SELECT ?propriedade ?valor
8     WHERE { <file:/home/ailton/Dados_Linux/Tese/Texto_Tese/
9         OntoHadoop.owl#unit1> ?propriedade ?valor }
10    ORDER BY ?propriedade
```

Resultado apresentado na Figura 7.1 após executar a consulta no ambiente Protégé.

7.1.2 Regras e Inferências Obtidas

Como parte da avaliação, as inferências obtidas apontam se os resultados obtidos são equivalentes aos resultados esperados, verificando se os conceitos, relações, axiomas e regras SWRL implementadas na ontologia desenvolvida atendem os objetivos propostos.

Com a utilização do raciocionador Pellet, dentro do ambiente Protégé, é possível inferir algumas informações relevantes para o processo de criação das Unidades de Sintonia. Caso a inferência responda corretamente de acordo com as regras estabelecidas, a construção da

propriedade	valor
flagTuningUnit	"approved by expert"^^<http://www.w3.org/2001/XMLSchema#string>
hasParameter	dfs_block_size_3
hasParameter	dfs_replication_default
hasParameter	mapreduce_combine_class_1
hasParameter	io_sort_spill_percent_default
hasParameter	mapred_reduce_tasks_1
hasParameter	mapred_child_java_opts_1
hasParameter	io_file_buffer_size_default
hasParameter	mapred_compress_map_output_1
hasParameter	io_sort_record_percent_default
hasParameter	mapred_map_tasks_1
hasParameter	io_sort_factor_default
hasParameter	io_sort_mb_default
hasWorkloadPattern	DataSummary

Figura 7.1: Resultado de consulta SPARQL no ambiente Protégé.

ontologia está correta e as regras bem implementadas. Isto significa que o objetivo foi cumprido e a consistência da ontologia comprovada.

Um primeiro exemplo de inferência é apresentado através da Figura 7.2. Esta inferência diz respeito a obtenção do padrão de carga de trabalho de acordo com os dados do *Job*. As regras levam em conta os dados de entrada, intermediários e de saída para se fazer a análise e inferir qual o padrão de carga de trabalho daquele do *Job* em questão.

The screenshot displays the Protégé interface with the following components:

- Members list (inferred):** A list on the left showing 'job1', 'job2' (highlighted), and 'job3'.
- Annotations:** A section for 'Annotations: job2' with a search bar.
- Rules:** A section containing several rules for *JobData*, such as:
 - `JobData(?x), durationJobSec(?x, ?dj), greaterThan(?dj, 1200), notEqual(?dj, 0) -> hasSpeed(?x, Slow)`
 - `JobData(?x), sizeInputJobKb(?x, ?i), sizeOutputJobKb(?x, ?o), greaterThan(?i, ?oo), multiply(?oo, ?o, 3), notEqual(?o, 0) -> hasWorkloadPattern(?x, DataSummary)`
 - `JobData(?x), durationJobSec(?x, ?dj), lessThanOrEqual(?dj, 1200), notEqual(?dj, 0) -> hasSpeed(?x, Fast)`
 - `JobData(?x), sizeInputJobKb(?x, ?i), sizeOutputJobKb(?x, ?o), sizeShuffleJobKb(?x, ?s), greaterThan(?i, ?s), greaterThan(?s, ?o), lessThan(?i, ?oo), multiply(?oo, ?o, 3.0), notEqual(?o, 0) -> hasWorkloadPattern(?x, Aggregate_Fast)`
- Description: job2:** A section for the class description of 'job2'.
- Property assertions: job2:** A section showing inferred assertions:
 - Object property assertions:**
 - `hasWorkloadPattern DataSummary`
 - `hasSpeed Fast`
 - Data property assertions:**
 - `durationJobSec 906.0`
 - `maptimeJobSec 104338.0`
 - `sizeOutputJobKb 491141.0`
 - `sizeShuffleJobKb 8776400.0`
 - `reducetimeJobSec 66760.0`
 - `sizeInputJobKb 2.30903926E8`

Figura 7.2: Resultado de inferência via Pellet para o "Job2".

Nesta figura, observamos que o Pellet classifica o "Job2" (em destaque na janela à esquerda) tem como *Object property assertions* duas propriedades inferidas:

- A primeira é a propriedade *hasWorkloadPattern* que define o *Job2* com sendo do padrão *DataSummary*. Este padrão foi inferido de acordo com a regra SWRL implementada na ontologia. A regra baseia-se na classificação proposta no Capítulo 4, na Seção 4.3.4. Abaixo é listada a regra que inferiu o padrão:

```
1 JobData(?x), sizeInputJobKb(?x, ?i), sizeOutputJobKb(?x, ?o),
    greaterThan(?i, ?oo), multiply(?oo, ?o, 3), notEqual(?o, 0) ->
    hasWorkloadPatternJ(?x, DataSummary)
```

São verificados os valores de entrada, intermediário e de saída do *job*.

- A segunda é a propriedade *hasSpeed* que define o *Job2* com sendo do tipo *Fast*. Este tipo de velocidade foi inferido de acordo com a regra SWRL implementada na ontologia. Abaixo é listada a regra que inferiu o padrão:

```
1 JobData(?x), durationJobSec(?x, ?dj), lessThanOrEqual(?dj, 1200),
    notEqual(?dj, 0) -> hasSpeedJ(?x, Fast)
```

São verificados os valores de entrada, intermediário e de saída do *job*.

7.1.3 Análise dos Resultados

Como os resultados preliminares obtidos com este trabalho, e com a análise da classificação da ontologia, seus conceitos, propriedades, axiomas e regras, verifica-se que a ontologia proposta e implementada torna-se essencial para explicitar e observar o cumprimento dos objetivos iniciais, tal como a classificação dos parâmetros de configuração do Hadoop e o papel que eles exercem na criação das Unidades de Sintonia.

Na verdade, a implementação inicial da ontologia já demonstra que sua funcionalidade é plausível. No entanto, a ontologia ainda carece de refinamentos e, principalmente, da implementação de todas as regras que podem inferir padrões de cargas de trabalho e todas as regras para cada um dos parâmetros de configuração do Hadoop.

Neste experimento inicial, foram implementadas apenas algumas regras como forma de ratificar a funcionalidade da ontologia em direção ao seu propósito maior. Além disso, faz-se necessário esclarecer que os experimentos foram realizados de forma manual. Outrossim, quando da automatização do ambiente semântico proposto, é possível a utilização de APIs para Java (tais com Jena² e OWL API³ que podem automatizar todo o processo de consultas e a integração com outros módulos em Java. Através de sua API, Jena tem como objetivo prover uma interface de programação consistente para o desenvolvimento de aplicações baseadas em ontologia.

7.2 SIMULAÇÕES

Outra avaliação que deve ser provida, é a avaliação da proposta da abordagem semântica baseada em ontologia. Nesta seção será apresentada uma simulação completa da criação e utilização de uma Unidade de Sintonia de parâmetros.

Para exemplificar o processo, este exemplo seguirá os passos do fluxo dos processos para geração de unidades de sintonia apresentado no Capítulo 6, na Seção 6.3.

Notas: Um processo de criação ou adaptação de uma Unidade de Sintonia para um determinado *job* inicia-se a partir da submissão de um *job* para análise. Dois pressupostos devem ser analisados em primeira instância:

²<https://jena.apache.org/>

³<http://owlapi.sourceforge.net/>

- Primeiro: Partindo do pressuposto que um usuário submete um *job*, sem nunca ter sido executado anteriormente. Neste caso, são necessárias duas informações: o tamanho da entrada de dados e o padrão da carga de trabalho (*Load Data, Aggregate, Data Summary*, etc).
- Segundo: Partindo do pressuposto que o usuário submete um *job* através do *log* histórico de alguma execução anterior. Neste caso, já é possível a extração, através de um *parser*, das informações dos dados de entrada, intermediários, saída e tempo de execução total do *job*.

Neste exemplo, seguiremos os passos partindo do primeiro pressuposto citado acima. Os passos a seguir demonstram o processo:

1. O usuário submete o *job* com as informações iniciais. Vamos chamar este *job* de instância *Job3*. Se necessário, quando da implementação do ambiente semântico, pode ser implementada uma interface de usuário que permita ao usuário inserir as informações através de um arquivo XML, por exemplo, consultar ou ainda inserir um *log* do histórico de execução.
2. Após submetido ao repositório de *jobs* (em nossa simulação, o repositório de *jobs* é a própria ontologia (OntoHadoop), as regras SWRL são executadas de modo a inferir qual o padrão da carga de trabalho. No entanto, a inferência não pode ser realizada, já que o *job* não possui informações necessárias para que sejam executadas. Observe-se na Figura 7.3 que o *Job3* não possui nenhuma inferência na seção *Object property assertions* e somente a propriedade *sizeInputJobKb* possui informação. Sendo assim, é criada uma Unidade de Sintonia *default* para ser executada pelo simulador, e assim, obter maiores informações sobre o *Job3*;

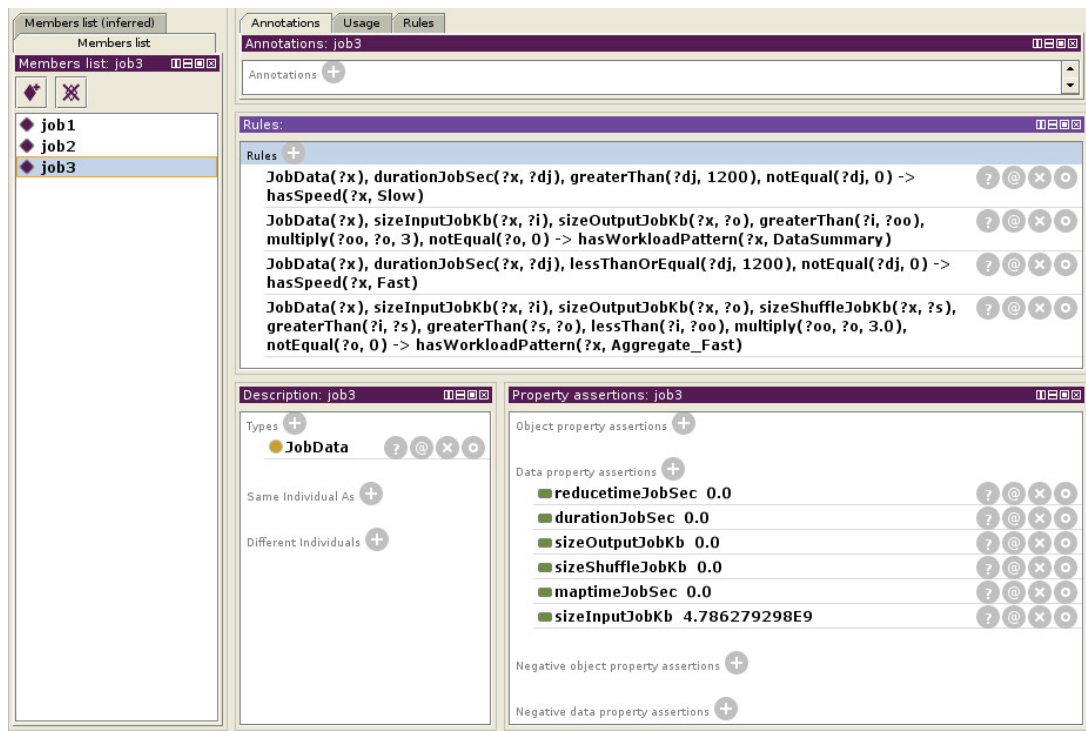


Figura 7.3: Informações de entrada do "Job3".

3. Após executar o *Job3*, com uma configuração *default* para os parâmetros, um *parser* lê os dados de saída gerados pelo simulador (listado abaixo) e instância a ontologia com as informações geradas;

```
Name:Job3_1
0,00 JsonJob:
Job3
Job Counters:
DURATION 1.734,928
FILE_BYTES_READ 11.442.480.896,
FILE_BYTES_WRITTEN 12.576.480.896,
HDFS_BYTES_READ 478.627.929,
HDFS_BYTES_WRITTEN 989.236,112
JOB_START_TIME 1,
JOB_STOP_TIME 901,
JOB_TOTAL_TIME 900,
MAP_INPUT_RECORDS 1.620.000,
MAP_INPUT_SIZE 478.627.929,
MAP_OUTPUT_BYTES 1.134.000.000,
MAP_OUTPUT_RECORDS 81.000.000,
REDUCE_INPUT_RECORDS 81.000.000,
REDUCE_OUTPUT_RECORDS 100.035,
REDUCE_SHUFFLE_BYTES 1.134.000.000,
SHUFFLE 1.134.000.000,
SPILLED_RECORDS 898.320.064,
START_TIME 848,
STOP_TIME 2.582,928
avMappersTime 844,809
avReducersTime 45,31
```

4. As regras SWRL são executadas de modo a inferir qual o padrão da carga de trabalho. Neste caso, a inferência determina que o padrão de carga de trabalho (propriedade *hasWorkloadPattern*) é *DataSummary* e a velocidade (propriedade *hasSpeed*) é *Slow*, conforme podemos observar na Figura 7.4;
5. Novamente se faz uma busca no repositório de configurações de *job* (ontologia, neste caso) para verificar se há configurações similares com o padrão inferido do *Job3*. Neste caso, verifica-se por inferência, que duas Unidades de Sintonia utilizam o padrão de carga de trabalho *DataSummary*, tal como o *Job3*. Observe na Figura 7.5 que o raciocinador, através da regra em destaque na janela *Rules*, identifica as Unidades de Sintonia *unit1* e *unit3*, inferidas através da propriedade *hasUnitTuning*;
6. Um *parser* gera os dados de entrada do simulador e gera as configurações dos parâmetros para serem simuladas. As configurações para a simulação são apresentadas na Tabela 7.1 abaixo:

As configurações apresentadas na tabela, indicam por ordem:

Members list (inferred)

Members list

Members list: job3

job1
job2
job3

Annotations: job3

Annotations +

Rules:

Rules +

JobData(?x), sizeInputJobKb(?x, ?i), sizeOutputJobKb(?x, ?o), greaterThan(?i, ?oo), multiply(?oo, ?o, 3), notEqual(?o, 0) -> hasWorkloadPatternJ(?x, DataSummary)

JobData(?x), durationJobSec(?x, ?dj), lessThanOrEqual(?dj, 1200), notEqual(?dj, 0) -> hasSpeedJ(?x, Fast)

JobData(?x), durationJobSec(?x, ?dj), greaterThan(?dj, 1200), notEqual(?dj, 0) -> hasSpeedJ(?x, Slow)

JobData(?x), sizeInputJobKb(?x, ?i), sizeOutputJobKb(?x, ?o), sizeShuffleJobKb(?x, ?s), greaterThan(?i, ?s), greaterThan(?s, ?o), lessThan(?i, ?oo), multiply(?oo, ?o, 3.0), notEqual(?o, 0) -> hasWorkloadPatternJ(?x, Aggregate_Fast)

Description: job3

Types +

JobData

Same Individual As +

Different Individuals +

Property assertions: job3

Object property assertions +

hasSpeedJ Slow

hasWorkloadPatternJ DataSummary

Data property assertions +

reducetimeJobSec 0.0

sizeOutputJobKb 989236.12

sizeShuffleJobKb 0.0

maptimeJobSec 0.0

durationJobSec 1734.92

sizeInputJobKb 4.78627929E8

Figura 7.4: Inferências para o "Job3".

Members list (inferred)

Members list

Members list: job3

job1
job2
job3

Annotations: job3

Annotations +

Rules:

Rules +

JobData(?x), sizeInputJobKb(?x, ?i), sizeOutputJobKb(?x, ?o), greaterThan(?i, ?oo), multiply(?oo, ?o, 3), notEqual(?o, 0) -> hasWorkloadPatternJ(?x, DataSummary)

JobData(?x), HadoopUnitTuning(?z), hasWorkloadPatternJ(?x, ?y), hasWorkloadPattern(?z, ?y) -> hasUnitTuning(?x, ?z)

JobData(?x), durationJobSec(?x, ?dj), lessThanOrEqual(?dj, 1200), notEqual(?dj, 0) -> hasSpeedJ(?x, Fast)

JobData(?x), durationJobSec(?x, ?dj), greaterThan(?dj, 1200), notEqual(?dj, 0) -> hasSpeedJ(?x, Slow)

Description: job3

Types +

JobData

Same Individual As +

Different Individuals +

Property assertions: job3

Object property assertions +

hasUnitTuning unit3

hasUnitTuning unit1

hasSpeedJ Slow

hasWorkloadPatternJ DataSummary

Data property assertions +

reducetimeJobSec 0.0

sizeOutputJobKb 989236.12

sizeShuffleJobKb 0.0

maptimeJobSec 0.0

durationJobSec 1734.92

sizeInputJobKb 4.78627929E8

Figura 7.5: Inferências de configurações similares para o "Job3".

Tabela 7.1: Parâmetros de Configuração para Simulação.

Parâmetros	Config1	Config2	Config3
dfs_replication	3	3	3
io_sort_factor	10	32	32
io_sort_mb	100	320	320
io_sort_record_percent	0,05	0,05	0,05
io_sort_spill_percent	0,8	0,8	0,8
mapred_child_jav_opts	200	512	512
mapred_inmem_merge_threshold	1000	1000	1000
mapred_job_reduce_input_buffer_percent	0	0	0
mapred_job_shuffle_input_buffer_percent	0,7	0,7	0,7
mapred_job_shuffle_merge_percent	0,66	0,66	0,66
mapred_reduce_parallel_copies	5	10	10
mapred_map_tasks	2	30	30
mapred_reduce_tasks	1	3	3
mapreduce_combine_class	false	false	true
mapred_compress_map_output	false	false	true

- **Config1:** Esta é a configuração *default* utilizada na primeira execução. O objetivo é extrair as informações da execução do *Job3* com o valores *default* dos parâmetros do Hadoop;
- **Config2:** Esta configuração é a inferida através de regras, de acordo com o padrão de carga de trabalho inferido. O Gerador de Unidades de Sintonia, gera no mínimo duas configurações para serem analisadas pelo simulador. Parâmetros com *io_sort_factor* (indica o número de partições de saída *map* que farão o *merge* ao mesmo tempo) e *io_sort_mb* (e espaço de buffer de memória em Mb utilizado para o *sort* dos fluxos de dados na fase intermediária) e *mapred_child_jav_opts* (quantidade de memória utilizada pelos *workers* para executar as tarefas), são alteradas e aumentadas em relação ao valor *default* pela característica do padrão de carga de trabalho. Ou seja, mais desempenho deve ser obtido através da redução do número *spills* no disco, com o aumento dos valores que lidam com memória, aumento o valor do parâmetro *io_sort_mb*, por exemplo. Além do aumento no número de tarefas *map* e *reduce* para uma melhor otimização da execução destas duas tarefas nos *workers* do Hadoop;
- **Config3:** A diferença desta configuração para a anterior é que os parâmetros *mapreduce_combine_class* e *mapred_compress_map_output* são configurados com o valor **true**. Isto significa, que no caso de um padrão de carga de trabalho do tipo *DataSummary*, que tem um comportamento similar a conhecida aplicação *WordCount*, muitos dados intermediários serão classificados, escritos, lidos, etc. Sendo assim, as regras de ouro para sintonia de parâmetros apontam que cargas de trabalho com este perfil, devem ser utilizados as funções *combine* e *compress*. Os resultados simulados apontam uma grande diferença nos tempos de execução quando estas funções são utilizadas nos dados intermediários.

Abaixo são listados um exemplo das entradas do simulador com os parâmetros que devem ser sintonizados de acordo com as configurações apresentadas na Tabela 7.1:

```
1 | "ioSortFactor":10,
```

```

2  "ioSortMb":100.0,
3  "ioSortRecordPercent":0.05,
4  "ioSortSpillPercent":0.8,
5
6  "mapredChildJavaOpts":200,
7  "mapredInmemMergeThreshold":1000,
8  "mapredJobReduceInputBufferPercent":0.0,
9  "mapredJobShuffleInputBufferPercent":0.7,
10 "mapredJobShuffleMergePercent":0.66,
11 "mapReduceParallelCopies":5,
12 "memoryLimit":1.4138736E8,
13
14 "numberOfMappers":60,
15 "numberOfReducers":1,
16
17 "useCombiner":false,
18 "useCompression":false
19 }

```

7. O simulador executa o *job* de acordo com as configurações acima e o resultado da execução é analisado e apresentado ao usuário. A execução do simulador se dá, neste ambiente simulado, através da linha de comando abaixo:

```

1  java -jar ontosim.jar "/home/ailton/Dados_Linux/Tese/Texto_Tese/
    OntoSim/jobs/job_1.json" "/home/ailton/Dados_Linux/Tese/
    Texto_Tese/OntoSim/rack" 2>out.csv

```

O primeiro parâmetro o arquivo *Json* que contém as configurações do *job*, o segundo o diretório do *rack* (33), a saída vai para o "out.csv". A saída contém o cabeçalho contendo os parâmetros de entrada e alguns dados de de saída da execução do *job*.

O arquivo de configuração do *rack* segue o padrão determinado pelo seu criador (33):

```

1  { "machines" : [
2
3      { "baudRate" : 70000000.0,
4        "cpu" : { "cores" : 4,
5                  "speed" : 500000000.0
6        },
7        "hardDisk" : { "capacity" : 40000.0,
8                      "read" : 40000000.0,
9                      "seekTime" : 1.0,
10                     "write" : 20000000.0
11          },
12        "hostName" : "m1",
13
14        "maxMapper" : 10,
15        "maxReducer" : 5
16      }
17    %% Definem-se quantas máquinas forem desejadas no cluster. Este
18    %% exemplo mostra apenas uma máquina (m1)
19
20    ],
21    "router" : "r_01",
22    "heartbeat":1.0,
23    "propDelay":1.0,
24    "maxIM":60000,
25    "deltaCPU":1000000.0,

```

```

25     "deltaHDD":1000000.0,
26     "deltaNEt":1000000.0,
27
28     "flowType":false,
29     "hlogLevel":"info"
30 }

```

Vale ressaltar que em nossa simulação estamos usando a configuração padrão do simulador que sugere um cluster de 3 máquinas com as configurações acima. Não estamos levando em conta, nesta simulação de ambiente, o critério de tamanho e disponibilidade de recursos. Sempre lembrando que todas as informações obtidas por este experimento levam a indicativos da melhor sintonia dos parâmetros e não resultados realísticos.

8. Após simuladas as configurações, as saídas são lidas, analisadas e apresentadas ao usuário. Então, o usuário, pode ou não interferir na Unidade de Sintonia criada. Se não houver interferência, a Unidade de Sintonia é criada levando-se em conta a configuração com os tempos de execução que foram mais baixos. Se houver interferência, o usuário pode gerar ou modificar os valores das Unidades de Sintonia, simular novamente e obter novos valores para serem analisados. Quando o usuário termina este processo e ratifica uma Unidade de Sintonia, ela a configura com *approved by expert*. A Figura 7.6 apresenta os tempos de execução para cada uma das configurações executadas.

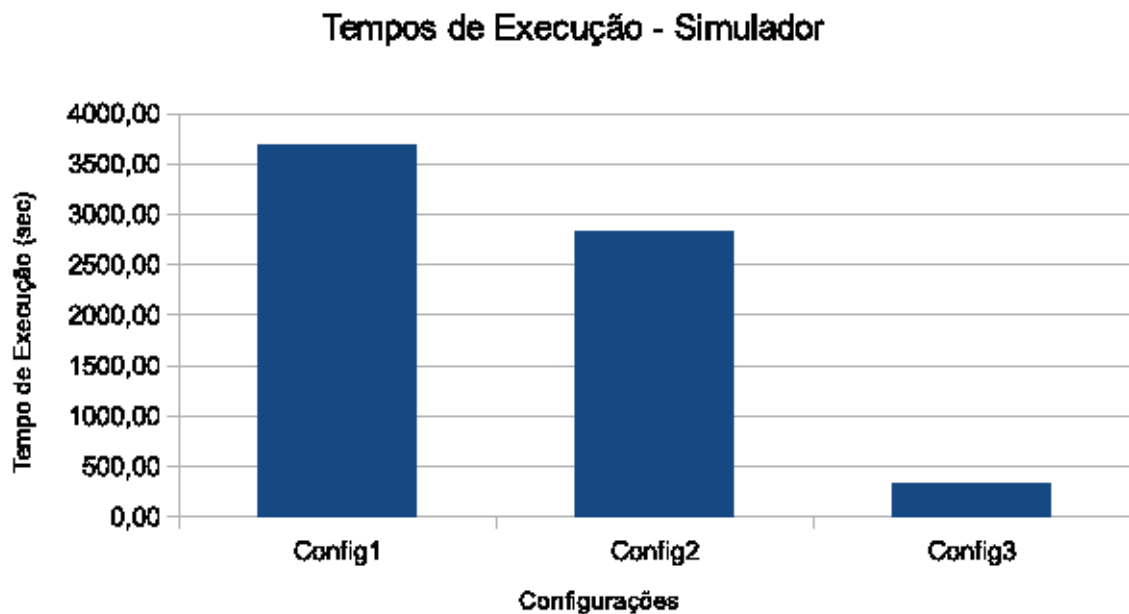


Figura 7.6: Tempos Total de execução do "Job3" em segundos por configuração.

Observa-se que há uma diferença de desempenho entre a *Config1* (default) e a *Config2*, prevalecendo um melhor desempenho para a configuração sintonizada. No entanto, quando se usa as funções *useCombiner: true* e *useCompression:true*, o desempenho se torna muito melhor. A função *combiner* agrupa registros intermediários na memória,

antes de escrevê-los no disco local. Por outro lado utilizam mais memória virtual nas tarefas, porém diminuem as operações de IO.

Como nota final da experimentação, observa-se que o simulador não serve apenas para a etapa final de análise da Unidade de Sintonia. Em um processo anterior, o simulador serve como forma de executar um *job* nunca antes executado e extrair informações básicas baseadas nos parâmetros com valores *default*.

7.3 CONSIDERAÇÕES FINAIS

Este capítulo apresentou alguns experimentos no sentido de experimentar a capacidade da ontologia em prover o conhecimento a respeito dos parâmetros de configuração do Hadoop, bem como o seu poder de estabelecer inferências para a aquisição de padrões de cargas de trabalho e sintonia dos parâmetros relacionados a estes padrões.

Foram apresentadas algumas consultas em SPARQL para demonstrar as questões de competência da ontologia proposta. Além disso, algumas inferências foram propostas através do raciocionador Pellet, com o objetivo de provar a capacidade que a ontologia oferece em inferir o conhecimento necessário para o que ela foi proposta.

Além disso, foi simulado um ambiente semântico em direção a um ciclo completo de criação das Unidades de Sintonia. Para isso foi utilizado um simulador já ratificado na comunidade científica chamado MRSim. Os resultados da simulação, através dos parâmetros sintonizados, mostrou um ganho significativo de desempenho em relação aos valores *default* dos parâmetros de configuração.

Os resultados apresentados ratificam o propósito de uma base de conhecimento para melhor entendimento de todas as nuances que envolvem a sintonia dos parâmetros do Hadoop. Assim, se mostram promissores no sentido de se alcançar resultados mais significativos, de acordo com a completa implementação das regras, para a sintonia dos parâmetros e criação das Unidades de Sintonia.

8 CONSIDERAÇÕES FINAIS

Este capítulo faz as considerações finais em relação a este trabalho e posiciona-se quanto a sua relevância, contribuições, limitações e trabalhos futuros.

8.1 RELEVÂNCIA

Este trabalho apresentou a proposta de uma abordagem semântica baseada em ontologias com o objetivo de proporcionar uma nova alternativa junto às propostas atuais, para se obter a melhor sintonia dos parâmetros de configuração do Hadoop MapReduce, em direção a um melhor desempenho das aplicações a ele submetidas.

Por se tratar de um assunto relativamente novo, o assunto da sintonia de parâmetros do Hadoop MapReduce torna-se bastante carente de soluções que discutam sobre técnicas para predição da melhor definição destes parâmetros de configuração associados às cargas de trabalho e suas características específicas. Também é de grande relevância a necessidade de difundir, para a comunidade usuária deste *framework*, o conhecimento sobre este assunto, bem como prover uma base de conhecimento modelada com este objetivo. Outrossim, até onde se sabe, não há nenhum trabalho realizado que aborde o conhecimento semântico com base ontológica para esta finalidade.

Como apontada na revisão sistemática apresentada por este trabalho, as pesquisas mais recentes, em sua maioria, apontam para métodos de experimentações empíricas, abordagens analíticas, *benchmarks* e simulações para propor meios de se avaliar e predizer a sintonia dos parâmetros do Hadoop. Para isso, alguns desses trabalhos são baseados na proposição de regras, outros são baseadas em histórico das execuções anteriores para caracterizar as aplicações, etc. No entanto, nenhum deles classifica, explicitamente, os parâmetros de configuração em relação as fases críticas para cada um deles. Além disso, não propõem nenhuma taxonomia para o melhor entendimento das características de cada um destes parâmetros, e não compartilham o conhecimento que está embutido em suas soluções. Dessa forma, este conhecimento não é disseminado para outros usuários e nem permite que os usuários mais experientes compartilhem seu conhecimento sobre o assunto, e nem aos inexperientes conhecerem mais sobre o assunto. Outrossim, nenhum dos trabalhos revisados propõe uma solução, ou melhor compreensão, a respeito de todos os parâmetros que influenciam no desempenho do Hadoop. Por fim, nenhum desses trabalhos, já apresentados na comunidade científica, baseiam suas soluções através da utilização de ontologias.

A proposta apresentada nesta tese, propõe uma abordagem promissora no sentido de cobrir estas lacunas. A abordagem semântica proposta utiliza algumas técnicas já empregadas em outros trabalhos, porém de uma forma que extrai o melhor de cada uma delas, propondo uma solução mais ampla, que seja útil para usuários experientes ou não no uso do Hadoop, quanto a sintonia de seus parâmetros de configuração em direção a obtenção de um melhor desempenho do sistema.

Os experimentos apresentados através de inferências na base de conhecimento ontológica, aliadas a simulação executada, já indicam que a abordagem direciona resultados significativos no tocante a sintonia de parâmetros do Hadoop MapReduce. Assim, pode auxiliar os usuários deste *framework* na tarefa árdua de entender e obter uma boa sintonia dos parâmetros em direção a um melhor desempenho, bem como segue rumo as tecnologias que servem à Web Semântica.

8.2 CONTRIBUIÇÕES DA PESQUISA

As contribuições mais relevantes desta pesquisa são enumeradas a seguir:

1. Revisão Sistemática de Literatura:

Foi realizada uma inédita revisão sistemática de literatura. Esta revisão investigou, na literatura científica mais recente, os trabalhos que abordam a sintonia dos parâmetros do Hadoop MapReduce para identificar quais parâmetros são os mais relevantes em relação ao desempenho do sistema. Além disso, foram pesquisadas as abordagens utilizadas nestes trabalhos, seus métodos de experimentação e validação. O objetivo era responder às seguintes perguntas: (1) Quais parâmetros de configuração do Hadoop têm influências e impactos no desempenho do sistema? (2) Quais parâmetros são influenciados pelas fases do Hadoop? (3) Quais parâmetros são influenciados pelas características das cargas de trabalho?

Através dessas respostas, foi possível propor uma classificação dos parâmetros relevantes em relação a sua influência nas fases de execução do Hadoop e suas relações com as características das cargas de trabalho. A revisão permitiu, ainda, a apresentação das principais regras para a melhor sintonia de cada um dos parâmetros identificados, e as principais regras para caracterização dos padrões das cargas de trabalho segundo as características dos seus dados de entrada, intermediários e de saída.

Esta revisão sistemática contribuiu, ainda, para fornecer várias referências relevantes que servirão de fonte de consulta para trabalhos futuros voltados a este assunto.

2. OntoHadoop:

Desenvolvimento de uma inédita ontologia para o Hadoop MapReduce. A ontologia criada teve maior ênfase na classificação dos parâmetros de configuração do Hadoop MapReduce que influenciam no desempenho das aplicações. A ontologia foi implementada com as classes, relações e axiomas referentes aos parâmetros de configuração do Hadoop e sua taxonomia, bem como a modelagem das características dos padrões das cargas de trabalho. Além disso, a ontologia permite a implementação de regras para a sintonia dos parâmetros de configuração de acordo com o padrão das cargas de trabalho. O conhecimento utilizado para a modelagem da ontologia vem da revisão sistemática e permite a disseminação do conhecimento sobre este assunto.

Esta ontologia visa fornecer suporte semântico para a construção de um ambiente computacional que possa alcançar a sintonia dos parâmetros do Hadoop e melhorar o desempenho das aplicações Hadoop.

Além disso, as regras que foram implementadas, e que podem ser estendidas, são fundamentais para se obter o perfil das cargas de trabalho e inferir os melhores parâmetros de configuração de acordo com este perfil.

3. Abordagem Semântica baseada em Ontologia

A contribuição mais relevante desta pesquisa é a proposição de uma inédita abordagem semântica baseada em ontologias para a sintonia de parâmetros do Hadoop, que foi a proposta inicial de deste trabalho. Esta abordagem segue em direção a um ambiente computacional de inferência semântica para a sintonia dos parâmetros, incluindo os módulos de classificação das cargas de trabalho, gerador de unidades de sintonia e

avaliador de sintonia através de simulações, para comparação do desempenho sintonizado em relação aos valores padrão dos parâmetros.

Essa abordagem é contemplada pela base de conhecimento (OntoHadoop), *parsers*, simulador e, caso necessário, técnicas de aprendizado de máquina. A abordagem se mostra modular através de sua arquitetura, a ponto de se desenvolver um ambiente de acordo com as tecnologias que se tem disponíveis. Pode ser usado um simulador para se avaliar as Unidades de Sintonia geradas, bastando apenas a adaptação do *parser* que extrai informações da ontologia e que o alimenta. Além disso, pode ser usada qualquer técnica de aprendizado para gerar novas populações de Unidades de Sintonia. Para isso, deve-se adaptar o *parser* que gera os dados de entrada para a técnica que for utilizada.

Essa flexibilidade, transforma a abordagem em um modelo bastante adaptativo para a sintonia de parâmetros. Até mesmo a base de conhecimento pode ser reusada e melhorada através das necessidades que forem surgindo e novas regras de sintonia e perfil de cargas de trabalho serem necessárias.

8.3 LIMITAÇÕES E TRABALHOS FUTUROS

Este trabalho propôs e apresentou uma abordagem alternativa baseada em ontologias para a sintonia dos parâmetros que podem melhorar o desempenho das aplicações Hadoop MapReduce.

Comparado as tradicionais abordagens, a proposta que este trabalho apresenta, agrega um pouco de cada uma destas tecnologias e adiciona o fator semântico. Isso a torna uma alternativa perfeitamente viável para a disseminação do conhecimento sobre esse assunto. No entanto, apesar de ser uma abordagem bastante promissora neste sentido, algumas lacunas ainda permanecem em aberto.

Uma grande limitação da abordagem diz respeito a dependência que ela possui em relação aos usuários especialistas. É fato que a abordagem depende do conhecimento prévio sobre o assunto do usuário que irá implementar as regras de sintonia para os parâmetros de configuração. Torna-se, então, uma abordagem colaborativa, que parte do princípio básico do compartilhamento do conhecimento. Dessa forma, os usuários comuns podem fazer proveito das vantagens que esta abordagem se compromete.

O trabalho pode ser estendido e melhorado em algumas direções, tanto no aspecto do modelo ontológico, quanto na abordagem semântica. A seguir é apresentada alguns trabalhos futuros.

Quanto ao modelo ontológico:

- Extensão das regras para determinar o padrão das cargas de trabalho;
- Extensão das regras para cada um dos parâmetros de configuração identificados, permitindo gradativamente uma melhor sintonia;
- Avaliação constante do modelo ontológico, e seu reuso por outros usuários, bem como trabalhos futuros no sentido de estender a ontologia, de forma a contemplar de maneira mais eficiente os recursos utilizados pelas aplicações;
- Criação de um método automatizado para a instanciar a ontologia, de forma que o maior números de usuários possível possa submeter *jobs* e *logs* de execuções. Também é sabido que, nesta abordagem, quanto mais informações a ontologia contiver, cada vez mais as sintonias podem ser refinadas.

Quanto a abordagem semântica em um aspecto geral:

- Realizar e produzir experimentos realísticos para que a abordagem seja totalmente ratificada. Porém, é sabido que grandes volumes de dados e grandes *clusters*, bem como a disponibilidade de *traces* de execução, são limitados a grandes empresas ou grandes instituições de pesquisa, sendo de difícil acesso para os usuários em geral;
- Implementação de todos os módulos da abordagem, a fim de se criar efetivamente um ambiente computacional integrado e automatizado;
- Definir um módulo específico, baseado em técnicas de aprendizado de máquina, para a caracterização e perfilização dos padrões das cargas de trabalho, bem como a relação destes padrões com os parâmetros de configuração do Hadoop.

REFERÊNCIAS

- [1] Abburu, S. (2012). A survey on ontology reasoners and comparison. *International Journal of Computer Applications*, 57.
- [2] Ahmad, F., Chakradhar, S. T., Raghunathan, A. e Vijaykumar, T. N. (2012). Tarazu: Optimizing mapreduce on heterogeneous clusters. *SIGARCH Comput. Archit. News*, 40(1):61–74.
- [3] Almeida, M. B. e Bax, M. P. (2003). Uma visão geral sobre ontologias: pesquisa sobre definições, tipos, aplicações, métodos de avaliação e de construção. *Ciência da Informação*, 32:7 – 20.
- [4] An, M., Wang, Y. e Wang, W. (2010). Using index in the mapreduce framework. Em *Proceedings of the 2010 12th International Asia-Pacific Web Conference, APWEB '10*, páginas 52–58, Washington, DC, USA. IEEE Computer Society.
- [5] Apache (2012a). Apache hadoop. <http://hadoop.apache.org/>. October 24, 2012.
- [6] Apache (2012b). Hdfs architecture guide. <http://hadoop.apache.org/docs/hdfs/current/index.html>.
- [7] Baader, F., Horrocks, I. e Sattler, U. (2008). Description Logics. Em van Harmelen, F., Lifschitz, V. e Porter, B., editores, *Handbook of Knowledge Representation*, capítulo: 3, páginas 135–180. Elsevier.
- [8] Babu, S. (2010). Towards automatic optimization of mapreduce programs. Em *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10*, páginas 137–142, New York, NY, USA. ACM.
- [9] Berners-Lee, T. e Fischetti, M. (1999). *Weaving the web: The original design and ultimate destiny of the world wide web by its inventor*. Harper, San Francisco.
- [10] Berners-Lee, T., Hendler, J. e Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.
- [11] Boley, H. (2006). The ruleml family of web rule languages. Em Alferes, J. J., Bailey, J., May, W. e Schwertel, U., editores, *PPSWR*, volume 4187 de *Lecture Notes in Computer Science*, páginas 1–17. Springer.
- [12] Bonifacio, A. S., Menolli, A. e Silva, F. (2014). Hadoop mapreduce configuration parameters and system performance: a systematic review. Em *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2014)*, Las Vegas, NV, USA.
- [13] Bortnikov, E., Frank, A., Hillel, E. e Rao, S. (2012). Predicting execution bottlenecks in map-reduce clusters. Em *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing, HotCloud'12*, páginas 18–18, Berkeley, CA, USA. USENIX Association.
- [14] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Eve e Yergeau, F., editores (2003). *Extensible Markup Language (XML) 1.0*. W3C Recommendation. W3C, fourth edition.

- [15] Brewster, C., Alani, H., Dasmahapatra, S. e Wilks, Y. (2004). Data driven ontology evaluation. Em *International Conference on Language Resources and Evaluation*. Event Dates: 24-30 May.
- [16] Brickley, D. e Guha, R. V. (2004). Rdf vocabulary description language 1.0: Rdf schema. Relatório técnico.
- [17] Chen, Y., Ganapathi, A., Griffith, R. e Katz, R. (2011). The case for evaluating mapreduce performance using workload suites. Em *Proceedings of the 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS '11*, páginas 390–399, Washington, DC, USA. IEEE Computer Society.
- [18] Chen, Y., Ganapathi, A. S., Griffith, R. e Katz, R. H. (2010). A methodology for understanding mapreduce performance under diverse workloads. Relatório Técnico UCB/EECS-2010-135, EECS Department, University of California, Berkeley.
- [19] Damme, C. V., Hepp, M. e Siorpaes, K. (2007). Folksonomy: An integrated approach for turning folksonomies into ontologies. Em *IN PROCEEDINGS OF THE ESWC WORKSHOP BRIDGING THE GAP BETWEEN SEMANTIC WEB AND WEB 2.0*. Springer.
- [20] Dancy, J. e Sosa, E. (1994). *A Companion to Epistemology*. Blackwell Companions to Philosophy. Wiley.
- [21] Davies, J., Fensel, D. e van Harmelen, F., editores (2003). *Towards the Semantic Web: Ontology-driven Knowledge Management*. Wiley, Chichester, UK.
- [22] Dean, J. e Ghemawat, S. (2004). Mapreduce: simplified data processing on large clusters. Em *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6, OSDI'04*, páginas 10–10, Berkeley, CA, USA. USENIX Association.
- [23] Decker, S., Melnik, S., Harmelen, F. V., Fensel, D., Klein, M., Broekstra, J., Erdmann, M. e Horrocks, I. (2000). The semantic web: The roles of xml and rdf.
- [24] Dentler, K., Cornet, R., ten Teije, A. e de Keizer, N. (2011). Comparison of reasoners for large ontologies in the owl 2 el profile. *Semant. web*, 2(2):71–87.
- [25] Do, A. V., Chen, J., Wang, C., Lee, Y. C., Zomaya, A. e Zhou, B. B. (2011). Profiling applications for virtual machine placement in clouds. Em *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, páginas 660 –667.
- [26] Elteir, M., Lin, H. e chun Feng, W. (2010). Enhancing mapreduce via asynchronous data processing. Em *ICPADS*, páginas 397–405. IEEE.
- [27] Falbo, R. A. (2004). Experiences in using a method for building domain ontologies. Em *16th International Conference on Software Engineering and Knowledge Engineering (SEKE 2004), International Workshop on Ontology in Action (OIA 2004)*, páginas 474–477, Alberta, Canada.
- [28] Fernandez-Lopez, M., Gomez-Perez, A. e Juristo, N. (1997). Methontology: from ontological art towards ontological engineering. Em *Proceedings of the AAAI97 Spring Symposium*, páginas 33–40, Stanford, USA.

- [29] Gruber, T. R. (1993a). Toward principles for the design of ontologies used for knowledge sharing. Em *International Journal of Human-Computer Studies*, páginas 907–928. Kluwer Academic Publishers.
- [30] Gruber, T. R. (1993b). A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220.
- [31] Guarino, N. e Giaretta, P. (1995). Ontologies and Knowledge Bases: Towards a Terminological Clarification. *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, páginas 25–32.
- [32] Guo, Z., Fox, G., Zhou, M. e Ruan, Y. (2012). Improving resource utilization in mapreduce. Em *CLUSTER*, páginas 402–410. IEEE.
- [33] Hammoud, S., Li, M., Liu, Y., Alham, N. e Liu, Z. (2010). Mrsim: A discrete event based mapreduce simulator. Em *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on*, volume 6, páginas 2993–2997.
- [34] Han, J., Ishii, M. e Makino, H. (2013). A hadoop performance model for multi-rack clusters. Em *Computer Science and Information Technology (CSIT), 2013 5th International Conference on*, páginas 265–274.
- [35] Heger, D. (2013). Hadoop performance tuning - a pragmatic & iterative approach. *CMG Journal*.
- [36] Herodotou, H. e Babu, S. (2011). Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs. *PVLDB: Proceedings of the VLDB Endowment*, 4(11):1111–1122.
- [37] Herodotou, H., Dong, F. e Babu, S. (2011a). Mapreduce programming and cost-based optimization? crossing this chasm with starfish. *PVLDB*, 4(12):1446–1449.
- [38] Herodotou, H., Dong, F. e Babu, S. (2011b). No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. Em *Proceedings of the 2Nd ACM Symposium on Cloud Computing, SOCC '11*, páginas 18:1–18:14, New York, NY, USA. ACM.
- [39] Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F. B. e Babu, S. (2011c). Starfish: A self-tuning system for big data analytics. Em *In CIDR*, páginas 261–272.
- [40] Horridge, M. e Bechhofer, S. (2011). The owl api: A java api for owl ontologies. *Semant. web*, 2(1):11–21.
- [41] Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosz, B. e Dean, M. (2004). Swrl: A semantic web rule language combining owl and ruleml. W3c member submission, World Wide Web Consortium.
- [42] Horrocks, I. e Tessaris, S. (2002). Querying the semantic web: A formal approach. Em *Proceedings of the First International Semantic Web Conference on The Semantic Web, ISWC '02*, páginas 177–191, London, UK, UK. Springer-Verlag.
- [43] Jurisica, I., Mylopoulos, J. e Yu, E. (2004). Ontologies for knowledge management: An information systems perspective. *Knowl. Inf. Syst.*, 6(4):380–401.

- [44] Kadirvel, S. e Fortes, J. A. B. (2011). Towards self-caring mapreduce: Proactively reducing fault-induced execution-time penalties. Em *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, páginas 63–71.
- [45] Kadirvel, S. e Fortes, J. A. B. (2012). Grey-box approach for performance prediction in map-reduce based platforms. Em *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, páginas 1–9.
- [46] Kambatla, K., Pathak, A. e Pucha, H. (2009). Towards optimizing hadoop provisioning in the cloud. Em *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing, HotCloud'09*, Berkeley, CA, USA. USENIX Association.
- [47] Kavulya, S., Tan, J., Gandhi, R. e Narasimhan, P. (2010). An analysis of traces from a production mapreduce cluster. Em *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGRID '10*, páginas 94–103, Washington, DC, USA. IEEE Computer Society.
- [48] Khousseinova, N., Balazinska, M. e Suciu, D. (2012). Perfxplain: debugging mapreduce job performance. *Proc. VLDB Endow.*, 5(7):598–609.
- [49] Kim, K., Jeon, K., Han, H., Kim, S.-g., Jung, H. e Yeom, H. Y. (2008). Mrbench: A benchmark for mapreduce framework. Em *Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems, ICPADS '08*, páginas 11–18, Washington, DC, USA. IEEE Computer Society.
- [50] Kitchenham, B. (2004). Procedures for performing systematic reviews. Keele university. technical report tr/se-0401, Department of Computer Science, Keele University, UK.
- [51] Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J. e Linkman, S. (2009). Systematic literature reviews in software engineering: A systematic literature review. *Information and Software Technology*, 51(1):7 – 15. Special Section - Most Cited Articles in 2002 and Regular Research Papers.
- [52] Klyne, G. e Carroll, J. J., editores (2004). *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. World Wide Web Consortium.
- [53] Koehler, M. e Benkner, S. (2012). Design of an adaptive framework for utility-based optimization of scientific applications in the cloud. Em *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, páginas 303–308.
- [54] Koehler, M., Kaniovskyi, Y. e Benkner, S. (2011). An adaptive framework for the execution of data-intensive mapreduce applications in the cloud. Em *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, IPDPSW '11*, páginas 1122–1131, Washington, DC, USA. IEEE Computer Society.
- [55] Kompella, R. R., Hu, Y. C. e Xie, D. (2012). On the performance projectability of mapreduce. Em *Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom), CLOUDCOM '12*, páginas 301–308, Washington, DC, USA. IEEE Computer Society.
- [56] Lama, P. e Zhou, X. (2012). Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. Em *Proceedings of the 9th International Conference on Autonomic Computing, ICAC '12*, páginas 63–72, New York, NY, USA. ACM.

- [57] Leonard, D. e Sensiper, S. (1998). The Role of Tacit Knowledge In Group Innovation. *California Management Review*, 40(3):112–133.
- [58] Li, W., Yang, H., Luan, Z. e Qian, D. (2011). Energy prediction for mapreduce workloads. Em *Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, DASC '11, páginas 443–448, Washington, DC, USA. IEEE Computer Society.
- [59] Lin, X., Meng, Z., Xu, C. e Wang, M. (2012a). A practical performance model for hadoop mapreduce. Em *Proceedings of the 2012 IEEE International Conference on Cluster Computing Workshops*, CLUSTERW '12, páginas 231–239, Washington, DC, USA. IEEE Computer Society.
- [60] Lin, X., Tang, W. e Wang, K. (2012b). Predator; an experience guided configuration optimizer for hadoop mapreduce. Em *Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, CLOUDCOM '12, páginas 419–426, Washington, DC, USA. IEEE Computer Society.
- [61] Liu, Y., Li, M., Alham, N. K. e Hammoud, S. (2013). Hsim: A mapreduce simulator in enabling cloud computing. *Future Gener. Comput. Syst.*, 29(1):300–308.
- [62] Ma, Y. B., Jang, S. H. e Lee, J. S. (2011). Ontology-based resource management for cloud computing. Em *Proceedings of the Third international conference on Intelligent information and database systems - Volume Part II*, ACIIDS'11, páginas 343–352, Berlin, Heidelberg. Springer-Verlag.
- [Mcguinness e van Harmelen] Mcguinness, D. L. e van Harmelen, F. OWL Web Ontology Language Overview.
- [64] Menolli, A., Reinehr, S. e Malucelli, A. (2013). Organizational learning applied to software engineering: A systematic review. *International Journal of Software Engineering and Knowledge Engineering*, 23(08):1153–1175.
- [65] Menolli, A. L. A. (2013). *Ambiente Colaborativo Social Semântico Voltado à Aprendizagem Organizacional para Empresas de Desenvolvimento de Software*. Tese de doutorado, Pontifícia Universidade Católica do Paraná.
- [66] Moraes, Fabio Jorge Almeida & Brasileiro, F. V. (2012). Avaliação de Serviços de Computação na Nuvem para Aplicações de e-Ciência. *SBRC*, páginas 786–799.
- [67] Moreira, M. A. (2010). *Mapas Conceituais e Aprendizagem Significativa*. Centauro.
- [68] Motik, B., Cuenca Grau, B., Horrocks, I. e Sattler, U. (2009). Representing ontologies using description logics, description graphs, and rules. *Artificial Intelligence*, 173(14):1275–1309.
- [69] Nonaka, I. e Konno, N. (1998). The concept of ? ba ? : Building a foundation for knowledge creation. *California Management Review*, 40(3):40–54.
- [70] Noy, N. F. (2004). Semantic integration: a survey of ontology-based approaches. *SIGMOD Rec.*, 33(4):65–70.
- [71] Nunez, A. e Merayo, M. G. (2013). A formal framework to analyze cost and performance in map-reduce based applications. *Journal of Computational Science*, (0):–.

- [72] Plantenga, T. D., Choe, Y. R. e Yoshimura, A. (2012). Using performance measurements to improve mapreduce algorithms. *Procedia Computer Science*, 9(0):1920 – 1929. Proceedings of the International Conference on Computational Science, {ICCS} 2012.
- [73] Premchaiswadi, W. e Romsaiyud, W. (2013). Optimizing and tuning mapreduce jobs to improve the large-scale data analysis process. *Int. J. Intell. Syst.*, 28(2):185–200.
- [74] Prud'hommeaux, E. e Seaborne, A. (2008). Sparql query language for rdf. Latest version available as <http://www.w3.org/TR/rdf-sparql-query/>.
- [75] Raj, A., Kaur, K., Dutta, U., Sandeep, V. e Rao, S. (2012). Enhancement of hadoop clusters with virtualization using the capacity scheduler. Em *Services in Emerging Markets (ICSEM), 2012 Third International Conference on*, páginas 50–57.
- [76] Ren, Z., Liu, Z., Xu, X., Wan, J., Shi, W. e Zhou, M. (2012). Waxelegant: A realistic hadoop simulator for parameters tuning and scalability analysis. Em *Proceedings of the 2012 Seventh ChinaGrid Annual Conference, CHINAGRID '12*, páginas 9–16, Washington, DC, USA. IEEE Computer Society.
- [77] Rizvandi, N. B., Taheri, J., Moraveji, R. e Zomaya, A. Y. (2012a). Network load analysis and provisioning of mapreduce applications. Em *Proceedings of the 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT '12*, páginas 161–166, Washington, DC, USA. IEEE Computer Society.
- [78] Rizvandi, N. B., Taheri, J., Moraveji, R. e Zomaya, A. Y. (2012b). On modelling and prediction of total cpu usage for applications in mapreduce environments. Em *Proceedings of the 12th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I, ICA3PP'12*, páginas 414–427, Berlin, Heidelberg. Springer-Verlag.
- [79] Schmidt-Schaubß, M. (1989). Subsumption in kl-one is undecidable. Em *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, páginas 421–431, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [80] Sim, T. H. . K. M. (2010). An ontology-enhanced cloud service discovery system. *Lecture Notes in Engineering and Computer Science*, 2180:644–649.
- [81] Stanford (2013). Hdfs architecture guide. <http://protege.stanford.edu>.
- [82] Tian, F. e Chen, K. (2011). Towards optimal resource provisioning for running mapreduce programs in public clouds. Em *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, CLOUD '11*, páginas 155–162, Washington, DC, USA. IEEE Computer Society.
- [83] Tiwari, D. e Solihin, D. (2012). Modeling and analyzing key performance factors of shared memory mapreduce. Em *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, páginas 1306–1317.
- [84] Uschold, M. e Gruninger, M. (2004). Ontologies and semantics for seamless connectivity. *SIGMOD Rec.*, 33(4):58–64.
- [85] Uschold, M., Gruninger, M., Uschold, M. e Gruninger, M. (1996). Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11:93–136.

- [86] Venner, J. (2009). Tuning your mapreduce jobs. Em *Pro Hadoop*, páginas 177–206. Apress.
- [87] Verma, A., Cherkasova, L. e Campbell, R. H. (2011). Play it again, simmr! Em *Proceedings of the 2011 IEEE International Conference on Cluster Computing, CLUSTER '11*, páginas 253–261, Washington, DC, USA. IEEE Computer Society.
- [88] W3C (2000). Xml schema. <http://www.w3.org/XML/Schema>.
- [89] W3C (2009). Owl web ontology language. <http://www.w3.org/TR/owl-features/>. January 16, 2009.
- [90] Wang, G., Butt, A., Pandey, P. e Gupta, K. (2009a). A simulation approach to evaluating design decisions in mapreduce setups. Em *Modeling, Analysis Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09. IEEE International Symposium on*, páginas 1–11.
- [91] Wang, G., Butt, A. R., Monti, H. e Gupta, K. (2011a). Towards synthesizing realistic workload traces for studying the hadoop ecosystem. Em *19th IEEE Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, páginas 400–408, Raffles Hotel, Singapore. IEEE Computer Society.
- [92] Wang, G., Butt, A. R., Pandey, P. e Gupta, K. (2009b). Using realistic simulation for performance analysis of mapreduce setups. Em *Proceedings of the 1st ACM Workshop on Large-Scale System and Application Performance, LSAP '09*, páginas 19–26, New York, NY, USA. ACM.
- [93] Wang, K., Tan, B., Shi, J. e Yang, B. (2011b). Automatic task slots assignment in hadoop mapreduce. Em *Proceedings of the 1st Workshop on Architectures and Systems for Big Data, ASBD '11*, páginas 24–29, New York, NY, USA. ACM.
- [94] Yang, H., Luan, Z., Li, W. e Qian, D. (2012a). Mapreduce workload modeling with statistical approach. *J. Grid Comput.*, 10(2):279–310.
- [95] Yang, H., Luan, Z., Li, W., Qian, D. e Guan, G. (2012b). Statistics-based workload modeling for mapreduce. Em *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IPDPSW '12*, páginas 2043–2051, Washington, DC, USA. IEEE Computer Society.
- [96] Yang, S.-Y., Hsu, C.-L. e Lee, D.-L. (2010). An ontology-supported ubiquitous interface agent for cloud computing - Example on Bluetooth wireless technique with Java programming. Em *ICMLC'10*, páginas 2971–2978.
- [97] Yang, X. e Sun, J. (2011). An analytical performance model of mapreduce. Em *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, páginas 306–310.
- [98] Yoo, H., Hur, C., Kim, S. e Kim, Y. (2009). An ontology-based resource selection service on science cloud. Em Slezak, D., Kim, T.-h., Yau, S. S., Gervasi, O. e Kang, B.-H., editores, *Grid and Distributed Computing*, volume 63 de *Communications in Computer and Information Science*, páginas 221–228. Springer Berlin Heidelberg.
- [99] Youseff, L., Butrico, M. e Da Silva, D. (2008). Toward a Unified Ontology of Cloud Computing. Em *Grid Computing Environments Workshop, 2008. GCE '08*, páginas 1–10.

- [100] Zelkowitz, M. V. e Wallace, D. R. (1998). Experimental models for validating technology. *Computer*, 31(5):23–31.
- [101] Zelkowitz, M. V., Wallace, D. R. e Binkley, D. W. (2003). Lecture notes on empirical software engineering. capítulo: Experimental Validation of New Software Technology, páginas 229–263. World Scientific Publishing Co., Inc., River Edge, NJ, USA.
- [102] Zhang, Z., Cherkasova, L. e Loo, B. T. (2013). Benchmarking approach for designing a mapreduce performance model. Em *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, páginas 253–258, New York, NY, USA. ACM.

APÊNDICE A – CÓDIGOS PARA IMPLEMENTAÇÃO DO PROTÓTIPO DA ABORDAGEM SEMÂNTICA

Neste apêndice são apresentados os códigos utilizados para a implementação do protótipo da abordagem semântica baseada em ontologias para sintonia de parâmetros do Hadoop Mapreduce.

A.1 VALIDAÇÃO DOS PARÂMETROS DE ENTRADA

Arquivos XSD (*XML Schema Definition*) são usados para descrever o "formatopadrão" que um arquivo XML deve seguir. Sendo assim, ele deve indicar quais *nodes* ele pode conter, quais *subnodes* e atributos esses *nodes* podem conter, entre outras coisas (88).

Abaixo apresenta-se o arquivo XSD para validação dos parâmetros de entrada oriundos do *log* de execução do Hadoop.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="
   qualified">
3   <xs:element name="configuration">
4     <xs:complexType>
5       <xs:sequence>
6         <xs:element maxOccurs="unbounded" ref="property"/>
7       </xs:sequence>
8     </xs:complexType>
9   </xs:element>
10  <xs:element name="property">
11    <xs:complexType>
12      <xs:sequence>
13        <xs:element ref="name"/>
14        <xs:element ref="value"/>
15      </xs:sequence>
16    </xs:complexType>
17  </xs:element>
18  <xs:element name="name" type="xs:NCName"/>
19  <xs:element name="value" type="xs:string"/>
20 </xs:schema>

```

A.2 LEITURA DOS LOGS

O código JAXB (Java Architecture for XML Binding) para leitura do arquivo de configuração gerado pelas execuções históricas do Hadoop, é apresentado abaixo:

```

1 package com.doc.main;
2
3 import generated.Configuration;
4 import generated.Property;
5 import java.io.File;
6 import java.util.List;
7 import javax.xml.bind.JAXBContext;
8 import javax.xml.bind.Unmarshaller;
9
10 public class JobConfiguration {

```

```

11     public final static List<Property> getProperties(String fileName)
12         throws Exception {
13         File file = new File(fileName);
14         Configuration configuration = null;
15         JAXBContext jaxbContext = JAXBContext.newInstance(Configuration.
16             class);
17         Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller()
18             ;
19         configuration = (Configuration) jaxbUnmarshaller.unmarshal(file)
20             ;
21         configuration.getProperty();
22         return (configuration != null) ? configuration.getProperty() :
23             null;
24     }
25 }

```

A.3 SIMULAÇÃO DOS JOBS

A classe Java abaixo representa as informações contidas na configuração do *job* e que geram o arquivo Json com os dados de entrada para o Simulador MRSim.

```

1  /**
2   * 1. unique_job_id
3   * 2. job_name
4   * 3. map_input_bytes
5   * 4. shuffle_bytes
6   * 5. reduce_output_bytes
7   * 6. submit_time_seconds (epoch format)
8   * 7. duration_seconds
9   * 8. map_time_task_seconds (2 tasks of 10 seconds = 20 task-seconds)
10  * 9. red_time_task_seconds
11  * 10. total_time_task_seconds
12  * 11. map_tasks_count
13  * 12. reduce_tasks_count
14  * 13. hdfs_input_path (if available in history log directory)
15  * 14. hdfs_output_path (if available in history log directory)
16  */
17
18  public class JobTrace {
19
20      private String uniqueJobID;
21      private String jobName;
22      private int mapInputBytes;
23      private int shuffleBytes;
24      private double reduceOutputBytes;
25      private float submitTimeSeconds;
26      private float durationSeconds;
27      private float mapTimeTaskSeconds;
28      private float redTimeTaskSeconds;
29      private float totalTimeTaskSeconds;
30      private int mapTasksCount;
31      private int reduceTasksCount;
32      private String hdfsInputPath;
33      private String hdfsOutputPath;
34
35      ...
36

```

37 }

Abaixo é listado o programa Java que faz o *parser* do arquivo de configuração e do *log* resultante da simulação do *Job*, gerando um arquivo "CSV" (*Comma Separated Value*) com os dados de saída da simulação e que servem para a análise do módulo *Evaluator* para a criação dos gráficos de análise.

```

1 package com.doc.main;
2
3 import generated.Property;
4
5 import java.io.BufferedInputStream;
6 import java.io.BufferedOutputStream;
7 import java.io.ByteArrayOutputStream;
8 import java.io.File;
9 import java.io.FileOutputStream;
10 import java.io.IOException;
11 import java.io.InputStream;
12 import java.util.ArrayList;
13 import java.util.HashMap;
14 import java.util.HashSet;
15 import java.util.List;
16 import java.util.Map;
17 import java.util.Set;
18
19
20 public class TraceExaminer {
21
22     private static final int STD_OUT = 1;
23     private static String PATH;
24     private static String OUTPUT;
25     private static final String CONFIG_SUFFIX = "_conf.xml";
26     private static final String CONF_PREFIX = "C__";
27     private static final String RESL_PREFIX = "R__";
28
29
30     public static void main(String[] args) {
31         Process process = null;
32         try {
33             if(args.length<2){
34                 System.err.println("Argumentos: <DIRETORIO_LOGS> <
35                 ARQUIVO_CSV_SAIDA>\n"+
36                 "ex: java -jar ontotrace.jar ~ailton/projects/hadoop_logs/data/ file.csf");
37                 return;
38             } else {
39                 PATH = args[0];
40                 OUTPUT = args[1];
41             }
42
43             process = Runtime.getRuntime().exec(
44                 "perl " + PATH + "parse-hadoop-jobhistory.pl " + PATH);
45
46             String result = getOutput(process.getInputStream(), STD_OUT);
47             String[] lines = result.split("\n");
48
49             Map<String, JobTrace> map = new HashMap<>();
50             List<JobTrace> list = new ArrayList<>();

```

```

51
52 JobTrace jobTrace = null;
53
54 for (String line : lines) {
55     jobTrace = new JobTrace();
56     int param = 1;
57
58     String[] tokens = line.split("\t");
59     for (String token : tokens) {
60         switch (param) {
61             case 1:
62                 jobTrace.setUniqueJobID(token);
63                 break;
64             case 2:
65                 jobTrace.setJobName(token);
66                 break;
67             case 3:
68                 jobTrace.setMapInputBytes(Integer.parseInt(token));
69                 break;
70             case 4:
71                 jobTrace.setShuffleBytes(Integer.parseInt(token));
72                 break;
73             case 5:
74                 jobTrace.setReduceOutputBytes(Double.parseDouble(token));
75                 break;
76             case 6:
77                 jobTrace.setSubmitTimeSeconds(Float.parseFloat(token));
78                 break;
79             case 7:
80                 jobTrace.setDurationSeconds(Float.parseFloat(token));
81                 break;
82             case 8:
83                 jobTrace.setMapTimeTaskSeconds(Float.parseFloat(token));
84                 break;
85             case 9:
86                 jobTrace.setRedTimeTaskSeconds(Float.parseFloat(token));
87                 break;
88             case 10:
89                 jobTrace.setTotalTimeTaskSeconds(Float
90                     .parseFloat(token));
91                 break;
92             case 11:
93                 jobTrace.setMapTasksCount(Integer.parseInt(token));
94                 break;
95             case 12:
96                 jobTrace.setReduceTasksCount(Integer.parseInt(token));
97                 break;
98             case 13:
99                 jobTrace.setHdfsInputPath(token);
100                break;
101             case 14:
102                 jobTrace.setHdfsOutputPath(token);
103                break;
104
105             default:
106                 break;
107         }
108         param++;

```

```

109     }
110
111     map.put(jobTrace.getUniqueJobID(), jobTrace);
112     list.add(jobTrace);
113 }
114
115
116 File outputFile = new File(OUTPUT);
117 FileOutputStream fos = new FileOutputStream(outputFile);
118 BufferedOutputStream bos = new BufferedOutputStream(fos);
119 boolean hasHeader = false;
120
121 List<List<Property>> allProperties = new ArrayList<List<Property>>();
122 Set<String> propertyNames = new HashSet<>();
123
124 // Check the amount of properties and place in a set
125 for (JobTrace jobt : list) {
126     String uniqueJobID = jobt.getUniqueJobID();
127     List<Property> properties = null;
128     String fName = null;
129     try{
130         fName = PATH + uniqueJobID + CONFIG_SUFFIX;
131         properties = JobConfiguration.getProperties(fName);
132         allProperties.add(properties);
133         for (Property property : properties) {
134             propertyNames.add(property.getName());
135         }
136     } catch(Exception e){
137         System.err.println("Arquivo de configuração com problema: " +
138             uniqueJobID + " (" + fName + ") " + e.getMessage());
139     }
140
141 for (JobTrace jobt : list) {
142     String uniqueJobID = jobt.getUniqueJobID();
143
144     List<Property> properties = null;
145     Map<String, Property> mProperties = new HashMap<>();
146     String fName = null;
147     try{
148         //TODO otimização: não precisa ler o arquivo 2x, mas isso
149         agora não importa
150         fName = PATH + uniqueJobID + CONFIG_SUFFIX;
151         properties = JobConfiguration.getProperties(fName);
152         for (Property property : properties) {
153             mProperties.put(property.getName(), property);
154         }
155     } catch(Exception e){
156         System.err.println("Arquivo de configuração com problema: " +
157             uniqueJobID + " (" + fName + ") " + e.getMessage());
158     }
159
160 if(properties == null){
161     continue;
162 }
163
164 if(!hasHeader){

```

```

164         // configuration parameters
165     for (String pName : propertyNames) {
166         bos.write((CONF_PREFIX + pName.toUpperCase().replaceAll("\\.", "_").
167             replaceAll("-", "_").replaceAll(", ", " ") + ", ").getBytes());
168     }
169     hasHeader = true;
170     bos.write((RESL_PREFIX + "UNIQUE_JOB_ID" + ", ").getBytes());
171     bos.write((RESL_PREFIX + "JOB_NAME" + ", ").getBytes());
172     bos.write((RESL_PREFIX + "MAP_INPUT_BYTES" + ", ").getBytes());
173     bos.write((RESL_PREFIX + "SHUFFLE" + ", ").getBytes());
174     bos.write((RESL_PREFIX + "REDUCE_OUTPUT_BYTES" + ", ").getBytes());
175     bos.write((RESL_PREFIX + "SUBMIT_TIME_SECONDS" + ", ").getBytes());
176     bos.write((RESL_PREFIX + "DURATION_SECONDS" + ", ").getBytes());
177     bos.write((RESL_PREFIX + "MAP_TIMETASK_SECONDS" + ", ").getBytes());
178     bos.write((RESL_PREFIX + "RED_TIMETASK_SECONDS" + ", ").getBytes());
179     bos.write((RESL_PREFIX + "TOTAL_TIMETASK_SECONDS" + ", ").getBytes());
180     bos.write((RESL_PREFIX + "MAP_TASKS_COUNT" + ", ").getBytes());
181     bos.write((RESL_PREFIX + "RED_TASKS_COUNT" + ", ").getBytes());
182     bos.write((RESL_PREFIX + "HDFS_INPUT_PATH" + ", ").getBytes());
183     bos.write((RESL_PREFIX + "HDFS_OUTPUT_PATH" + ", ").getBytes());
184
185 } else {
186     for (String name : propertyNames) {
187         if (mProperties.get(name) != null) {
188             Property property = mProperties.get(name);
189             bos.write((property.getValue().toUpperCase().replaceAll("
190                 \\", "_").replaceAll("-", "_").replaceAll(", ", " ") +
191                 ", ").getBytes());
192         }
193     }
194
195     bos.write((jobt.getUniqueJobID() + ", ").getBytes());
196     bos.write((jobt.getJobName() + ", ").getBytes());
197     bos.write((jobt.getMapInputBytes() + ", ").getBytes());
198     bos.write((jobt.getShuffleBytes() + ", ").getBytes());
199     bos.write((jobt.getReduceOutputBytes() + ", ").getBytes());
200     bos.write((jobt.getSubmitTimeSeconds() + ", ").getBytes());
201     bos.write((jobt.getDurationSeconds() + ", ").getBytes());
202     bos.write((jobt.getMapTimeTaskSeconds() + ", ").getBytes());
203     bos.write((jobt.getRedTimeTaskSeconds() + ", ").getBytes());
204     bos.write((jobt.getTotalTimeTaskSeconds() + ", ").getBytes());
205     bos.write((jobt.getMapTasksCount() + ", ").getBytes());
206     bos.write((jobt.getReduceTasksCount() + ", ").getBytes());
207     bos.write((jobt.getHdfsInputPath() + ", ").getBytes());
208     bos.write((jobt.getHdfsOutputPath() + ", ").getBytes());
209 }
210 bos.write("\n".getBytes());
211 }
212
213     bos.close();
214     fos.close();
215
216 } catch (IOException e1) {
217     e1.printStackTrace();
218 }

```

```

219 try {
220     if (process != null)
221         process.waitFor();
222 } catch (InterruptedException e) {
223     e.printStackTrace();
224 }
225
226 }
227
228 private static String getOutput(InputStream is, int type) {
229     BufferedInputStream bufferedInputStream = null;
230     ByteArrayOutputStream baos = new ByteArrayOutputStream();
231     BufferedOutputStream bos = new BufferedOutputStream(baos);
232     try {
233         bufferedInputStream = new BufferedInputStream(is);
234
235         byte[] buffer = new byte[1024];
236         int count;
237         while ((count = bufferedInputStream.read(buffer)) > 0) {
238             bos.write(buffer, 0, count);
239         }
240         bos.close();
241
242     } catch (IOException e) {
243         e.printStackTrace();
244     } finally {
245         try {
246             bufferedInputStream.close();
247         } catch (IOException e) {
248             e.printStackTrace();
249         }
250         try {
251             bos.close();
252         } catch (IOException e) {
253             e.printStackTrace();
254         }
255     }
256     return baos.toString();
257 }
258
259 }

```

A.3.1 Simulador MRSim

Abaixo é listado o código adaptado do simulador MRSim para aceitar os arquivos gerados pela abordagem.

```

1 package atest;
2
3 import hasim.HJobTracker;
4 import hasim.RF;
5 import hasim.gui.HMonitor;
6 import hasim.gui.HMonitor.DebugMode;
7 import hasim.gui.HSimulator;
8 import hasim.json.JsonJob;
9 import hasim.json.JsonRealRack;
10

```

```

11 import org.apache.log4j.Logger;
12
13 public class StartSim {
14     /**
15      * Logger for this class
16      */
17     private static final Logger logger = Logger.getLogger(StartSim.class);
18
19     private static final int SIMULATION_TIME = 3000;
20
21     public static void main(String[] args) {
22         try {
23             final HSimulator sim = new HSimulator();
24             String rackFile = RF.firstFileInDir("data/json/rack");
25             if (HSimulator.running()) {
26                 sim.stopSimulator();
27             }
28             JsonRealRack jsonRack = JsonJob.read(rackFile,
29                 JsonRealRack.class);
30             HSimulator.initSimulator(jsonRack.isFlowType());
31             RF.copy(rackFile, RF.get(sim.resultDir, RF.jsonRack));
32             sim.jobTracker = new HJobTracker("JobTracker", rackFile,
33                 null);
34             HMonitor.setDebugMode(DebugMode.STEP);
35             HMonitor.setSLEEP(600);
36             sim.jobTracker.createEntities(sim.resultDir);
37             Runnable r= new Runnable() {
38                 public void run() {
39                     logger.info("simulator has started simulator")
40                     ;
41                     sim.startSimulator();
42                 }
43             };
44             Thread t = new Thread(r);
45             t.start();
46
47             String jobFile = "/Users/eduardo/Data/Professional/Hadoop/
48                 wks/MrSim
49                 /data/json/jobs/job_1.json";
50             sim.submitJob(jobFile);
51             Thread.sleep(SIMULATION_TIME);
52             System.err.println("Waiting until simulation end");
53             sim.join();
54             System.err.println("END");
55         } catch (Exception e) {
56             e.printStackTrace();
57         }
58     }
59 }

```

A.4 INSTANCIADOR DE UNIDADES DE SINTONIA - ONTOHADOOP

Exemplo de instância da ontologia utilizando OWL Api

```

1 OntoHadoopFactory ontoHadoopFactory = new OntoHadoopFactory(ontology);
2 List<Parameters> parameters = new ArrayList<>();
3 SizeInput sizeInput = ontoHadoopFactory.createSizeInput("sizeInput1");

```

```

4 sizeInput.addSizeInputMinMb(new Double(1));
5 sizeInput.addSizeInputMaxMb(new Double(512000));
6 SpeedJob speedJob = ontoHadoopFactory.createSpeedJob("speedJob1");
7 System_Resource systemResource = ontoHadoopFactory.createSystem_Resource("
    systemResource1");
8 systemResource.addNameInstance("large");
9 systemResource.addNumberCPU(4);
10 systemResource.addMemGB(new Double(7.5));
11 systemResource.addStorageGB(new Double(32));
12 TypeJob typeJob = ontoHadoopFactory.createTypeJob("jobType1");
13 typeJob.addHasSize(sizeInput);
14 typeJob.addHasSpeed(speedJob);
15 typeJob.addHasSystemResource(systemResource);
16 WorkloadPattern workloadPattern = ontoHadoopFactory.createWorkloadPattern("
    workloadPatternAggregate");
17 workloadPattern.addHasTypeJob(typeJob);
18 Parameters dfsBlockSize = ontoHadoopFactory.createDfs_block_size("
    dfsBlockSize");
19 dfsBlockSize.addNameParameter("DFS_BLOCK_SIZE");
20 dfsBlockSize.addValueParameter("64m");
21 parameters.add(dfsBlockSize);
22 Parameters ioFileBufferSize = ontoHadoopFactory.createDfs_block_size("
    ioFileBufferSize");
23 ioFileBufferSize.addNameParameter("IO_FILE_BUFFER_SIZE");
24 ioFileBufferSize.addValueParameter("4096");
25 parameters.add(ioFileBufferSize);
26 HadoopUnitTuning tuningUnit = ontoHadoopFactory.createHadoopUnitTuning("
    hadoopUnitTuning1");
27 for (Parameters param : parameters) {
28     tuningUnit.addHasParameter(param);
29 }
30 tuningUnit.addHasWorkloadPattern(workloadPattern);

```

APÊNDICE B – PUBLICAÇÕES

Neste apêndice são listadas as publicações obtidas durante o desenvolvimento desta tese.

B.1 TRABALHOS PUBLICADOS

B.1.0 Trabalhos Publicados em Anais de Congressos

1. **BONIFACIO, A. S.**; MENOLLI, Andre; SILVA, Fabiano. *Hadoop MapReduce Configuration Parameters and System Performance: a Systematic Review*. In: International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2014), 2014, Las Vegas, NV, USA.
2. **BONIFACIO, A. S.**; MENOLLI, Andre; SILVA, Fabiano. *Towards an Ontology-based Semantic Approach to Tuning Parameters to Improve Hadoop Application Performance*. In: 9th International Conference on Information Technology and Applications (ICITA 2014), 2014, Sydney, Australia.

B.1.0 Trabalhos em Periódicos

1. **BONIFACIO, A. S.**; MENOLLI, Andre; SILVA, Fabiano. *Towards an Ontology-based Semantic Approach to Tuning Parameters to Improve Hadoop Application Performance*. IT in Industry (ITII) 2(2):56-61, 2014.