UNIVERSIDADE FEDERAL FLUMINENSE

DOUGLAS ERICSON MARCELINO DE OLIVEIRA

OTIMIZAÇÃO DA EXECUÇÃO DE WORKFLOWS INTENSIVOS DE DADOS EM FRAMEWORKS MAPREDUCE

NITERÓI

2017

UNIVERSIDADE FEDERAL FLUMINENSE

DOUGLAS ERICSON MARCELINO DE OLIVEIRA

OTIMIZAÇÃO DA EXECUÇÃO DE WORKFLOWS INTENSIVOS DE DADOS EM FRAMEWORKS MAPREDUCE

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Redes e Processamento Paralelo e Distribuído

Orientador:
DANIEL CARDOSO MORAES DE OLIVEIRA

Co-orientadores:
MARIA CRISTINA DA SILVA BOERES
FÁBIO ANDRÉ MACHADO PORTO

NITERÓI

2017

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

O48 Oliveira, Douglas Ericson Marcelino de
Otimização da execução de workflows intensivos de dados em frameworks *MapReduce* / Douglas Ericson Marcelino de Oliveira. – Niterói, RJ : [s.n.], 2017.
161 f.

Tese (Doutorado em Computação) - Universidade Federal Fluminense, 2017.
Orientadores: Daniel Cardoso Moraes de Oliveira, Maria Cristina da Silva Boeres, Fábio André Machado Porto.

1. Sistemas paralelos e distribuído. 2. Fluxo de trabalho. 3. Aprendizado de máquina. I. Título.

CDD 004.35

DOUGLAS ERICSON MARCELINO DE OLIVEIRA

OTIMIZAÇÃO DA EXECUÇÃO DE *WORKFLOWS* INTENSIVOS DE DADOS EM FRAMEWORKS *MAPREDUCE*

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Redes e Processamento Paralelo e Distribuído

Aprovada em Novembro de 2017.

BANCA EXAMINADORA

Prof. DANIEL CARDOSO MORAES DE OLIVEIRA - Orientador, UFF

Prof. MARIA CRISTINA DA SILVA BOERES - Co-Orientador, UFF

Prof. FÁBIO ANDRÉ MACHADO PORTO - Co-Orientador, LNCC

Prof. LÚCIA MARIA DE ASSUMPÇÃO DRUMMOND - UFF

Prof. VANESSA BRAGANHOLO MURTA - UFF

Prof. MARTA LIMA DE QUEIRÓS MATTOSO - UFRJ

Prof. BRUNO RICHARD SCHULZE - LNCC

Prof. ESTHER PACITTI - INRIA & LIRMM, UNIVERSITY OF MONTPELLIER

Niterói

2017

Dedico este trabalho à minha esposa Juliana que muito me apoiou e incentivou a realizá-lo.

Agradecimentos

Elaborar uma tese de doutorado foi uma das tarefas mais desafiadores que já enfrentei. Certamente não seria possível realizá-la sozinho e muitas pessoas me ajudaram durante esta caminhada, a quem preciso agradecer neste momento.

Agradeço primeiramente à Deus, por me dar esta oportunidade e estar comigo em todo o tempo me ajudando e dando forças para continuar nesta jornada.

À minha esposa Juliana que sempre me motivou e esteve ao meu lado em todos os momentos, nas horas de desânimo e ansiedade, nas viagens de férias adiadas, etc. Não existem palavras que podem expressar minha gratidão, Te amo muito!. Sou muito grato por tê-la como esposa, amiga e companheira.

À minha filha Melissa que nasceu durante a realização deste trabalho e que nos proporcionou muita alegria e, consequentemente, motivação para a conclusão do mesmo, Papai te ama muito!

Aos meus pais Rogério e Joelma que sempre me apoiaram e incentivaram a prosseguir nos estudos, dando continuamente todo o suporte necessário para isso. Ao meu irmão André e minha cunhada Aline, meu cunhado Adriel e minha irmã Raquel, meus sogros Hélio e Helena, cunhados Jeferson e Simone, Helinho e Daniele e meus sobrinhos Lucas, Davi e Enzo. Sou muito grato a todos pelo compreensão, companheirismo e pelas palavras de apoio.

À professora Cristina Boeres que me orientou desde o início do trabalho de forma muito responsável, atenciosa e dedicada. Ao professor Daniel de Oliveira que prontamente aceitou me orientar durante a realização da tese e contribuiu com muitas ideias e sugestões decisivas para a elaboração e conclusão da mesma. Ao meu amigo e orientador, professor Fábio Porto, que é um exemplo para mim, e me ajudou muitíssimo compartilhando sugestões criativas e inovadoras diante do seu vasto conhecimento e também pela disponibilidade, amizade e paciência que contribuíram de forma decisiva para o meu crescimento acadêmico desde o mestrado;

As professoras Lúcia Maria de Assumpção Drummond, Vanessa Braganholo Murta,

Marta Lima de Queirós Mattoso, Esther Pacitti e ao professor Bruno Richard Schulze por terem aceitado fazer parte desta banca;

Aos meus amigos, Matheus e Henrique, pelo trabalho em conjunto e amizade durante todo o doutorado, em especial pela parceria na realização das disciplinas e nas viagens Petrópolis-Niterói. Ao Adolfo Simões, Rodrigo Botelho, João Guilherme pela ajuda com os ambientes de execução dos experimentos desta tese;

Ao casal de amigos, Dáfnis e Ester, pelos valiosos conselhos, apoio e ajuda nos momentos mais difíceis e por estarem na defesa da tese compartilhando este momento comigo. Ao casal de amigos, Leonardo e Meire, por sempre incentivar, de forma descontraída e espirituosa, e disponibilizar todo o apoio possível;

A todos os demais familiares, amigos, professores, entre outros, que nos bastidores me ajudaram de alguma maneira e contribuíram para a realização deste trabalho;

Aos revisores de artigos, que de alguma forma contribuíram para a melhoria dos trabalhos aqui referenciados. À CAPES, pela concessão da bolsa de doutorado entre 2012 e 2016 que me permitiu realizar o doutoramento;

Agradeço.

Resumo

Na ciência a análise de grandes volumes de dados é modelada como experimento científico, envolvendo algumas questões como o armazenamento dos dados e formatos dos mesmos, encadeamento dos programas e definição do ambiente de execução usados durante as simulações. Cientistas têm usado *workflows* científicos para exprimir e modelar computacionalmente análises e experimentos sobre dados. Devido à complexidade de processamento dos *workflows* e também o volume de dados envolvido, estes têm sido executados em ambientes distribuídos, em conjunto com modelos de programação paralela do *workflow*. O modelo *MapReduce* (MR) tem sido muito utilizado na especificação de experimentos científicos, em especial, aqueles que analisam um grande volume de dados. A partir do MR foram criados *frameworks*, como Hadoop e Spark, que permitem a manipulação e análise dos dados de forma distribuída, além de realizarem o gerenciamento da execução dos experimentos em ambientes distribuídos. No entanto, a execução de *workflows* intensivos de dados em ambientes distribuídos gerenciados por *frameworks* MR ainda apresenta desafios em aberto. Embora exista uma certa facilidade na instalação desses *frameworks*, há muitos parâmetros a serem configurados para execução de um *workflow*. Além disso, para explorar o paralelismo oferecido pelo ambiente é necessário o particionamento dos dados de entrada. Existem diversas estratégias de particionamento de dados e aspectos como: conhecimento do critério de particionamento por parte da aplicação, tamanho das partições e o balanceamento de carga interferem no desempenho do *workflow*. Com isso, para executar um *workflow* MR de forma eficiente, o cientista deve ajustar diversos parâmetros de configuração dos *frameworks* e do particionamento dos dados de entrada. As correlações que existem entre estes parâmetros, o *workflow* e o ambiente de execução tornam o ajuste da configuração de tais parâmetros uma tarefa complexa e difícil para o cientista. Nesta tese, é proposta uma abordagem que pode ser aplicada no ajuste da configuração dos parâmetros de execução de *workflows* MR em ambientes distribuídos. A abordagem é baseada em (i) coletar o tempo de execução do *workflow* utilizando diversos valores na configuração dos parâmetros, (ii) aplicar técnicas de aprendizado de máquina afim de encontrar os valores e parâmetros que executam o *workflow* de forma eficiente e (iii) utilizar as mesmas técnicas para gerar o modelo preditivo para conhecer previamente o desempenho de uma configuração de parâmetros em execuções posteriores do *workflow* MR. Os experimentos apresentados nesta tese mostraram que a abordagem proposta para configuração de parâmetros conduz a um desempenho eficiente do *workflow* MR em um ambiente distribuído.

Palavras-chave: *Workflows* Científicos, Configuração de Parâmetros, Aprendizado de Máquina.

Abstract

In science, an analysis of large volumes of data is modeled as a scientific experiment, involving some issues such as data storage and formatting, program chaining and the definition of execution environment during simulations. Scientists have used scientific workflows to express and model computations and experiments on data. Due to complexity of the workflows and also the volume of data involved, these have been executed on distributed environments, through workflow parallel programming models. The MapReduce (MR) model has been widely used in the specification of scientific experiments, especially those that analyze a large volume of data. From the MR, frameworks such as Hadoop and Spark were created, which allow the manipulation and analysis of the data in a distributed way, as well as managing the execution of the experiments on distributed environments. However, the execution of intensive data workflows on distributed environments managed by MR frameworks still presents open challenges. Although it is not a complex task to install these frameworks, there are many parameters to be configured to execute a workflow. In addition, to exploit the parallelism offered by the environment it is necessary to partition the input data. There are several data partitioning strategies and aspects such as: knowledge of the partitioning criterion by the application, partition size and load balancing impact the workflow performance. Thus, in order to execute an MR workflow efficiently, the scientist must tune several configuration parameters related to the framework and data partitioning. The correlations between these parameters, workflow, and the execution environment make the configuration of such parameters a complex and difficult task for the scientist. In this thesis, an approach is proposed that can be applied in tuning the execution parameters configuration of workflows MR in distributed environments. The approach is based on (i) collecting the workflow execution time using several values in the parameters configuration, (ii) applying machine learning techniques in order to find the values and parameters that execute the workflow efficiently and (iii) use the same techniques to generate the predictive model to previously know the performance of a parameter configuration in later executions of workflow MR. The experiments presented in this thesis showed that the proposed approach to parameter setting leads to efficient performance of MR workflow in a distributed environment.

Keywords: Scientific Workflows, Parameter Tuning, Machine Learning.

Lista de Figuras

2.1 Passo a passo do FRANCE [FREIRE, 2016] 15

2.2 Etapas da Criação de uma Quadtree [FREIRE, 2016] 16

2.3 Organização de uma árvore de classificação 26

3.1 Arquitetura com SGBD Centralizado e Particionado 32

3.2 Tempo de Execução de cada Atividade nas Arquiteturas: HDFS, SGBD Particionado, SGBD Centralizado 35

3.3 *Workflow* da Aplicação *SkyMap* e Alternativas de Execução 37

3.4 Tempo de Execução Total e de cada Atividade com as alternativas - Hadoop, HaQoop, HaQoop-HDFS e Spark 41

4.1 Abordagem de Otimização de desempenho de *Workflows* MR em Clusters Computacionais 48

4.2 Coleta dos dados para treinamento do modelo preditivo 49

5.1 Sistema de coordenadas RA e DEC 53

5.2 *Einstein Cross* 54

5.3 *Workflow* que representa a aplicação da astronomia SkyMap 54

5.4 Modelo de Execução do SkyMap com Particionamento Espacial dos Dados 56

5.5 Modelo de Execução do SkyMap com Particionamento AdHoc dos Dados . 57

5.6 Distâncias relativas entre os pontos do *Einstein Cross* 57

5.7 *Workflow* que representa a aplicação da astronomia Constellation Queries . 58

5.8 Tempo de Execução do *Workflow* Skymap no framework Spark e no cluster Petrus 61

5.9 Tempo Total de Execução do Workflow SkyMap no Petrus considerando particionamento FRANCE e AdHoc 63

5.10 Tempo Total de Execução do Workflow SkyMap no Petrus considerando as estratégias de particionamento FRANCE, QuadTree e AdHoc 63

5.11 Tempo de Execução total do *workflow* SkyMap e de cada atividade variando o número de partições com estratégia de particionamento FRANCE no *cluster* Petrus 65

5.12 Tempo de Execução total do *workflow* SkyMap e de cada atividade variando o número de partições com estratégia de particionamento AdHoc no *cluster* Petrus 65

5.13 Tempo de Execução total obtido com a execução do *workflow* SkyMap no Petrus com as configurações de parâmetros $C1$ e $C2$ 68

5.14 Tempo de Execução total obtido com a execução do *workflow* SkyMap no Petrus com as configurações $C3$ e $C4$ 69

5.15 Tempo de Execução total obtido com a execução do *workflow* SkyMap no *cluster* Petrus variando a quantidade de *Cores por Executor* conforme a configuração $C5$ 70

5.16 Tempo de Execução total obtido com a execução do *workflow* SkyMap no *cluster* Petrus variando a quantidade de *Memória/Executor* conforme a configuração $C6$ 71

6.1 Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 1: *Baixo*, *Medio* e *Alto* 75

6.2 Árvore de Classificação do Grupo de execução com 2 níveis para o Cenário 1 76

6.3 Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 1 78

6.4 Árvore de Classificação do Grupo de execução com 5 níveis para o Cenário 1 81

6.5 Árvore de Classificação de Falhas com 3 níveis para o Cenário 1 83

6.6 Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 2: Baixo, Medio e Alto 85

6.7 Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 2 86

6.8 Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 3: Baixo, Medio e Alto 88

6.9 Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 3 89

6.10 Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 4: Baixo, Medio e Alto 90

6.11 Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 4 91

6.12 Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 5: Baixo, Medio e Alto 93

6.13 Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 5 94

6.14 Matriz de Confusão Resultante da Aplicação dos dados do Cenário 4 (Actual) no modelo Preditivo gerado a partir do Cenário 1 (Predicted) 97

6.15 Matriz de Confusão Resultante da Aplicação dos dados do Cenário 5 (Actual) no modelo Preditivo gerado a partir do Cenário 1 (Predicted) 100

6.16 Matriz de Confusão Resultante da Aplicação dos dados do Cenário 2 (Actual) no modelo Preditivo gerado a partir do Cenário 1 (Predicted) 101

6.17 Matriz de Confusão Resultante da Aplicação dos dados do Cenário 3 (Actual) no modelo Preditivo gerado a partir do Cenário 1 (Predicted) 102

6.18 Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 6: Baixo, Medio e Alto 105

6.19 Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 6105

6.20 Matriz de Confusão Resultante da Aplicação dos dados do Cenário 1 (Actual) no modelo Preditivo gerado a partir do Cenário 6 (Predicted) 106

6.21 Matriz de Confusão Resultante da Aplicação dos dados do Cenário 4 (Actual) no modelo Preditivo gerado a partir do Cenário 6 (Predicted) 107

6.22 Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 7: Baixo, Medio e Alto 110

6.23 Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 7111

6.24 Matriz de Confusão Resultante da Aplicação dos dados do Cenário 1 (Actual) no modelo Preditivo gerado a partir do Cenário 7 (Predicted) 112

6.25 Matriz de Confusão Resultante da Aplicação dos dados do Cenário 5 (Actual) no modelo Preditivo gerado a partir do Cenário 7 (Predicted) 113

6.26 Abordagem Proposta para Criação do Modelo Preditivo 114

Lista de Tabelas

2.1 Exemplo de discretização utilizando o método *Equal-Frequency* 24

2.2 Matriz de Confusão de um Classificador [Monard and Baranauskas, 2003] . 27

3.1 Estratégias de Armazenamento de Dados 33

4.1 Parâmetros Selecionados da Abordagem Proposta 46

5.1 Configuração de Hardware dos Ambientes de Execução 59

5.2 Configuração do Spark nos Ambientes de Execução 60

5.3 Conjunto de parâmetros utilizados nos experimentos descritos nesta seção . 60

5.4 Configuração dos Parâmetros para execução do workflow SkyMap no cluster Petrus variando a estratégia de particionamento 61

5.5 Configuração dos Parâmetros para execução do *workflow* SkyMap no Petrus variando o número de partições em cada estratégia de particionamento 64

5.6 Variação do Tamanho da Partição, Número de Partições e Objetos por partição do *dataset* de 24GB utilizada nestes experimentos 64

5.7 Conjuntos de Configuração de Parâmetros Avaliados na Execução do *Workflow* SkyMap no *cluster* Petrus 67

5.8 Conjuntos de Configuração de Parâmetros Avaliados na Execução do *Workflow* SkyMap no *cluster* Petrus 68

5.9 Valores dos Parâmetros Fixados na Execução do *Workflow* SkyMap no Petrus variando a quantidade de *cores por executor* 69

5.10 Valores dos Parâmetros Fixados na Execução do *Workflow* SkyMap no Petrus variando a quantidade de *Memória por executor* 70

6.1 Valores dos Parâmetros Avaliados na Execução do Cenário 1 75

6.2 Resultado das Métricas utilizadas para avaliação do método árvore de classificação com dois níveis no Cenário 1 77

6.3 Resultado das Métricas utilizadas para avaliação do método árvore de classificação com três níveis no Cenário 1 79

6.4 Resultado das Métricas utilizadas para avaliação do método árvore de classificação com cinco níveis no Cenário 1 82

6.5 Resultado das Métricas utilizadas para avaliação do método árvore de classificação de falhas com três níveis no Cenário 1 84

6.6 Configuração do Spark no Ambientes de Execução no Cenário 2 84

6.7 Valores dos Parâmetros Avaliados na Execução do Cenário 2 85

6.8 Resultado das Métricas utilizadas para avaliação do método árvore de classificação com três níveis no Cenário 2 86

6.9 Configuração do Spark no Ambientes de Execução 87

6.10 Valores dos Parâmetros Avaliados na Execução do Cenário 3 87

6.11 Resultado das Métricas utilizadas para avaliação do método árvore de classificação com três níveis no Cenário 3 89

6.12 Valores dos Parâmetros Avaliados na Execução do Cenário 4 90

6.13 Resultado das Métricas utilizadas para avaliação do método árvore de classificação com três níveis no Cenário 4 91

6.14 Valores dos Parâmetros Avaliados na Execução do Cenário 5 92

6.15 Resultado das Métricas utilizadas para avaliação do método árvore de classificação com três níveis no Cenário 5 94

6.16 Quadro Comparativo entre os Cenários: *Workflow*, *Cluster*, Hardware dos Nós Trabalhadores e Regras que de configuração de parâmetros que levam a uma execução classificada como *Baixo* 94

6.17 Quadro que apresenta o custo de treinamento em cada Cenário 96

6.18 Métricas de Avaliação: Cenário 1 (Modelo) versus Cenário 4 (Teste) 99

6.19 Métricas de Avaliação: Cenário 1 (Modelo) versus Cenário 5 (Teste) 100

6.20 Métricas de Avaliação: Cenário 1 (Modelo) versus Cenário 2 (Teste) 102

6.21 Métricas de Avaliação: Cenário 1 (Modelo) versus Cenário 3 (Teste) 103

6.22 Valores dos Parâmetros Avaliados na Execução do Cenário 6 104

6.23 Métricas utilizadas para avaliação do método de classificação 106

6.24 Métricas de Avaliação: Cenário 6 (Modelo) versus Cenário 1 (Teste) 107

6.25 Métricas de Avaliação: Cenário 6 (Modelo) versus Cenário 4 (Teste) 108

6.26 Quadro Comparativo entre os Cenários: *Workflow*, *Cluster*, Hardware dos Nós Trabalhadores e Regras de configuração de parâmetros que levam a uma execução classificada como *Baixo* 108

6.27 Valores dos Parâmetros Avaliados na Execução do Cenário 7 109

6.28 Métricas utilizadas para avaliação do método de classificação 111

6.29 Métricas de Avaliação: Cenário 7 (Modelo) versus Cenário 1 (Teste) 112

6.30 Métricas de Avaliação: Cenário 7 (Modelo) versus Cenário 5 (Teste) 113

6.31 Quadro Comparativo entre os Cenários: *Workflow*, *Cluster*, Hardware dos Nós Trabalhadores e Regras que de configuração de parâmetros que levam a uma execução classificada como *Baixo* 114

7.1 Quadro comparativo entre as principais abordagens de otimização da Execução de Aplicações Intensivas de Dados 122

Lista de Abreviaturas e Siglas

API	:	*Application Programming Interface*;
BD	:	Banco de Dados;
CPU	:	*Central Process Unit*;
CSV	:	*Comma-Separated Values*;
DEC	:	*Declination*;
DES	:	*Dark Energy Survey*;
DEXLLAB	:	*Extrem Data Lab*;
DW	:	*Data Warehouse*;
FRANCE	:	FRAgmeNtador de Catálogos Espaciais;
GAD	:	Grafo Acíclico Direcionado;
GB	:	GigaByte;
GCD	:	Grafo Cíclico Direcionado;
GD	:	Grafo Direcionado;
GFS	:	*Google File System*;
HDFS	:	*Hadoop Distributed File System*;
HFS	:	*Hadoop Fair Scheduler*;
HPC	:	*High Performance Computing*;
JSON	:	*JavaScript Object Notation*;
JVM	:	*Java Virtual Machine*;
LIneA	:	Laboratório Interinstitucional de e-Astronomia;
LNCC	:	Laboratório Nacional de Computação Científica;
LSST	:	*Large Synoptic Survey Telescope*;
MB	:	MegaByte;
MR	:	MapReduce;
NFS	:	*Network File System*;
ON	:	Observatório Nacional;

PB	:	PetaByte;
PEW	:	Plano de Execução do *Workflow*;
QEF	:	*Query Evaluation Framework*;
QEP	:	*Query Execution Plan*;
RA	:	*Right Ascension*;
RAM	:	*Random Access Memory*;
RDD	:	*Resilient Distributed Dataset*;
RNP	:	Rede Nacional de Ensino e Pesquisa;
SBBD	:	Simpósio Brasileiro de Banco de Dados;
SDSS	:	*Sloan Digital Sky Survey*;
SGBD	:	Sistema Gerenciador de Banco de Dados;
SGWC	:	Sistema de Gerenciamento de *Workflows* Científicos;
SKA	:	*Square Kilometer Array*;
SPACE	:	*Spark Configuration Engine*;
SQL	:	*Structured Query Language*;
SVM	:	*Support Vector Machine*;
SVR	:	*Support Vector Regression*;
TB	:	TeraByte;
UDF	:	*User-Defined Functions*;
XML	:	*eXtensible Markup Language*;
YARN	:	*Yet Another Resource Negotiator*;

Sumário

1 Introdução **1**

1.1 Caracterização do Problema: Parametrização da Execução de um Workflow MapReduce . . . 3

1.2 Abordagem Proposta . . . 5

1.3 Organização da Tese . . . 8

2 Referencial Teórico **9**

2.1 *Workflows* Científicos . . . 9

2.1.1 Sistemas de Gerenciamento de *Workflows* Científicos . . . 11

2.1.2 Execução Paralela de *Workflows* Científicos . . . 11

2.2 Particionamento dos Dados . . . 12

2.2.1 FRANCE . . . 13

2.2.2 QuadTree . . . 15

2.2.3 AdHoc . . . 17

2.2.4 Parâmetros relacionados ao particionamento dos dados . . . 17

2.3 *MapReduce* . . . 18

2.3.1 Hadoop . . . 19

2.3.2 *Framework* Spark . . . 20

2.4 Aprendizado de Máquina . . . 22

2.4.1 Discretização . . . 23

2.4.2 Árvores de Classificação . . . 25

2.4.3 Métricas de Avaliação da Predição . . . 26

2.5 Resumo do Capítulo 28

3 Experimentos Preliminares 30

3.1 Armazenamento de Dados de Entrada 31

3.1.1 Aplicação 31

3.1.2 Estratégias de Armazenamento de Dados 32

3.1.3 Ambiente 33

3.1.4 Análise dos experimentos 34

3.2 Localização dos Dados Intermediários 36

3.2.1 Aplicação 37

3.2.2 Exploração de Alternativas para o tratamento de arquivos intermediários 37

3.2.3 Ambiente 39

3.2.4 Análise dos experimentos 39

3.2.4.1 Comparação do tempo total de execução 39

3.2.4.2 Leitura e Tratamento dos dados 40

3.2.4.3 SkyMap 41

3.2.4.4 SkyMapAdd 42

3.3 Resumo do Capítulo 43

4 Abordagem Proposta 45

4.1 Definição e Modelagem do Problema 45

4.2 A Abordagem 47

4.3 Resumo do Capítulo 50

5 Modelagem dos Experimentos 51

5.1 Estudos de Caso 51

5.1.1 Astronomia 52

5.1.1.1 Catálogo 52
5.1.1.2 Sistema de Coordenadas 52
5.1.1.3 Padrões Geométricos 53
5.1.2 *Workflow SkyMap* 54
5.1.3 *Workflow Constellation Queries* 56
5.2 Configuração do Ambiente de Execução 58
5.3 Seleção dos Parâmetros 60
5.3.1 Análise das Estratégias de Particionamento 61
5.3.2 Análise do Tamanho da Partição 64
5.3.3 Variando os parâmetros do Spark e fixando os parâmetros relacionados ao particionamento dos dados 67
5.3.4 Variando os parâmetros relacionados ao particionamento dos dados e fixando os parâmetros do Spark 67
5.3.5 Variando o número de *Cores por Executor* 69
5.3.6 Variando a quantidade de *Memória por Executor* 70
5.4 Resumo do Capítulo 72

6 Avaliação Experimental 73

6.1 Análise das Árvores de Classificação em cada Cenário 73
6.1.1 Cenário 1 - Análise da Árvore de Classificação gerada pela Execução do *Workflow* SkyMap no *Cluster* Petrus 74
6.1.2 Cenário 2 - Análise da Árvore de Classificação gerada pela Execução do *Workflow* SkyMap utilizando 50% de CPU e RAM do *cluster* Petrus 84
6.1.3 Cenário 3 - Análise da Árvore de Classificação gerada pela Execução do *Workflow* SkyMap utilizando 10% de CPU e RAM do *cluster* Petrus 87
6.1.4 Cenário 4 - Análise da Árvore de Classificação gerada pela Execução do *Workflow* SkyMap no Minicluster 89

6.1.5 Cenário 5 - Análise da Árvore de Classificação gerada pela Execução do *Workflow* Constelation Queries no Cluster Petrus 92

6.2 Avaliação dos Modelos Preditivos 95

6.2.1 Um Modelo Para Todos os Cenários 96

6.2.1.1 Teste de Validação do Modelo Preditivo do Cenário 1 com os Dados de Execução do Cenário 4 97

6.2.1.2 Teste de Validação do Modelo Preditivo do Cenário 1 com os Dados de Execução do Cenário 5 99

6.2.2 Um modelo para cada cenário 100

6.2.2.1 Teste de Validação do Modelo Preditivo do Cenário 1 com os Dados de Execução do Cenário 2 100

6.2.2.2 Teste de Validação do Modelo Preditivo do Cenário 1 com os Dados de Execução do Cenário 3 102

6.2.3 Um Modelo Preditivo Agrupado por Aplicação 103

6.2.3.1 Cenário 6 - Análise da Árvore de Classificação gerada pela Execução do *Workflow* SkyMap no Cluster Petrus e no Minicluster 103

6.2.3.2 Teste de Validação do Modelo Preditivo do Cenário 6 com os Dados de Execução do Cenário 1 106

6.2.3.3 Teste de Validação do Modelo Preditivo do Cenário 6 com os Dados de Execução do Cenário 4 107

6.2.4 Um Modelo Preditivo Agrupado por Ambiente de Execução 109

6.2.4.1 Cenário 7 - Análise da Árvore de Classificação gerada pela Execução dos *Workflows* SkyMap e Constellation Queries no Cluster Petrus 109

6.2.4.2 Teste de Validação do Modelo Preditivo do Cenário 7 com os Dados de Execução do Cenário 1 111

6.2.4.3 Teste de Validação do Modelo Preditivo do Cenário 7 com os Dados de Execução do Cenário 5 112

6.2.5 Abordagem Proposta para Criação do Modelo Preditivo 114

6.3 Resumo do Capítulo 115

7 Literatura Relacionada 116

7.1 Abordagens Existentes para Execução de Aplicações Intensivas de Dados . 116

7.1.1 Otimização do MapReduce e SGBDs 117

7.1.2 Otimização de *Workflows* MR Intensivos de Dados em Diferentes Ambientes de Execução 119

7.1.3 Considerações sobre as Abordagens Existentes para Execução de Aplicações Intensivas de Dados 121

7.2 Abordagens Existentes para Otimização de *Workflows* MR através da Configuração de Parâmetros 122

7.2.1 Hadoop 122

7.2.2 Spark 124

7.2.3 Considerações sobre as Abordagens Existentes para Otimização de *Workflows* MR através da Configuração de Parâmetros 124

8 Conclusões 126

8.1 Contribuições 128

8.2 Limitações 129

8.3 Trabalhos Futuros 130

Referências 132

Capítulo 1

Introdução

Processar grandes volumes de dados de forma eficiente tem se tornado uma questão a ser solucionada por empresas, ciência, governo, entre outros. Por exemplo, na área científica como astronomia, biologia e física, o processo experimental tem sido transposto para o ambiente computacional. Com isso, instrumentos como telescópios e sequenciadores de DNA, e as simulações computacionais estão gerando um grande volume de dados a serem analisados pelos cientistas [Deelman et al., 2009].

Realizar a modelagem, implementação e execução dos experimentos científicos que analisam grandes volumes de dados envolve algumas questões de grande complexidade para os cientistas, tais como a diversidade de fontes de dados e formatos dos mesmos, encadeamento dos programas que são usados durante as simulações, entre outros [de Oliveira, 2012] [Liu et al., 2015]. Neste contexto, pode-se destacar duas áreas que têm sido exploradas [Stonebraker et al., 2010] [Deelman et al., 2009] [Liu et al., 2015]: sistemas de gerência de bancos de dados (SGBD) e sistemas de gerência de *workflows* científicos (SGWC). De um lado, em SGBDs relacionais alcança-se desempenho satisfatório para processamento de dados segundo as operações da álgebra relacional, cuja semântica é conhecida. Adicionalmente, otimiza-se o armazenamento e acesso aos dados através de técnicas de distribuição de dados por discos e pela criação de índices de suporte às consultas, exemplificando alguns princípios elementares de melhoria de desempenho em SGBDs [Shasha and Bonnet, 2002]. Por outro lado, cientistas têm usado *workflows* científicos para exprimir e modelar computacionalmente análises e experimentos científicos sobre dados *in-silico* [Deelman et al., 2009] [Romano, 2008] [Zhao et al., 2008].

Um *workflow* científico pode ser definido como o arcabouço funcional que permite a composição de programas em uma sequência de execução com o objetivo de gerar um resultado final. *Workflows* científicos definem um processo computacional composto por

um conjunto de atividades estruturadas segundo uma ordem parcial, obedecendo a uma dependência de dados entre as mesmas. Desta forma, uma atividade recupera o resultado em dados da atividade da qual depende. Finalmente, um *workflow* científico intensivo de dados se caracteriza por ler, processar/analisar e produzir grandes volumes de dados durante a execução.

Devido à complexidade de processamento dos *workflows* científicos e também ao grande volumes de dados envolvido, dificilmente estes podem ser executados em ambientes computacionais com um único processador ou cuja execução seja sequencial, pois a execução do *workflow* neste ambiente poderia levar um tempo de processamento relativamente alto e com isso impedir a continuidade do processo de pesquisa [Gorton et al., 2008] [Gray et al., 2005] [Hey et al., 2009]. Portanto, é importante para uma execução eficiente, a utilização de ambientes paralelos e distribuídos, de grande capacidade de processamento e armazenamento de dados, em conjunto com ferramentas de implantação paralela do *workflow*. Como exemplos destes ambientes distribuídos podem ser destacados os *clusters* de computadores [Dantas, 2005], as grades computacionais [Dantas, 2005] [Foster and Kesselman, 2004] e as nuvens de computadores [Vaquero et al., 2009].

Quanto ao gerenciamento da execução dos experimentos científicos em ambientes paralelos, é necessário que atenda as demandas de desempenho e eficiência de *workflows* intensivos de dados e com isso diversos *frameworks* de processamento distribuído têm surgido para solucionar este problema, como por exemplo: Hadoop [Hadoop,] [White, 2009], Spark [Zaharia et al., 2010b] [Spark,], Storm [Storm,], entre outros. O modelo *MapReduce* (MR), empregado pelos *frameworks* Hadoop e Spark, têm sido muito utilizado na especificação de experimentos em ambientes paralelos, pois permite que os cientistas analisem um grande volume de dados de uma forma mais simples. A partir deste modelo de programação foram criados *frameworks* que permitem a manipulação e análise de grandes volumes de dados de forma paralela e distribuída, além de prover tolerância à falhas, monitoramento e outros serviços como Hadoop [Hadoop,] [White, 2009] e o Spark [Zaharia et al., 2010b]. Dentre estes *frameworks*, o Apache Spark tem se tornado uma plataforma de processamento distribuído muito utilizada pelos cientistas e grandes empresas, por causa da maior variedade de modelos de *workflows* intensivos de dados que ele suporta e também por seu desempenho na execução [Gu and Li, 2013]. Diferentemente do Hadoop, os *workflows* no Spark se beneficiam do armazenamento dos dados intermediários em memória RAM, proporcionando assim um melhor desempenho quando comparado com o Hadoop, se houver disponibilidade de memória para a demanda de execução do workflow em questão [Gu and Li, 2013].

A execução destes experimentos intensivos de dados em ambientes paralelos gerenciados por *frameworks* MR ainda é um problema em aberto e importante para a comunidade científica [Herodotou, 2012] [Wang et al., 2016]. Neste contexto, surgem diversos problemas que devem ser tratados para obter uma execução de um *workflow* científico em *frameworks* MR com um bom desempenho.

1.1 Caracterização do Problema: Parametrização da Execução de um Workflow MapReduce

Para realizar a execução paralela de *workflows* científicos intensivos de dados em *frameworks* MR de forma eficiente é necessário o particionamento dos dados de entrada, de forma a explorar o paralelismo oferecido pelo ambiente. Existem diversas estratégias de particionamento de dados e aspectos como: conhecimento do critério de particionamento por parte da aplicação, tamanho das partições e o balanceamento de carga, entre outros, que interferem no desempenho do *workflow*. Estes aspectos são caracterizados nesta tese como *parâmetros associados aos dados de entrada* relacionados à execução eficiente do *workflow*. As correlações que existem entre estes parâmetros, o *workflow* e o ambiente de execução tornam o ajuste da configuração de tais parâmetros uma tarefa complexa e difícil para o cientista. Portanto, extrair o ajuste destes parâmetros que mais beneficiam o desempenho da execução paralela de um *workflow* é um dos problemas estudados nesta tese.

Além do mais, não somente parâmetros associados ao particionamento de dados são de crucial relevância, mas também, *parâmetros de configuração* do ambiente computacional que levam a um melhor desempenho de execução do workflow, devem ser identificados. Devido a grande quantidade de parâmetros destes *frameworks* a serem configurados e a diversidade de aplicações, surge um problema relacionado ao desempenho da aplicação, pois é difícil extrair a configuração que proporciona o melhor desempenho em um determinado hardware (ambiente computacional). Por exemplo, o Spark possui mais de 180 parâmetros de configuração, os quais podem ser ajustados pelos usuários de acordo com a aplicação para obter um melhor desempenho [Spark,]. Portanto, por um lado, a grande quantidade de parâmetros a serem configurados abre diversas oportunidades para alcançar um ganho significativo de desempenho na execução de um *workflow* específico. Por outro lado, a grande quantidade de parâmetros e as correlações que existem entre eles tornam o ajuste da configuração dos parâmetros uma tarefa complexa e difícil para o cientista.

Os *frameworks* Spark e Hadoop são tipicamente executados em *clusters*, que por sua vez podem ser construídos através de diversas plataformas de nuvem de computadores, como por exemplo a Amazon. *Elastic MapReduce*, por exemplo, é um serviço da Amazon onde um usuário pode solicitar o uso de um *cluster* Hadoop executando em um número qualquer de máquinas do *Elastic Compute Cloud* (EC2). O *cluster* pode ser utilizado para executar tarefas MR intensivas de dados, e após a execução encerrar o uso do *cluster*. O usuário precisa pagar somente pelo tempo em que utilizou as máquinas do *cluster*. Devido a essa facilidade, atualmente plataformas de processamento em nuvem tornam o MR uma solução atrativa para pequenas e médias organizações que precisam processar grandes volumes de dados, mas não possuem recursos computacionais e técnicos como as grandes empresas para tratar o problema. Em poucos minutos, um usuário comum pode solicitar e obter um *cluster* virtual de qualquer tamanho em uma nuvem para tratar a necessidade de processamento dos dados. Portanto a nuvem representa uma grande vantagem e facilidade para o usuário, porém ao utilizar ferramentas de análise dos dados, como *frameworks* MR, questões como particionamento de dados e configuração de parâmetros tornam o trabalho complexo. Existem poucos trabalhos voltados para o tratamento destas questões em um ambiente de nuvem [Herodotou, 2012].

Para demonstrar a complexidade neste contexto, podemos supor um cenário em que um usuário quer executar um *workflow* MR em um *cluster* fornecido por uma plataforma de nuvem computacional. O usuário tem por objetivo executar o *workflow* com o menor custo monetário possível em um tempo de execução máximo de duas horas. A fim de atender a essas exigências, o usuário precisa tomar algumas decisões. Primeiramente, precisa decidir o tamanho do *cluster* e o tipo dos recursos a serem usados no *cluster* a partir de várias opções oferecidas pela plataforma. Para aumentar a complexidade ainda mais, o usuário deve especificar um grande conjunto de configurações dos parâmetros do *framework*, Spark ou Hadoop, como por exemplo o número máximo de tarefas *map* e *reduce* a serem executadas por máquina e o máximo de memória disponível para cada tarefa, entre outros.

De acordo com [Wang et al., 2016] existem duas maneiras de ajustar a configuração de parâmetros em plataformas de processamento paralelo MR como Hadoop e Spark. Na primeira, esses parâmetros são ajustados manualmente através do método "Tentativa e Erro" (*Trial and Error*). Este método é pouco eficiente e oneroso devido ao grande número de parâmetros e as correlações que existem entre eles. Um outro método é proposto em [Herodotou et al., 2011b], baseado em custo de desempenho para ajuste dos parâmetros do *framework* Hadoop. Embora tanto o Hadoop quanto o Spark sejam considerados

frameworks MR, a forma como o Spark é implementado é muito diferente do Hadoop para aplicarmos diretamente este último método no Spark. Em [Wang et al., 2016] argumenta-se que o método baseado em custo é uma abordagem do tipo "caixa branca" (em inglês *white box*) onde é necessário um conhecimento aprofundado do funcionamento interno do sistema como um todo. Considerando-se a complexidade do sistema que envolve um conjunto de softwares como: sistema operacional, *Java Virtual Machine* (JVM) e Spark; e as características de hardware, se torna muito difícil capturar esta complexidade com o método baseado em custo. Estes desafios têm motivado a exploração de métodos de desempenho baseados em técnicas de aprendizado de máquina, os quais são métodos "caixa preta" (em inglês *black box*) [Wang et al., 2016]. Comparando com as abordagens existentes, métodos "caixa preta" possuem duas vantagens principais. Primeiro, as recomendações para um auto-ajuste dos parâmetros são baseadas nas observações reais de desempenho do sistema na execução de um determinado *workflow* em um determinado *cluster*. Segundo, métodos "caixa preta" geralmente são mais simples de construir comparados com métodos "caixa branca", pois não é necessário nenhuma informação detalhada sobre as características internas do sistema (software e hardware).

Apesar de existirem algumas soluções para obtenção de uma configuração dos parâmetros que levem a uma execução paralela eficiente de *workflows* em *frameworks* MR, nenhuma delas leva em consideração todas estas características importantes. Por exemplo, algumas soluções não abordam *workflows* intensivos de dados e consequentemente os aspectos relacionados ao particionamento dos mesmos como acontece com o Starfish [Herodotou et al., 2011b], o Stubby [Lim et al., 2012] e a solução descrita em [Wang et al., 2016]. Desta forma, a questão de pesquisa a ser investigada nesta tese é:

"No contexto de frameworks MapReduce para experimentos científicos em larga escala executados em um cluster shared-nothing, é possível definir uma abordagem que aponte para uma configuração de execução considerando as características do ambiente e do particionamento dos dados de entrada que maximize o desempenho da execução destes experimentos por parte dos cientistas?".

1.2 Abordagem Proposta

Como o cenário acima demonstra, os usuários estão diante de problemas de ajustes da execução do *workflow* que envolvem configuração de parâmetros e dados para alcançarem um tempo de execução desejado. A hipótese geral deste trabalho considera que é possível

realizar a execução paralela de *workflows* científicos de maneira eficiente em um ambiente dedicado e homogêneo (*cluster*) através do particionamento dos dados de entrada e da escolha correta dos parâmetros relacionados ao *framework* de execução *MapReduce*. No cenário de experimentos científicos em larga escala, executar um experimento de maneira eficiente implica fazer-se valer das características do ambiente de *cluster*, do *framework* MR e do conjunto de dados de entrada do *workflow* para gerar uma configuração de parâmetros de execução que melhor se adeque ao cenário. Técnicas de mineração de dados aliadas a um modelo de custo específico para o ambiente de *cluster* permitem a escolha de uma configuração de parâmetros de execução que levam ao menor tempo de processamento do *workflow*. Assim, a adoção desta abordagem proposta permite realizar a execução paralela de *workflows* científicos de forma sistemática, i.e., de modo que o cientista não tenha que se preocupar com configurações do ambiente, dos dados, do *framework* ou com o *workflow*, mas somente com a especificação do experimento.

Como base para a formulação desta hipótese e das propostas que são apresentadas nesta tese foram realizados diversos experimentos de execução paralela de *workflows* científicos em um ambiente de *cluster* [de Oliveira et al., 2014] [de Oliveira et al., 2015] descritos no Capítulo 3. Estes experimentos forneceram subsídios para uma proposta mais consistente no que se refere à execução paralela de *workflows* MR.

Nesta tese, é proposta uma abordagem de otimização que pode ser aplicada no ajuste da configuração dos parâmetros de execução de *workflows* MR em *clusters* computacionais. A abordagem é baseada em (i) coletar o tempo de execução do *workflow* utilizando diversos valores na configuração dos parâmetros, (ii) aplicar técnicas de aprendizado de máquina a fim de encontrar os valores e parâmetros que executam o *workflow* de forma eficiente e (iii) utilizar as mesmas técnicas para gerar o modelo preditivo para conhecer previamente o desempenho de uma nova configuração.

Para demonstrar a abordagem para tratar as possíveis configurações dos parâmetros, vamos supor um *workflow* W que deve ter os parâmetros ajustados. Inicialmente, o sistema possui conhecimentos limitados sobre as configurações possíveis a serem usadas na execução do *workflow*, e uma configuração inicial C_0 é selecionada. Estas execuções se repetem para diversas configurações $C_1, C_2, ..., C_n$. Após as execuções, aplica-se técnicas de aprendizado de máquina para conhecer os parâmetros e valores que determinam o desempenho do *workflow*. Após isso, aplica-se a configuração C_{opt} que possui os parâmetros e valores que levam a um desempenho eficiente da execução do *workflow* W. Com isso, a abordagem pode ser usada para diferentes *workflows*, dados e recursos computacionais.

Para tal, foi implementado o *Spark Configuration Engine* (SPACE), que implementa a abordagem proposta para prover a configuração dos parâmetros e captura de proveniência de *workflows* MR executados em paralelo em *clusters*. Esta abordagem é baseada em um modelo de custo que considera o tempo de execução do *workflow*. A seguir são resumidas as principais contribuições relacionadas a área de atuação deste trabalho:

i. Levantamento do estado da arte na execução paralela de aplicações intensivas de dados e na otimização da execução paralela de *workflows* científicos MR sobre *clusters* computacionais;

ii. Levantamento de desafios a partir de uma análise experimental utilizando um *workflow* da astronomia e publicação dos resultados iniciais descritos no Capítulo 3;

iii. Definição dos parâmetros que contribuem para execução eficiente dos *workflows* intensivos de dados através dos experimentos descritos na Seção 5.3. Os parâmetros estão relacionados com o particionamento dos dados de entrada e o *framework* MR;

iv. Definição da abordagem baseada em coleta de dados, treinamento do modelo preditivo e predição de parâmetros;

v. Uma avaliação experimental de *workflows* reais da área da astronomia em dois *clusters* no Laboratório Nacional de Computação Científica (LNCC), mostrando os benefícios da abordagem proposta;

vi. Protótipo que implementa a abordagem proposta denominado SPACE.

O desenvolvimento desta tese recebeu o apoio da equipe do (LIneA[1]) que nos forneceu o *workflow* SkyMap e os *datasets* que foram utilizados nos experimentos descritos nos capítulos 3 e 6. O LIneA é Laboratório Interinstitucional de e-Astronomia que envolve o Observatório Nacional (ON), o Laboratório Nacional de Computação Científica (LNCC), e a Rede Nacional de Ensino e Pesquisa (RNP), que foi criado com a finalidade de dar suporte à participação brasileira em levantamentos astronômicos gerando grandes volumes de dados. O *workflow Constelation Queries* utilizado nos experimentos descritos no Capítulo 6 foi criado pela equipe do *Extrem Data Lab* (DEXLLAB) do LNCC.

[1]http://www.linea.gov.br

1.3 Organização da Tese

O restante da tese está estruturado da seguinte forma: o Capítulo 2 apresenta os conceitos considerados relevantes para o melhor entendimento deste trabalho. No Capítulo 3 são descritos os experimentos realizados que demonstram a importância do armazenamento e particionamento dos dados. O Capítulo 4 apresenta a abordagem proposta, utilizando técnicas de aprendizado de máquina para a configuração dos parâmetros na execução de *workflows*, desenvolvida para melhorar o desempenho dos mesmos. O Capítulo 5 descreve a modelagem experimental do estudo de caso considerando os *workflows* científicos, ambientes de execução e o subconjunto de configurações de parâmetros relacionados aos dados de entrada e ao *framework* MR. O Capitulo 6 apresenta a avaliação experimental da abordagem proposta contemplando a avaliação de desempenho. No Capítulo 7 são descritos os trabalhos que estão relacionados aos desafios discutidos nesta tese. Finalmente, o Capítulo 8 conclui a tese apresentando os principais resultados obtidos e os trabalhos futuros.

Capítulo 2

Referencial Teórico

Este capítulo tem como objetivo fornecer a fundamentação teórica e conceitual relacionada à execução de *workflows* científicos intensivos de dados e as ferramentas utilizadas no tratamento destas aplicações. A Seção 2.1 caracteriza o conceito de *workflows* científicos. A Seção 2.2 descreve algumas estratégias de particionamento de dados de entrada dos *workflows* científicos relevantes para a compreensão desta tese. A Seção 2.3 define algumas características dos *frameworks* MR, Hadoop e Apache Spark, apresentando sua arquitetura. Por fim, são descritos na Seção 2.4 alguns conceitos e algoritmos de Aprendizado de Máquina (AM) que foram utilizados na abordagem proposta apresentada no Capítulo 4 e como ferramenta de apoio na análise dos experimentos.

Como discutido no Capítulo 1, alcançar uma execução eficiente em *frameworks* MR envolve ajustar diversos parâmetros, levando em consideração as características tanto da aplicação quanto do ambiente arquitetural de execução. Nesta tese, são exploradas aplicações sensíveis ao critério de particionamento dos dados e alguns parâmetros de configuração dos *frameworks* MR. O conjunto de parâmetros de otimização considerados nesta tese são apresentados nesta seção destacando a relevância dos mesmos na execução de *workflows* científicos intensivos de dados de forma eficiente.

2.1 *Workflows* Científicos

Um *workflow* é um conjunto de atividades que realizam o processamento de dados de acordo com um conjunto de regras. Tais regras são representadas pela modelagem do *workflow* que determinam a sequência de atividades e também como se dá o fluxo e a dependência de dados entre elas. Pode-se dizer que existem dois tipos de *workflows*: científico e empresariais. O último é resumidamente um processo composto de um conjunto de

atividades para alcançar um determinado objetivo dentro de uma empresa, normalmente dentro do contexto de estrutura organizacional definindo regras funcionais e relacionamentos [Liu et al., 2015].

Os *workflows* científicos são tipicamente utilizados para modelar e executar experimentos científicos. Através destes, pode-se automatizar a execução das atividades de processamento de dados científicos, resultando em uma redução no tempo total de execução da aplicação. Um *workflow* científico pode possuir um ou mais sub-*workflows* que são constituídos de um subconjunto de operações com as respectivas dependências de dados.

Os *workflows* científicos podem ser representados de diversas formas [Liu et al., 2015]. A mais comum e utilizada neste trabalho, é a modelagem através de um Grafo Direcionado (GD), onde, os vértices correspondem às atividades de processamento e as arestas representam a dependência de dados entre as atividades. Adicionalmente, muitas aplicações são representadas por *workflows* científicos que seguem o modelo de um Grafo Acíclico Direcionado (GAD).

Cada atividade possui o esquema dos dados, dados de entrada, operação e recursos computacionais de processamento. O esquema de dados fornece as informações relacionadas ao formato esperado pela atividade, e a operação define quais serão os métodos de processamento sobre os dados (ex: filtrar, agregar, *map*, *reduce*). A atividade do *workflow* em execução é denominada tarefa. Se uma atividade estiver sendo executada em paralelo, então existem várias tarefas para uma mesma atividade e cada tarefa estará atuando sobre um conjunto de dados diferente [Liu et al., 2015].

Portanto, conforme definido em [de Oliveira, 2012], um *workflow* é representado por um GAD denominado $W = (A, Dep)$, onde A é um conjunto de atividades modelados pelos nós ou vértices que representam as etapas do *workflow* a serem executados (em paralelo ou sequencialmente) e um conjunto de arestas (Dep) que representam as dependências de dados entre as atividades de A.

Dada uma atividade $a_i \in A$, seja $D(a_i)$ o conjunto possível de dados de entrada para a atividade a_i. Da mesma forma, considere-se $O(a_i)$ como sendo o conjunto possível de dados de saída produzidos por a_i. Uma dependência entre duas atividades a_i, a_j é expressa por: $(a_i, a_j) \in Dep \leftrightarrow \exists\, d_k \in D(a_j) \mid d_k \in O(a_i)$.

Uma vez que cada atividade pode ser paralelizada, esta pode ser dividida em diversas tarefas. Desta forma, um conjunto $T = \{t_1, ..., t_n\}$ de tarefas é criado para a execução paralela das atividades do conjunto A. Uma atividade a_i pode ser execu-

tada por várias tarefas, onde cada tarefa t_j irá processar um conjunto de dados diferente [Ogasawara et al., 2011].

2.1.1 Sistemas de Gerenciamento de *Workflows* Científicos

Como definido em [Liu et al., 2015] um sistema de gerenciamento de *workflows* científicos (SGWC) é um sistema que modela, constrói e gerencia a execução de *workflows* em diversos ambientes computacionais, tanto sequenciais quanto paralelos. Para realizar a execução de um *workflow* em um determinado ambiente, um SGWC produz o Plano de Execução do *Workflow* (PEW) de uma determinada aplicação, a partir da definição das atividades, operações e dependências de dados, ou seja, o PEW representa o GD do *workflow* que pode ser tanto acíclico quanto cíclico, e paralelo ou sequencial. Além disso, o PEW proporciona diretivas de execução e decisões de otimização para a execução das atividades de acordo com as dependências de dados, recursos de processamento e armazenamento, e paralelismo. Em suma, esta produção do PEW é similar à compilação de um programa e geração do código executável.

Vários SGWC como Pegasus [Deelman et al., 2005] [Deelman et al., 2015], Swift [Zhao et al., 2007], Kepler [Altintas et al., 2004], Taverna [Oinn et al., 2004], Chiron [Dias et al., 2013] entre outros têm sido muito utilizados em diversas áreas científicas como astronomia, biologia, física e engenharia computacional.

2.1.2 Execução Paralela de *Workflows* Científicos

Em [Hey et al., 2009] afirma-se que a ciência intensiva de dados está surgindo como o quarto paradigma científico, sendo que os outros são as ciências empírica (i.e. descrição de fenômenos naturais baseando-se em evidências), teórica (i.e. lei de Newton, Moore, etc) e computacional (i.e. simulações científicas). Diversas áreas científicas têm utilizado as tecnologias computacionais para expressarem suas aplicações, tais como: astronomia, meteorologia, bioinformática, entre outras. A classe de aplicações intensivas de dados se caracterizam por processar e produzir grandes volumes de dados [Deelman et al., 2009] de alta importância científica e econômica, e que requerem investimentos no seu processamento eficiente. Por exemplo, na área da astronomia, a cada ano o *Large Synoptic Survey Telescope* (LSST[1]) produz novas imagens e dados na ordem de petabytes, e o telescópio *Square Kilometer Array* (SKA[2]) irá gerar aproximadamente 200GB de dados por segundo,

[1] www.lsst.org
[2] www.skatelescope.org

sendo necessário uma grande capacidade de processamento destas imagens do céu.

A paralelização de um *workflow* científico intensivo de dados tem como objetivo obter um melhor desempenho na execução dos *workflows*, pois, principalmente na execução desta classe de *workflows*, as técnicas de paralelismo utilizando infraestruturas paralelas/distribuídas contribuem para uma diminuição no tempo de processamento. Basicamente, a paralelização do *workflow* consiste em transformar um PEW sequencial em um PEW paralelo, através da identificação de tarefas que podem ser executadas em paralelo. O PEW paralelo possui uma semelhança muito grande com o conceito de *Query Execution Plan* (QEP) apresentado por [Özsu and Valduriez, 2011] no contexto de bancos de dados distribuídos.

Em [Liu et al., 2015] é definido um tipo de paralelismo de *workflows*, que executa atividades iguais em paralelo sobre um conjunto de dados particionado, caracterizando o paralelismo de dados. Neste trabalho é considerado o paralelismo de dados para um melhor aproveitamento do particionamento dos dados e consequentemente melhor desempenho. Paralelismo de dados é conhecido por possuir múltiplas tarefas realizando a mesma atividade, cada uma sobre um conjunto de dados diferente, e pode ser estático, dinâmico ou adaptativo [Pautasso and Alonso, 2006], sendo que, nesta tese é usado o estático.

- Estático: o número de partições de dados é calculado antes da execução e se mantém o mesmo ao longo desta;
- Dinâmico: o número de partições de dados é determinado em tempo de execução;
- Adaptativo: determina o número de partições de acordo com o ambiente de execução.

2.2 Particionamento dos Dados

Afim de realizar a execução paralela eficiente de *workflows* científicos intensivos de dados em *clusters* é necessário o particionamento dos dados de entrada. Dessa forma, os dados devem ser divididos em partes menores, denominadas partições. Existem diversas estratégias de particionamento de dados relacionado com a classe de *workflows* científicos considerado e alguns aspectos como o conhecimento do critério de particionamento, tamanho das partições e o balanceamento de carga interferem no desempenho de execução do respectivo *workflow*.

Nesta tese são utilizados *workflows* da astronomia no estudo de caso para avaliar a abordagem proposta. Os dados de entrada destes *workflows* são dados espaciais, e com isso, as estratégias de particionamento descritas nesta seção estão relacionadas com o particionamento de dados com esta característica. Aplicações de outras áreas, por exemplo processamento de imagens, também utilizam dados espaciais na análise de imagens médicas, tomografias, radiografias. Portanto, os dados podem ser particionados de acordo com as estratégias de particionamento:

- ***Equi-depth*** [Muralikrishna and DeWitt, 1988]: Algoritmos de particionamento que dividem os dados de acordo com as posições dos elementos, procurando criar cada partição com um conjunto de elementos vizinhos, ou seja, pertencente ao intervalo determinado pela partição. Nesta tese foi utilizado o FRAgmeNtador de Catálogos Espaciais (FRANCE) [Daniel Gaspar, 2014], que é um algoritmo iterativo para particionar dados que implementa a estratégia *Equi-depth*;
- **Randômico**: Os dados são divididos em partições de forma aleatória e sem um critério estabelecido. Nesta tese a aplicação desta estratégia é denominada AdHoc;
- **Hierárquico** [Samet, 1984]: Os dados são divididos seguindo uma hierarquia. Assim como as estratégias baseadas em intervalos, as estruturas de dados hierárquicas têm se tornado uma importante técnica de particionamento de dados em áreas como computação gráfica, processamento de imagens e astronomia. Na astronomia, por exemplo, este tipo de particionamento é baseado no princípio de decomposição recursiva dos dados de acordo com a região do espaço. Nesta tese foi utilizado o algoritmo QuadTree [Samet, 1984] para particionar dados seguindo a estratégia Hierárquica;

A seguir são descritos os algoritmos de particionamento de dados utilizados nesta tese, que são: FRANCE [Daniel Gaspar, 2014], AdHoc e QuadTree [Samet, 1984]. Para simplificar, nas análises experimentais é utilizado a nomenclatura FRANCE, AdHoc e QuadTree.

2.2.1 FRANCE

Em [Daniel Gaspar, 2014], é proposto o FRAgmeNtador de Catálogos Espaciais (FRANCE), que é um algoritmo iterativo para particionar dados em histogramas *equi-depth*. O FRANCE calcula iterativamente os pontos de fragmentação (intervalos) segundo uma das dimensões espaciais, se preocupando em manter as partições com uma quantidade

equivalente de elementos e considerando a vizinhança dos elementos, mesmo em catálogos com densidade não uniforme (como o espaço).

No FRANCE, uma partição P_i de um conjunto de dados espacial D representa um subconjunto de D de tal forma que todos os objetos em P_i encontram-se entre dois pontos de fragmentação espaciais fi e f_{i+1}. Adicionalmente, a coordenada espacial associada a f_i tem um valor inferior à coordenada espacial associada à f_{i+1}.

As partições são geradas iterativamente até que a quantidade de elementos seja próxima em δ do balanceamento ideal. Considera-se ideal o balanceamento em que as partições de dados apresentam $\frac{N}{\beta}$ elementos, onde N é o total de elementos e β a quantidade de partições. Portanto, dadas duas partições P_i e P_j , dizemos que estão balanceadas se o número de objetos em P_i , $|P_i|$, é aproximadamente igual ao de P_j, $|P_j|$. Logo, $|P_i| + \delta = |P_j|$, tal que δ é um valor tolerável na diferença da quantidade de objetos das duas partições. O valor de δ pode ser alterado no algoritmo e, nos experimentos descritos no capítulo 6, no FRANCE, ele se apresenta como $0,5\%$ da quantidade de objetos desejada. Logo, todas as partições geradas terão $(\frac{N}{\beta}) \pm 0.005 \times \frac{N}{\beta}$ elementos. Com esse δ em $0,5\%$ temos uma variação muito pequena no tamanho das partições.

O FRANCE recebe como entrada um *dataset* D com N elementos e a quantidade de partições desejada β. Na primeira iteração o *dataset* é dividido em duas partições, onde cada partição contém uma metade dos dados. Uma variável *diff* é definida como metade do tamanho de cada partição. Para cada partição, enquanto não atingir um particionamento com $\frac{N}{\beta}$ elementos (com uma tolerância de δ definida a priori), ela será reduzida por *diff*. Para cada passo, $diff = \frac{diff}{2}$. Quando a partição tiver a quantidade de elementos dentro do limite de δ, ela será fixada. O algoritmo executa recursivamente no espaço disponível entre as partições fixas. Nesta etapa o FRANCE produz como saída os pontos de fragmentação F onde ocorreram as divisões do espaço, ou seja, os valores que determinam as coordenadas inicial e final de cada partição. Vale observar que atualmente o particionamento do FRANCE é feito em apenas uma dimensão espacial.

Uma vez que os pontos de fragmentação F tenham sido definidos, podem-se gerar as partições de dados a partir da associação de cada objeto do catálogo a um intervalo. As partições assim criadas podem ser distribuídas pelos nós do *cluster* para armazenamento e processamento. É importante ressaltar que como o FRANCE precisa da visão completa dos dados, atualmente a sua execução não é paralela e executa de forma centralizada em apenas uma máquina.

O funcionamento do FRANCE é ilustrado na Figura 2.1, onde os retângulos vermelhos

correspondem às partições com uma quantidade de elementos maior que $\frac{N}{\beta} + \delta$ e que ainda precisam ser divididas para conter o número de objetos ideal. Os retângulos verdes representam as partições que contêm a quantidade de objetos dentro do limite de $\frac{N}{\beta} + \delta$ e não precisam mais ser subdivididas, ou seja, elas são partições fixadas.

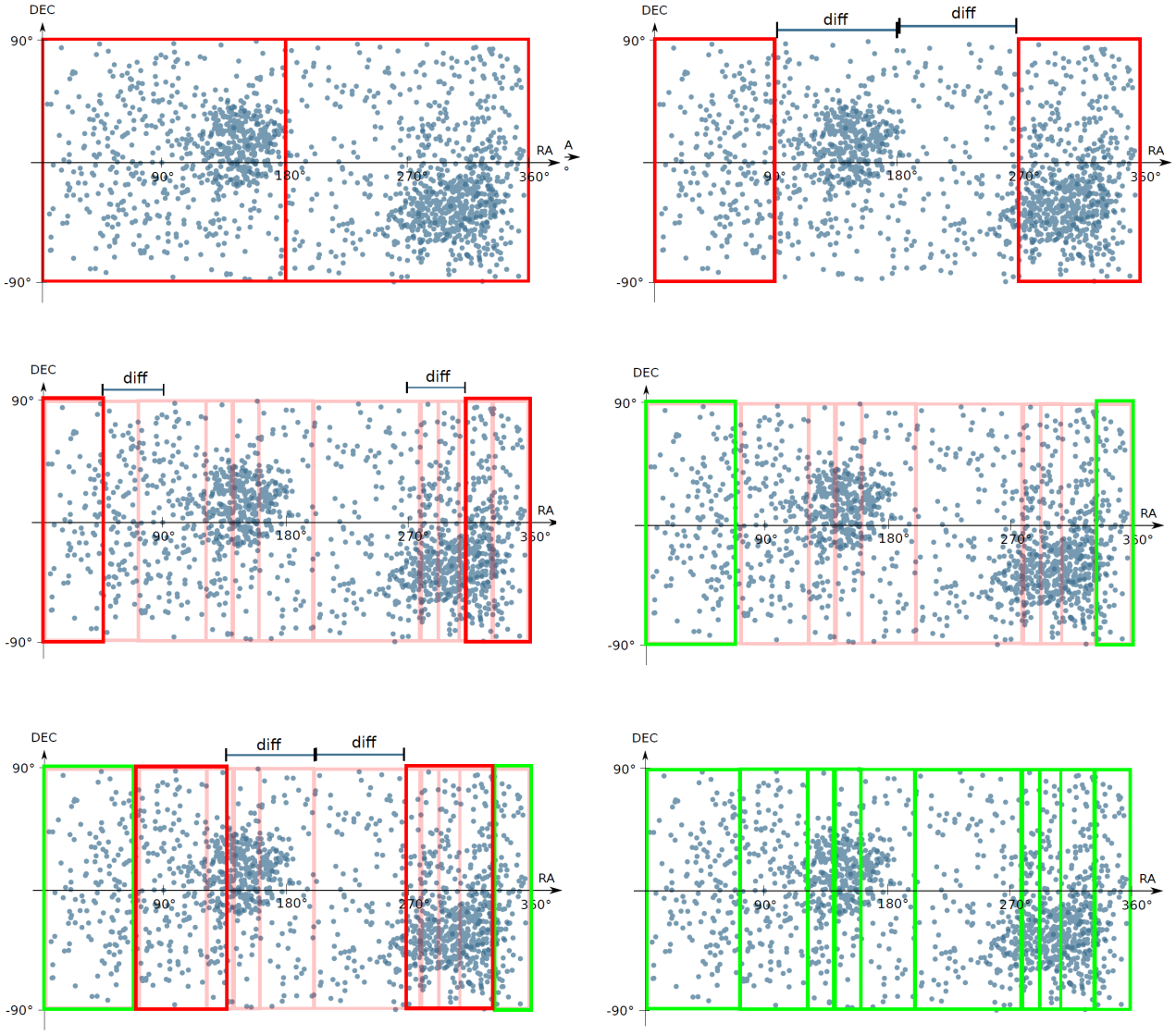

Figura 2.1: Passo a passo do FRANCE [FREIRE, 2016]

2.2.2 QuadTree

O algoritmo QuadTree [Samet, 1984] segue o modelo de particionamento em árvore (hierárquico), que é um modelo frequentemente utilizado em problemas 2D ou 3D, pois estes problemas tendem a alocar uma grande quantidade de memória devido à sua propensão em exigir muitos e grandes vértices [Zäschke et al., 2014].

A estrutura da árvore é criada pela subdivisão sucessiva de quadrantes em subquadrantes, representando uma área no plano. Isto significa que, no caso de duas dimensões, cada vértice interno da árvore tem quatro filhos, um para cada quadrante. Essa subdivisão pode ser exemplificada através da Figura 2.2. A Figura 2.2 mostra que a numeração dos quadrantes é feita em sentido horário partindo do quadrante superior esquerdo: 1 (NW), 2 (NE), 3 (SE), e 4 (SW). Também é apresentado na Figura 2.2 que ocorre uma subdivisão recursiva do espaço, no qual os locais em amarelo correspondem às áreas ocupadas, ou seja, que contêm objetos e que precisam ser subdivididas. Estas subdivisões acontecem até que um determinado nível da árvore pré-determinado seja alcançado. Partições que não possuem elementos não são subdivididas, como demonstrado no quadrante 1. As áreas que possuem muitas subdivisões são áreas mais densas do espaço.

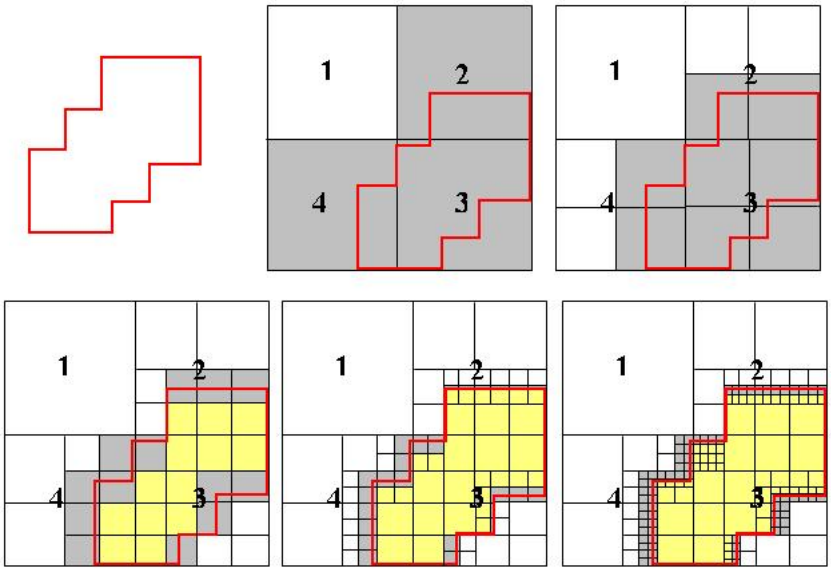

Figura 2.2: Etapas da Criação de uma Quadtree [FREIRE, 2016]

Pode-se observar que este modelo produz partições com tamanhos diferentes (desbalanceadas), pois o ponto de fragmentação é pela dimensão e não pela quantidade de elementos. Com isso, uma partição pode possuir mais elementos do que outra partição, como pode ser observado na Figura 2.2. Por exemplo, na área da astronomia o espaço analisado é modelado na forma de um retângulo desbalanceado, pois o espaço não possui uma distribuição uniforme dos corpos celestes, tendo regiões muito densas e outras muito esparsas.

2.2.3 AdHoc

Na estratégia de particionamento AdHoc o *dataset* é dividido em partições com a mesma quantidade de elementos. Neste particionamento não são consideradas as dimensões espaciais na divisão dos objetos pelas partições, como nas estratégias Equi-depth ou Hierárquica, e com isso uma partição pode ter elementos independentemente da região do espaço. Inversamente, no FRANCE uma partição irá conter somente os elementos da região determinada pelos pontos de fragmentação. O algoritmo recebe como entrada um *dataset* com N elementos e a quantidade de partições desejada β. O *dataset* é dividido em β partições, cada uma com $\frac{N}{\beta}$ elementos. As partições assim criadas podem ser distribuídas pelos nós do cluster para armazenamento e processamento.

A estratégia randômica possui um critério simples de divisão dos dados e produz um particionamento balanceado. Porém nesta estratégia não é conhecida a semântica dos dados de cada partição como acontece nas outras duas estratégias de particionamento, ou seja, para uma determinada partição gerada pelas estratégias Equi-depth ou Hierárquica é possível afirmar que todos os elementos desta estão entre os dois pontos de fragmentação que definem a mesma. Na estratégia randômica não é possível fazer esta afirmação, uma vez que os elementos são divididos pelas partições sem levar em consideração a posição (coordenada) que ocupam no espaço.

2.2.4 Parâmetros relacionados ao particionamento dos dados

Um dos objetivos desta tese é apontar que é importante para o desempenho de *workflows* intensivos de dados realizar o particionamento dos dados de acordo com uma determinada semântica. Para tal, alguns parâmetros de execução do *workflow* relacionados aos dados de entrada foram avaliados nesta tese, como:

- **Tamanho do *dataset***: É o volume total de dados da aplicação;
- **Estratégia de Particionamento do *dataset***: Este parâmetro define a estratégia de particionamento que pode ser adotada. Nesta tese foram avaliadas 3 estratégias, que são: Equi-depth, Hierárquica e Random;
- **Tamanho das partições**: Este parâmetro define o tamanho das partições nas quais o *dataset* será dividido;

Nas análises experimentais serão apresentados também, para uma melhor avaliação, o *número de partições*, que pode ser derivado do Tamanho do *dataset* e tamanho da partição em uma estratégia que produza partições balanceadas, como por exemplo, Equi-depth e Random. Ainda, nestas estratégias de particionamento, o número de elementos por partição é considerado.

2.3 *MapReduce*

O *MapReduce* [Dean and Ghemawat, 2004] é um modelo de programação proposto pelo Google para facilitar o processamento de grandes volumes de dados. Basicamente, *MapReduce* aplica o método de divisão e conquista, reduzindo problemas complexos em vários problemas menores, até que estes possam ser resolvidos trivialmente. Tais problemas podem ser executados em paralelo e suas soluções parciais combinadas em seguida resultando na solução final para o problema. Este método de divisão e conquista é implementado por duas etapas: *Map* e *Reduce*.

O modelo MR permite que os cientistas analisem dados da ordem de *TeraBytes* (TB) de uma forma menos complexa [Dean and Ghemawat, 2004]. Com isso, um grande número de aplicações reais tem sido expressas através deste modelo de programação [Dean and Ghemawat, 2004]. A operação de *Map* recebe um par <chave,valor> e gera um conjunto intermediário de dados, também no formato <chave,valor>. Em geral a operação de *Map* é usada para encontrar algo, dado algum valor de entrada. A operação de *Reduce* é executada para cada chave intermediária, com todos os conjuntos de valores intermediários associados àquela chave combinados.

A partir deste modelo de programação foram criados *frameworks* que permitem a manipulação e análise de grandes volumes de dados de forma paralela e distribuída, além de prover tolerância à falhas, monitoramento e outros serviços. Alguns desses *frameworks* que implementam o *MapReduce* são: Hadoop [White, 2009], Spark [Zaharia et al., 2010b] [Spark,], Dryad [Isard et al., 2007], Phoenix [Ranger et al., 2007], Mars [He et al., 2008] e Sphere [Gu and Grossman, 2009]. A seguir são resumidos o Hadoop e o Spark, pois realizam a implementação do *MapReduce*, são muito utilizados atualmente, possuem o código aberto e foram utilizados nos experimentos descritos nesta tese, fazendo parte da validação do problema e avaliação da abordagem proposta.

2.3.1 Hadoop

O *framework* Hadoop [White, 2009] é uma implementação do algoritmo *MapReduce* desenvolvido em Java pela *Apache Foundation*, voltada para soluções de alto desempenho baseadas em *cluster* com o propósito de processar grandes volumes de dados aplicando o paralelismo.

HDFS [HDFS, 2012] é um sistema de arquivos distribuído nativo do Hadoop, que permite o armazenamento e transmissão de grandes conjuntos de dados em máquinas de baixo custo. Além disso, possui mecanismos de replicação para tolerar falhas. O HDFS é uma implementação de código aberto, baseado no GFS (*Google File System*) [Ghemawat et al., 2003], e oferece suporte ao armazenamento e ao processamento de grandes volumes de dados em um ambiente de processamento paralelo. Essa quantidade de dados pode chegar à ordem de *petabytes* (PB), tal quantidade não seria possível armazenar em um sistema de arquivos tradicional.

O HDFS tem como princípio promover tolerância, detecção e recuperação automática de falhas. Essas funcionalidades permitem que, no caso de alguma máquina do conjunto vir a falhar, a aplicação como um todo não seja interrompida, pois o processamento que estava sendo realizado nessa máquina poderá ser reiniciado, ou no pior caso, repassado para uma outra máquina disponível de forma transparente ao usuário.

O HDFS gerencia os dados distribuídos pelos nós e armazenados em unidades chamadas de *blocos*. Cada *bloco* representa uma unidade física de dados, por exemplo de 128MB de tamanho. Arquivos armazenados em HDFS são particionados em unidade de *blocos*, distribuídos e replicados pelas máquinas do sistema. O fator de replicação padrão é três, significando que um *bloco* é armazenado em três nós separados.

Quanto ao aspecto de infraestrutura do Hadoop, há dois tipos de recursos (máquinas): *mestre* e trabalhadores. Resumidamente, os dados de entrada estão divididos no HDFS, e tendo a localização dos dados como base o nó *mestre* distribui o processamento para cada trabalhador na fase *Map*. Em seguida, o *mestre* obtém as respostas de cada trabalhador e as combina para formar a saída da etapa de *Reduce*.

No Hadoop, enquanto as tarefas são executadas pelos trabalhadores, o *mestre* fica monitorando a execução. Se um trabalhador falhar, o *mestre* automaticamente verifica se um outro trabalhador pode executar a tarefa que falhou. Para que isto seja realizado pelo *mestre*, as aplicações devem especificar quais são seus dados de entrada e a localização dos dados de saída.

Com isso, uma tarefa *MapReduce* deve ser configurada de acordo com algumas propriedades, tais como: quantidade de *maps* e *reduces*, localização dos dados de entrada e saída, localização das classes Java e classes Hadoop, entre outros. Estes parâmetros são usados pelo *framework* Hadoop para escalonar as tarefas *MapReduce* durante sua execução.

2.3.2 *Framework* Spark

O Spark [Zaharia et al., 2010b] é um *framework* para processamento paralelo de aplicações que possui a característica de reutilizar um determinado conjunto de dados na execução de várias atividades subsequentes do *workflow*. Este aspecto está intimamente relacionado à classe de aplicações denominadas iterativas, embora também beneficie outros tipos de aplicações que possuam o princípio de re-utilização de um conjunto de dados.

Quanto a paralelização, todos os aspectos relacionados ao gerenciamento de uma execução paralela/distribuída é fornecida pelo Spark e o HDFS pode ser utilizado no armazenamento de dados de entrada/saída para o Spark. Neste caso, pode-se realizar uma integração da aplicação/*workflow* que se deseja executar em paralelo nestas ferramentas. Neste cenário, o Spark fornece o gerenciamento paralelo da execução do *workflow*, e o HDFS pode ser utilizado para armazenar os dados de entrada e saída necessários para o processamento.

Uma das diferenças entre o Spark e o Hadoop é a utilização da memória principal, através do RDD (Resilient Distributed Dataset) [Zaharia et al., 2012], para o armazenamento dos dados intermediários das tarefas de um *workflow*. No Hadoop estes dados obrigatoriamente devem ser persistidos em um sistema de arquivos, sendo o HDFS o mais utilizado. Consequentemente, isto pode dar um ganho de desempenho ao Spark devido a memória principal possuir uma maior velocidade de acesso do que a memória secundária, considerando que os dados caibam na memória principal.

Outra característica do Spark frente ao Hadoop é a maior variedade de operações fornecidas. No Hadoop as operações se resumem em *map* e *reduce*. Por outro lado o Spark suporta operações como: *filter*, *join* e *groupBy*, além das funções *map* e *reduce*, entre outras. Esta maior variedade de operações permite uma melhor adaptação de aplicações comparado ao modelo *MapReduce*, além de permitir que aplicações que não podem ser adaptadas ao *MapReduce* sejam adaptadas ao Spark e consequentemente beneficiadas pelo paralelismo.

Uma aplicação Spark [Zaharia et al., 2010b] consiste de um único processo *driver* e um conjunto de processos *executors* distribuídos pelos nós do cluster. O *driver* é o processo que é responsável pelo gerenciamento da execução da aplicação e executa no nó mestre. Cada processo *executor* é responsável pela execução do trabalho nas máquinas trabalhadoras, na forma de tarefas. Cada *executor* deve ser configurado com o número de *cores* para executar as tarefas, e poderá executar várias concorrentemente durante a execução da aplicação. Inicializar estes processos no *cluster* é responsabilidade do *cluster manager* em questão, que podem ser: Yet Another Resource Negotiator (YARN) [Vavilapalli et al., 2013], Mesos [Hindman et al., 2011], ou Spark Standalone [Spark,] [Zaharia et al., 2010b]. O Spark possui diversos parâmetros que controlam a utilização dos recursos durante a execução de uma aplicação. Estas configurações de parâmetros podem se comportar de forma diferente dependendo do *cluster manager*. Neste trabalho os experimentos foram realizados utilizando o YARN como *cluster manager*, devido ao ambiente disponibilizado para este trabalho de tese. Resumidamente, o YARN realiza o gerenciamento de recursos e monitora todos os recursos no *cluster*, e garante que esses recursos sejam alocados de modo dinâmico para realizar as tarefas no trabalho de processamento. O YARN pode gerenciar as cargas de trabalho do Hadoop MapReduce e do Apache Spark, como também de outras estruturas distribuídas, como o HBase [George, 2011].

Os dois principais recursos considerados pelo Spark e o YARN são CPU e memória. Recursos de I/O, como disco e rede, também impactam o desempenho do Spark, porém tanto o Spark como o YARN atualmente não gerenciam estes recursos ativamente. O Spark e o YARN possuem como base a execução em um *cluster* onde a latência de rede não impacta no desempenho, e aplicações Spark tradicionalmente realizam pouco acesso a disco. Assim, neste trabalho foram avaliados somente os parâmetros do Spark relacionados a CPU e Memória, conforme definidos a seguir:

- **Quantidade de *Executors***: Como já descrito anteriormente, o *executor* é um processo responsável pela execução do trabalho nas máquinas trabalhadoras, na forma de tarefas. Este parâmetro define quantos *executors* (processos) serão criados e alocados para execução da aplicação no Spark;

- **Quantidade de *cores/executor***: Cada *executor* em uma aplicação possui o mesmo número fixo de *cores*. Esta propriedade controla o número de tarefas concorrentes (*threads*) que um *executor* pode executar e que efetivamente executam as tarefas da aplicação. Por exemplo, se for 5, significa que cada *executor* pode

executar no máximo 5 tarefas ao mesmo tempo;

- **Quantidade de memória/*executor***: Esta propriedade controla o tamanho fixo de memória de cada *executor*.

Caso os parâmetros de configuração não sejam especificados pelo cientista na execução do *workflow*, então os valores pré-estabelecidos (*default*), pelo *framework* MR ou pelo administrador do sistema, são utilizados. Uma outra opção que pode ser empregada para o ajuste dos parâmetros é a utilização de valores que são recomendados e divulgados publicamente em tutoriais pelos mantenedores dos *frameworks* [Tutorial, 2016] [Lipcon, 2009] [Spark,] [Tutorial, 2017] [White, 2009], com base no conhecimento prático do *framework*, denominados de *Rules-of-Thumb*. Porém, os resultados obtidos em [Herodotou, 2012] [Wang et al., 2016], apontam que as configurações de parâmetros com valores *default* e/ou *rules-of-thumb* realizam uma execução menos eficiente da aplicação quando comparado com as soluções propostas nestes trabalhos, mostrando que através de um ajuste "correto" dos parâmetros efetivamente pode-se melhorar o desempenho dos *workflows* em *frameworks* MR.

2.4 Aprendizado de Máquina

A quantidade de configurações de parâmetros distintos envolvidas na execução de um *framework* MR, como mencionado anteriormente (no caso Spark), juntamente com características específicas da aplicação, como particionamento de dados, tornam a seleção de parâmetros um problema complexo. Nesta seção, são resumidos os principais conceitos de aprendizado de máquina que serão empregados na construção do modelo preditivo. Este modelo será usado no ajuste da configuração de parâmetros necessários em execuções do *workflow*.

Aprendizado de máquina pode ser descrito como um sub-domínio da Inteligência Artificial que avalia estratégias computacionais para obter conhecimentos novos de maneira automática e também atualizar o conhecimento já existente. Um sistema de AM é um programa computacional que concebe decisões apoiado em experiências anteriores [Weiss and Kulikowski, 1991]. Essas experiências anteriores integram o conjunto de exemplos de treinamento fornecidos ao algoritmo de aprendizado. Os sistemas de AM possuem características que possibilitam sua classificação quanto ao paradigma, modo, forma de aprendizado e linguagem de descrição utilizada para representar exemplos e conhecimento.

Os conjuntos de exemplos (observações) fornecidos para o algoritmo de aprendizado supervisionado são rotulados com suas respectivas classes. Nesse caso, o objetivo do algoritmo de classificação é determinar corretamente a classe de novos exemplos ainda não rotulados. Formalmente, em classificação, um exemplo é um par $(x_i, f(x_i))$ onde x_i é a entrada e $f(x_i)$ é a saída. A tarefa do classificador é, dado um conjunto de exemplos, induzir uma função h que aproxima f, normalmente desconhecida. Neste caso, h é chamada hipótese sobre a função objetivo f, ou seja, $h(x_i) \approx f(x_i)$.

Geralmente, um conjunto de exemplos é dividido em dois subconjuntos disjuntos denominados de conjunto de treinamento e de teste. O conjunto de treinamento é usado para o aprendizado do conceito e o de teste é usado para medir o grau de efetividade do conceito aprendido. Esses subconjuntos são disjuntos para garantir que as medidas obtidas, utilizando o conjunto de teste, sejam de um conjunto diferente do utilizado para realizar o aprendizado, tornando a medida de desempenho estatisticamente válida [Monard and Baranauskas, 2003].

2.4.1 Discretização

O processo de discretização pode ser definido como uma etapa do pré-processamento de dados, cujo objetivo é transformar atributos contínuos em atributos discretos [Yang et al., 2005]. Por exemplo, um intervalo contínuo $[x, y]$ pode ser particionado em dois novos intervalos $[x, k]$ e $(k, y]$, sendo k o ponto de corte adotado. Dessa forma, os valores contínuos contidos nesse intervalo serão divididos em dois grupos: os que pertencem à primeira partição e os que pertencem à segunda. A discretização também pode ser definida como um método de redução de dados, pois o conjunto de valores contínuos é reduzido à um subconjunto de valores discretos.

Um método de discretização também pode ser definido como supervisionado quando emprega a informação existente em outro atributo durante o processo de discretização de um atributo contínuo. Em contrapartida, os métodos não-supervisionados realizam a discretização de um atributo contínuo sem considerar outros atributos. Os métodos Equal-Width e Equal-Frequency [Li and Wang, 2002] são exemplos de métodos de discretização não-supervisionados. Já os métodos de discretização MDLP [Fayyad and Irani, 1993] e ChiMerge [Kerber, 1992] são exemplos de métodos de discretização supervisionados.

O método Equal-Frequency (EF) [Li and Wang, 2002] necessita que os valores contínuos do atributo sejam ordenados no início do processo de discretização. Além disso, é necessário definir previamente o número de grupos de valores a serem formados, cujo

conjunto de grupos é definido como $C = \{C_1, C_2, \ldots, Ck\}$. Diferentemente do que ocorre no método Equal-Width, no EF cada um dos k grupos terá aproximadamente a mesma quantidade de elementos. Assim, dados n valores contínuos distintos, cada grupo terá n/k valores contínuos. Porém, múltiplas ocorrências de um determinado valor contínuo devem ser mantidas no mesmo grupo e, com isso, nem sempre é possível produzir k grupos com a mesma quantidade de elementos.

A Tabela 2.1 fornece um exemplo de execução do método de discretização EF. Na primeira linha da tabela estão disponíveis os valores ordenados de um atributo contínuo. Na segunda linha da Tabela 2.1 é definido o número de grupos k, enquanto que na terceira linha são calculados os valores dos elementos de fronteira de cada grupo. Finalmente, na última linha da Tabela 2.1, são apresentados os valores discretos obtidos.

Valores Contínuos Ordenados	4,5	5,3	5,4	5,6	5,75	6,05	6,2	6,3	7
Parâmetro	$k = 3$								
Qtde. de Valores por Partição	9/3, cada grupo terá 3 valores								
Valores Discretos	A	A	A	B	B	B	C	C	C

Tabela 2.1: Exemplo de discretização utilizando o método *Equal-Frequency*

O método de discretização EF pode ser categorizado como estático, divisivo, uni-variado, direto e não-supervisionado [Garcia et al., 2013]. Um método de discretização é definido como estático quando a execução da mesma ocorre anteriormente à execução do algoritmo de aprendizagem. Os métodos categorizados como divisivos iniciam o processo de discretização com apenas um grupo, contendo todos os valores contínuos do atributo e, em seguida, sucessivas subdivisões são executadas até que o critério de parada seja alcançado. São conhecidos como uni-variados os métodos que acessam apenas um atributo por vez. Os métodos que requerem a predefinição do número de partições a serem criadas são caracterizados como métodos diretos. Os métodos diretos também incluem em seu grupo, métodos que executam a discretização em apenas um passo e métodos que adotam mais de um ponto de corte a cada etapa do processo.

Por se tratarem de métodos de discretização não-supervisionados, não é imperativo que a base de dados possua informações referente a outros atributos. Desse modo, esses métodos são diretamente aplicáveis a diferentes tipos de técnicas de classificação, incluindo por exemplo, árvores de classificação.

2.4.2 Árvores de Classificação

Árvore de classificação é uma das técnicas utilizadas em AM e que tem sido aplicada com sucesso em uma grande gama de tarefas de aprendizado em aplicações de diversas áreas da ciência. No processo experimental destas aplicações muitas vezes são produzidos dados com grandes quantidades de parâmetros, valores e combinações, tornando necessário realizar análises complexas que por vezes geram resultados de difícil interpretação. A análise deste conjunto de dados experimentais envolvendo diversos parâmetros vem sendo realizada através de metodologias flexíveis como árvore de classificação, uma vez que estas têm sido capazes de beneficiar a compreensão dos resultados frente a dificuldades como a presença de números elevados de parâmetros e os diferentes graus de combinações entre os mesmos.

A criação de árvores de classificação proporciona a modelagem de um atributo classe resposta discretizado, com base em um conjunto de parâmetros. Neste contexto os modelos de classificação por árvores [Breiman et al., 1984] aparecem como uma alternativa de grande valia para uma melhor compreensão dos dados devido à simplicidade e versatilidade desta técnica. A construção de uma árvore de classificação é dividida basicamente em quatro etapas: definição (e execução) de um critério de partição das amostras, aplicação do processo de poda, seleção do melhor modelo e classificação dos nós finais.

A elaboração de modelos de classificação por árvores possibilita a explicação de uma classe discretizada por meio de um conjunto de parâmetros e de suas eventuais combinações. Existem diversos métodos de classificação, como o método *Classification And Regression Trees* (CART), que é baseado na execução de sucessivas divisões binárias de uma amostra, com base nos resultados dos parâmetros, buscando a definição de subamostras internamente homogêneas. A classificação dessas subamostras é realizada conforme alguma medida descritiva. Já a predição de novos elementos é executada por meio da estrutura de classificação definida.

O aprendizado consiste em um conjunto de regras disjuntas que podem ser apresentadas como uma árvore [Mitchell, 1982]. Nesta tese, o termo regra faz referência a uma regra extraída diretamente de uma árvore de classificação e representada da seguinte forma:

$$(\text{Condição 1}) \wedge (\text{Condição 2}) \wedge \cdots \wedge (\text{Condição n}) \rightarrow Classe = C_i$$

onde C_i é um dos possíveis valores para a classe, isto é, $C_i \in \{C_1, C_2, \ldots, Cn\}$.

Alguns termos utilizados de forma recorrente na caracterização dos componentes de

uma árvore de classificação são: nó raiz, nó intermediário, nó folha e ramos. Denomina-se nó raiz à amostra original, nós intermediários às subamostras que originam novas subamostras e nós folhas às subamostras não partidas. Além disso, as divisões executadas podem ser denominadas ramos. Portanto, a árvore é a representação gráfica de nós e ramos. Tais termos são representados na Figura 2.3, que ilustra uma árvore de classificação.

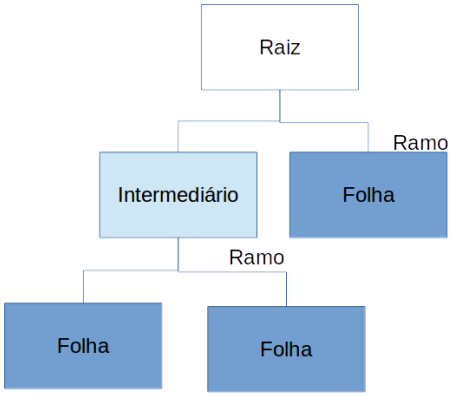

Figura 2.3: Organização de uma árvore de classificação

O algoritmo C4.5 foi criado por J. R. Quinlan, na década de 90, e tem sido considerado o algoritmo de referência para o desenvolvimento e análise de novos algoritmos de classificação [Ruggieri, 2004]. O C4.5 realiza a construção da árvore através de um algoritmo guloso, utilizando a técnica de dividir-e-conquistar. No caso de árvores de decisão, o algoritmo C4.5 calcula a melhor característica que possa servir como um nó de decisão e trabalha recursivamente a cada nó até que a árvore esteja montada.

A partir de um conjunto de dados de treinamento T, o algoritmo inicia com todas as instâncias de treinamento em um nó, que será a raiz da árvore. Este conjunto de dados de treinamento é dividido em subconjuntos menores à medida em que a árvore é construída, até que todos os subconjuntos presentes em um nó N sejam da mesma classe C_i. Então, estes nós são chamados de nós folha, e a construção da árvore está completa.

2.4.3 Métricas de Avaliação da Predição

A Matriz de Confusão (MC) de uma hipótese h oferece uma medida efetiva do modelo de classificação ao mostrar o número de classificações corretas versus as classificações preditas para cada classe, sobre um conjunto de exemplos ou dados de treinamento T. Os resultados são totalizados em duas dimensões, para as classes verdadeiras e as preditas, para k classes diferentes $C_1, C_2, ..., C_k$, conforme apresentado na Tabela 2.2. Cada elemento

$M(C_i, C_j)$ da matriz, i, $j = 1, 2, ..., k$, calculado pela Equação 2.1, representa o número de exemplos de T que realmente pertencem à classe C_i, mas foram classificados como sendo da classe C_j. O número de acertos para cada classe localiza-se na diagonal principal, enquanto os demais elementos representam erros na classificação [Monard and Baranauskas, 2003].

$$M(C_i, C_j) = \sum_{\forall (x,y) \in T : y = C_i} ||h(x) = C_j|| \tag{2.1}$$

Classe	Predita C_1	Predita $C_2 \ldots$	Predita C_k
verdadeira C_1	$M(C_1, C_1)$	$M(C_1, C_2) \ldots$	$M(C_1, C_k)$
verdadeira C_2	$M(C_2, C_1)$	$M(C_2, C_2) \ldots$	$M(C_2, C_k)$
$\vdots$	$\vdots$	$\vdots \ddots$	$\vdots$
verdadeira C_k	$M(C_k, C_1)$	$M(C_k, C_2) \ldots$	$M(C_k, C_k)$

Tabela 2.2: Matriz de Confusão de um Classificador [Monard and Baranauskas, 2003]

Por exemplo, neste trabalho o conjunto de dados T utilizado corresponde aos dados coletados da execução dos experimentos nos diferentes ambientes de execução e *workflows*. Foram coletados mais de 1200 exemplos de treinamento e um total de 10 atributos. Os atributos são apresentados no Capítulo 6, onde a precisão e a MC serão utilizadas para avaliação dos experimentos que utilizaram esse conjunto de dados.

O maior desafio é, a partir de um conjunto de exemplos de treinamento, montar uma árvore de classificação que possua um poder preditivo satisfatório. Para avaliação da capacidade preditiva podem ser utilizadas algumas métricas como: *Precision*, *Recall*, *F-Measure* (F1), e *Accuracy*. As métricas *precision e recall* são as mais utilizadas numa classificação binária. Dada uma árvore de classificação, *precision* e *recall* associados a esta árvore são definidos como:

$$recall = \frac{\text{Número de Predições Positivas Corretas}}{\text{Número de Exemplos Positivos}} \tag{2.2}$$

$$precision = \frac{\text{Número de Predições Positivas Corretas}}{\text{Número de Predições Positivas}} \tag{2.3}$$

Para um classificador ser considerado bom não é suficiente que ele tenha uma boa precisão ou boa cobertura isoladamente. Por isso também calcula-se uma terceira medida, denominada *F-Measure*, que faz uma avaliação conjunta delas da seguinte forma:

$$F1 = \frac{(\beta^2 + 1)precision \times recall}{\beta^2 precision + recall} \tag{2.4}$$

Utilizamos $\beta = 1$ dando a mesma importância para as duas medidas e desta forma calculando a métrica chamada de F1.

Adicionalmente em alguns experimentos utiliza-se também a métrica *Accuracy*. Ela sozinha não é uma boa métrica de desempenho, pois poder-se-ia alcançar valores altos sempre classificando as instâncias na classe negativa, mas para os experimentos relativos ao aprendizado supervisionado esta métrica tem sido utilizada. Esta métrica é na verdade a *precision* considerando todas as classes da seguinte forma:

$$accuracy = \frac{\text{Número de Predições Positivas Corretas}}{\text{Total de Amostras}} \tag{2.5}$$

Nesta tese utilizou-se o modo de aprendizado supervisionado por meio do método Árvore de Classificação, descrito na subseção 2.4.2. Além disso, foi empregado o método de discretização não-supervisionado *Equal-Frequency*. A fim de implementar a ideia proposta, foi utilizado a ferramenta Orange [Demsar et al., 2013], que implementa muitos algoritmos amplamente conhecidos de AM e mineração de dados, além de ferramentas de visualização, sendo ainda gratuito. O Orange implementa o algoritmo C4.5 [Quinlan, 1993] para construção da árvore de classificação.

Nesta tese o uso das técnicas de AM se limitou a serem auxiliares no desenvolvimento da abordagem e interpretação dos dados coletados, e não para o desenvolvimento de novas técnicas ou algoritmos para a área de AM. Apesar da utilização de técnicas de AM, mesmo como ferramentas auxiliares, deve ser ressaltado que o foco deste texto não é se aprofundar nesse contexto. Assim, aborda-se superficialmente o funcionamento das técnicas de AM utilizadas neste trabalho para apresentar uma notação a ser empregada nos experimentos, permitindo uma melhor avaliação e interpretação dos resultados obtidos.

2.5 Resumo do Capítulo

Neste capítulo foram destacados os principais conceitos relacionados a esta tese. Foram caracterizados resumidamente aplicações intensivas de dados, *workflows*, SGWC, paralelismo de *workflows* e paradigma *MapReduce*. Também foram apresentadas algumas estratégias de particionamento de dados de entrada dos *workflows* científicos relevantes para a compreensão desta tese. Além disso, *frameworks* MR utilizados na execução para-

lela de *workflow*, como o Hadoop e Spark também foram abordados. Descrevem-se ainda os principais conceitos e métodos de aprendizado de máquina utilizados na abordagem proposta. Dentre eles, os métodos de discretização Equal-Frequency e Equal-Width, o método de classificação por árvore de decisão, e também os critérios e métricas de avaliação e validação do modelo preditivo: acurácia, F-*measure*, cobertura e precisão. A caracterização destes conceitos e a descrição do funcionamento das ferramentas são importantes para a compreensão da abordagem proposta e dos experimentos realizados e descritos ao longo desta tese.

Capítulo 3

Experimentos Preliminares

Neste capítulo são apresentados os experimentos realizados com o objetivo de demonstrar a importância do problema relacionado ao particionamento de dados. Basicamente, o particionamento de dados analisado nestes experimentos está relacionado a dois aspectos: dados de entrada e os dados intermediários. Nesta tese, são denominados dados intermediários aqueles dados que são produzidos entre as atividades do *workflow*.

De acordo com a avaliação experimental realizada em [de Oliveira et al., 2014], foram constatadas algumas limitações quanto ao aspecto de armazenamento dos dados de entrada. Em [de Oliveira et al., 2014] foi possível identificar as arquiteturas de armazenamento que favorecem o processamento paralelo de *workflows* intensivos de dados. As alternativas de arquiteturas de armazenamento foram avaliadas tendo como aplicação exemplo um *workflow* real da área de astronomia [1] denominado SkyMap, descrito na no Capítulo 4.

Além disso, outra avaliação experimental foi realizada em [de Oliveira et al., 2015], onde foram investigadas alternativas de execução de *workflows* que se beneficiem do particionamento dos dados de entrada aumentando o desempenho da aplicação. As alternativas de execução avaliadas se diferem no tratamento e armazenamento dos dados intermediários de um *workflow*.

Portanto, a seguir são realizadas algumas análises relacionadas aos dois aspectos supracitados. Na seção 3.1 são analisadas as arquiteturas de armazenamento dos dados de entrada com relação ao particionamento. Na seção 3.2 são avaliadas as alternativas de execução paralela e sua relação com os dados intermediários.

[1] http://ogando.linea.gov.br/

3.1 Armazenamento de Dados de Entrada

Uma vez que os *workflows* científicos tratados nesta tese são classificados como intensivos de dados foi realizada uma análise de alternativas de armazenamento de grandes volumes de dados que favoreçam o processamento destes e avalia dois tipos de paralelismo na execução: processo e dados.

Com isso, foram avaliadas diferentes estratégias de acesso a dados tendo como aplicação exemplo, um *workflow* real da área de astronomia. De fato, a questão colocada sobre a eficiência da arquitetura de armazenamento de dados frente à de processamento de *workflows* foi motivada por esta cooperação.

Basicamente, as estratégias de acesso a dados para a execução de *workflows* avaliadas podem se resumir em duas alternativas básicas: paralelismo de processo (controle) e paralelismo de dados. No primeiro, o paralelismo do *workflow* é independente da localidade dos dados, enquanto o segundo explora a execução que privilegia a localidade de dados. A combinação da estratégia de armazenamento com a arquitetura de processamento impacta no tempo total de execução, escalonamento, grau de concorrência, transferência de dados pela rede e tolerância à falhas.

Os experimentos resultaram na produção do artigo "Análise de Estratégias de Acesso a Grandes Volumes de Dados" publicado no Simpósio Brasileiro de Banco de Dados (SBBD - 2014). No artigo são descritas as arquiteturas analisadas: Sistema Gerenciador de Banco de Dados (SGBD) centralizado, SGBD particionado, SGBD distribuído e sistema de arquivos distribuído, além da aplicação utilizada e do ambiente de execução. A seguir são resumidos a aplicação, estratégias avaliadas, ambiente e os resultados obtidos.

3.1.1 Aplicação

O *workflow* da aplicação é um *pipeline* composto das seguintes atividades em ordem:

(1) **Leitura de dados**: Obtenção dos dados;

(2) **Tratamento dos dados**: Preparação para formato esperado pelo *SkyMap*;

(3) **SkyMap**: Execução da aplicação *SkyMap* que com base na indicação do posicionamento de cada objeto no arquivo de entrada, colore o ponto respectivo em um histograma da região do céu sendo analisada. Sua implementação realiza a leitura do arquivo gerado na atividade anterior e gera um outro arquivo ".pkl"gravado no NFS (*Network File System*).

(4) **SkyMapAdd**: Execução da aplicação *SkyMapAdd* que consolida os diversos histogramas gerados em um único. Uma instância do *Reduce* acessa os vários ".pkls"gerados na atividade (3) e produz o histograma final;

Sua implementação no *framework* Hadoop requer o mapeamento do *workflow* em duas etapas *Map* e *Reduce*. Sendo assim as atividades (1), (2) e (3) fazem parte da etapa *Map* e a atividade (4) compõe a etapa *Reduce*.

3.1.2 Estratégias de Armazenamento de Dados

Foram avaliadas cinco estratégias de armazenamento que estão descritas a seguir:

(1) **Banco de Dados Centralizado**: Na Figura 3.1(A) apresenta-se a arquitetura que foi utilizada para execução da aplicação. Esta estratégia é a mais comum e se traduz no armazenamento dos dados inteiramente em um único SGBD. Considera-se que instâncias paralelas de *workflows* requisitam subconjuntos disjuntos dos dados armazenado no BD centralizado.

Do ponto de vista da arquitetura de processamento de *workflow*, apenas a estratégia de paralelismo de processos se mostra possível na medida em que o *workflow* executa em nós remotos em relação ao SGBD centralizado. Desta forma, ao paralelizar a execução do *workflow* em diferentes nós do *cluster*, tem-se que cada um submeterá uma consulta ao SGBD centralizado recuperando sua partição dos dados, logo, existirá concorrência na leitura dos dados. Como os dados deste modelo estão armazenados no nó mestre não há localidade de dados, pois os dados devem ser transferidos do nó mestre para os nós trabalhadores, ou seja, quando a execução da aplicação é iniciada, os vários nós trabalhadores realizam a leitura de seus respectivos dados do SGBD PostgreSQL localizado no nó mestre.

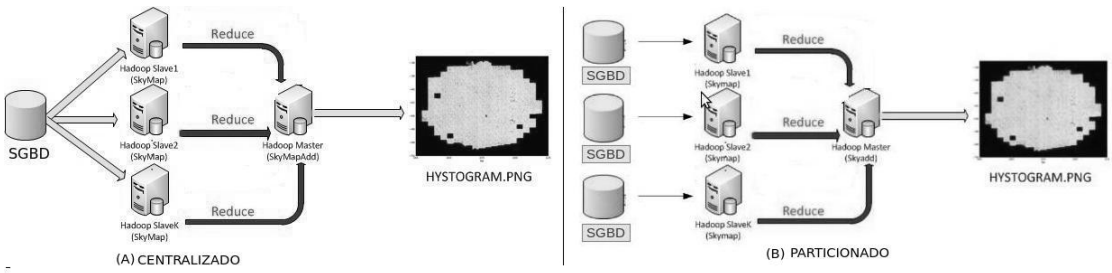

Figura 3.1: Arquitetura com SGBD Centralizado e Particionado

(2) **Banco de Dados Particionado**: Na Figura 3.1(B) apresenta-se a arquitetura que foi utilizada para execução da aplicação. Nesta estratégia, os dados são gerenciados

por instâncias de SGBD instaladas em cada nó do *cluster*. Sendo assim, cada instância recebe um subconjunto dos dados e recupera o conjunto de dados de interesse acessando o SGBD local. Pode-se perceber que não há uma visão global dos dados e os *workflows* conhecem o critério de particionamento. Note-se ainda que essa estratégia coincide com o fato dos *workflows* acessarem uma grande partição dos dados e terem interesse em uma leitura do tipo varredura. Nesta estratégia podem-se adotar arquiteturas de processamento de *workflow* tanto baseadas em paralelismo de processo quanto de dados. Como os dados deste modelo estão armazenados em cada nó trabalhador, quando a execução da aplicação é iniciada, os vários nós trabalhadores realizam a leitura de seus respectivos dados que se encontram no SGBD PostgreSQL localmente. Com isso, não existe tráfego de dados na rede nem concorrência de acesso aos dados.

(3) **Sistema de Arquivos Distribuído**: Este modelo é semelhante ao utilizado pelo SGBD particionado com relação à localidade dos dados, pois os processos *Map* podem ser executados sobre os mesmos nós de armazenamento. Neste cenário somente o subconjunto de dados do SGBD relevante ao *workflow* foi importado para o HDFS.

As principais características de cada estratégia, descritas anteriormente, são destacadas a seguir na Tabela 3.1:

Estratégia	Localidade de Dados	Concorrência	Replicação dos Dados
SGBD Centralizado	Não	Sim	Não
SGBD Particionado	Sim	Não	Não
HDFS	Sim	Não	Sim

Tabela 3.1: Estratégias de Armazenamento de Dados

3.1.3 Ambiente

O ambiente de processamento utilizado nos experimentos possui as seguintes características:

(1) *Hardware*: *Cluster* SGI com 81 nós de processamento, sendo 1 nó mestre (com dois processadores Intel Xeon X5650, 2.67GHz, 6 núcleos reais por processador, com possibilidade de *Hyper-threading*, 24GB memória, 1,5TB de disco rígido) e 80 nós trabalhadores com a mesma configuração do nó mestre, porém contando com 500GB de disco rígido localmente.

(2) *Software*: *Framework* Hadoop 0.20; BD Centralizado (PostgreSQL 8.4), BD Particionado (PostgreSQL 8.4) e HDFS; Dados para aplicação foram obtidos do catálogo do

Laboratório LIneA com aproximadamente 30 milhões de objetos.

3.1.4 Análise dos experimentos

As avaliações experimentais foram realizadas considerando-se três cenários quanto ao ambiente computacional: 20, 40 e 80 máquinas. Para cada configuração de ambiente foram realizadas 10 execuções e foi calculada uma média dos tempos de execução. Estas médias são apresentadas na Figura 3.2 analisada nesta Subseção.

Nos experimentos, cada arquitetura é representado por <ARQUITETURA><n>, podendo $ARQUITETURA$ ser $CENT$ representando um SGBD *centralizado*, $PART$, representando um SGBD particionado e HDFS_RA, obviamente utilizando HDFS, com n processos (ou nós de processamento) que executam a etapa Map, com n nós de armazenamento, quando a arquitetura incluir armazenamento distribuído, de tal forma, que cada processo obtém $1/n$ dos dados. Por exemplo, $CENT20$ representa a execução do *workflow* com 20 processos paralelos, obtendo os dados a partir de um banco de dados centralizado. Os resultados obtidos usando a arquitetura HDFS são denominados nos experimentos como HDFS<n>_RA.

Diante dos experimentos realizados alguns aspectos foram identificados. O modelo de distribuição adotado pelos dados de entrada no *workflow* impacta significativamente no custo total do processamento. Por exemplo, a estratégia centralizada (CENT20, CENT40 e CENT70) apresenta um *speedup* de 0,77, para uma redução de leitura por cada instância do *workflow* de um fator de 3,5. Neste cenário, ao reduzir o número de dados a serem lidos, aumenta-se de forma inversa o número de instâncias do *workflow* que acessam concorrentemente o SGBD. Este aumento no grau de paralelismo acessando um mesmo local diminui substancialmente o ganho obtido na redução de dados lidos por cada instância. Consequentemente, impacta a intenção de ganho de desempenho com o paralelismo de processo como apresentado na estratégia centralizada apresentada na Figura 3.2.

Com isso, o tempo gasto na atividade de Leitura na estratégia centralizada é claramente pior devido à concorrência no acesso aos dados. Esta concorrência gera uma maior variação no tempo de resposta às requisições, uma vez que todas as máquinas solicitam os dados a um nó central, algumas máquinas recebem os dados mais rapidamente e outras demoram mais a receber.

Por outro lado, percebe-se que na estratégia particionada (PART20, PART40 E PART80) o tempo gasto na atividade de Leitura dos dados é proporcional à quanti-

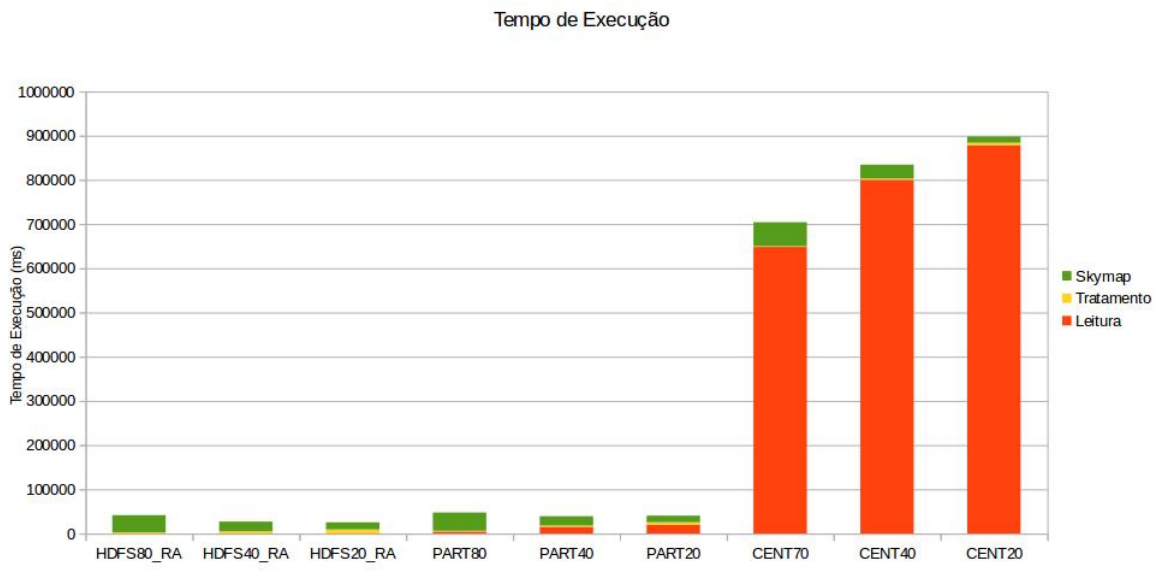

Figura 3.2: Tempo de Execução de cada Atividade nas Arquiteturas: HDFS, SGBD Particionado, SGBD Centralizado

dade de dados a serem lidos por cada *Map*. Neste cenário explora-se a execução paralela dos processos privilegiando a localização dos dados, ou seja, os dados necessários para a execução de qualquer *Map* estão na mesma máquina em que executam, não havendo concorrência de acesso aos dados. Pode-se concluir que o fato de não existir um elemento centralizador que gerencia os dados contribui para um melhor desempenho destas estratégias. Na estratégia centralizada a máquina que armazena todos os dados recebe as requisições dos processos paralelos, centralizando assim o acesso aos dados, e gerando uma concorrência na resposta as requisições. Esta perda de desempenho gerada pela concorrência no acesso aos dados não ocorre na estratégia particionada, uma vez que, cada processo acessa localmente os seus dados.

Pode-se verificar nos experimentos que a estratégia particionada dos dados e a estratégia utilizada no HDFS (HDFS20_RA, HDFS40_RA E HDFS80_RA) apresentam um acesso mais eficiente para o problema como apresentado na Figura 3.2. Isto se dá, principalmente, pelo fato dos processos serem alocados de acordo com o local de armazenamento dos dados. A principal vantagem destas estratégias é não realizar nenhum tipo de transferência de dados entre as máquinas de processamento. O Hadoop realiza esta alocação de processos de forma automática e em tempo de execução, na etapa de divisão dos processos *Map*.

Portanto, o paralelismo de processos é válido em aplicações que utilizam pequenos volumes de dados, onde somente a etapa de processamento é relevante para o desempenho

da aplicação. Principalmente o maior desempenho é atingido se os dados poderem ser levados até o processo. Também, em aplicações *CPU-Intensive* onde o processamento é o "gargalo", o modelo de paralelismo de processos é uma boa solução.

Porém, em aplicações com grandes volumes de dados e/ou aplicações *data-intensive* verifica-se que somente o paralelismo de processo caracterizado pelo fato de levar os dados até o processo, como na estratégia centralizada, não se mostra como solução eficiente em termos de desempenho. Neste tipo de aplicação, chamada atualmente de *data-intensive*, o paralelismo de dados mostra uma execução paralela dos processos de acordo com a localização dos dados, eliminando a concorrência e a transferência de dados pela rede entre as máquinas de processamento.

3.2 Localização dos Dados Intermediários

Um outro aspecto é relativo à gravação de resultados intermediários nas atividades do *workflow*. Adicionalmente, o perfil de leitura/gravação de dados pelas atividades têm impacto significativo no desempenho de uma aplicação.

Os experimentos descritos a seguir verificaram o impacto da co-alocação de atividades sobre um mesmo *Map*. Com isso, as atividades co-alocadas em um único *Map* podem ler e armazenar localmente os dados, e apresentam uma redução linear associada ao tempo em relação ao número de nós envolvidos. Por outro lado, quando cada atividade corresponde a um *Map*, os dados devem ser armazenados em um sistema global de armazenamento, causando um *overhead* no desempenho. Em uma solução de *workflow* tradicional, esse fenômeno aconteceria a cada atividade, quando ocorresse a gravação dos resultados intermediários.

Estes experimentos resultaram no artigo "Avaliação da Localidade de Dados Intermediários na Execução Paralela de *Workflows BigData*" publicado no Simpósio Brasileiro de Banco de Dados (SBBD - 2015). No artigo é descrito detalhadamente as alternativas de execução analisadas: Hadoop, HaQoop-Local, HaQoop-HDFS e Spark. Além disso, é descrita a aplicação utilizada nos experimentos assim como o ambiente de execução. A seguir são apresentados somente os resultados que demonstram a importância da localidade de dados de acordo com a alternativa de execução.

3.2.1 Aplicação

A aplicação utilizada nos experimentos descritos na seção 3.1 é a mesma utilizada nestes testes, porém o *workflow* foi modelado com três atividades, que são: (1) **Leitura e Tratamento dos dados**, (2) **SkyMap** e (3) **SkyMapAdd**.

Observa-se que o *workflow* pode ser mapeado para o modelo *MapReduce*, sendo as atividades (1) e (2) associadas a processos do tipo *Map*, enquanto a atividade (3) seria mapeada para um processo do tipo *Reduce*. O *workflow*, apesar de simples, oferece oportunidades de investigação no tratamento de localidade de dados para o arquivo intermediário compartilhado entre as atividades (1) e (2).

3.2.2 Exploração de Alternativas para o tratamento de arquivos intermediários

A partir do *workflow* real discute-se alternativas para o tratamento de localidade de dados associada a arquivos intermediários em *workflows* científicos. Consideram-se quatro alternativas de execução das tarefas (1) e (2) do *workflow SkyMap* que se comunicam por meio de um arquivo intermediário como demonstrado na Figura 3.3.

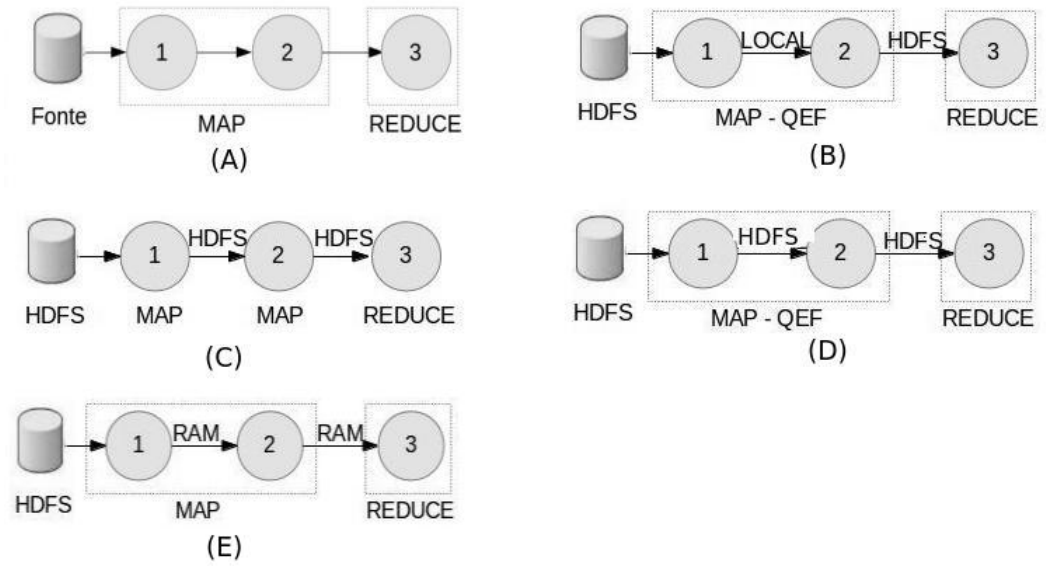

Figura 3.3: *Workflow* da Aplicação *SkyMap* e Alternativas de Execução

As avaliações exploram alternativas de mapeamento do *workflow* para os sistemas *Hadoop* e *Spark*. As variações adotadas são apresentadas a seguir:

(1) **Alternativa Hadoop**: Na Figura 3.3-(C) apresenta o mapeamento adotado nesta

alternativa. Sua implementação no Hadoop requer o mapeamento do *workflow* em dois *jobs*. Sendo assim a atividade (1) corresponde a um *job* que só possui a etapa *Map*, as atividades (2) e (3) compõem o segundo *job* com as etapas *Map* e *Reduce* respectivamente. Neste modelo, os dados produzidos pelo primeiro *job* devem ser armazenados no HDFS para serem consumidos pelo segundo *job*. Ainda, o *Map* implementando a atividade (2) pode ser iniciado em qualquer nó, local ou não em relação ao nó que produziu os dados a serem consumidos na execução da atividade (1). O mesmo processo ocorre na atividade (3) na etapa *Reduce*. Neste caso, é obrigatório o armazenamento dos dados produzidos pela atividade (2) no HDFS, pois o *Reduce* é processado por somente uma máquina escolhida aleatoriamente pelo Hadoop.

A discussão acima realça o problema da ativação de *jobs* diferentes, pelo *framework* Hadoop. Dois elementos entram em cena nesta discussão: a alocação das atividades que compartilham arquivos intermediários, assim como, o armazenamento destes arquivos que deve ser realizada no HDFS.

(2) Alternativa HaQoop: A Figura 3.3-(B) apresenta este mapeamento, considerando sua implementação no *framework* Hadoop segundo um mapeamento do *workflow* em um único *Map* e um *Reduce*. Alocação das tarefas com garantia de execução em um mesmo nó das atividades (1) e (2) que compartilham o arquivo intermediário, usando o QEF com Hadoop. Sendo assim, as atividades (1) e (2) são integradas via QEF em um único *Map*, formando assim o fragmento de execução local, e a atividade (3) compõe a etapa *Reduce*. A função *Map* ativa a execução do QEF, garantindo o tratamento local ao fragmento do *workflow*.

Adicionalmente, garante-se o acesso local aos dados intermediários gravando-se os mesmos no sistema de arquivos local. Nesta alternativa, a execução deste conjunto de atividades ocorre garantidamente de forma local, diferentemente da alternativa *Hadoop*. A execução local de atividades se traduz na localidade de acesso aos arquivos intermediários envolvidos.

(3) **Alternativa HaQoop-HDFS**: O modelo apresentado na Figura 3.3-(D) apresenta alternativa semelhante a HaQoop-Local. Alocação das tarefas compartilhando arquivos intermediários como a alternativa (2), porém o arquivo intermediário é armazenado no HDFS, o que pode proporcionar seu armazenamento fora do nó em que as atividades são alocadas. Semelhante ao modelo anterior, baseado em Hadoop e QEF, quanto à alocação das atividades em um mesmo nó. Os dados intermediários, no entanto, são armazenados no sistema de arquivos distribuído HDFS.

Esta alternativa explora as consequências da execução local de atividades quando os dados intermediários compartilhados não são necessariamente acessados localmente. Para isso, considera-se o armazenamento de arquivos intermediários no HDFS obtendo assim a tolerância à falhas.

(4) **Alternativa Spark**: Na Figura 3.3-(E) apresenta-se o mapeamento visando a execução do *workflow* no *framework* Spark. Neste modelo os dados intermediários são armazenados em memória RAM, com o objetivo de investigar o efeito na gravação de arquivos intermediários neste tipo de memória.

3.2.3 Ambiente

Para a realização dos testes foi preparado um ambiente de processamento com as seguintes características: 11 máquinas físicas cada uma com processador Intel Xeon ES4607, 2.2GHz, 24 núcleos reais por processador, com possibilidade de *Hyper-threading*, 128GB memória e 15TB de disco rígido. Um ambiente com 11 máquinas virtuais de criadas com o *hypervisor* KVM (*Kernerl-based Virtual Machine*) sendo 1 mestre e 10 trabalhadores, uma por máquina real cada uma com 2 núcleos, 12GB memória, 200GB de disco rígido. Os *frameworks* utilizados foram Hadoop, QEF, Spark.

3.2.4 Análise dos experimentos

Nas subseções a seguir são analisados os experimentos realizados utilizando as quatro alternativas descritas anteriormente na seção 3.2.2. O objetivo é demonstrar como o modelo de execução do *workflow* pode impactar o desempenho da aplicação e com isso destacar a importância da localidade dos dados intermediários. Os testes foram feitos com as seguintes variações no número de *Maps*: 2, 4, 6, 8, 10 e 20. Embora as máquinas utilizadas para os testes possuam mais de um núcleo, a fase de *Map* inicia um único processo em cada máquina utilizando somente um núcleo. Para cada ambiente e variação de *Maps* foram realizadas 15 execuções consecutivas. Foram extraídos os valores máximos de cada execução e obtida a média desses valores.

3.2.4.1 Comparação do tempo total de execução

A Figura 3.4 (A) apresenta uma avaliação do tempo total de execução do *workflow* entre as alternativas de avaliação. Aparece em realce a diferença de desempenho entre o Hadoop e o HaQoop. Por exemplo, com 20 *Maps* percebe-se claramente que a diferença entre ambas

as alternativas é bem pequena nas três atividades que compõem o *workflow*, porém o tempo total de execução apresenta uma diferença muito grande entre eles.

A justificativa para este resultado está no custo de inicialização do segundo *Map* na alternativa Hadoop. O Hadoop realiza duas chamadas *Map* e uma *Reduce*. Com isso, o *framework* Hadoop é solicitado entre as atividades (1) e (2) no modelo Hadoop. O HaQoop realiza somente um *Map* uma vez que as atividades (1) e (2) fazem parte do fragmento de execução local executado pelo QEF. Além da garantia da localidade de dados isto também beneficia o desempenho do *workflow* pelo fato de permitir atividades de um mesmo tipo serem unidas em um único fragmento e executadas como uma só.

Uma análise de cada atividade modelada pelo *workflow* é avaliada separadamente. Em todos os casos, observando a Figura 3.4, é visto que a alternativa Spark teve o melhor desempenho em todas as atividades e consequentemente no tempo de execução total. Percebe-se que a utilização dos RDDs que mantém os dados intermediários na memória RAM beneficia o desempenho do *workflow*. Sabe-se que a memória RAM possui maior velocidade do que a memória secundária, porém uma menor capacidade de armazenamento. Nos experimentos aqui realizados, a quantidade de dados em cada partição era alocada devidamente na memória disponível.

3.2.4.2 Leitura e Tratamento dos dados

Na Figura 3.4 (B) apresentam-se os tempos de execução da primeira atividade - Leitura e Tratamento dos Dados. As diferenças nesta atividade entre as alternativas de execução avaliadas são: gravação dos dados produzidos por esta atividade e utilização do QEF.

É possível observar um melhor desempenho do HaQoop com relação aos demais, a partir da execução com 6 *Maps*. Porém, nas execuções com 2 e 4 *Maps*, o desempenho inferior do HaQoop com relação ao Hadoop justifica-se pelo fato de possuir uma camada de *software* adicional (*i.e. QEF*). Com isso, o HaQoop produz um custo ligeiramente maior de processamento do que no Hadoop simples, impactando no desempenho das execuções com 2 e 4 *Maps*.

Na alternativa Hadoop, a gravação é realizada no HDFS, porém localmente. O HDFS atua como um intermediário na gravação dos dados locais de cada *Map*. Consequentemente, a medida em que aumenta-se o número de *Maps* paralelos, aumenta também a concorrência no HDFS como intermediário único na gravação dos dados de vários *Maps*. Pode-se verificar que nas execuções com 2 e 4 *Maps*, o Hadoop possui um desempenho

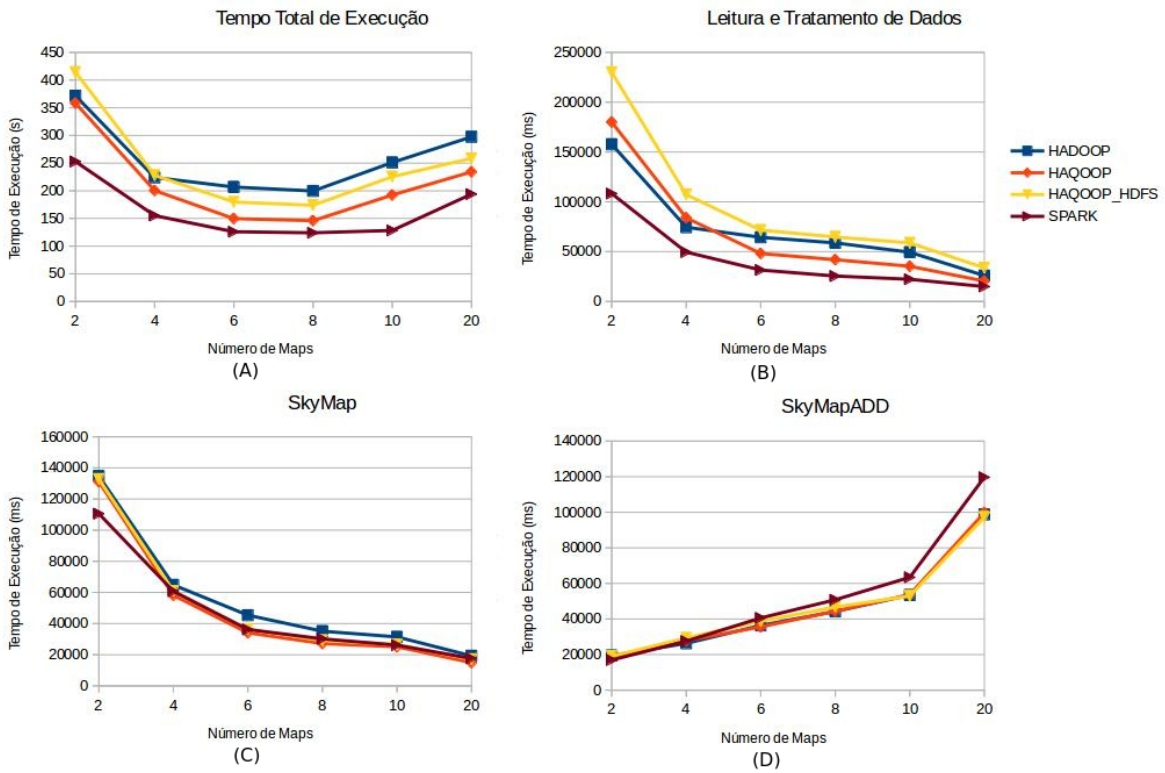

Figura 3.4: Tempo de Execução Total e de cada Atividade com as alternativas - Hadoop, HaQoop, HaQoop-HDFS e Spark

superior ao HaQoop, mas o impacto da concorrência entre Maps supera aquele devido ao QEF quando o número de *Maps* aumenta, como pode-se observar na Figura 3.4 (B).

O HaQoop-HDFS une as duas desvantagens apresentadas acima. A desvantagem do Hadoop, representada pela concorrência na gravação dos dados, e a do HaQoop causada pela utilização do QEF. Com isso, observa-se um desempenho inferior comparado as demais em todas as variações realizadas.

3.2.4.3 SkyMap

Na Figura 3.4 (C) apresentam-se os tempos de execução da atividade *SkyMap*. As diferenças na execução desta atividade, entre as alternativas discutidas, estão no modelo de obtenção dos dados intermediários, gerados pela atividade (1), e na utilização do QEF, nas estratégias HaQoop e HaQoop-HDFS. É possível observar um melhor desempenho do HaQoop-local com relação aos demais, pois os dados são lidos diretamente a partir do sistema de arquivos local. Apesar do melhor desempenho, o HaQoop não garante a tolerância à falhas, em cada atividade, fornecida pelos demais através do HDFS.

No HaQoop-HDFS, os dados são obtidos a partir do HDFS, que prioriza o armaze-

namento local. Com o QEF sendo invocado pelo *Map*, garante-se que a atividade (2) será executada localmente ao mesmo conjunto de dados da atividade (1) . O desempenho levemente inferior com relação ao HaQoop deve-se somente ao fato de utilizar o HDFS como um intermediário nesta obtenção dos dados.

No Hadoop, existem dois aspectos que influenciam no desempenho. O primeiro, que também ocorre no HaQoop-HDFS, é a utilização do HDFS para obtenção dos dados gerados pela atividade (1). O segundo aspecto se dá pelo fato de não possuir o QEF. Isto implica em realizar dois *Maps*. Com isso, o último *Map* pode ser executado em um nó diferente do que foi utilizado para execução do primeiro *Map*, ou seja, o conjunto de dados do segundo *Map* será o mesmo produzido pelo primeiro *Map*, porém estes irão executar em máquinas diferentes, implicando em uma transferência de dados entre as máquinas. Por esta razão o Hadoop possui um desempenho levemente inferior quando comparado ao HaQoop-HDFS, mesmo que ambos realizem a leitura através do HDFS.

Concluí-se que é importante atentar para a localidade dos dados intermediários, uma vez que isto influencia diretamente na execução da aplicação. O HaQoop possui como desvantagem o fato de não garantir a tolerância à falhas, fornecida pelos demais por causa do HDFS. Além disso, é uma vantagem garantir várias atividades executando sobre um mesmo *Map*, como o faz o QEF, pois, verifica-se que no HaQoop - HDFS existe a localidade de dados por causa do QEF e também a tolerância à falhas através do HDFS.

3.2.4.4 SkyMapAdd

Na Figura 3.4 (D) apresentam-se os tempos de execução somente da atividade *SkyMapAdd*, que é realizada como uma tarefa *Reduce*. Esta atividade possui a mesma implementação em todas as estratégias analisadas. Com isso constata-se que o tempo de execução é semelhante para todas as estratégias. Uma vez que só um nó irá executar a etapa *Reduce*, e este só é definido em tempo de execução, os dados de entrada devem estar no HDFS permitindo assim que independentemente do nó escolhido este tenha acesso aos dados.

Ainda, pode-se perceber que o desempenho desta atividade piora a medida que aumenta o número de *Maps*. Este problema está diretamente relacionado a estratégia de particionamento dos dados de entrada. Nestes experimentos optou-se por uma particionamento balanceado dos dados sem se importar com a localização espacial do objeto. Por exemplo, possuindo 10 nós trabalhadores e 30 milhões de objetos, cada recurso irá receber 3 milhões de objetos.

A atividade paralela *SkyMap* recebe os objetos correspondentes a partição, e o intervalo espacial (*ra* e *dec*) em que estes objetos se encontram, gerando um histograma parcial proporcional ao tamanho do intervalo. Como o particionamento não considerou a localização espacial, o intervalo foi definido como o mínimo e o máximo possível para todas as atividades paralelas *SkyMap*. Portanto, o tamanho dos histogramas produzidos por cada *Map* é o mesmo independente da quantidade de objetos processada. Por exemplo, possuindo 1 nó trabalhador e 30 milhões de objetos, será gerado um histograma com 10 MB. Em outro caso, com 10 nós e 30 milhões de objetos, cada nó irá processar 3 milhões de objetos, porém será produzido 10 histogramas cada um com 10 MB.

A atividade *SkyMapAdd* recebe como entrada estes histogramas parciais. Pode-se concluir que a quantidade de arquivos de entrada desta atividade é diretamente relacionada ao grau de paralelismo. Quanto maior o grau de paralelismo maior a quantidade de histogramas gerados e maior será o tempo de processamento do *SkyMapADD*, o que pode se tornar um gargalo.

3.3 Resumo do Capítulo

Neste capítulo foram apresentados alguns experimentos realizados para ilustrar a importância da localidade de dados. Em [de Oliveira et al., 2014] observou-se que a partir da estratégia de armazenamento dos dados abrem-se oportunidades de definição de arquiteturas de processamento de *workflows*. Principalmente, observou-se que o consumo de dados da fonte tem impacto relevante no custo total do processamento. Adicionalmente, o perfil de leitura/gravação de dados pelas atividades também tem impacto significativo. Basicamente, duas estratégias de paralelismo foram exploradas: paralelismo de processo e paralelismo de dados. No primeiro, o paralelismo do *workflow* é independente da localização dos dados, enquanto o segundo explora a execução que privilegia a localização de dados. Os resultados demonstraram a importância, através da otimização no desempenho da aplicação, desta última estratégia de paralelismo.

Também neste capítulo foram apresentados os experimentos realizados mostrando que *frameworks* de execução de aplicações como Hadoop e Spark melhoram, de fato, a eficiência na execução de *workflows* sobre grandes volumes de dados. Porém foi destacado que nestes sistemas, perde-se a perspectiva global da execução e reduz-se a possibilidade de obtenção de uma estratégia de execução de maior eficiência. Neste trabalho, foram avaliadas quatro alternativas de execução para se obter um eficiente processamento de

grandes volumes de dados por *workflows* científicos.

No Hadoop verificou-se que a execução do *workflow* utilizando dois *Maps* influenciou o desempenho. Neste caso, o último *Map* pode ser executado em um nó diferente do primeiro *Map*, implicando em transferência de dados devido a não localidade destes. Por estas razões, o Hadoop possui um desempenho levemente inferior quando comparado aos demais. Conclui-se que a unidade de trabalho local influencia diretamente na execução da aplicação. Ela garante várias atividades executando sobre um mesmo *Map* e consequentemente a localidade de dados entre as atividades do *workflow*, verificada no HaQoop-HDFS e HaQoop-Local. Adicionalmente, o HaQoop-HDFS dispõe da tolerância à falhas fornecida pelo HDFS que é relevante caso algum recurso se torne indisponível. Por outro lado perde-se em desempenho de leitura e escrita, mas principalmente na escrita de dados, utilizando HDFS ao invés do sistema de arquivos local usado no HaQoop-Local.

Por fim, a alternativa de execução utilizando o Spark levou ao melhor desempenho em todas as atividades e consequentemente no tempo de execução total comparada as demais alternativas avaliadas. Percebe-se que a utilização dos RDDs que mantém os dados intermediários na memória RAM beneficia o desempenho do *workflow* sem perder a perspectiva global de execução.

Capítulo 4

Abordagem Proposta

Neste capítulo é apresentada a abordagem aplicada para a otimização de *workflows* científicos MR em *clusters* computacionais através da configuração de parâmetros relacionados ao sistema utilizado e aos dados a serem processados. A abordagem proposta segue o modelo de otimização "caixa preta". Portanto, as recomendações para o ajuste dos parâmetros são baseadas nas observações reais de desempenho do sistema na execução de um determinado *workflow* em um determinado *cluster* variando as configurações dos parâmetros. Em algumas etapas da abordagem são utilizadas algumas definições, algoritmos e métodos de Aprendizado de Máquina, descritas na Seção 2.4. Foram utilizados algoritmos de aprendizado de máquina relacionados à discretização, classificação e validação, para o ajuste da configuração dos parâmetros. São descritos neste capítulo os critérios utilizados na seleção dos parâmetros, obtenção dos dados, treinamento e teste do modelo preditivo. A Seção 4.1 apresenta a formalização do problema e a Seção 4.2 apresenta a abordagem proposta para otimização de *workflows* científicos MR.

4.1 Definição e Modelagem do Problema

Nesta seção é apresentada a definição formal do problema, relacionada aos aspectos relevantes que são: estrutura do *workflow*, dados de entrada, ambiente de execução e configuração de parâmetros. São apresentados seus elementos e como foram utilizados na otimização da execução de um *workflow* MR em um ambiente de *cluster*.

Quanto ao ambiente de processamento, considera-se $R = \{r_1, ..., r_k\}$ como sendo o conjunto de máquinas que compõem o *cluster* e se encontram disponíveis e dedicadas para a execução de *workflows*. Nesta tese, os recursos que compõem o *cluster* são homogêneos, possuindo hardware (CPU, Memória RAM, Disco) e software idênticos.

Conforme as definições formais descritas nas Seções 2.1 e 2.2, dado um *workflow* W, um conjunto de dados de entrada D, e um conjunto de recursos R, seja $C = \{c_1, c_2, ..., c_m\}$ o conjunto de configurações de parâmetros para execução do *workflow* W. Consideremos ainda que cada configuração de parâmetros c_i representa os valores dos parâmetros de execução do *workflow*, como por exemplo: número de *cores* por *executor*, quantidade de memória por *executor*, os parâmetros relacionados com o particionamento dos dados, os parâmetros vinculados ao *cluster*, e assim por diante, conforme discutidos a seguir.

A fim de realizar a execução de um *workflow*, diversos valores podem ser especificados na configuração de parâmetros. Por exemplo, no framework Spark há mais de 180 parâmetros que podem ser configurados. Porém, nesta tese, foi selecionado um pequeno conjunto (6 parâmetros), de acordo com os experimentos anteriores [de Oliveira et al., 2014] [de Oliveira et al., 2015]. Estes possuem um impacto mais significativo no desempenho do *workflow*, como analisado na Seção 5.3. Além disso, estes parâmetros estão relacionados diretamente com os principais recursos de um *cluster* como CPU e Memória RAM. Finalmente, como observado em [Wang et al., 2016], os usuários geralmente encontram problemas de desempenho causados pela configuração errada destes parâmetros.

A Tabela 4.1 apresenta o subconjunto de parâmetros utilizado nesta tese na primeira coluna, a simbologia adotada na segunda coluna e na terceira o domínio dos valores usados na avaliação experimental descrita no Capítulo 6. Note que, o parâmetro DN é obtido de acordo com o valor definido nos parâmetros DS, DE e DP. Se a estratégia de particionamento dos dados DE realizar uma divisão balanceada, então $DN = \frac{DS}{DP}$. Além disso, o parâmetro EN em uma configuração c_i não pode ser superior a $\frac{CM \times CN}{EM}$. Desta forma, cada configuração $c_i = \{EN_i, EC_i, EM_i, DE_i, DN_i, DP_i\}$, sendo $EN_i \in EN, EC_i \in EC, EM_i \in EM, DE_i \in DE, DP_i \in DP, DN_i \in DN, DP_i \in DP$.

Variável	Significado	Valores
EN	Número de processos *Executors*	1 - 200
EC	Número de *Cores* a ser usado por processo *executor*	1 - 32
EM	Quantidade de memória utilizada por processo *executor*	1 - 64(GB)
DS	Tamanho do Conjunto de Dados de Entrada	1 - 24(GB)
DE	Estratégia de particionamento do Conjunto de Dados	FRANCE,ADHOC
DN	Número de Partições do Conjunto de Dados	1 - 512
DP	Tamanho de cada Partição do Conjunto de Dados	1 - 400(MB)
CN	Quantidade de Nós do *cluster*	1 - 6
CC	Número de *Cores* por Nó do *cluster*	1 - 32
CM	Quantidade de Memória por Nó do *cluster*	1 - 96(GB)

Tabela 4.1: Parâmetros Selecionados da Abordagem Proposta

Conforme definido em [de Oliveira, 2012], consideremos também que cada *cluster* computacional possui um valor associado chamado de índice de retardo de computação (do inglês *computational slowdown index – csi*). Uma vez que o ambiente de *cluster* é homogêneo o *csi* é obtido através de uma estimativa extra modelo: um valor é atribuído para cada máquina sem que se tenha realizado nenhum experimento nas mesmas.

Ainda, considera-se $P(W, R, C)$ como a proveniência do tempo de execução de um *workflow* W em um *cluster* R usando um conjunto de configurações C, assim sendo $P(W, R, C) = \{(W, csi(R), c_1, et_1), (W, csi(R), c_2, et_2), ..., (W, csi(R), c_m, et_m)\}$, onde et_i é o tempo de execução do *workflow* W no *cluster* R e com a configuração de parâmetros c_i.

Ainda define-se uma função de predição $Pred(W, c_i, P(W, R, C)) \rightarrow et_i$, então uma execução de W com menor tempo possível sob um conjunto de configurações C é dado por:

$$\min_{i=1,m} Pred(W, c_i, P(W, R, C)) \tag{4.1}$$

4.2 A Abordagem

A fim de encontrar o menor tempo de execução para um determinado *workflow* MR, deve-se encontrar a configuração correta dos parâmetros, a qual não é uma tarefa trivial, uma vez que deve ser avaliado uma grande variedade de combinações possíveis. Com isso, esta tese propõe uma solução para este problema, conforme especificada pelas etapas enumeradas a seguir e apresentada na Figura 4.1.

1 - Coleta de Dados: Primeiramente, deve-se produzir os dados para treinamento do modelo a partir de algumas execuções utilizando diferentes configurações de parâmetros. De acordo com a Figura 4.2, os dados de treinamento são produzidos para um dado *workflow*, um determinado conjunto de dados de entrada e um determinado *cluster*. Os dados de treinamento são gerados através de diversas execuções do *workflow* variando os valores dos parâmetros, especificados na Tabela 4.1, com o objetivo de obter o tempo de cada execução realizada. Uma questão importante a ser abordada neste contexto é a definição dos valores dos parâmetros que devem ser explorados na coleta dos dados de treinamento. Uma vez que o espaço de valores é muito grande e é praticamente impossível avaliar todos eles, foram definidos alguns valores entre os intervalos definidos na Tabela 4.1. Estes valores foram definidos de acordo com o conhecimento adquirido com os experimentos realizados e descritos na Seção 5.3;

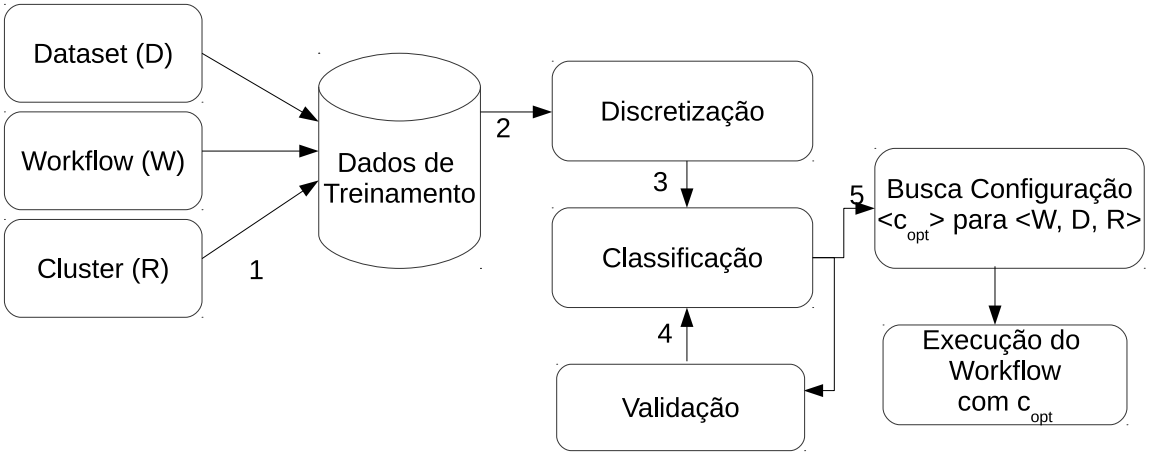

Figura 4.1: Abordagem de Otimização de desempenho de *Workflows* MR em Clusters Computacionais

2 - Discretização: Para representar o desempenho foi utilizado o tempo de execução total do *workflow* MR. Portanto, um problema é realizar a predição do tempo de execução de um *workflow* MR em um *cluster* sob um determinada configuração dos parâmetros. Todavia, realizar uma predição do valor específico do tempo de execução é muito difícil e por isso nesta tese é realizado uma discretização do tempo de execução em grupos. Com isso, o conjunto de dados de treinamento é discretizado pelo método *Equal-Frequency* em três grupos de acordo com o tempo de execução: *Baixo*, *Medio* e *Alto*;

3 - Classificação: Nesta etapa a árvore de classificação é construída a partir dos dados discretizados, gerando assim o modelo preditivo. Diante disso, o problema original é transformado em um problema de aprendizado de máquina, o qual é composto pelos métodos de discretização e árvore de classificação;

4 - Validação: Nesta etapa o modelo preditivo obtido na etapa anterior é submetido ao método de validação Cross-Validation[1] para avaliação das métricas *Precision*, *Recall*,

[1]O método *10-fold cross-validation* é realizado dividindo-se os dados em 10 subconjuntos mutuamente

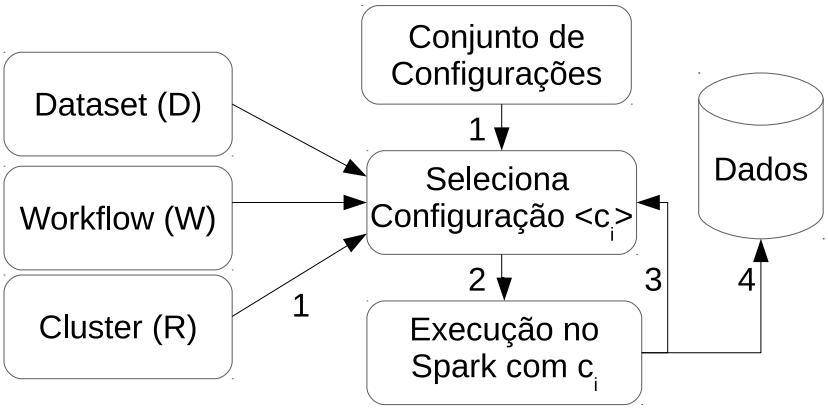

Figura 4.2: Coleta dos dados para treinamento do modelo preditivo

F-Measure (F1), e *Accuracy*;

5 - Busca Configuração: Na última etapa, antes de iniciar a execução do *workflow* MR é extraído do modelo preditivo um conjunto de regras que definem os parâmetros e valores a serem configurados que levam a uma execução eficiente, *i.e*, regras classificadas como *Baixo* de acordo com a árvore de decisão;

exclusivos de casos aproximadamente do mesmo tamanho. O modelo é treinado e testado 10 vezes, cada vez testado com um subconjunto (10% dos exemplos) e treinado com o restante do conjunto de dados (90% dos exemplos). A estimativa da precisão é a média das precisões estimadas usando-se os 10 subconjuntos.

4.3 Resumo do Capítulo

Neste capítulo foi descrita a formalização do problema através da definição de *workflows*, atividades, ambiente de execução e configuração de parâmetros. Por fim, foi apresentada a abordagem composta basicamente pelas etapas de treinamento, discretização, classificação e validação.

Capítulo 5

Modelagem dos Experimentos

Para análise da abordagem proposta, este capítulo apresenta a modelagem experimental considerando as aplicações científicas, ambientes de execução e o subconjunto de configurações de parâmetros relacionados aos dados de entrada e ao Spark. Os parâmetros avaliados envolvem as estratégias de particionamento de dados, tamanho das partições e os parâmetros diretamente relacionados ao *framework* Spark. Para esta avaliação, modelamos os *workflows* SkyMap e Constellation Queries conforme apresentado na Seção 5.1 no *framework* Spark. Primeiramente, na Seção 5.1 são apresentados os *workflows* científicos reais da astronomia utilizados para avaliação da abordagem, assim como alguns conceitos da área de astronomia necessários para compreensão do funcionamento dos *workflows*. Na Seção 5.2 são descritos os ambientes de execução dos experimentos avaliados nesta tese. Por fim, na Seção 5.3 são apresentados os resultados de experimentos que apontam o impacto da configuração de parâmetros no desempenho do *workflow*.

5.1 Estudos de Caso

Para avaliação da abordagem dois tipos de *workflows* científicos da área de astronomia foram selecionados, denominados SkyMap e *Constellation Queries*. Uma vez que os *workflows* utilizados são da astronomia, nesta seção são descritos alguns conceitos importantes desta área científica e relevantes para a compreensão do funcionamento dos *workflows*. O catálogo é o conjunto de dados de entrada dos *workflows* e na seção 5.1.1.1 é apresentado como esses dados são obtidos. Na seção 5.1.1.2, é descrito o conceito do sistema de coordenadas que está atrelado ao problema tratado pelo *workflow* SkyMap e, na seção 5.1.1.3, é descrito o conceito de padrões geométricos que está relacionado ao problema atacado pelo *workflow* Constellation Queries. Por fim, também são descritos o funcionamento de

cada *workflow* e suas atividades assim como foi modelado a execução paralela dos mesmos no sistema Spark.

5.1.1 Astronomia

Astronomia é uma ciência que busca estudar os corpos celestes, desenvolver e testar teorias confrontando-as com a observação dos fenômenos. Uma sub-área de estudo é a análise da características físicas dos astros, os seus movimentos e a disposição no espaço em relação aos demais corpos da região. Alguns exemplos das características físicas são: análise da sua massa e também a composição química de sua superfície.

5.1.1.1 Catálogo

Os catálogos são criados a partir várias fotografias do espaço obtidas através de câmeras dos telescópios, unificadas e processadas por algoritmos que identificam os corpos celestes. O resultado desse processo é um conjunto de dados contendo todas as informações tais como as coordenadas *Right ascension(ra)*, *declination(dec)*, luminosidade e outros atributos de cada objeto identificado. O conjunto de objetos celestes identificados em um levantamento astronômico formam um catálogo, em que cada entrada corresponde a um objeto.

Existem diversos projetos voltados para a criação e análise de catálogos da astronomia como por exemplo o *Sloan Digital Sky Survey*[1] projeto iniciado em 2000, que utilizam câmeras digitais de alta resolução para criar mapas tridimensionais detalhados do universo com milhões de galáxias.

5.1.1.2 Sistema de Coordenadas

Para encontrar um planeta, estrela e outros objetos no céu, é necessário ter um sistema de coordenadas. Este sistema composto por *ra*, *dec* especifica sua posição de modo único, assim como a latitude e longitude de um objeto na superfície da Terra definem uma localização única. Em outras palavras, o sistema de numeração usado tanto em *ra* quanto em *dec*, vem com um valor zero definido inicialmente. O *ra* varia de 0 a 360 enquanto *dec* varia de -90 até 90 como mostra a Figura 5.1. O ponto zero da escala em *ra* e *dec* são medidos em horas, minutos e segundos, sendo 24 horas equivalente a um círculo completo,

[1]Disponível em `www.sdss.org`

porque determina a localização de uma estrela cronometrando sua passagem pelo ponto mais alto do céu à medida que a Terra gira.

O ponto mais alto do céu, chamado meridiano (perpendicular ao equador), é a projeção da longitude sobre a esfera celeste. O círculo completo contém 24 horas de *ra* ou 360° (graus de arco), uma hora de *ra* corresponde a 15°. Um minuto de *ra* equivale a $\frac{15}{60}^{\circ}$.

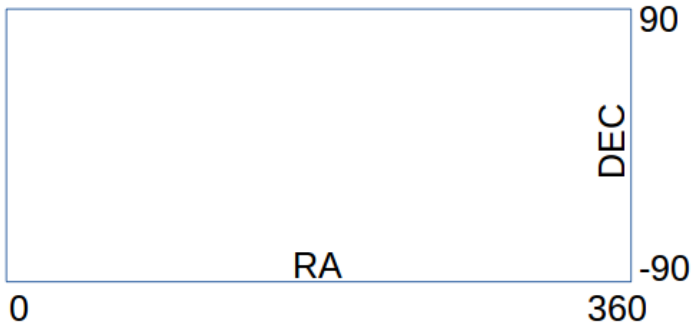

Figura 5.1: Sistema de coordenadas RA e DEC

5.1.1.3 Padrões Geométricos

Padrões geométricos representam um conjunto de pontos que formam uma figura, como por exemplo: um quadrado, retângulo, triângulo, círculo e figuras mais complexas. Desta forma, pode-se falar que este conjunto de pontos expressa uma consulta do tipo geométrica na qual a forma geométrica é dada pelas distâncias entre cada ponto. Na astronomia, um exemplo de padrão geométrico é o Einstein Cross, um fenômeno da astronomia previsto na teoria da relatividade de Albert Einstein. O Einstein Cross ocorre pelo efeito das lentes gravitacionais e a sua representação é semelhante a uma cruz como mostra a Figura 5.2.

Encontrar o padrão Einstein Cross, assim como outros padrões, é um desafio, pois a quantidade de combinações de pontos que podem definir o padrão é exponencialmente grande dentro de um catálogo. Em geral, para um conjunto de dados D e uma quantidade de pontos k no padrão, um limite superior de combinações candidatas $\frac{|D|}{k}$ é a quantidade de formas para escolher k itens a partir de D. Este problema se torna ainda mais complexo se considerarmos como solução uma composição de pontos similar ao padrão geométrico em uma escala espacial relativamente diferente.

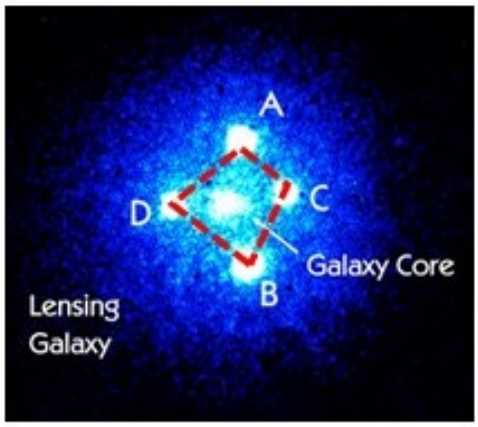

Figura 5.2: *Einstein Cross*

5.1.2 *Workflow SkyMap*

Existem diversas aplicações que processam os catálogos da astronomia, e neste caso, o *workflow* científico SkyMap tem por objetivo produzir um histograma da região do céu investigada. Cada objeto celeste identificado no levantamento astronômico aparece como um ponto na posição indicada por sua coordenada esférica. O *workflow* SkyMap acessa um catálogo astronômico produzido pelo levantamento *Dark Energy Survey* (DES) [2]. Estes objetos são acessados pelo *workflow* a partir do catálogo armazenado de forma distribuída no sistema de arquivos HDFS. O *workflow* SkyMap produz o histograma do céu pintando cada um dos objetos celestes contidos no catálogo em uma representação $2D$ da região de interesse do céu. O processo é realizado modelando um *workflow* com três atividades, conforme apresentado na Figura 5.3:

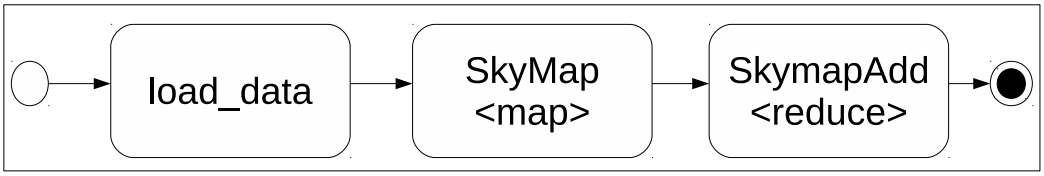

Figura 5.3: *Workflow* que representa a aplicação da astronomia SkyMap

1. **load-data**: recuperação dos dados a partir do catálogo e tratamento para adequação ao formato esperado pela próxima atividade, SkyMap. O resultado desta atividade

[2] www.darkenergysurvey.org

é mantido na memória principal;

2. **SkyMap**: Execução da aplicação *SkyMap* com base na indicação do posicionamento de cada objeto no arquivo de entrada. A atividade colore cada objeto em um histograma da região do céu sendo analisada. Sua implementação na etapa *Map* realiza a leitura dos dados gerados na atividade anterior, processa e armazena os histogramas em memória, pois serão consumidos pela atividade SkymapAdd;

3. **SkyMapAdd**: Execução da aplicação *SkyMapAdd* que consolida os diversos histogramas gerados em um único. Uma instância do *Reduce* acessa os vários dados gerados na atividade SkyMap e produz o histograma final. Este histograma final é armazenado na memória secundária.

Nas Figuras 5.4 e 5.5 são apresentados os modelos de execução paralela do *workflow* SkyMap com os diferentes tipos de particionamento dos dados de entrada descritos na Seção 2.2. A Figura 5.4 apresenta como o *workflow* é executado quando os dados de entrada estão particionados de acordo com os métodos FRANCE ou com QuadTree.

Observa-se que as atividades *load-data* e SkyMap são executadas uma após a outra de forma paralela e a atividade SkyMapAdd é executada por último de forma sequencial. Adicionalmente, cada tarefa da atividade *load-data* recebe uma partição p_j e um metadado $[i_j,f_j]$ que indica o intervalo de dados daquela partição. Neste trabalho, como os dados a serem processados são coordenadas de objetos da astronomia, este metadado $[i_j,f_j]$ corresponde ao menor e ao maior *ra* da partição p_j.

Ainda na Figura 5.4, cada tarefa da atividade SkyMap é responsável por gerar um histograma e enviá-lo à atividade SkyMapAdd que irá produzir a imagem final. O tamanho do histograma é equivalente ao intervalo de dados processado. As coordenadas dos objetos estão por padrão incluídas entre um *ra* mínimo e máximo que é 0 e 360 respectivamente. Como neste modelo é conhecido como os dados estão particionados, seja pelo método FRANCE ou QuadTree, sabe-se previamente o intervalo de dados de cada partição especificamente. Portanto, o histograma produzido por cada tarefa paralela é de acordo com o intervalo de dados da partição i_j,f_j e não com o intervalo total (0 e 360). Sendo assim, cada histograma é exatamente proporcional à região dos dados da partição analisada.

Por exemplo, a Figura 2.2 ilustra um particionamento dos dados usando o método QuadTree. Cada partição, representada por um quadrado, possui limites mínimos e máximos de *ra* e *dec*. Portanto, a atividade SkyMap irá gerar diversos histogramas, sendo

cada um correspondente a uma e somente uma partição.

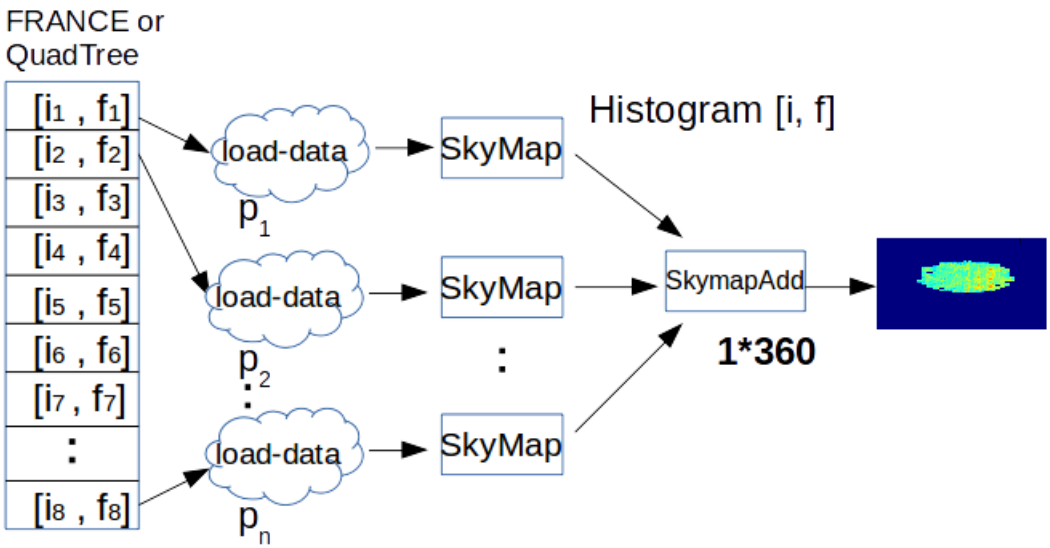

Figura 5.4: Modelo de Execução do SkyMap com Particionamento Espacial dos Dados

A Figura 5.5 apresenta como o *workflow* é executado quando os dados de entrada estão particionados sem a utilização de um método ou critério específico. Com isso, cada tarefa da atividade *load-data* recebe uma partição. Como neste caso é desconhecido o intervalo de dados contidos na partição, uma mudança ocorre nas atividades Skymap e SkymapAdd. Na atividade SkyMap o histograma produzido por cada tarefa paralela é de acordo com o intervalo total dos dados (0 e 360). Portanto serão produzidos n histogramas com mesmo tamanho (0 e 360), onde n é o número de tarefas da atividade SkyMap. Cada tarefa irá enviar o histograma à atividade SkyMapAdd que irá produzir a imagem final.

Conclui-se que o modelo AdHoc realiza um maior processamento nas atividades SkyMap e SkymapAdd quando comparado aos modelos FRANCE e QuadTree. Também ocorre uma maior transferência de dados entre as atividades SkyMap e SkyMapAdd no modelo AdHoc do que nos demais. Estas características apontam que o tipo de particionamento dos dados afetam a execução de uma aplicação. Os experimentos descritos na Seção 5.3 ratificam a importância de utilizar um método de particionamento dos dados e também que a aplicação conheça o critério de particionamento.

5.1.3 *Workflow Constellation Queries*

Este *workflow* da astronomia tem como objetivo encontrar padrões geométricos, especificamente o Einstein Cross, em um determinado conjunto de dados [Porto et al., 2017] [Khatibi et al., 2017]. No *workflow*, o padrão Einstein Cross a ser encontrado no *dataset*

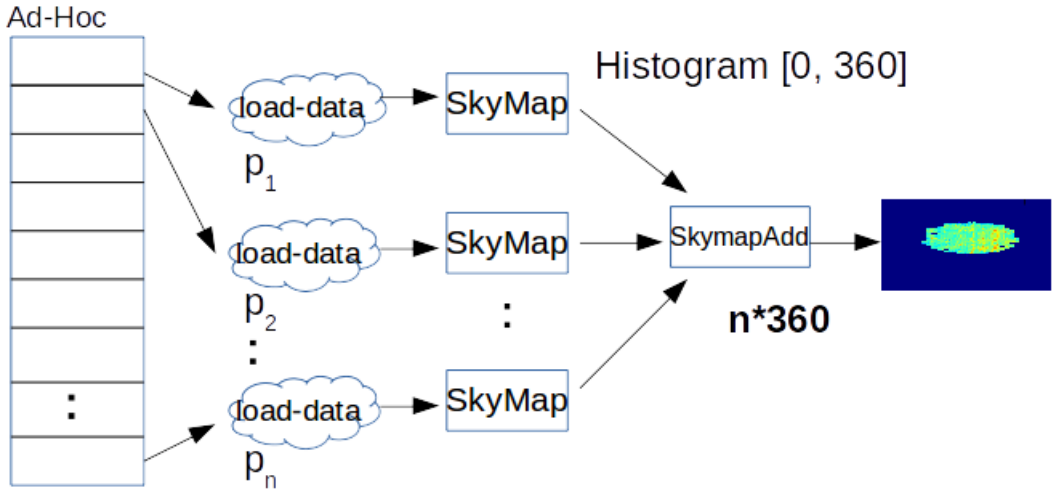

Figura 5.5: Modelo de Execução do SkyMap com Particionamento AdHoc dos Dados

é especificado por um conjunto de pontos ($E1$, $E2$, $E3$ e $E4$), como mostra a Figura 5.6, cada ponto com seus atributos (*ra*, *dec*, *id*, etc). Portanto, o conjunto de dados é processado de forma a buscar conjuntos de pontos cuja forma seja similar à amostra ($E1$, $E2$, $E3$ e $E4$). Assim como o *workflow* SkyMap, o *Constellation Queries* acessa um catálogo astronômico armazenado de forma distribuída no sistema de arquivos HDFS. De acordo com a solução implementada por este *workflow*, os dados devem estar obrigatoriamente particionados de acordo com a estratégia de particionamento Hierárquica ou **Equi-depth**, descrita na Seção 2.2.2. O artigo [Porto et al., 2017] descreve com maiores detalhes a solução implementada neste *workflow*.

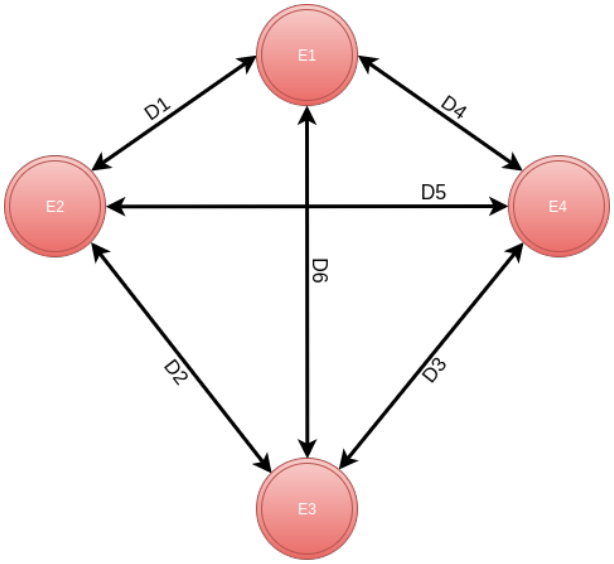

Figura 5.6: Distâncias relativas entre os pontos do *Einstein Cross*

O processo é realizado modelando um *workflow* com três atividades, conforme apre-

sentado na Figura 5.7:

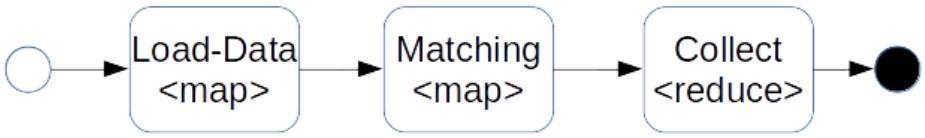

Figura 5.7: *Workflow* que representa a aplicação da astronomia Constellation Queries

1. **Load-Data**: recuperação dos dados a partir do catálogo e tratamento para adequação ao formato esperado pela próxima atividade, Matching. Além disso, deve ser fornecido o conjunto de pontos que formam o padrão a ser buscado. O resultado desta atividade é mantido na memória principal. Implementado como um Map;

2. **Matching**: Realiza a busca por padrões geométricos similares à amostra com base na indicação do posicionamento de cada objeto no arquivo de entrada;

3. **Collect**: Uma instância do *Reduce* acessa os vários dados gerados na atividade (2) e consolida os diversos padrões geométricos encontrados em um arquivo, que por sua vez, é armazenado na memória secundária.

Devido à semelhança de operações Spark entre os *workflows* Constellation Queries e SkyMap, a arquitetura de execução paralela do primeiro é a mesma arquitetura do *workflow* SkyMap, apresentado na Figura 5.4, substituindo apenas o nome das atividades (2) e (3) e restringindo a estratégia de particionamento Hierárquica.

5.2 Configuração do Ambiente de Execução

Para a realização dos experimentos deste trabalho, os *frameworks* Spark (versão 2.0) e YARN (versão 2.7) foram utilizados em dois ambientes, cada um composto de um *cluster* de computadores de características distintas, conforme descrito na Tabela 5.1. Os *workflows* foram implementados através da linguagem de programação Python (versão 2.7).

- Petrus: todos os nós utilizam processadores Intel Xeon E5-2630 V3 2.4GHz, memória RAM 96GB (6x 16GB) RDIMMs e sistema operacional CentOS 6.7 (64 bits).

- Minicluster: o nó mestre utiliza processadores Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, memória RAM 16GB e os nós trabalhadores possuem processadores AMD Phenom(tm) II X2 B55 Processor e memória RAM 8GB. Todos os nós deste *cluster* possuem o sistema operacional Ubuntu 14.04 (64 bits).

Uma vez que o modelo de processamento Spark é baseado no modelo mestre-trabalhador, ambos os *clusters* possuem um nó físico chamado aqui de *mestre* e os nós chamados de trabalhadores, onde serão executadas as tarefas. A Tabela 5.1 apresenta as características físicas complementares de cada *cluster*.

Cluster	Tipo do Nó	Qtd. Nós	Cores/Nó	Mem/Nó(GB)	HD/Nó(TB)
Petrus	Mestre	1	8	96	0.5
	Trab.	6	32	96	16
Minicluster	Mestre	1	8	16	1.5
	Trab.	4	2	8	0.5

Tabela 5.1: Configuração de Hardware dos Ambientes de Execução

A Tabela 5.2 apresenta as configurações gerais do Spark em cada *cluster*. Especificamente, no *cluster* Petrus, além do Spark, existem diversos outros *frameworks* disponíveis para realização de experimentos e execução de aplicações paralelas, como: Ambari, YARN [Vavilapalli et al., 2013], HBase [George, 2011], Hive [Thusoo et al., 2010], Pig [Pig,], Storm [Storm,], dentre outros. Por exemplo, o *framework* Ambari é utilizado para gerenciamento e monitoramento de *clusters* Hadoop através de uma interface Web. Cada *framework* possui uma finalidade específica dentro do *cluster* Petrus, que é utilizado por vários cientistas. Com isso, a quantidade de memória e *cores* disponibilizada em cada máquina do *cluster* Petrus para o Spark foram limitados em 80GB e 25 *cores* respectivamente, para que estes outros serviços continuem executando e o *cluster* também possa ser acessado pelos usuários. Observa-se porém que nenhuma outra aplicação de outro usuário era executada concorrentemente e/ou competia por recursos durante a realização dos experimentos analisados nesta tese. Também, destaca-se que nos experimentos apresentados nesta tese foram utilizados somente os *frameworks* Spark e o YARN definidos na seção 2.3.2.

Em ambos os *clusters* todos os nós foram configurados para serem acessados utilizando SSH sem verificação de senha. Em termos de software, todos os nós, não importando seu tipo, executam os mesmos programas e configurações.

Cluster	Tipo do Nó	Qtd. Nós	Cores/Nó	Mem/Nó(GB)
Petrus	Mestre	1	8	80
	Trab.	6	25	80
Minicluster	Mestre	1	8	16
	Trab.	4	2	8

Tabela 5.2: Configuração do Spark nos Ambientes de Execução

5.3 Seleção dos Parâmetros

Um problema que surge nesse contexto é a seleção e configuração dos parâmetros relacionados ao *framework* de execução do experimento. No Spark, por exemplo, pode-se definir a quantidade de processos paralelos (Quantidade de *Executors*), tamanho da memória de cada processo, quantidade de núcleos a serem utilizados, particionamento dos dados, tamanho da partição, dentre outros. Cada configuração de parâmetros irá afetar de alguma forma o desempenho da aplicação como foi constatado nos experimentos realizados e descritos nesta seção. Portanto, obter um bom desempenho é extremamente importante, mas não é um exercício trivial devido à grande quantidade de parâmetros que estão relacionados à execução de um *workflow*.

Portanto, nesta seção é apresentada a relevância dos ajustes de configuração de parâmetros de execução do Spark para a otimização do desempenho do *workflow* SkyMap no *cluster* Petrus. Neste contexto, foram selecionados um total de seis parâmetros que estão descritos na Tabela 5.3. Os três primeiros (EN, EC, EM) estão relacionados diretamente com o Spark e os outros três parâmetros (DS, DE, DP) vinculados ao particionamento dos dados de entrada. A escolha de cada parâmetro foi baseada nos experimentos descritos a seguir nesta seção.

Variável	Significado
EN	Número de processos *Executors*
EC	Número de Cores por *Executor*
EM	Quantidade de Memória por *Executor*
DS	Tamanho do *dataset* de entrada
DE	Estratégia de particionamento do *dataset*
DP	Tamanho da Partição
DN	Número de Partições

Tabela 5.3: Conjunto de parâmetros utilizados nos experimentos descritos nesta seção

5.3.1 Análise das Estratégias de Particionamento

Para inicialmente mostrar a importância da especificação de uma estratégia de particionamento e sua possível relação com a classe de aplicação considerada, o experimento a seguir compara os particionamentos FRANCE e AdHoc na execução do *Workflow* SkyMap no cluster Petrus considerando a configuração de parâmetros conforme Tabela 5.4. Os valores dos parâmetros foram escolhidos de acordo com as dimensões do *cluster* Petrus. Os resultados dos experimentos apresentados nas Figuras 5.8, 5.9 e 5.10 foram gerados com a configuração fixa dos parâmetros especificados na Tabela 5.4 variando a estratégia de particionamento. Para cada estratégia de particionamento avaliada foram realizadas cinco execuções:

Configuração	EN	EC	EM	DS	DN	DP
$C1$	18	2	16GB	24GB	256	96MB

Tabela 5.4: Configuração dos Parâmetros para execução do workflow SkyMap no cluster Petrus variando a estratégia de particionamento

A Figura 5.8 apresenta o tempo médio de execução de cada atividade do *workflow* SkyMap (*Load-Data*, SkyMap e SkymapAdd) utilizando as estratégias de particionamento FRANCE e AdHoc no *cluster* Petrus.

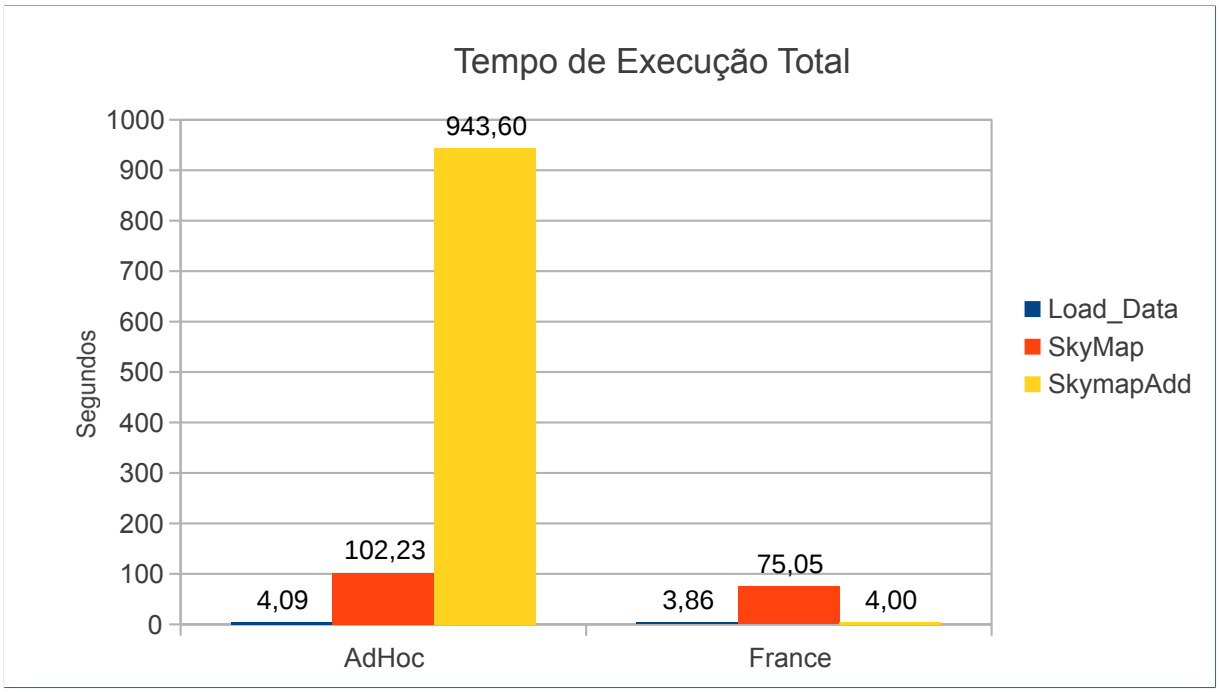

Figura 5.8: Tempo de Execução do *Workflow* Skymap no framework Spark e no cluster Petrus

Primeiramente, pode-se observar que o tempo de execução da atividade *Load-data* é semelhante nas duas estratégias de particionamento, visto que independentemente da estratégia, cada tarefa paralela deve ler o mesmo volume de dados, que é definido pelo parâmetro DP da Tabela 5.4.

Na atividade SkyMap observa-se uma ligeira diferença entre as estratégias de particionamento: o tempo desta fase no caso do particionamento FRANCE é aproximadamente 30% melhor quando comparado com o AdHoc. Isto pode ser justificado no fato das tarefas que executam a atividade SkyMap sobre dados particionados AdHoc realizarem um volume maior de processamento. Como descrito na Seção 5.1.2, na atividade SkyMap e particionamento AdHoc, o histograma produzido por cada tarefa paralela é de acordo com o intervalo total dos dados (0 e 360). Por outro lado, na estratégia de particionamento FRANCE, o histograma produzido por cada tarefa paralela é de acordo com o intervalo de dados de cada partição, uma vez que este intervalo de dados é conhecido. Portanto, esta menor carga de processamento proporcionada pela estratégia de particionamento FRANCE leva também a um menor tempo de execução da atividade SkyMap.

Já a atividade SkymapAdd teve um desempenho muito pior com a estratégia de particionamento AdHoc do que o particionamento FRANCE. No modelo de particionamento AdHoc não é conhecido o critério de particionamento e cada tarefa SkyMap gera um histograma com o tamanho total do céu e envia para a tarefa SkymapAdd. No FRANCE o critério de particionamento é conhecido e consequentemente cada tarefa SkyMap gera um histograma proporcional ao tamanho da partição. Isto gera menor transferência de dados na rede e também menor carga de processamento para a atividade SkymapAdd.

Portanto, pode-se observar que conhecer o critério de particionamento dos dados é muito importante para o desempenho do *workflow* SkyMap. A Figura 5.9 mostra o impacto disto no tempo total de execução desta aplicação considerando as duas estratégias de particionamento. Quando usado o particionamento AdHoc o tempo total é aproximadamente nove vezes maior do que o tempo total sob a estratégia de particionamento FRANCE.

O ponto principal destes resultados não é apresentar que a estratégia de particionamento FRANCE leva a um melhor desempenho do que a estratégia AdHoc, mas sim mostrar que particionar os dados e conhecer o critério de particionamento é importante para o desempenho deste *workflow*. Para tal, na Figura 5.10 observa-se o tempo total de execução do *Workflow* SkyMap nas estratégias de particionamento Quadtree, AdHoc e FRANCE. Observa-se que o tempo de execução no particionamento QuadTree é aproxi-

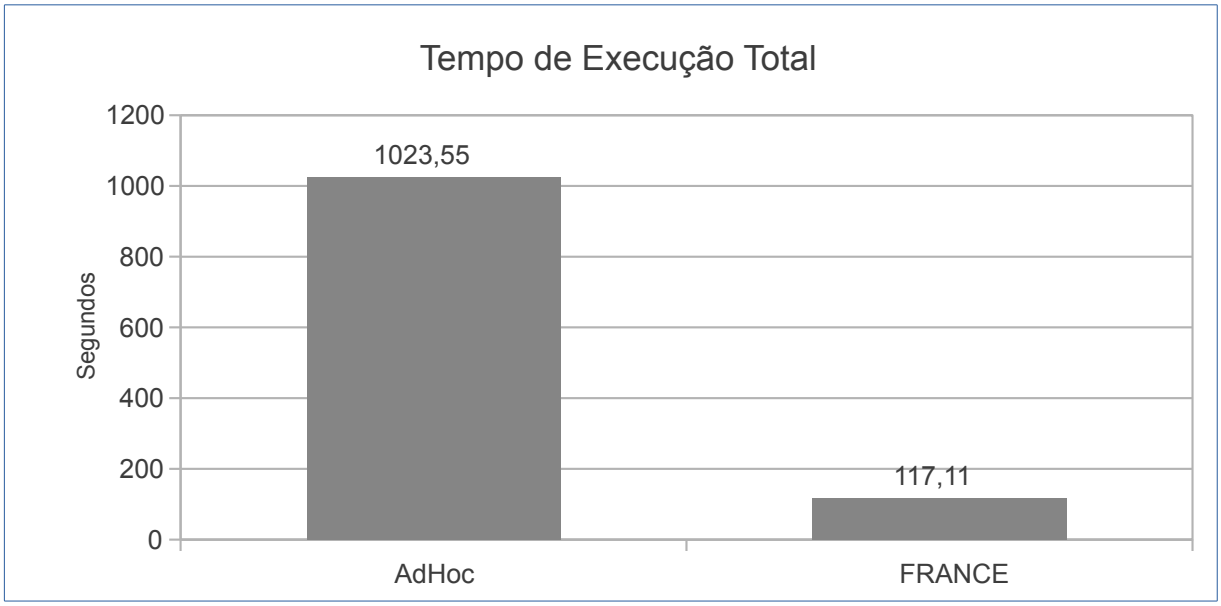

Figura 5.9: Tempo Total de Execução do Workflow SkyMap no Petrus considerando particionamento FRANCE e AdHoc

madamente oito vezes menor do que o tempo total no AdHoc. De uma forma geral, as mesmas vantagens de processamento das atividades do *workflow* obtidas pelo FRANCE frente ao AdHoc também são observadas na estratégia QuadTree. Portanto, estes resultados apresentados ratificam a importância de utilizar um método de particionamento dos dados e também que a aplicação conheça o critério de particionamento.

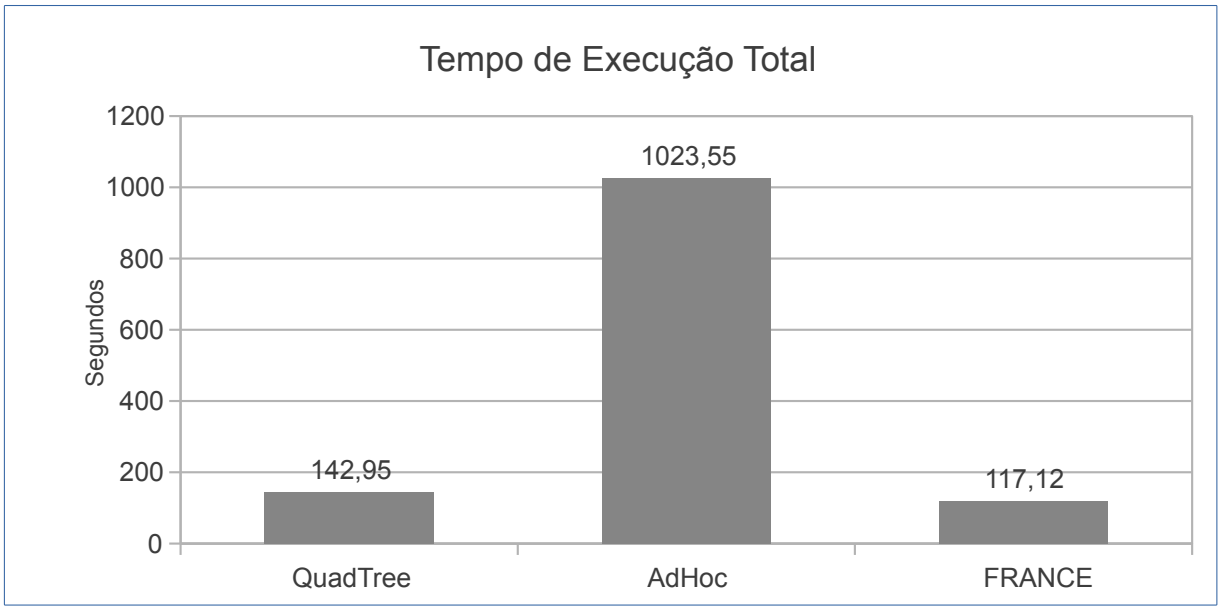

Figura 5.10: Tempo Total de Execução do Workflow SkyMap no Petrus considerando as estratégias de particionamento FRANCE, QuadTree e AdHoc

5.3.2 Análise do Tamanho da Partição

A fim de apresentar a importância do tamanho da partição, os experimentos descritos nesta seção varia o tamanho das partições na execução do *workflow* SkyMap no Petrus. O experimento apresentados nas Figuras 5.12 e 5.11 utilizou a configuração fixa dos parâmetros descrita na Tabela 5.5 variando o tamanho das partições e a estratégia de particionamento. Os valores dos parâmetros foram escolhidos de acordo com as dimensões do *cluster* Petrus. Para cada tamanho de partição avaliado foram realizadas cinco execuções:

Configuração	EN	EC	EM	DS
$C2$	18	1	16GB	24GB

Tabela 5.5: Configuração dos Parâmetros para execução do *workflow* SkyMap no Petrus variando o número de partições em cada estratégia de particionamento

A Tabela 5.6 apresenta a variação do número de partições, tamanho da partição e o número de objetos por partição utilizados nos experimentos. O número total de objetos do *dataset* é de 1.177.512.149 (um pouco mais de 1 bilhão de objetos).É importante destacar que, quanto mais partições menor será o tamanho de cada partição, uma vez que o tamanho do *dataset* está fixado em 24GB. Por exemplo, se o número de partições for 64 então cada partição é de 384MB e possui 18.398.627 objetos. Também pode-se destacar que o número de processos (Qtd. *Executors*), assim como a quantidade de memória para cada *executor*, para todos os testes está fixado de acordo com a Tabela 5.5. Portanto não há um aumento no número de processos (grau de paralelismo) quando o número de partições aumenta.

Número de Partições (DN)	Tam. da Partição(MB) (DP)	Número de Objetos / Partição
512	48	2.299.828
256	96	4.599.656
128	192	9.199.313
96	256	12.265.751
64	384	18.398.627

Tabela 5.6: Variação do Tamanho da Partição, Número de Partições e Objetos por partição do *dataset* de 24GB utilizada nestes experimentos

As Figuras 5.11 e 5.12 apresentam o tempo médio de execução de cada atividade do *workflow* (Load-Data, SkyMap e SkymapAdd) e o tempo total variando o número de partições do *dataset* e consequentemente o tamanho da partição. Na Figura 5.11 as

partições foram obtidas pela estratégia FRANCE e na Figura 5.12 as partições foram geradas pela estratégia AdHoc.

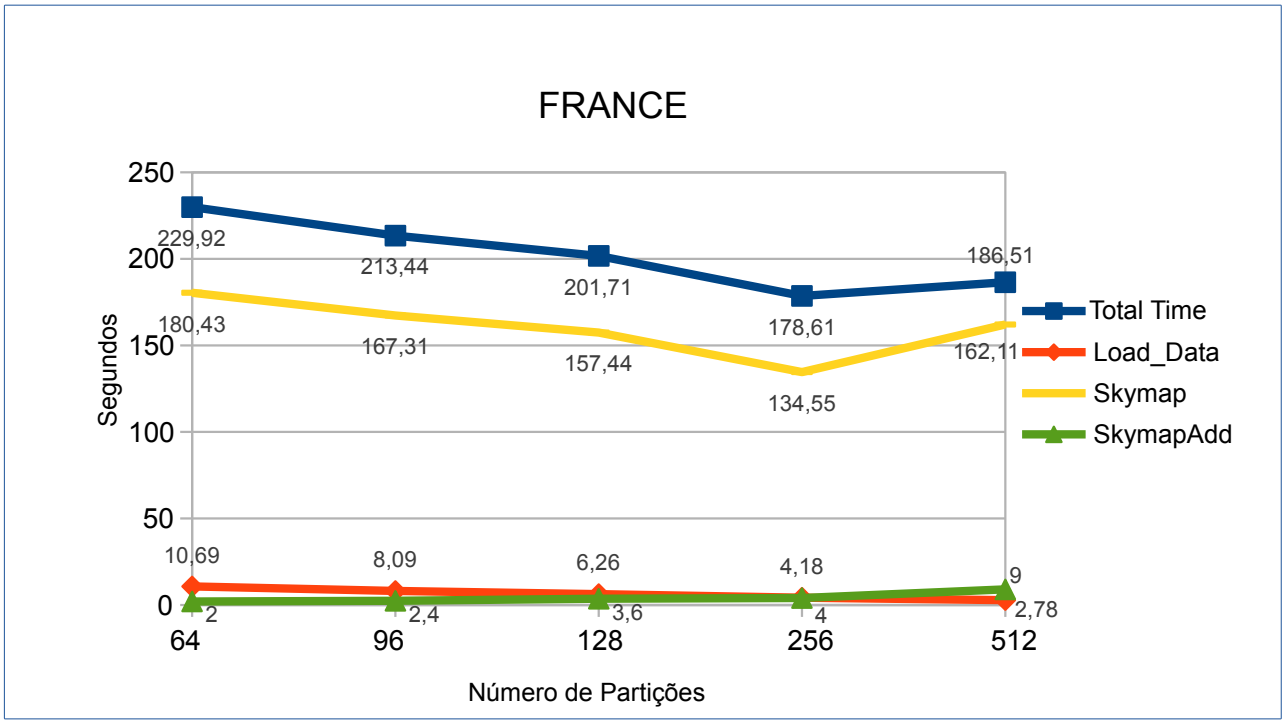

Figura 5.11: Tempo de Execução total do *workflow* SkyMap e de cada atividade variando o número de partições com estratégia de particionamento FRANCE no *cluster* Petrus

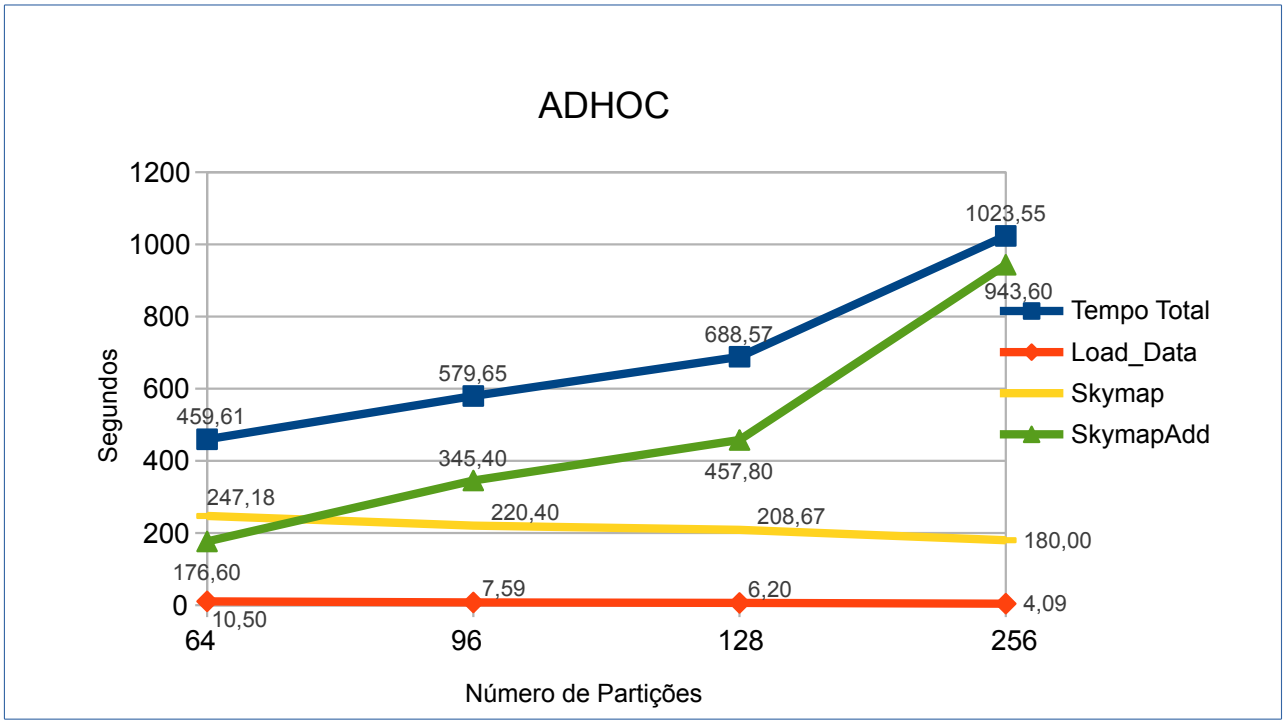

Figura 5.12: Tempo de Execução total do *workflow* SkyMap e de cada atividade variando o número de partições com estratégia de particionamento AdHoc no *cluster* Petrus

Analisando as Figuras 5.11 e 5.12 pode-se observar que, independente da estratégia de particionamento, as atividades Load-Data e SkyMap são beneficiadas pelo aumento do número de partições. Isto aponta que partições menores influenciam positivamente o tempo de execução das atividades paralelas do *workflow* SkyMap no Spark. Porém, pode-se observar na Figura 5.11 que há uma piora no tempo de execução da atividade SkyMap quando o número de partições é igual a 512. Neste caso, cada partição possui 48MB, e o tempo de processamento de cada partição é muito rápido, levando a um pior resultado quando comparado à execução com 256 partições. Na Figura 5.11 o tempo de execução da atividade SkyMap com 256 partições é aproximadamente 18% mais rápido do que com 512 partições.

Analisando especificamente a atividade SkymapAdd, o tempo de execução desta atividade aumenta 60% em média a cada aumento do número de partições no particionamento FRANCE, como apresentado na Figura 5.11. Para explicar este aumento é necessário atentar para o funcionamento desta atividade em ambos os particionamentos, como apresentado na Seção 5.1.2. Quanto mais partições maior será o número de histogramas gerados na atividade Skymap e recebidos pela atividade SkymapAdd. Como a atividade SkymapAdd é sequencial, este maior número de histogramas a serem processados leva a um tempo maior de execução conforme aumentam as partições, e isto justifica o leve aumento com o particionamento FRANCE. Porém, o tempo total não é influenciado na mesma proporção uma vez que o peso desta atividade no tempo total é muito pequeno, cerca de 2,2% em média. Como explicado anteriormente, o tempo de execução desta atividade com o FRANCE é favorecido pelo conhecimento do critério de particionamento pelo *workflow*.

De acordo com a Figura 5.12 o tempo de execução aumenta 80% em média, conforme o número de partições aumenta, no particionamento AdHoc. Isto se dá pelo fato deste modelo de execução desconhecer o critério de particionamento dos dados. No AdHoc, o tempo da atividade SkymapAdd é definido pelo número de partições multiplicado pelo espaço total dos dados (360), apresentado na Figura 5.5 na Seção 5.1.2. Com este fato a mais, conclui-se que quanto mais partições, pior será o tempo de execução desta atividade e consequentemente o tempo total, pois o peso desta atividade no tempo total é em média de 75%.

5.3.3 Variando os parâmetros do Spark e fixando os parâmetros relacionados ao particionamento dos dados

Com o objetivo de mostrar a importância do ajuste dos parâmetros, foram selecionados dois conjuntos de parâmetros $C1$ e $C2$ que possuem os valores apresentados na Tabela 5.7. Nesta avaliação, os parâmetros relacionados ao particionamento de dados são tais como definidos em $C1$ e $C2$, há somente alteração dos parâmetros do Spark (EN, EC, EM). Para cada configuração de parâmetros $C1$ e $C2$ avaliada foram realizadas cinco execuções.

Configuração	EN	EC	EM	DS	DE	DN	DP
$C1$	66	1	4MB	24GB	FRANCE	256	96MB
$C2$	18	2	16MB	24GB	FRANCE	256	96MB

Tabela 5.7: Conjuntos de Configuração de Parâmetros Avaliados na Execução do *Workflow* SkyMap no *cluster* Petrus

A Figura 5.13 apresenta a média dos tempos de execução de $C1$ e $C2$. Estes resultados experimentais têm como objetivo observar que ajustando um pequeno número de parâmetros do Spark o tempo de execução pode ser reduzido em aproximadamente 35%. Este melhor desempenho obtido pela configuração $C1$ é devido ao maior grau de paralelismo proporcionado por esta configuração, sendo este grau calculado da seguinte forma: $EN \times EC$. Com isso, a configuração $C1$ possui 66 processos paralelos e a configuração $C2$ possui somente 36, justificando o melhor desempenho de $C1$, visto a disponibilidade de recursos computacionais para a execução paralela. Por outro lado, para aumentar o grau de paralelismo na execução de *workflows* intensivos de dados deve ser levado em consideração a quantidade de memória para cada *executor* (EM) e também o tamanho da partição (DN). Caso estes parâmetros não sejam considerados, a execução pode vir a falhar por falta de espaço na memória dos processos *executors*, como é apontado pela análise dos experimentos descritos na seção 5.3.5.

5.3.4 Variando os parâmetros relacionados ao particionamento dos dados e fixando os parâmetros do Spark

Nesta avaliação foram selecionados dois conjuntos de parâmetros $C3$ e $C4$, cujos valores estão apresentados na Tabela 5.8. Nesta avaliação, os parâmetros relacionados ao Spark são iguais em $C3$ e $C4$, há somente alteração dos parâmetros vinculados ao particionamento de dados (DE, DN). Para cada configuração de parâmetros $C3$ e $C4$ avaliado

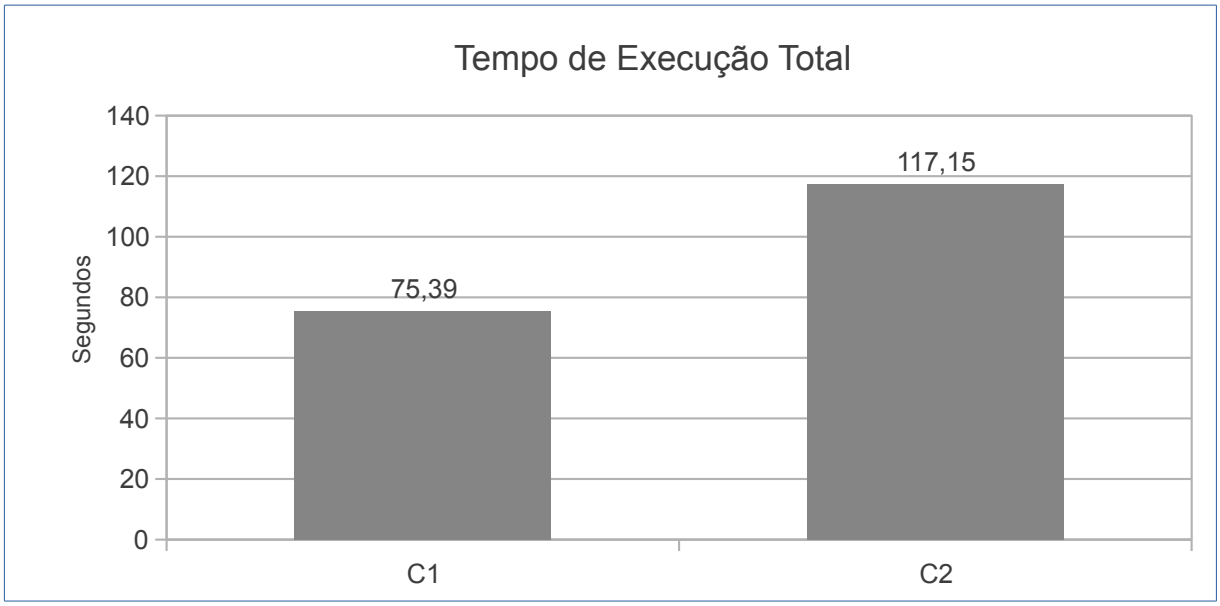

Figura 5.13: Tempo de Execução total obtido com a execução do *workflow* SkyMap no Petrus com as configurações de parâmetros $C1$ e $C2$

foram realizadas cinco execuções.

Configuração	EN	EC	EM	DS	DE	DN	DP
$C3$	12	4	32MB	24GB	FRANCE	128	192MB
$C4$	12	4	32MB	24GB	AdHoc	256	96MB

Tabela 5.8: Conjuntos de Configuração de Parâmetros Avaliados na Execução do *Workflow* SkyMap no *cluster* Petrus

A Figura 5.14 apresenta a média dos tempos de execução das configurações $C3$ e $C4$. Estes resultados experimentais mostram que o tempo de execução com a configuração $C3$ é aproximadamente 8 vezes menor do que $C4$, ajustando somente a estratégia de particionamento de dados e o número de partições. Este melhor desempenho obtido pela configuração $C3$ é devido à escolha da estratégia de particionamento FRANCE nesta configuração. Pode-se observar, mais uma vez, que conhecer o critério de particionamento dos dados é muito importante para o desempenho do *workflow* SkyMap. É importante relembrar que a Seção 5.3.1 apresenta a comparação entre as estratégias de particionamento FRANCE e AdHoc, e como a estratégia FRANCE melhora o tempo de execução do *workflow*.

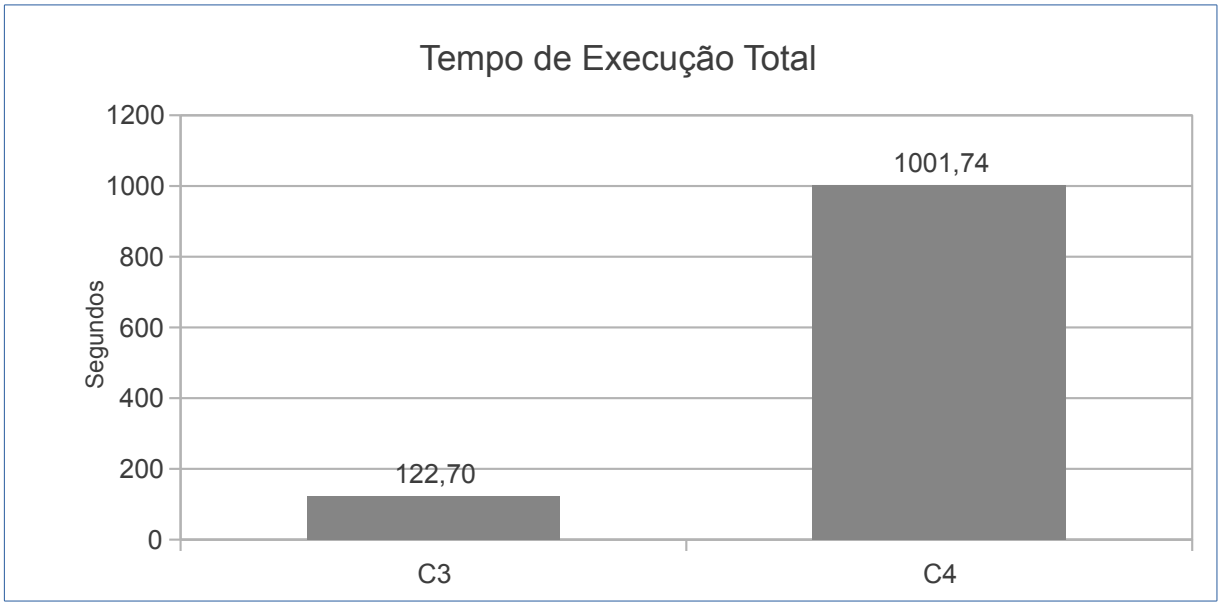

Figura 5.14: Tempo de Execução total obtido com a execução do *workflow* SkyMap no Petrus com as configurações $C3$ e $C4$

5.3.5 Variando o número de *Cores por Executor*

Esta seção apresenta os tempos de execução do *workflow* SkyMap no Petrus sob diferentes valores do parâmetro EC (*Cores por Executor*), o qual representa a quantidade de *cores* disponíveis para cada processo *executor*. Os demais parâmetros estão fixados de acordo com a configuração $C5$ especificada na Tabela 5.9. Para cada variação do parâmetro EC foram realizadas cinco execuções e a Figura 5.15 apresenta a média dos tempos de execução.

Configuração	EN	EM	DS	DE	DN	DP
$C5$	18	16MB	24GB	FRANCE	256	96MB

Tabela 5.9: Valores dos Parâmetros Fixados na Execução do *Workflow* SkyMap no Petrus variando a quantidade de *cores por executor*

Estes resultados, apresentados na Figura 5.15, mostram que o tempo de execução melhora cerca de 35% conforme o número de *cores/executor* aumenta. Observa-se que, independentemente da quantidade de cores/executor avaliada, no *cluster* Petrus há disponibilidade de núcleos físicos, 25 núcleos por máquina. Quanto mais *cores/executor* maior é a quantidade de tarefas que podem ser executadas em paralelo (grau de paralelismo), contribuindo assim para este ganho de desempenho. Porém, observa-se na Figura 5.15 que com 8 *cores/executor* as cinco execuções do *workflow* SkyMap falham por falta de espaço na memória do *executor*. Conclui-se que aumentar o grau de paralelismo melhora

o desempenho da aplicação, mas este aumento não deve ser realizado sem considerarmos o espaço de memória disponível para cada *executor* e também o tamanho de cada partição a ser processada pelo *executor*.

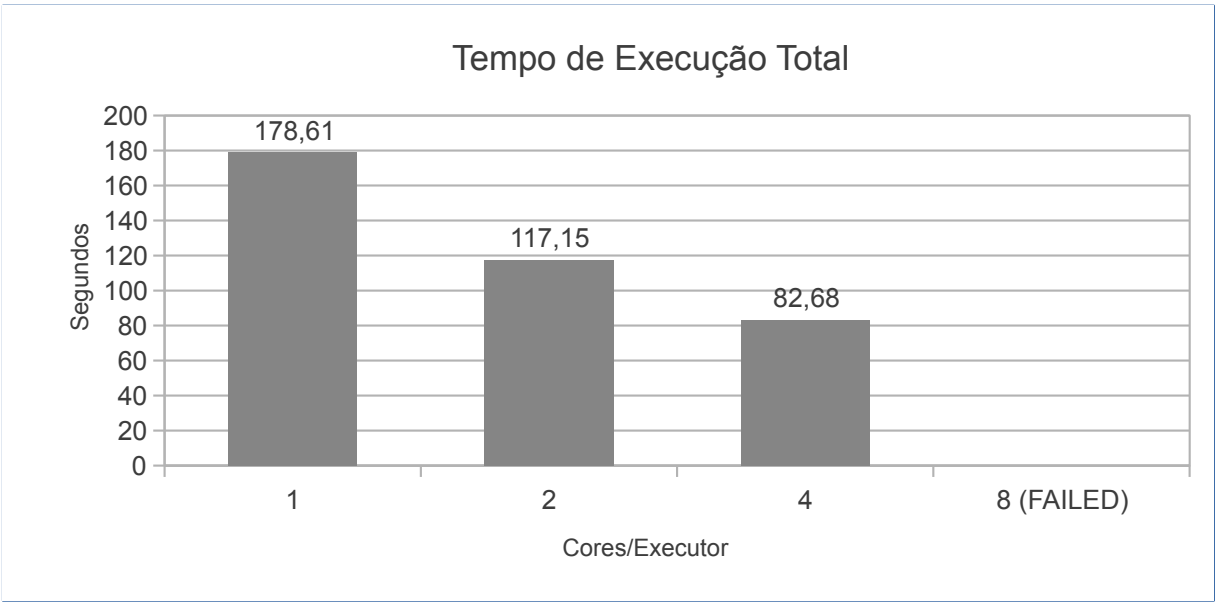

Figura 5.15: Tempo de Execução total obtido com a execução do *workflow* SkyMap no *cluster* Petrus variando a quantidade de *Cores por Executor* conforme a configuração $C5$

5.3.6 Variando a quantidade de *Memória por Executor*

Esta seção apresenta os tempos de execução do *workflow* SkyMap no Petrus sob diferentes valores do parâmetro EM (*Memória por Executor*), o qual representa a quantidade de memória disponível para cada processo *executor*. Este parâmetro influencia diretamente a quantidade máxima de EN (Número de *Executors*), quanto maior for o volume de Memória por *Executor*, menor será o número de processos *executors* e consequentemente o grau de paralelismo, conforme é apresentado na Figura 5.16. Os demais parâmetros estão fixados de acordo com a configuração $C6$ da Tabela 5.10. Para cada variação dos parâmetros EM e EN foram realizadas cinco execuções.

Configuração	EC	DS	DE	DN	DP
$C6$	1	24GB	FRANCE	256	96

Tabela 5.10: Valores dos Parâmetros Fixados na Execução do *Workflow* SkyMap no Petrus variando a quantidade de *Memória por executor*

A Figura 5.16 apresenta a média dos tempos de execução que mostram que este varia bastante sob os diferentes valores dos parâmetros memória por *executor* EM e

número de *executors* EN, sendo a taxa máxima de variação do tempo de execução de 43%. Pode-se observar que, conforme a quantidade de memória (EM) diminui (para os casos $EM = 16, 8e4$), o tempo de execução diminui, devido ao aumento do número de processos *executors* (EN) executando em paralelo. Porém, a Figura 5.16 mostra que este mesmo comportamento não se repete quando $EM = 2$ e $EN = 137$, pois ocorre uma maior concorrência entre os processos pelos recursos do *cluster* Petrus. Os 137 processos *executors* devem ser distribuídos e alocados pelas máquinas do *cluster*. Como o *cluster* Petrus é composto por 6 máquinas, e o Spark realiza uma distribuição balanceada, cada máquina irá receber aproximadamente 23 processos *executors*. Por exemplo, quando $EM = 4$ e $EN = 66$ cada máquina recebe somente 11 processos paralelos.

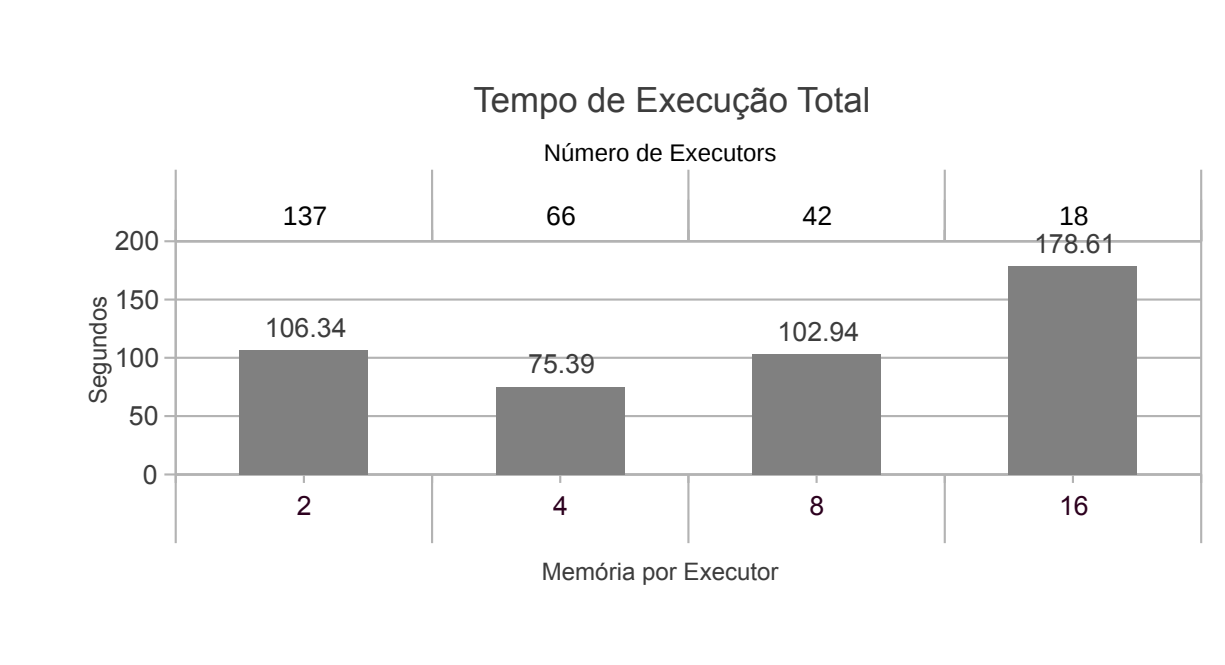

Figura 5.16: Tempo de Execução total obtido com a execução do *workflow* SkyMap no *cluster* Petrus variando a quantidade de *Memória/Executor* conforme a configuração $C6$

A partir dos resultados experimentais descritos nesta seção, é possível concluir que existem muitas oportunidades para melhorar o desempenho de *workflows* no Spark através de um ajuste preciso dos parâmetros de execução. Porém, o ajuste manual por tentativa e erro é um método que despende muito tempo para encontrar o ideal, uma vez que o espaço de variação e combinação dos parâmetros é grande. Além disso, é muito difícil explorar todo este espaço de busca manualmente. Finalmente, o desempenho do *workflow* pode ser afetado por vários outros fatores como o particionamento de dados, características do *workflow* e a configuração de hardware do *cluster*. Portanto, isto aponta a importância de uma abordagem que possua um método de auto-ajuste dos parâmetros de configuração

de execução do *workflow* para o cientista.

5.4 Resumo do Capítulo

Neste Capítulo foi realizada a modelagem do estudo de caso para avaliação da abordagem proposta envolvendo *workflows*, ambientes de execução e a seleção dos parâmetros. Foram descritos dois *workflows* da área da astronomia, SkyMap e Constellation Queries, assim como o funcionamento da execução paralela de cada *workflow* no Spark. Além disso foram descritos dois ambientes de *cluster* reais, Petrus e Minicluster, utilizados para executar os *workflows*. Também foi apresentada a avaliação dos parâmetros de configuração da execução selecionados e a importância da configuração correta dos mesmos para o desempenho do *workflow*.

Capítulo 6

Avaliação Experimental

Para avaliação da abordagem proposta, este capítulo descreve a análise experimental considerando um subconjunto de configurações de parâmetros relacionados aos dados de entrada e Spark na execução dos *workflows* SkyMap e Constellation Queries em dois ambientes de *clusters* dedicados e homogêneos. Nesta tese é denominado Cenário a combinação entre *workflow* e o ambiente de execução. Diante disso, o principal objetivo deste capítulo é analisar a aplicação da abordagem proposta através da configuração dos parâmetros em cada Cenário. Para tal, na Seção 6.1 são apresentadas as etapas de treinamento, discretização, classificação e também as regras de ajuste e configuração dos parâmetros relacionadas ao tempo de execução dos *workflows* e por fim uma análise e validação do modelo preditivo obtido em cada Cenário é descrito na Seção 6.2.

6.1 Análise das Árvores de Classificação em cada Cenário

Para extrair informação dos resultados obtidos nos experimentos a seguir, técnicas de aprendizado de máquina foram utilizadas para identificar os parâmetros que mais influenciam no tempo de execução do *workflow* em um determinado ambiente. Nas análises realizadas nesta seção, foram usados métodos de discretização e de classificação com árvores de classificação [Han et al., 2011], os quais definem as regras relacionadas à execução do *workflow*. A ferramenta de mineração de dados e aprendizado de máquina Orange [Demsar et al., 2013] foi usada para a discretização e geração dos modelos de classificação. O modelo preditivo obtido através do método árvore de classificação, utilizado para classificação dos parâmetros, foi avaliado de acordo com as seguintes métricas definidas por [Cherkassky and Mulier, 1998] [Duda et al., 2000]: acurácia (CA), *F-measure* (F1), Pre-

cisão (Precision) e cobertura (Recall). Estas métricas de avaliação do modelo preditivo foram obtidas através da ferramenta Orange utilizando o método *10-fold cross-validation* (10-cv) para todas as árvores de classificação apresentadas nesta seção. O método *10-fold cross-validation* é realizado dividindo-se os dados em 10 subconjuntos mutuamente exclusivos de casos, aproximadamente do mesmo tamanho. O modelo é treinado e testado 10 vezes, sendo cada vez testado com um subconjunto (10% dos exemplos) e treinado com o restante do conjunto de dados (90% dos exemplos). A estimativa da precisão é a média das precisões estimadas usando-se os 10 subconjuntos. O tempo de treinamento e predição no Orange em uma máquina com 8 núcleos e 16GB de memória RAM é de no máximo dois segundos, em todos os experimentos descritos nesta seção, sendo aproximadamente 200KB o tamanho dos dados de treinamento de um cenário, e fornecidos como entrada para o Orange. Nesta tese, considera-se que o modelo preditivo alcança um resultado satisfatório quando a métrica F1 é maior ou igual a 70%.

A seguir são apresentadas as etapas de treinamento, discretização, classificação e validação do modelo em cada cenário. Cada cenário é composto pelo *workflow* e pelo ambiente computacional onde foram realizados os experimentos. Na Subseção 6.1.1 é descrito como foi aplicada a abordagem proposta na execução da *workflow* SkyMap no *cluster* Petrus. Nas Subseções 6.1.2 e 6.1.3 a abordagem é avaliada com a execução do *workflow* SkyMap sobre o *cluster* Petrus com a configuração da execução limitada, respectivamente, em 50% e 12% dos recursos do *cluster* (CPU e RAM). Na Subseção 6.1.4 é descrita a aplicação da abordagem na execução da aplicação SkyMap no Minicluster. Por fim, na Subseção 6.1.5 é descrita a aplicação da abordagem na execução do *workflow* Constellation Queries no *cluster* Petrus.

6.1.1 Cenário 1 - Análise da Árvore de Classificação gerada pela Execução do *Workflow* SkyMap no *Cluster* Petrus

Nesta subseção é avaliada a execução do *workflow* SkyMap no *cluster* Petrus com o objetivo de realizar uma classificação dos parâmetros que levam a um desempenho eficiente de execução através da abordagem proposta.

Treinamento: As execuções foram realizadas de acordo com a variação de parâmetros apresentada na Tabela 6.1. Diversas combinações dos parâmetros na Tabela 6.1 foram avaliadas e um exemplo desta combinação do conjunto parâmetros é $C1 = [135, 1, 2, 24, FRANCE, 48, 512]$, seguindo a ordem dos parâmetros apresentados na Tabela 6.1. Para cada combinação do conjunto de parâmetros C_i avaliado foram realizadas

cinco execuções. Nos experimentos descritos nesta subseção foram avaliadas 77 combinações dos parâmetros, gerando assim um total de 385 execuções.

Spark	Qtd. Executors	EN	6, 12, 18, 42, 66, 135
	Cores/Exec	EC	1, 2, 4, 8, 16, 32, 64
	Mem/Exec(GB)	EM	2, 4, 8, 16, 32, 64
Entrada	Tam. Dataset	DS	24GB
	Estratégia de Particionamento	DE	FRANCE, ADHOC
	Tam. da Partição(MB)	DP	48, 96, 192, 256, 384
	Núm. de Partições	DN	64, 96, 128, 256, 512

Tabela 6.1: Valores dos Parâmetros Avaliados na Execução do Cenário 1

Discretização: Os tempos de execução obtidos com os 385 experimentos foram submetidos ao método de discretização não supervisionado *EqualFrequency* implementado pela ferramenta Orange. Com a discretização, conforme observado no gráfico na Figura 6.1, foram obtidos três *Grupos* de acordo com o tempo de execução: *Baixo*, *Medio* e *Alto*. Foram classificadas como *Baixo* as execuções cujos tempos são menores que 117,29 segundos, *Medio* aquelas cujos tempos estão entre 117,29 e 480,58 segundos e como *Alto*, as execuções que obtiveram tempo acima de 480,58 segundos.

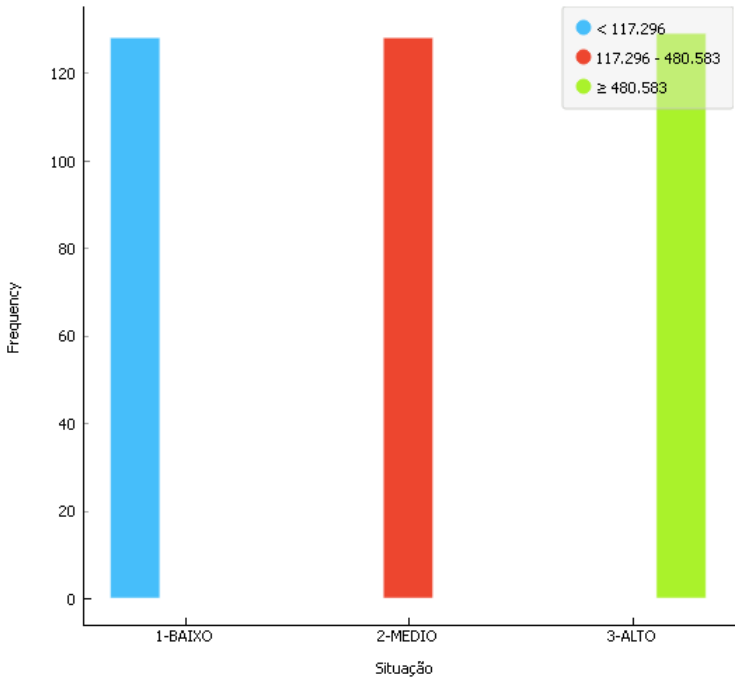

Figura 6.1: Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 1: *Baixo*, *Medio* e *Alto*

Classificação: A Figura 6.2 apresenta a árvore de classificação gerada com dois níveis. O principal conhecimento obtido com esse modelo foi encontrar os principais

parâmetros que definem o grupo de execução e os intervalos de valores que cada parâmetro assume, para cada grupo de execução. Esse tipo de conhecimento quantitativo sobre os parâmetros é muito difícil de ser obtido de forma visual e sem a utilização de métodos de aprendizado de máquina.

Conforme a Figura 6.2 verifica-se que o *Grupo* em que uma execução será classificada é definido pelos parâmetros: estratégia de particionamento (DE) e número de partições (DN). Desta forma, é visto que a execução do *workflow* SkyMap no *cluster* Petrus é impactada principalmente pela forma como os dados de entrada estão organizados (o tipo de particionamento). Uma execução classificada com o tempo *Baixo* pode ser obtida através do $DE = FRANCE$ e $DN > 128$. Uma execução classificada com o tempo *Medio* pode ser obtida através do $DE = FRANCE$ e $DN \leq 128$ ou $DE = ADHOC$ e $DN \leq 64$. Uma execução classificada com o tempo *Alto* pode ser obtida através do $DE = ADHOC$ e $DN > 64$.

A árvore de classificação produzida a partir dos dados de treinamento realça a relevância do método de particionamento na execução do *workflow*. Observa-se igualmente que quanto maior o número de partições melhor o desempenho, uma vez que permite maior paralelismo das tarefas, disponível no ambiente Petrus, que dispõe de 150 núcleos e 480GB de RAM. Esse conhecimento não é novo, pois ambos os parâmetros, DE e DN, foram analisados de forma específica nas Seções 5.3.1 e 5.3.2. Porém, com a análise oriunda da árvore de classificação, o conhecimento dos valores e relações dos parâmetros do Spark ficam mais evidentes.

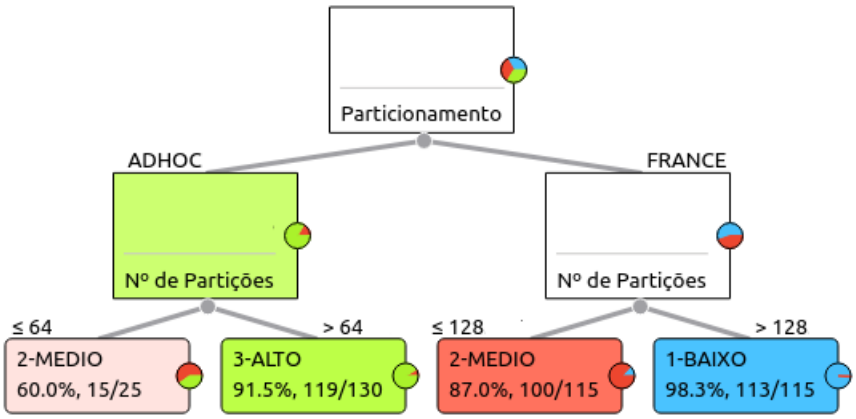

Figura 6.2: Árvore de Classificação do Grupo de execução com 2 níveis para o Cenário 1

Além disso, este experimento mostra que os parâmetros do Spark (EN, EC, EM) não

influenciam o grupo de execução e podem ser configurados com qualquer valor dentre aqueles definidos pela Tabela 6.1. Podem-se obter as seguintes regras de acordo com a árvore de classificação apresentada pela Figura 6.2:

(1) $(DE = FRANCE) \wedge (DN > 128) \rightarrow Baixo$

(2) $(DE = FRANCE) \wedge (DN \leq 128) \rightarrow Medio$

(3) $(DE = ADHOC) \wedge (DN > 64) \rightarrow Alto$

(4) $(DE = ADHOC) \wedge (DN \leq 64) \rightarrow Medio$

Validação: Os resultados de avaliação do modelo preditivo utilizando o método 10-cv estão apresentados na Tabela 6.2. Nesta avaliação, a predição do *Grupo* foi correta em 88,6% dos casos (F1), considerada como satisfatória, além de ser um modelo preditivo compacto, com regras de boa cobertura e produzir um conhecimento consistente com o problema.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,886	0,886	0,890	0,886

Tabela 6.2: Resultado das Métricas utilizadas para avaliação do método árvore de classificação com dois níveis no Cenário 1

Para avaliar o impacto da geração de um número maior de níveis da árvore de classificação na abordagem proposta, o mesmo treinamento e discretização foi aplicado novamente juntamente com o método de classificação com maior número de níveis. A Figura 6.3 apresenta a árvore de classificação gerada com três níveis. Conforme a Figura 6.3 verifica-se que o *Grupo* em que uma execução será classificada como *Baixo* é definido pelos parâmetros: particionamento (DE), nº de partições (DN) e Cores/Executor (EC). Constata-se que a estratégia de particionamento e o tamanho da partição continuam sendo parâmetros determinantes para o desempenho do *workflow*. Diante do aumento do número de folhas da árvore, uma maior quantidade de regras é produzida para cada grupo de execução. Como o objetivo nesta tese é a configuração que leva a uma execução eficiente são destacadas apenas as regras que classificam o grupo de execução *Baixo*. Conforme Figura 6.3 podem-se obter as seguintes regras para execuções classificadas como *Baixo*:

(1) $(DE = FRANCE) \wedge (DN > 128) \rightarrow Baixo$

(2) $(DE = FRANCE) \wedge (DN \leq 128) \wedge (EC > 4) \rightarrow Baixo$

Pode-se observar que na árvore com dois níveis, apresentada na Figura 6.2, a configuração de parâmetros $(DE = FRANCE) \wedge (DN \leq 128)$ leva a um tempo de execução classificado como $Medio$ em 87% dos casos. Isso mostra que em 13% dos casos esta mesma configuração pode levar a um tempo de execução $Baixo$ ou $Alto$. Nesse caso, com 2 níveis o algoritmo generalizou para a maior quantidade de exemplos que são $Medio$. Diante disso, observa-se que na árvore com três níveis é encontrado um parâmetro que aponta a possibilidade de alcançar o $Baixo$ nos 13% dos casos, ou seja, adicionando $(EC > 4)$ na configuração de parâmetros $(DE = FRANCE) \wedge (DN \leq 128)$, pode levar a um tempo de execução classificado como $Baixo$. Além disso, a árvore mostra que se $(EC \leq 4)$ for adicionado a configuração de parâmetros $(DE = FRANCE) \wedge (DN \leq 128)$, continuará levando um tempo de execução classificado como $Medio$ em 94,7% dos casos, aumentando a precisão da predição e a taxa de acerto. É importante observar que estas regras mais específicas geradas na árvore com 3 níveis cobrem menos casos comparado as regras mais "genéricas" na árvore com 2 níveis.

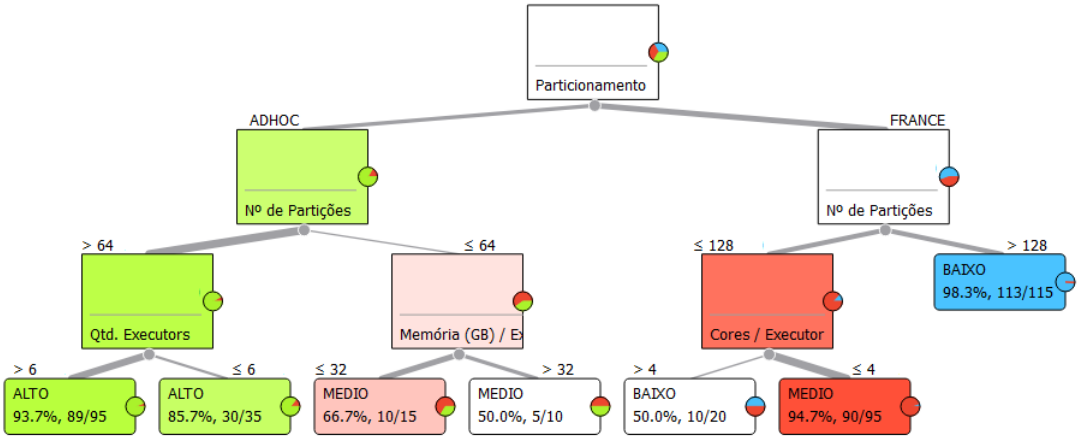

Figura 6.3: Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 1

Apesar do objetivo desta tese ser escolher a configuração de parâmetros que levem ao menor tempo de execução ($Baixo$), sabe-se que nem sempre isso é possível. Por exemplo, caso o cientista não deseje realizar o particionamento dos dados de acordo com algum critério e use a estratégia de particionamento AdHoc, mesmo assim o modelo apresenta as soluções eficientes neste contexto. Pode-se observar na Figura 6.3 que poderá alcançar um tempo de execução classificado como $Medio$, através da configuração dos parâmetros $(DE = ADHOC) \wedge (DN \leq 64)$. No entanto, considerando a árvore de classificação, ao utilizar este particionamento, o tempo de execução não será $Baixo$.

Pode-se verificar, mais uma vez, a importância da utilização da árvore de classificação,

que produz um conhecimento quantitativo sobre os parâmetros. Através desta foi possível encontrar os parâmetros que definem o grupo de execução *Medio* e os intervalos de valores que os parâmetros assumem. Isto possibilita, no exemplo descrito acima, alcançar o melhor desempenho possível usando a estratégia de particionamento AdHoc. Adicionalmente, sem a utilização de métodos de aprendizado de máquina seria muito difícil de forma visual obter os parâmetros e valores que levam a uma execução eficiente.

Os resultados de avaliação do modelo preditivo utilizando o método *10-fold cross-validation* (10-cv) estão apresentados na Tabela 6.3. Nesta avaliação, a predição do *Grupo* foi correta em 88,5% dos casos (F1). Portanto, quando aumentamos em um nível a árvore pode-se constatar que: mais um parâmetro influencia a predição do *Baixo*, *EC* (Cores/Executor), conforme apresentado na Figura 6.2.

É possível observar que ainda assim os 7 parâmetros, definidos na Tabela 6.1, não foram utilizados na construção da árvore e por isso o modelo é chamado de compacto. Quando o modelo gerado é muito grande, há muitas regras que cobrem poucos exemplos, e é como se cada uma delas descrevesse um exemplo do conjunto de dados. Nesse tipo de situação, o modelo não é útil, pois seria inviável testar centenas ou milhares de regras para se classificar um novo exemplo. O modelo gerado, além de ser compacto (poucas regras), tem alta cobertura. Por exemplo, somente a regra (1) classifica 115 dos 127 exemplos da classe $BAIXO$, correspondendo à aproximadamente 90% dos exemplos.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,886	0,885	0,888	0,886

Tabela 6.3: Resultado das Métricas utilizadas para avaliação do método árvore de classificação com três níveis no Cenário 1

Já a Figura 6.4 apresenta a árvore de classificação gerada sem um limite de níveis definido na ferramenta Orange. Neste caso, foi produzida uma árvore com cinco níveis. Verifica-se que o *Grupo* em que uma execução será classificada como *Baixo* é definido pelos parâmetros: particionamento (DE), nº de partições (DN), Cores/Executor (EC) e Qtd. Executors (EN). Conforme a Figura 6.4 pode-se obter as seguintes regras para execuções classificadas como *Baixo*:

(1) $(DE = FRANCE) \wedge (DN > 128) \rightarrow Baixo$

(2) $(DE = FRANCE) \wedge (DN \leq 128) \wedge (EC > 8) \rightarrow Baixo$

(3) $(DE = FRANCE) \wedge (DN \leq 128) \wedge (EC \leq 4) \wedge (EN > 42) \rightarrow Baixo$

(4) $(DE = FRANCE) \wedge (DN \leq 128) \wedge (EC > 4) \wedge (EC <= 8) \wedge (EN > 6) \rightarrow Baixo$

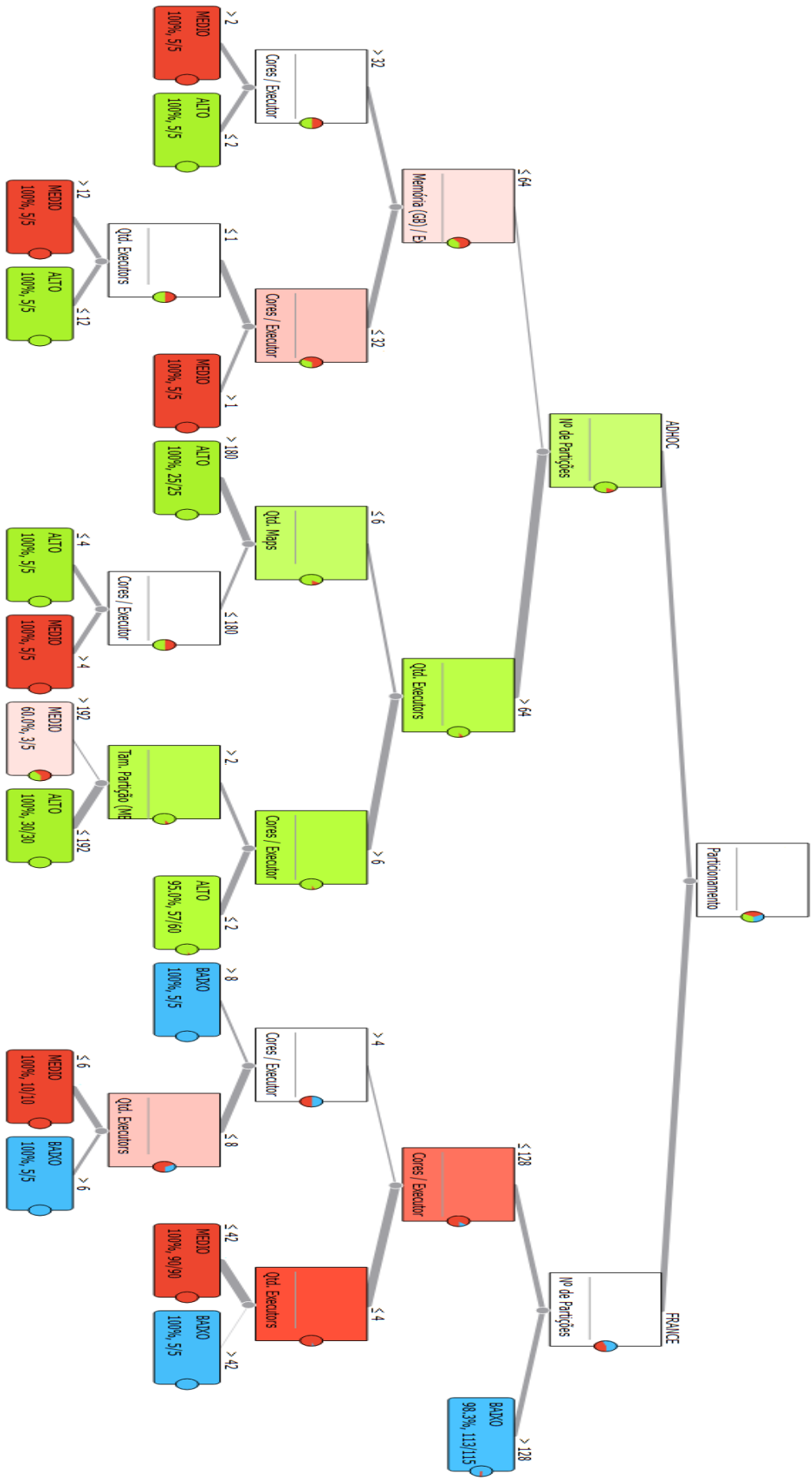

Figura 6.4: Árvore de Classificação do Grupo de execução com 5 níveis para o Cenário 1

Os resultados de avaliação do modelo preditivo utilizando o método 10-cv são apresentados pela Tabela 6.4. Nesta avaliação, a predição do *Grupo* foi correta em 95,3% dos casos (F1). Portanto, nesta situação pode-se constatar que: mais um parâmetro influencia a predição do *Baixo* (Qtd. Executors) e há um aumento na precisão da predição frente as árvores de classificação apresentadas anteriormente nas Figuras 6.2 e 6.3.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,953	0,953	0,954	0,953

Tabela 6.4: Resultado das Métricas utilizadas para avaliação do método árvore de classificação com cinco níveis no Cenário 1

Pode-se concluir com os experimentos apresentados nesta seção que do conjunto de parâmetros avaliados, um desempenho eficiente pode ser alcançado com a configuração de pelo menos 2 parâmetros. Além disso, quando a altura da árvore não é definida o modelo gerado é muito grande, e consequentemente, há um maior número de regras e também regras com muitos parâmetros que cobrem poucos exemplos. Portanto, apesar de existir uma maior precisão do modelo preditivo obtido com uma árvore de classificação com maior altura, para simplificação da visualização, compreensão e quantidade de parâmetros nas regras (modelo compacto), nos experimentos apresentados nesta tese as árvores de classificação estão fixadas em 3 níveis. Adicionalmente, observa-se que todos os modelos com três níveis avaliados nesta tese possuem a métrica F1 maior do que 70%, caracterizando uma boa precisão e taxa de acerto.

Árvore de Classificação de Falhas: Anteriormente foram classificados os grupos de execução de acordo com a configuração de parâmetros. Porém, algumas combinações de parâmetros levaram a execução a falhar e com isso não fazem parte dos dados de treinamento para classificação dos grupos de execução. Portanto, consideramos importante realizar uma classificação das falhas para analisar e compreender quais parâmetros e valores levam a uma falha de execução. Portanto, a seguir é avaliada a execução do Cenário 1 (*workflow* SkyMap no *cluster* Petrus) com o objetivo de realizar uma classificação dos parâmetros que levam a execução a falhar ou finalizar com sucesso.

Treinamento: As execuções foram realizadas de acordo com a variação de parâmetros apresentada na Tabela 6.1. Diversas combinações dos parâmetros na Tabela 6.1 foram avaliadas e para cada combinação do conjunto de parâmetros avaliado foram realizadas cinco execuções. Nos experimentos descritos nesta subseção foram avaliadas 125 combinações dos parâmetros gerando assim um total de 600 execuções.

Discretização: As execuções que finalizaram com sucesso foram definidas como NOT_FAILED e as que não finalizaram como $FAILED$.

Classificação: A Figura 6.5 apresenta uma árvore de classificação de falhas com três níveis. Pode-se observar na árvore de classificação os parâmetros que levam uma execução a falha, que são: Cores por executor (EC), Memória por executor (EM) e o tamanho da partição (DP).

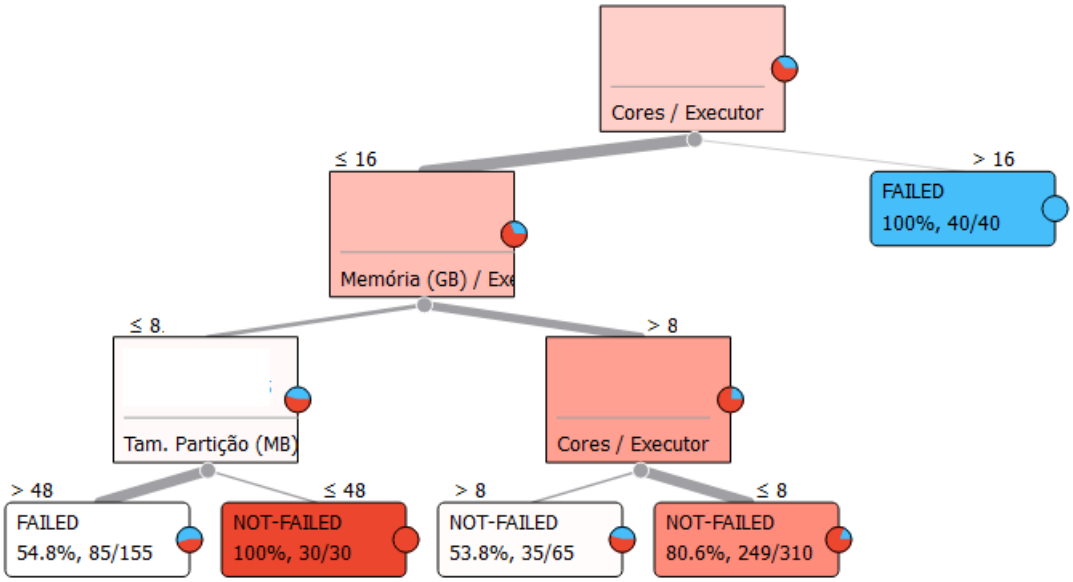

Figura 6.5: Árvore de Classificação de Falhas com 3 níveis para o Cenário 1

De acordo com o apresentado na Figura 6.5 pode-se obter as seguintes regras:

(1) $(EC > 16) \rightarrow FAILED$

(2) $(EC \leq 16) \wedge (EM \leq 8) \wedge (DP > 48) \rightarrow FAILED$

(3) $(EC \leq 16) \wedge (EM \leq 8) \wedge (DP \leq 48) \rightarrow NOT_FAILED$

(4) $(EC \leq 16) \wedge (EM > 8) \rightarrow NOT_FAILED$

As falhas ocorrem devido a falta de espaço em memória para o processo *executor*. Por exemplo, quando o parâmetro EC é maior do que 16 significa que cada executor possui mais de 16 *threads* executando em mais de 16 *cores* diferentes, porém há um compartilhamento da memória do *executor*. Com isso, quanto maior for o parâmetro EC menor é o espaço de cada *thread* dentro da memória do *executor*. Este mesmo problema

ocorre quando a quantidade de memória dada a cada *executor* não é suficiente para processar partições maiores do que 48MB, mesmo que o parâmetro EC seja menor ou igual a 16, como observado na segunda regra descrita acima.

Validação: Os resultados de avaliação do modelo preditivo utilizando o método 10-cv estão na Tabela 6.5. Nesta avaliação, a predição foi correta em 71,5% dos casos, considerada como satisfatória, além de ser um modelo preditivo compacto e com regras de boa cobertura.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,727	0,715	0,719	0,727

Tabela 6.5: Resultado das Métricas utilizadas para avaliação do método árvore de classificação de falhas com três níveis no Cenário 1

6.1.2 Cenário 2 - Análise da Árvore de Classificação gerada pela Execução do *Workflow* SkyMap utilizando 50% de CPU e RAM do *cluster* Petrus

Nestes experimentos é apresentada a aplicação da abordagem de otimização neste Cenário, que está relacionado à execução do *workflow* SkyMap sobre o *cluster* Petrus com a configuração da execução limitada em 50% dos recursos (CPU e RAM). A quantidade de memória e *cores* disponibilizada no *cluster* Petrus para o Spark neste Cenário foram limitados no YARN em 40GB e 12 *cores* respectivamente, conforme destacado pela Tabela 6.6. O YARN é o sistema responsável pela definição da quantidade máxima de recursos que o Spark pode utilizar na execução de uma aplicação.

Cluster	Tipo do Nó	Qtd. Nós	Cores/Nó	Mem/Nó(GB)
Petrus	Mestre	1	8	80
	Trab.	6	12	40

Tabela 6.6: Configuração do Spark no Ambientes de Execução no Cenário 2

Treinamento: Diversas combinações dos parâmetros na Tabela 6.7 foram avaliadas, como por exemplo, a combinação $C1 = [71, 1, 2, 24, FRANCE, 96, 256]$, seguindo a ordem dos parâmetros apresentados na Tabela 6.7. Para cada combinação do conjunto de parâmetros avaliado foram realizadas cinco execuções. Nos experimentos descritos nesta subseção foram avaliadas 68 combinações dos parâmetros gerando assim um total de 340 execuções.

Spark	Qtd. Executors	EN	7, 12, 24, 36, 71
	Cores/Exec	EC	1, 2, 4, 8, 16
	Mem/Exec(GB)	EM	2, 4, 8, 16, 32
Entrada	Tam. Dataset	DS	24GB
	Estratégia de Particionamento	DE	FRANCE, ADHOC
	Tam. da Partição(MB)	DP	48, 96, 192, 256, 384
	Núm. de Partições	DN	64, 96, 128, 256, 512

Tabela 6.7: Valores dos Parâmetros Avaliados na Execução do Cenário 2

Discretização: Os tempos de execução obtidos foram submetidos ao método *Equal-Frequency* e foram gerados três *Grupos* de classificação conforme a Figura 6.6: *Baixo* as execuções que obtiveram tempo abaixo de 148,75 segundos; como *Medio* as execuções que obtiveram tempo entre 148,75 e 630,71 segundos; e como *Alto* as execuções que obtiveram tempo acima de 630,71 segundos.

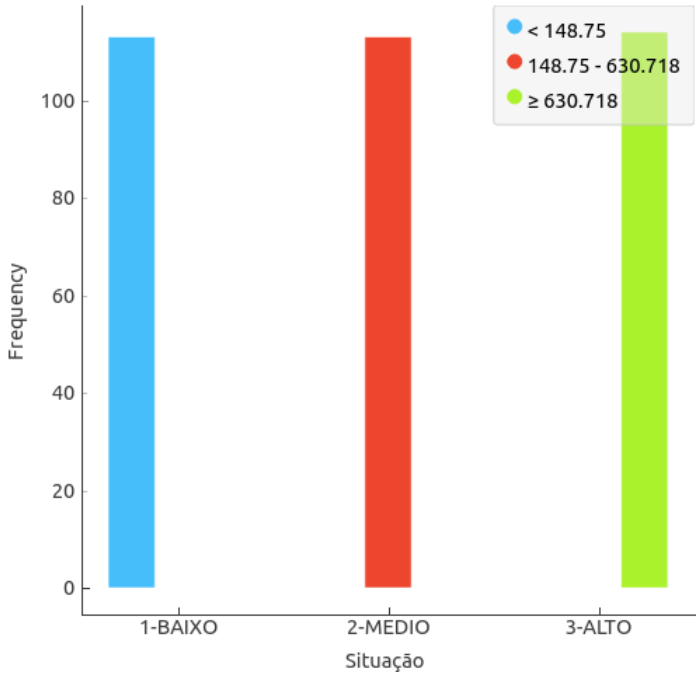

Figura 6.6: Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 2: Baixo, Medio e Alto

Classificação: A Figura 6.7 apresenta a árvore de classificação com três níveis obtida através das execuções do *workflow* Skymap. Nesta análise, pode-se afirmar que o *Grupo Baixo* pode ser definido pelos seguintes parâmetros: particionamento (DE), número de partições (DN). Observa-se que esses parâmetros relacionados aos dados continuam sendo determinantes no desempenho do *workflow* SkyMap mesmo com uma menor quantidade de recursos e menor grau de paralelismo. Conforme a Figura 6.7 pode-se obter a seguinte regra para execução classificadas como *Baixo*:

(1) $(DE = FRANCE) \wedge (DN > 128) \rightarrow Baixo$

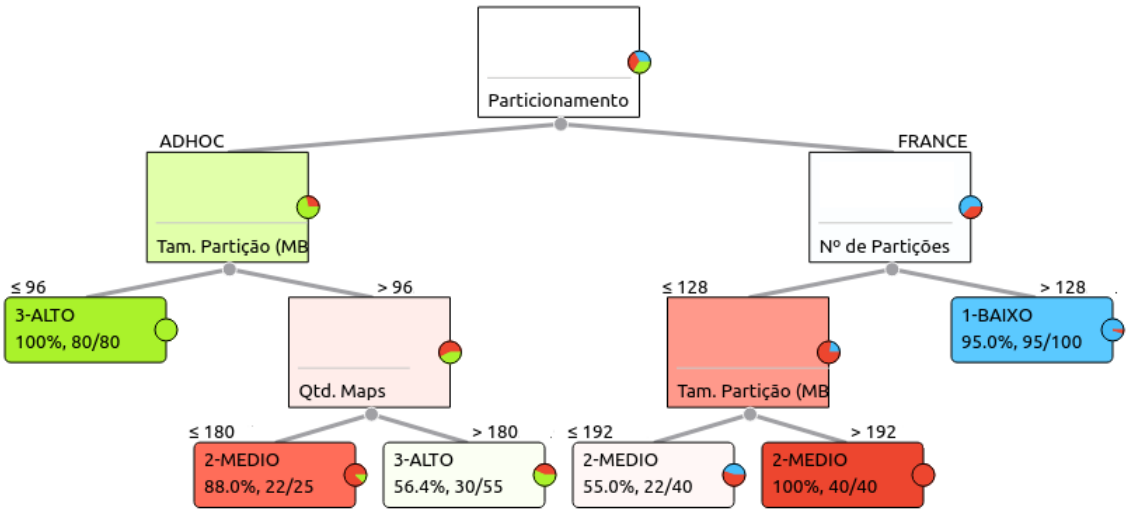

Figura 6.7: Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 2

Pode-se novamente verificar que, caso o cientista tenha disponível somente a estratégia de particionamento AdHoc, mesmo assim o modelo obtido por meio da árvore de classificação apresenta uma solução eficiente neste contexto. Portanto, pode-se observar na Figura 6.7 que um tempo de execução classificado como *Medio* pode ser alcançado através da regra $(DE = ADHOC) \wedge (DP > 96)$.

Validação: Os resultados de avaliação do modelo preditivo utilizando o método 10-cv estão na Tabela 6.8. Nesta avaliação, a predição do grupo foi correta em 86,2% dos casos, considerada como satisfatória, além de ser um modelo preditivo compacto com boa cobertura.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,862	0,862	0,864	0,862

Tabela 6.8: Resultado das Métricas utilizadas para avaliação do método árvore de classificação com três níveis no Cenário 2

6.1.3 Cenário 3 - Análise da Árvore de Classificação gerada pela Execução do *Workflow* SkyMap utilizando 10% de CPU e RAM do *cluster* Petrus

Nestes experimentos são apresentadas as configurações de parâmetros para uma execução eficiente obtidas através da abordagem proposta na execução do *workflow* SkyMap sobre o *cluster* Petrus com a configuração da execução limitada em 10% dos recursos (CPU e RAM). Com isso, a quantidade de memória e *cores* disponibilizada em cada máquina do *cluster* Petrus para o Spark foram limitados em 10GB e 4 *cores* respectivamente, conforme destacado na Tabela 6.9.

Cluster	Tipo do Nó	Qtd. Nós	Cores/Nó	Mem/Nó(GB)
Petrus	Mestre	1	8	80
	Trab.	6	4	10

Tabela 6.9: Configuração do Spark no Ambientes de Execução

Treinamento: Diversas combinações dos parâmetros na Tabela 6.10 foram avaliadas e um exemplo desta combinação do conjunto parâmetros é $C1 = [71, 1, 2, 24, FRANCE, 96, 256]$, seguindo a ordem dos parâmetros apresentados na Tabela 6.7. Para cada combinação do conjunto de parâmetros avaliado foram realizadas cinco execuções. Nos experimentos descritos nesta subseção foram avaliadas 24 combinações dos parâmetros gerando assim um total de 120 execuções.

Spark	Qtd. Executors	EN	4, 5, 11
	Cores/Exec	EC	1, 2, 4
	Mem/Exec(GB)	EM	2, 4, 8
Entrada	Tam. Dataset	DS	24GB
	Estratégia de Particionamento	DE	FRANCE, ADHOC
	Tam. da Partição(MB)	DP	48, 96, 192, 256
	Núm. de Partições	DN	96, 128, 256, 512

Tabela 6.10: Valores dos Parâmetros Avaliados na Execução do Cenário 3

Discretização: Os tempos de execução obtidos foram submetidos ao método *Equal-Frequency* e foram gerados três *Grupos* de classificação de acordo com o tempo de execução: *Baixo*, *Medio* e *Alto*. Conforme a Figura 6.8 foram classificadas como *Baixo* as execuções que obtiveram tempo abaixo de 513,96 segundos, como *Medio* as execuções que obtiveram tempo entre 513,96 e 1184,45 segundos e foram classificadas como *Alto* as execuções que obtiveram tempo acima de 1184,45 segundos.

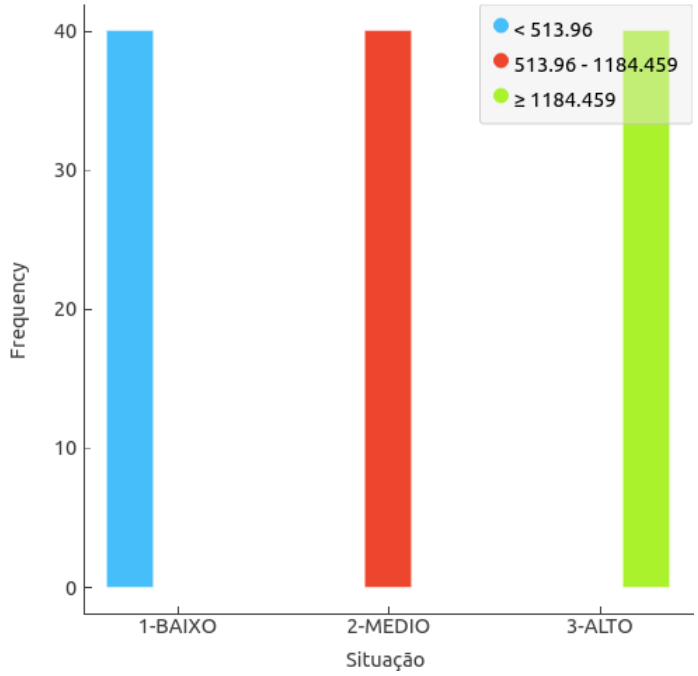

Figura 6.8: Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 3: Baixo, Medio e Alto

Classificação: A Figura 6.9 apresenta a árvore de classificação com três níveis obtida através das execuções do *workflow* Skymap. Nesta análise, pode-se afirmar que o *Grupo Baixo* pode ser definido pelos seguintes parâmetros: particionamento (DE) e tamanho da partição (DP). Observa-se que os parâmetros relacionados aos dados continuam sendo determinantes no desempenho do *workflow* SkyMap mesmo limitado a 10% dos recursos do *cluster*. Conforme a Figura 6.9 pode-se obter as seguintes regras para execuções classificadas como *Baixo*:

(1) $(DE = FRANCE) \wedge (DP \leq 96) \rightarrow Baixo$

Pode-se observar que, caso o cientista opte por usar a estratégia de particionamento AdHoc, o modelo obtido através da árvore de classificação apresenta uma solução eficiente neste contexto. Pode-se observar na Figura 6.9 que poderá ser alcançado um tempo de execução classificado como *Medio*, através da configuração dos parâmetros $(DE = ADHOC) \wedge (EC \leq 2) \wedge (DP > 48)$.

Validação: Os resultados de avaliação do modelo preditivo utilizando o método 10-cv estão na Tabela 6.11. Nesta avaliação, a predição do grupo foi correta em 76,9% dos casos, considerada como satisfatória.

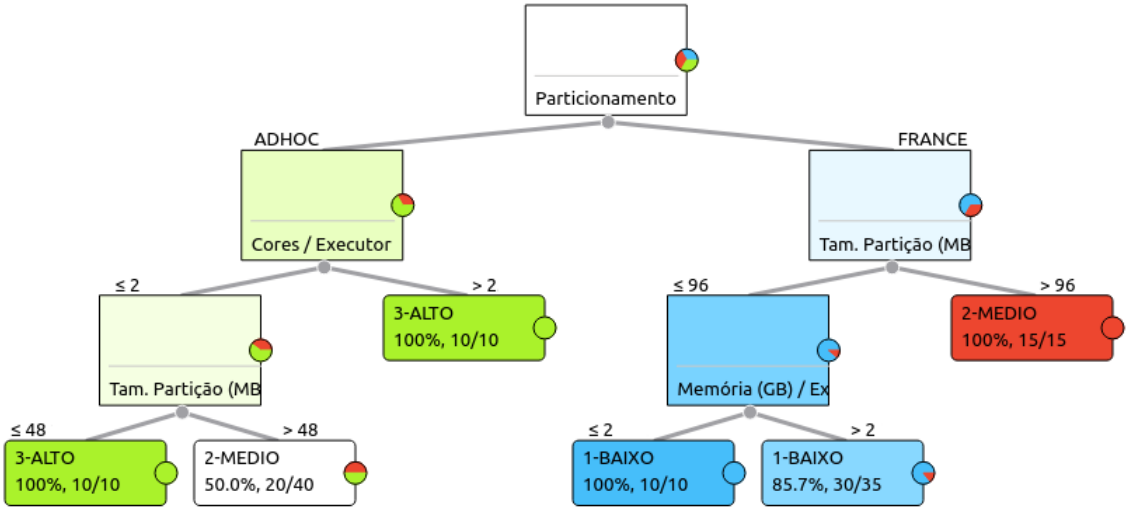

Figura 6.9: Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 3

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,775	0,769	0,767	0,775

Tabela 6.11: Resultado das Métricas utilizadas para avaliação do método árvore de classificação com três níveis no Cenário 3

6.1.4 Cenário 4 - Análise da Árvore de Classificação gerada pela Execução do *Workflow* SkyMap no Minicluster

Nestes experimentos são apresentadas as configurações de parâmetros para uma execução eficiente obtidas através da abordagem proposta na execução do *workflow* SkyMap sobre o Minicluster, cuja configuração está apresentada na Tabela 5.1. O Minicluster é um ambiente com uma capacidade computacional, especialmente CPU e RAM, inferior ao Petrus. Portanto, foi aplicada a abordagem proposta nesta tese com o objetivo de descobrir quais parâmetros e valores influenciam o tempo de execução neste ambiente com um poder computacional mais restrito.

Treinamento: Diversas combinações dos parâmetros na Tabela 6.12 foram avaliadas seguindo a ordem dos parâmetros. Para cada combinação do conjunto de parâmetros foram realizadas cinco execuções. Nos experimentos descritos nesta subseção foram avaliadas 27 combinações dos parâmetros gerando assim um total de 135 execuções.

Discretização: Os tempos de execução obtidos foram submetidos ao método *Equal-Frequency* e foram gerados três *Grupos* de classificação de acordo com o tempo de exe-

Spark	Qtd. Executors	EN	3, 4, 8
	Cores/Exec	EC	1, 2, 4
	Mem/Exec(GB)	EM	2, 4, 8
Entrada	Tam. Dataset	DS	24GB
	Estratégia de Particionamento	DE	FRANCE, ADHOC
	Tam. da Partição(MB)	DP	96, 192, 256, 384
	Núm. de Partições	DN	64, 96, 128, 256

Tabela 6.12: Valores dos Parâmetros Avaliados na Execução do Cenário 4

cução: *Baixo*, *Medio* e *Alto*. Conforme a Figura 6.10 foram classificadas como *Baixo* as execuções que obtiveram tempo abaixo de 1273,31 segundos, como *Medio* as execuções que obtiveram tempo entre 1273,31 e 2756,88 segundos e foram classificadas como *Alto* as execuções que obtiveram tempo acima de 2756,88 segundos.

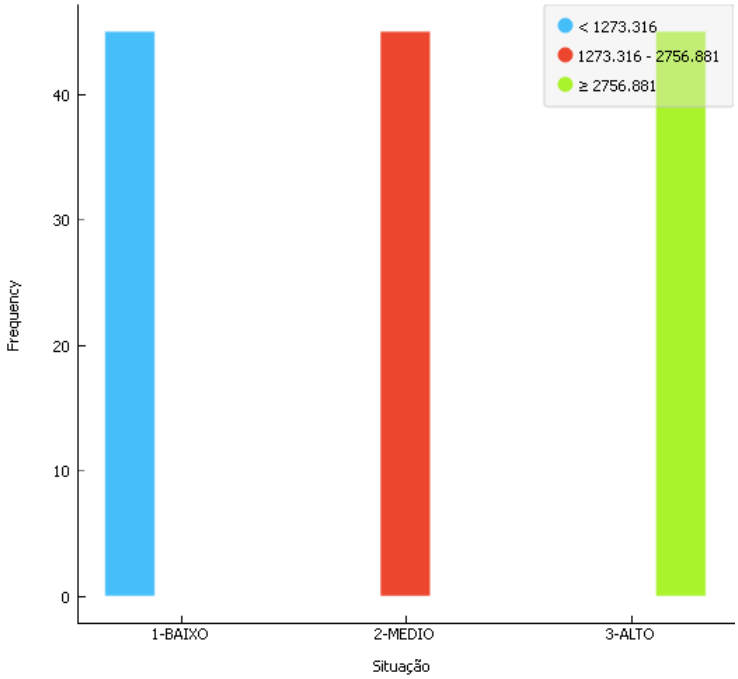

Figura 6.10: Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 4: Baixo, Medio e Alto

Classificação: A Figura 6.11 apresenta a árvore de classificação com três níveis obtida através das execuções do *workflow* Skymap. Nesta análise, pode-se afirmar que o *Grupo Baixo* pode ser definido pelos seguintes parâmetros: Qtd. de *Executors* (EN), particionamento (DE) e o número de partições (DN). Observa-se que os parâmetros relacionados aos dados continuam sendo determinantes no desempenho do *workflow* SkyMap mesmo em um *cluster* diferente, porém o parâmetro principal (nó raiz da árvore) passa a ser a número de partições. Conforme a Figura 6.11 pode-se obter as seguintes

regras para execuções classificadas como *Baixo*:

(1) $(DN > 128) \wedge (EN > 3) \rightarrow Baixo$

(2) $(DN > 128) \wedge (EN \leq 3) \wedge (DE = FRANCE) \rightarrow Baixo$

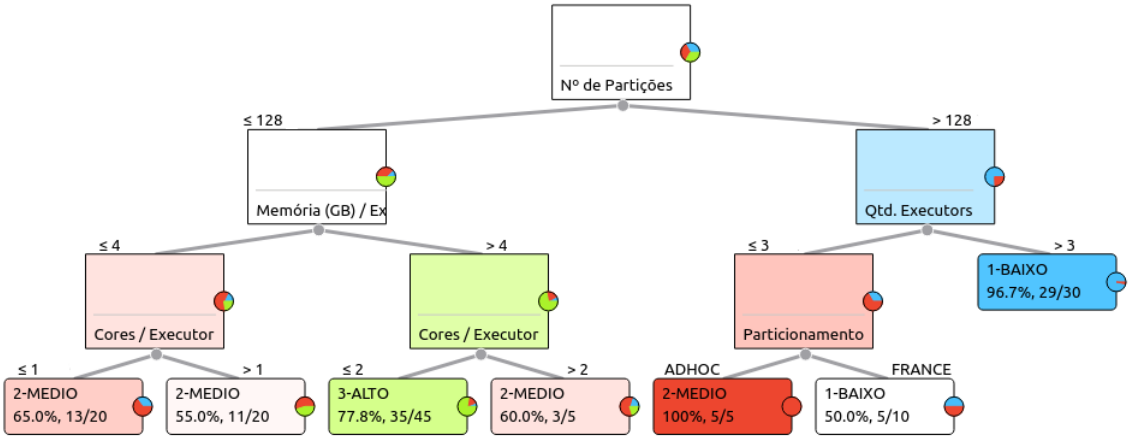

Figura 6.11: Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 4

A árvore de classificação aponta que com um número maior de partições de tamanhos menores a execução se beneficia do paralelismo disponível no ambiente Minicluster. Este mesmo comportamento foi identificado nos Cenários 1,2 e 3 executados no ambiente Petrus. Esse conhecimento não é novo, porém, com a análise oriunda da árvore de classificação, é possível identificar os valores que este parâmetro pode assumir para levar a uma execução classificada como *Baixo*.

Validação: Os resultados de avaliação do modelo preditivo utilizando o método 10-cv estão na Tabela 6.13. Nesta avaliação, a predição do grupo foi correta em 71,4% dos casos, considerada satisfatória.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,711	0,714	0,725	0,711

Tabela 6.13: Resultado das Métricas utilizadas para avaliação do método árvore de classificação com três níveis no Cenário 4

6.1.5 Cenário 5 - Análise da Árvore de Classificação gerada pela Execução do *Workflow* Constelation Queries no Cluster Petrus

Nesta subseção é avaliado a execução do *workflow Constellation Queries* no *cluster* Petrus com o objetivo de realizar uma classificação dos parâmetros que levam a um desempenho eficiente de execução deste *workflow*. Neste Cenário, foram utilizados a quantidade total de núcleos e memória do *cluster* Petrus, conforme definido na Tabela 5.1.

Treinamento: As execuções foram realizadas de acordo com a variação de parâmetros apresentada na Tabela 6.14. Diversas combinações dos parâmetros na Tabela 6.14 foram avaliadas seguindo a ordem dos parâmetros apresentados na Tabela 6.1. Para cada combinação do conjunto de parâmetros C avaliado foram realizadas cinco execuções. Nos experimentos descritos nesta subseção foram avaliadas 44 combinações dos parâmetros gerando assim um total de 220 execuções. Uma vez que o *workflow Constellation Queries* tem como objetivo encontrar padrões geométricos, isso envolve que os objetos que serão analisados sejam vizinhos. Portanto, não foi utilizado a estratégia de particionamento ADHOC, pois não há garantia de que os objetos de uma determinada partição são vizinhos.

Spark	Qtd. Executors	EN	6,12,18,24,30,42
	Cores/Exec	EC	1, 2
	Mem/Exec(GB)	EM	1,2,4,8,16,32,64
Entrada	Tam. Dataset(GB)	DS	1
	Estratégia de Particionamento	DE	FRANCE
	Tam. da Partição(MB)	DP	58,115,230,460
	Núm. de Partições	DN	2,4,8,16

Tabela 6.14: Valores dos Parâmetros Avaliados na Execução do Cenário 5

Discretização: Os tempos de execução obtidos com os vários experimentos foram submetidos ao método de discretização *EqualFrequency*. Com isso, foram obtidos três *Grupos* de classificação de acordo com o tempo de execução: *Baixo*, *Medio* e *Alto*. Conforme a Figura 6.12 foram classificadas como *Baixo* as execuções que obtiveram tempo abaixo de 646,35 segundos. Foram classificadas como *Medio* as execuções que obtiveram tempo entre 646,35 e 731,63 segundos e foram classificadas como *Alto* as execuções que obtiveram tempo acima de 731,63 segundos.

Classificação: De acordo com a Figura 6.13 pode-se afirmar que o *Grupo Baixo* é definido pelo número de partições (DN). Observa-se, mais uma vez, que um parâmetro

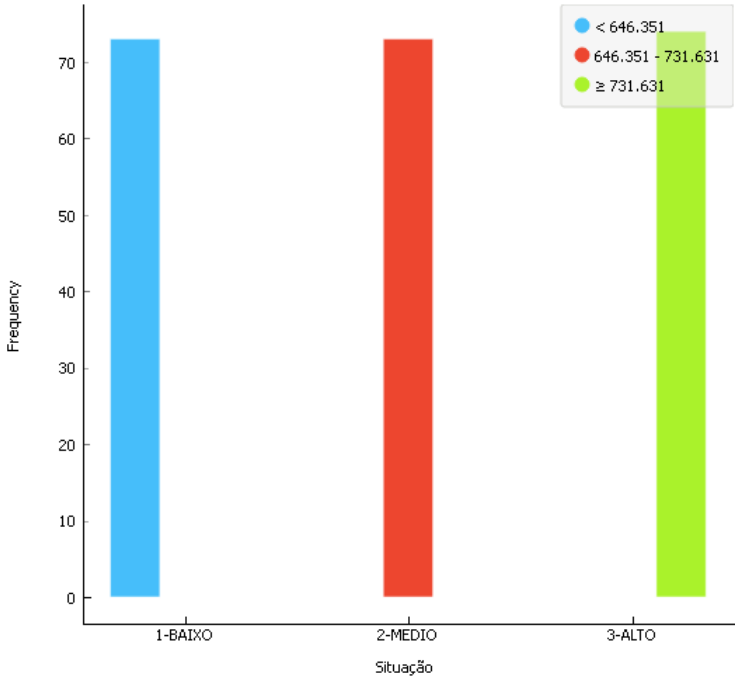

Figura 6.12: Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 5: Baixo, Medio e Alto

relacionado aos dados é determinante no desempenho do *workflow Constellation Queries*. De acordo com o apresentado na Figura 6.13 pode-se obter as seguintes regras:

(1) $(DN > 8) \rightarrow Baixo$

(2) $(DN \leq 8) \wedge (DN > 4) \rightarrow Medio$

(3) $(DN \leq 8) \wedge (DN \leq 4) \rightarrow Alto$

A árvore de classificação aponta que com um maior número de partições a execução do *workflow Constellation Queries* se beneficia do paralelismo disponível no ambiente Petrus. Este mesmo comportamento foi identificado na execução do *workflow SkyMap* descrita nos Cenários 1, 2, 3 e 4. Uma vez que o número de partições está relacionado ao tamanho da partição (DP), verifica-se que $DP < 115$ também determina o grupo *Baixo*. Com a análise oriunda da árvore de classificação, é possível identificar os valores que os dois parâmetros (DN e DP) podem assumir para levar a uma execução classificada como *Baixo*.

Validação: Os resultados de avaliação do modelo preditivo utilizando o método 10-cv estão na Tabela 6.15.Nesta avaliação, a predição do *Grupo* foi correta em 90,4% dos casos.

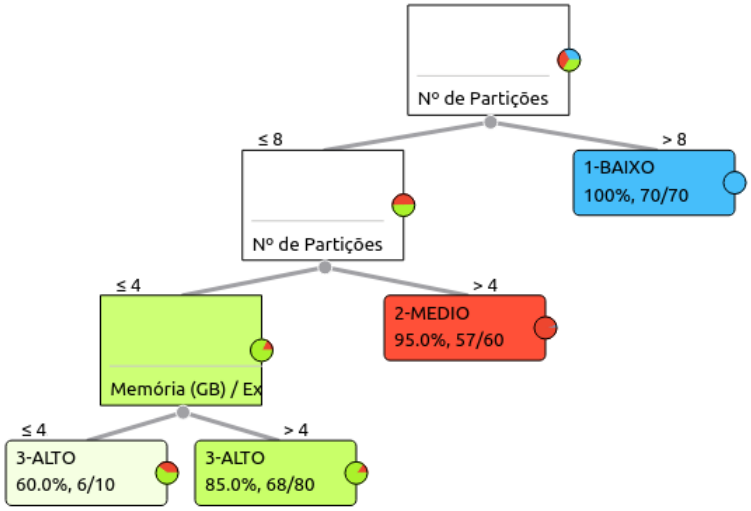

Figura 6.13: Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 5

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,905	0,904	0,912	0,905

Tabela 6.15: Resultado das Métricas utilizadas para avaliação do método árvore de classificação com três níveis no Cenário 5

Por fim, a Tabela 6.16 apresenta os aspectos mais relevantes, descritos anteriormente, de cada cenário para avaliação da abordagem proposta. Conforme definido na Tabela 4.1, CC significa o número de *cores* por nó trabalhador e CM a quantidade de memória RAM disponível para o Spark em cada nó trabalhador.

Cenário	Workflow	Cluster	CC	CM	Regras
1	SkyMap	Petrus	25	80(GB)	DE=FRANCE, DN>128
2	SkyMap	Petrus	12	40(GB)	DE=FRANCE, DN>128
3	SkyMap	Petrus	4	10(GB)	DE=FRANCE, DP>=96
4	SkyMap	Minicluster	2	8(GB)	DN>128, EN>3, DE=FRANCE
5	Const. Queries	Petrus	25	80(GB)	DN>8

Tabela 6.16: Quadro Comparativo entre os Cenários: *Workflow*, *Cluster*, Hardware dos Nós Trabalhadores e Regras que de configuração de parâmetros que levam a uma execução classificada como *Baixo*

6.2 Avaliação dos Modelos Preditivos

Na abordagem proposta, os dados de treinamento são coletados separadamente para cada cenário, aplicação e *cluster*, e consequentemente, este processo de coleta pode levar um tempo relativamente alto. Sendo que, em muitos casos a proveniência já existe tornando desnecessário realizar o processo de coleta novamente. A Tabela 6.17 apresenta o tempo total gasto na etapa de treinamento do modelo preditivo para cada Cenário descrito na Seção 6.1. Por exemplo, para gerar o modelo preditivo correspondente ao Cenário 1, foram realizadas execuções com 77 configurações de parâmetros diferentes. Neste Cenário, cada execução do *workflow* SkyMap no *cluster* Petrus utilizando uma determinada configuração de parâmetros leva um tempo médio de $5,8$ minutos. Com isso, o tempo total de treinamento (quantidade de execuções $\times$ tempo médio) do Cenário 1 foi de aproximadamente $446,6$ minutos.

Adicionalmente a Tabela 6.17 apresenta o ganho que pode ser alcançado em cada Cenário, usando a configuração de parâmetros que leva a uma execução classificada como *Baixo* comparado à configuração de parâmetros que leva a uma execução classificada como *Alto*. Para calcular este ganho, foram utilizados os tempos obtidos na fase de discretização que classificam as execuções como *Baixo* e *Alto*, de acordo com o Cenário. Por exemplo, no Cenário 1, o tempo de execução de uma configuração classificada como *Baixo* é no mínimo $4,1$ vezes mais rápida do que uma configuração de parâmetros classificada como *Alto*, ambas executando o *workflow* SkyMap sobre o *cluster* Petrus. Os valores de cada grupo (*Baixo*, *Medio* e *Alto*), relacionados ao Cenário 1, estão apresentados na Figura 6.1. Ainda no Cenário 1, o tempo de execução de uma configuração classificada como *Baixo* é no mínimo três vezes mais rápida do que o tempo médio de execução. Portanto, neste Cenário o tempo gasto com a etapa de treinamento rapidamente seria compensado com o potencial de ganho que foi possível alcançar.

Pode-se verificar que o Cenário 5 apresenta um ganho muito pequeno comparado aos demais Cenários. Isto se deve ao fato do *workflow Constellation Queries* ser intensivo de CPU, e também os dados devem estar particionados de acordo com a estratégia FRANCE, não sendo possível a execução deste *workflow* com os dados particionados de acordo com a estratégia ADHOC. Com isso, neste Cenário o tempo gasto com a etapa de treinamento iria demorar para ser compensado diante do ganho que foi possível alcançar.

É importante ressaltar que o tamanho total do conjunto de dados de entrada não faz parte do conjunto de parâmetros avaliado e/ou definem um Cenário. Os parâmetros

Cenário	Exec. de Treinamento	Tempo Médio (Min)	Tempo Total (Min)	Ganho
1	77	5,8	446,6	4,1
2	68	8,6	587	4,25
3	24	12,4	298	1,9
4	27	37,61	1015	2,1
5	44	11,3	500,8	1,13

Tabela 6.17: Quadro que apresenta o custo de treinamento em cada Cenário

relacionados aos dados são: tamanho da partição e particionamento. Com isso, o modelo preditivo de um Cenário permanece o mesmo diante de uma mudança do conjunto de dados processado pela *workflow*, desde que, o tamanho das partições e a estratégia de particionamento sejam os mesmos utilizados na etapa de treinamento deste Cenário usando o conjunto de dados de entrada antigo. O tempo de execução do *workflow* intensivo de dados pode ser diferente, uma vez que um volume de dados maior provavelmente irá levar mais tempo para ser processado e vice-versa. Porém, os parâmetros e valores que levam a uma execução classificada como *Baixo*, *Medio* ou *Alto* permanecem os mesmos, não sendo necessário realizar a etapa de treinamento quando o conjunto de dados de entrada a ser processado pelo *workflow* for diferente. Este é um aspecto importante pois os *workflows* científicos intensivos de dados são executados pelos cientistas diversas vezes, alterando somente os dados de entrada [Liu et al., 2015].

A seguir nesta seção são avaliados os modelos preditivos gerados por cada cenário afim de definir e apontar em qual cenário deve ser gerado um novo modelo preditivo ou se já existe um modelo obtido com um outro cenário que realiza uma predição eficiente neste novo cenário, evitando assim a etapa de treinamento para este novo cenário.

6.2.1 Um Modelo Para Todos os Cenários

O objetivo desta subseção é verificar a possibilidade de um modelo preditivo obtido para um cenário ser capaz de realizar a predição para qualquer outro cenário. Para tal, será utilizado como padrão o modelo de predição obtido no Cenário 1 e será avaliado a sua capacidade preditiva sobre os Cenários 4 e 5. Portanto, os dados de execução obtidos com os Cenários 4 e 5 serão utilizados como teste para validação do modelo preditivo do Cenário 1. A Tabela 6.16 apresenta a aplicação e o ambiente de execução que especificam cada cenário apresentado anteriormente.

6.2.1.1 Teste de Validação do Modelo Preditivo do Cenário 1 com os Dados de Execução do Cenário 4

Nesta avaliação o Cenário 1 é utilizado como modelo preditivo e os dados de execução obtidos no Cenário 4 são aplicados como teste de validação do modelo preditivo. A principal diferença entre os cenários é o ambiente de execução Petrus (Cenário 1) e Minicluster (Cenário 4). O modelo preditivo gerado a partir da árvore de classificação e das regras relacionadas ao Cenário 1 estão descritas na subseção 6.1.1 e os dados de teste do Cenário 4 foram discretizados conforme apontado na subseção 6.1.4.

Na Figura 6.14 é apresentada a matriz de confusão obtida com o resultado do modelo preditivo (Predicted) e a classificação real do Cenário 4 (Actual). Pode-se observar que o total de elementos é 135, que representa a quantidade total de execuções do Cenário 4. Com o método de discretização *EqualFrequency* foram gerados três grupos *Baixo*, *Medio* e *Alto* com a mesma quantidade de elementos, ou seja, cada grupo do Cenário 4 possui 45 elementos. Observa-se pela Figura 6.14 que 45 elementos foram classificados como *Baixo* no Cenário 4, mas de acordo com a predição do modelo somente 44,4% seriam classificadas corretamente como *Baixo*. Isto ocorre também nos grupos *Medio* e *Alto* apresentando uma taxa de acerto baixa deste modelo preditivo para este cenário.

		Predicted			
		1-BAIXO	**2-MEDIO**	**3-ALTO**	**∑**
Actual	**1-BAIXO**	44.4 %	22.2 %	33.3 %	**45**
	2-MEDIO	11.1 %	40.0 %	48.9 %	**45**
	3-ALTO	0.0 %	48.9 %	51.1 %	**45**
	∑	**25**	**50**	**60**	**135**

Figura 6.14: Matriz de Confusão Resultante da Aplicação dos dados do Cenário 4 (Actual) no modelo Preditivo gerado a partir do Cenário 1 (Predicted)

A principal diferença entre os cenários é a maior disponibilidade de recursos computacionais para execução do *workflow* no Cenário 1, especialmente núcleos de processamento em cada máquina. No *cluster* Petrus cada máquina possui 25 núcleos de processamento dedicados para o Spark e um total de seis máquinas, levando a um total de 150 núcleos disponíveis. Vale ressaltar, que no *cluster* Petrus cada máquina possui 32 núcleos reais, mas o Sistema Operacional e os demais serviços não compartilham os núcleos dedicados ao Spark. No Minicluster cada máquina possui 2 núcleos de processamento, estando disponível um total de quatro máquinas, levando a um total de 8 núcleos disponíveis compar-

tilhados por Spark, Sistema Operacional e outros processos. No Cenário 1 a configuração que gera maior número de processos paralelos é com $EN = 135, EC = 1, EM = 2$, tendo 135 processos paralelos, consequentemente, ocupando a mesma quantidade de núcleos. Portanto no Cenário 1 não é utilizado a capacidade total dos 150 núcleos, pois sobram 15 na execução que leva ao maior grau de paralelismo.

Por outro lado, no Cenário 4 a configuração que produz o maior número de processos paralelos é com $EN = 3, EC = 4, EM = 8$, tendo 12 processos paralelos e ocupando todos os núcleos do *cluster*, restando 4 processos a serem alocados e competindo por recursos. Esta concorrência entre os processos leva a um pior desempenho impactando na árvore de classificação do próprio Cenário 4. Por exemplo, a quantidade de *executors* e Memória / *Executor* são parâmetros do segundo nível da árvore que representa o Cenário 4, o que não ocorre na árvore de classificação do Cenário 1. Em outra palavras, no Cenário 4, há um maior compartilhamento de *executors* por núcleos físicos, o que não aconteceu na geração do modelo preditivo do Cenário 1, onde há uma grande disponibilidade de núcleos de processamento.

No Spark, a quantidade total de *executors* EN é dada pela quantidade total de memória RAM RM dividido pela quantidade de memória para cada executor EM ($EN = RM/EM$). A quantidade de núcleos do *cluster* RC não é levada em consideração na criação dos *executors* e cada *executor* realiza a execução em um núcleo. Portanto, quanto maior for a razão RM/RC maior é a probabilidade de serem criados mais *executors* do que a quantidade de núcleos disponíveis no *cluster*. No Minicluster esta razão é igual a 4, sendo $RM = 8 * 4 = 32$ e $RC = 2 * 4 = 8$, logo $EN = RM/RC = 32/8 = 4$, ocorrendo neste caso o compartilhamento dos quatro núcleos entre os *executors*, sistema operacional e seus processos. Por outro lado, no Cenário 1 esta razão é igual a $3,2$, sendo $RM = 480$ e $RC = 150$. Com isso, os parâmetros EN, EC, EM são mais relevantes para o Cenário 4 e por isso o modelo preditivo do Cenário 1 não possui uma predição satisfatória para este Cenário.

Na Tabela 6.18 são apresentados os resultados de avaliação do modelo preditivo utilizado sobre os dados de execução do Cenário 4. Nesta avaliação, a predição do *Grupo* foi correta em apenas 41,5% dos casos de acordo com a métrica F1, comprovando que o modelo preditivo obtido pelo Cenário 1 não obtém uma boa probabilidade de acerto na predição dos elementos no Cenário 4, e com isso seria necessário gerar um novo modelo preditivo para o Cenário 4.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,415	0,430	0,485	0,415

Tabela 6.18: Métricas de Avaliação: Cenário 1 (Modelo) versus Cenário 4 (Teste)

6.2.1.2 Teste de Validação do Modelo Preditivo do Cenário 1 com os Dados de Execução do Cenário 5

Nesta avaliação o Cenário 1 é utilizado como modelo preditivo e os dados de execução no Cenário 5 são aplicados como teste de validação do modelo preditivo. A principal diferença entre os cenários é o *workflow* SkyMap (Cenário 1) e Constellation Queries (Cenário 5). Os dados de teste do Cenário 5 foram discretizados conforme apontado na subseção 6.1.5. Na Figura 6.15 é apresentada a matriz de confusão obtida com o resultado do modelo preditivo (Predicted) e a classificação real do Cenário 5 (Actual). Pode-se observar que o total de elementos é 220, que representa a quantidade total de execuções do Cenário 5. Com o método de discretização *EqualFrequency* foram gerados três grupos com a mesma quantidade de elementos, ou seja, cada grupo do Cenário 5 possui aproximadamente 73 elementos. Observa-se pela Figura 6.15 que nesta situação o modelo preditivo iria classificar todos os 220 elementos como *Medio*. Conforme a matriz de confusão todos os elementos dos grupos *Baixo* e *Alto* seriam classificados equivocadamente.

Devido ao fato do *workflow Constellation Queries* possuir um tempo de execução alto em relação ao SkyMap, optamos por reduzir para 1GB o volume de dados de entrada do Cenário 5 para a realização dos experimentos. Com isso foram geradas no máximo 16 partições com tamanho mínimo de 58MB. Além disso, no Cenário 5, os dados foram particionados somente com a estratégia FRANCE. Também, as execuções foram executadas com no máximo 2 *cores / executor* pelo fato de precisar de somente 16 processos paralelos. Com o Cenário 5 sendo bem diferente do Cenário 1 que gerou o modelo preditivo, espera-se que a predição seja considerada insatisfatória, comprovando o fato de que um modelo preditivo não pode ser aplicado para todos os cenários.

Na Tabela 6.19 são apresentados os resultados de avaliação do modelo preditivo utilizado sobre os dados de execução do Cenário 5. Nesta avaliação, a predição do *Grupo* foi correta em apenas 16,5% dos casos de acordo com a métrica F1. Pode-se concluir que o modelo preditivo obtido Cenário 1 não obtém uma boa probabilidade de acerto na predição dos elementos no Cenário 5, e com isso seria necessário gerar um novo modelo preditivo para o Cenário 5.

		Predicted			
		1-BAIXO	**2-MEDIO**	**3-ALTO**	**∑**
Actual	**1-BAIXO**	0.0 %	100.0 %	0.0 %	**73**
	2-MEDIO	0.0 %	100.0 %	0.0 %	**73**
	3-ALTO	0.0 %	100.0 %	0.0 %	**74**
	∑		**220**		**220**

Figura 6.15: Matriz de Confusão Resultante da Aplicação dos dados do Cenário 5 (Actual) no modelo Preditivo gerado a partir do Cenário 1 (Predicted)

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,332	0,165	0,110	0,332

Tabela 6.19: Métricas de Avaliação: Cenário 1 (Modelo) versus Cenário 5 (Teste)

Estes experimentos mostraram que o modelo preditivo do Cenário 1 não realiza uma boa predição para os Cenários 4 e 5. Destaca-se que estes Cenários não são totalmente distintos, ou seja, Cenário 1 e 4 executam a mesma aplicação em ambientes diferentes, já os Cenários 1 e 5 executam aplicações diferentes em um mesmo ambiente. Apesar dessa diferença parcial entre os Cenários observa-se que não é possível construir um modelo preditivo baseado em um Cenário que alcance uma predição satisfatória para todos os outros Cenários. Também é possível afirmar que, quando dois cenários X e Y são totalmente diferentes, aplicação e ambiente, dificilmente o modelo preditivo obtido com o Cenário X alcançaria uma predição satisfatória para o Cenário Y.

6.2.2 Um modelo para cada cenário

O objetivo desta experimento é verificar se é necessário gerar um modelo preditivo agrupado para cada novo cenário. Para tal, será utilizado como padrão o modelo de predição obtido no Cenário 1 e será avaliado a sua capacidade preditiva sobre os Cenários 2 e 3. Portanto os dados de execução obtidos com os Cenários 2 e 3 serão utilizados como teste para validação do modelo preditivo do Cenário 1.

6.2.2.1 Teste de Validação do Modelo Preditivo do Cenário 1 com os Dados de Execução do Cenário 2

Nesta avaliação o Cenário 1 é utilizado como modelo preditivo e os dados de execução no Cenário 2 são aplicados como teste de validação do modelo preditivo. A principal

diferença entre os cenários é o ambiente de execução Petrus (Cenário 1) e Petrus limitado a 50% dos recursos (Cenário 2). Os dados de teste do Cenário 2 foram discretizados conforme apontado na subseção 6.1.2. Na Figura 6.16 é apresentada a matriz de confusão obtida com o resultado do modelo preditivo (Predicted) e a classificação real do Cenário 2 (Actual). Pode-se observar que o total de elementos é 340, que representa a quantidade total de execuções do Cenário 2. Com o método de discretização *EqualFrequency* foram gerados três grupos com a mesma quantidade de elementos, ou seja, cada grupo do Cenário 2 possui aproximadamente 113 elementos. Observa-se pela Figura 6.16 que todos os elementos classificados no grupo *Alto* no Cenário 2 são também classificados pelo modelo preditivo como *Alto*. Os elementos classificados como *Medio* no Cenário 2 são classificados pelo modelo preditivo como *Baixo*, *Medio* e *Alto*, tendo maior probabilidade de serem classificados corretamente, uma vez que 63,7% dos elementos são classificados como *Medio* e os demais 36,3% seriam erroneamente classificados como *Baixo* ou *Alto* pelo modelo preditivo. Os elementos classificados como *Baixo* no Cenário 2 são classificados pelo modelo preditivo como *Baixo* e *Medio* tendo maior probabilidade de serem classificados corretamente como *Baixo*, uma vez que 88,5% dos elementos são classificados neste grupo e os demais 11,5% seriam equivocadamente classificados como *Medio* pelo modelo preditivo.

		Predicted			
		1-BAIXO	**2-MEDIO**	**3-ALTO**	**∑**
Actual	**1-BAIXO**	88.5 %	11.5 %	0.0 %	**113**
	2-MEDIO	4.4 %	63.7 %	31.9 %	**113**
	3-ALTO	0.0 %	0.0 %	100.0 %	**114**
	∑	**105**	**85**	**150**	**340**

Figura 6.16: Matriz de Confusão Resultante da Aplicação dos dados do Cenário 2 (Actual) no modelo Preditivo gerado a partir do Cenário 1 (Predicted)

Na Tabela 6.20 são apresentados os resultados de avaliação do modelo preditivo utilizando sobre os dados de execução do Cenário 2. Nesta avaliação, a predição do *Grupo* foi correta em 82,3% dos casos de acordo com a métrica F1. Pode-se concluir que o modelo preditivo obtido Cenário 1 obtém uma boa probabilidade de acerto na predição dos elementos no Cenário 2, e dessa forma não seria necessário gerar um novo modelo para este cenário.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,826	0,823	0,834	0,826

Tabela 6.20: Métricas de Avaliação: Cenário 1 (Modelo) versus Cenário 2 (Teste)

6.2.2.2 Teste de Validação do Modelo Preditivo do Cenário 1 com os Dados de Execução do Cenário 3

Nesta avaliação o Cenário 1 é utilizado como modelo preditivo e os dados de execução no Cenário 3 são aplicados como teste de validação do modelo preditivo. A principal diferença entre os cenários é o ambiente de execução Petrus (Cenário 1) e Petrus limitado a 10% dos recursos (Cenário 3). Os dados de teste do Cenário 3 foram discretizados conforme apontado na subseção 6.1.3. Na Figura 6.17 é apresentada a matriz de confusão obtida. Pode-se observar que o total de elementos é 120, que representa a quantidade total de execuções do Cenário 3. Com o método de discretização *EqualFrequency* foram gerados três grupos com a mesma quantidade de elementos, ou seja, cada grupo do Cenário 3 possui 40 elementos. Observa-se pela Figura 6.17 que todos os elementos classificados nos grupos *Baixo* e *Alto* no Cenário 3 o modelo preditivo do Cenário 1 também os classificaria como *Baixo* e *Alto*, respectivamente. Por outro lado, os elementos classificados como *Medio* no Cenário 3 poderiam ser classificados pelo modelo como *Baixo*, *Medio* e *Alto*, tendo maior probabilidade de serem classificados de forma equivocada, uma vez que somente 37,5% dos elementos seriam classificados como *Medio* e os demais 62,5% seriam erroneamente classificados como *Baixo* ou *Alto* pelo modelo preditivo.

		Predicted			
		1-BAIXO	**2-MEDIO**	**3-ALTO**	**∑**
Actual	**1-BAIXO**	100.0 %	0.0 %	0.0 %	**40**
	2-MEDIO	12.5 %	37.5 %	50.0 %	**40**
	3-ALTO	0.0 %	0.0 %	100.0 %	**40**
	∑	**45**	**15**	**60**	**120**

Figura 6.17: Matriz de Confusão Resultante da Aplicação dos dados do Cenário 3 (Actual) no modelo Preditivo gerado a partir do Cenário 1 (Predicted)

Na Tabela 6.21 são apresentados os resultados de avaliação do modelo preditivo utilizando sobre os dados de execução do Cenário 3. Nesta avaliação, a predição do *Grupo* foi correta em 76,2% dos casos de acordo com a métrica F1. Pode-se concluir que o modelo preditivo obtido Cenário 1 alcança uma boa probabilidade de acerto na predição dos

elementos no Cenário 3, e com isso não é necessário gerar um novo modelo preditivo para o Cenário 3.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,792	0,762	0,852	0,792

Tabela 6.21: Métricas de Avaliação: Cenário 1 (Modelo) versus Cenário 3 (Teste)

Através dos experimentos pode-se verificar que um modelo preditivo gerado pelo Cenário 1 é capaz de realizar a predição dos cenários 2 e 3 com uma boa probabilidade de acerto, e com isso não seria necessário gerar um novo modelo para os Cenários 2 e 3. Neste contexto específico, os Cenários 1,2 e 3 possuem em comum a aplicação analisada. Isto permite concluir que dado dois Cenários X e Y parcialmente diferentes, mesma aplicação e ambientes de execução diferentes, existe a possibilidade de um modelo preditivo obtido com o Cenário X realizar uma boa predição para o Cenário Y.

6.2.3 Um Modelo Preditivo Agrupado por Aplicação

Diante das conclusões obtidas na Seção anterior, nesta seção é realizada a avaliação de um modelo preditivo agrupado por aplicação, ou seja, utilizando os dados de treinamentos obtidos através da execução de uma aplicação SkyMap em dois ambientes computacionais diferentes. O objetivo é avaliar se este modelo agrupado é capaz de realizar a predição de execução da aplicação SkyMap para cada ambiente computacional de forma separada. Para tal, será utilizado como padrão um novo modelo de predição obtido no Cenário 6, descrito na subseção 6.2.3.1, e será avaliado a sua capacidade preditiva sobre os Cenários 1 e 4. Portanto, os dados de execução obtidos com os Cenários 1 e 4 serão utilizados como teste para validação do modelo preditivo do Cenário 6.

6.2.3.1 Cenário 6 - Análise da Árvore de Classificação gerada pela Execução do *Workflow* SkyMap no Cluster Petrus e no Minicluster

Nestes experimentos é realizada a aplicação da abordagem proposta na construção de um modelo agrupado por aplicação, neste caso, execução do *workflow* SkyMap sobre diferentes infraestruturas de execução.

Treinamento: Foram utilizados para os experimentos dois *clusters* com a configuração de hardware apresentada na Tabela 5.1. Embora o *cluster* Petrus possua muito mais memória e núcleos do que o Minicluster, as configurações do Spark foram as mesmas

em ambos os *clusters* afim de uma comparação mais justa. Com isso, os parâmetros do Spark em ambos os *clusters* foram limitados pelo hardware do Minicluster. Portanto, as configurações dos parâmetros usadas nos dois *clusters* são apresentadas na Tabela 6.22.

Diversas combinações dos parâmetros na Tabela 6.22 foram avaliadas seguindo a ordem dos parâmetros apresentados na Tabela 6.22. Para cada combinação do conjunto de parâmetros avaliado foram realizadas cinco execuções. Nos experimentos descritos nesta subseção foram avaliadas 56 combinações dos parâmetros gerando assim um total de 280 execuções.

Spark	Qtd. Executors	EN	3, 4, 8
	Cores/Exec	EC	1, 2, 4
	Mem/Exec (GB)	EM	2, 4, 8
Entrada	Tam. Dataset	DS	24GB
	Estratégia de Particionamento	DE	FRANCE, ADHOC
	Tam. da Partição(MB)	DP	96, 192, 256, 384
	Núm. de Partições	DN	64, 96, 128, 256
Hardware	*Cluster*	R	Petrus, Minicluster

Tabela 6.22: Valores dos Parâmetros Avaliados na Execução do Cenário 6

Discretização: Os tempos de execução obtidos foram submetidos ao método *Equal-Frequency* e foram gerados três *Grupos* de classificação de acordo com o tempo de execução: *Baixo* as execuções que obtiveram tempo abaixo de 565,18 segundos, como *Medio* as execuções que obtiveram tempo entre 565,18 e 1261,94 segundos e foram classificadas como *Alto* as execuções que obtiveram tempo acima de 1261,94 segundos.

Classificação: A Figura 6.19 apresenta a árvore de classificação com três níveis obtida através das execuções do workflow Skymap. Observa-se que o modelo preditivo do Cenário 6 é dividido de acordo com os Cenários 1 e 4. O lado direito da árvore de classificação do Cenário 6 está relacionado ao Cenário 4 e o esquerdo ao Cenário 1, uma vez que o parâmetro raiz da árvore é a quantidade de nós. No Cenário 1 as execuções foram realizadas em 6 nós enquanto no Cenário 4 as execuções foram realizadas em 4 nós. Com isso, o lado direito da árvore só classifica em *Alto* e *Medio* e o lado esquerdo classifica em *Baixo* e *Medio*. Isto se reflete na matriz de confusão, pois o modelo preditivo classifica todas as execuções do Cenário 1 como *Baixo* e *Medio*. Inversamente, a matriz de confusão apresentada na seção 6.2.3.3 classifica todas as execuções do Cenário 4 como *Alto* e *Medio*.

Nesta análise, pode-se afirmar que o *Grupo Baixo* pode ser definido pelos seguintes parâmetros: Qtd. de Nós (QN), Cores por executor (EC), particionamento (DE) e o

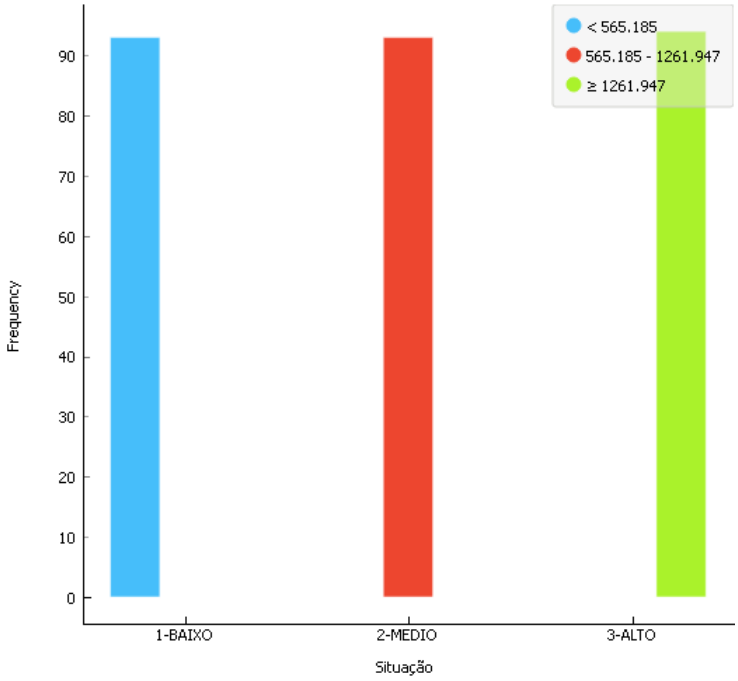

Figura 6.18: Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 6: Baixo, Medio e Alto

tamanho da partição (DP). Conforme a Figura 6.19 pode-se obter as seguintes regras para execuções classificadas como $Baixo$:

(1) $(CN > 4) \wedge (EC > 1) \wedge (DE = FRANCE) \rightarrow Baixo$

(2) $(CN > 4) \wedge (EC > 1) \wedge (DE = ADHOC) \rightarrow Baixo$

(3) $(CN > 4) \wedge (EC \leq 1) \wedge (DP \leq 96) \rightarrow Baixo$

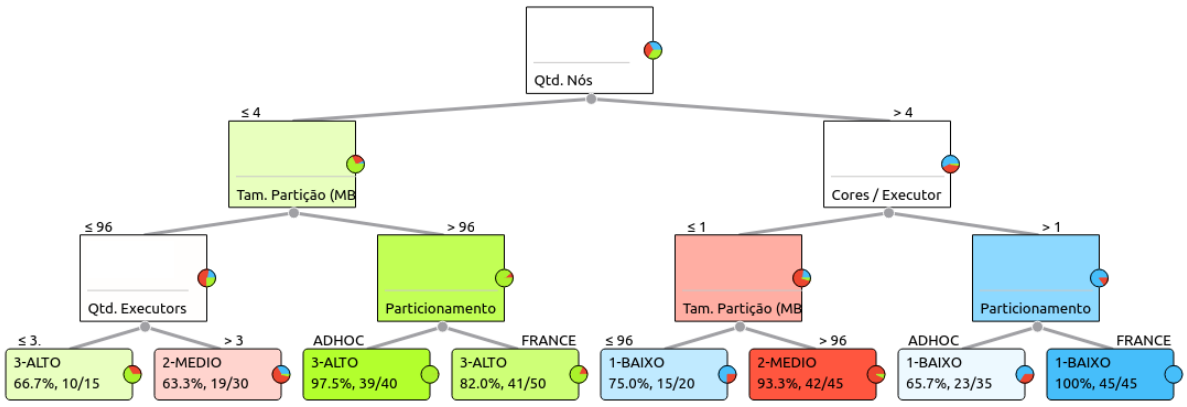

Figura 6.19: Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 6

Validação: Os resultados de avaliação do modelo preditivo utilizando o método 10-cv estão na Tabela 6.23. Nesta avaliação, a predição do grupo foi correta em 79,4% dos casos.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,800	0,794	0,795	0,800

Tabela 6.23: Métricas utilizadas para avaliação do método de classificação

6.2.3.2 Teste de Validação do Modelo Preditivo do Cenário 6 com os Dados de Execução do Cenário 1

Nesta avaliação o Cenário 6 é utilizado como modelo preditivo e os dados de execução no Cenário 1 são aplicados como teste de validação do modelo preditivo. Os dados de teste do Cenário 1 foram discretizados conforme apontado na Subseção 6.2.3.1. Na Figura 6.20 é apresentada a matriz de confusão obtida com o resultado do modelo preditivo (Predicted) e a classificação real do Cenário 1 (Actual). Pode-se observar que o total de elementos é 145, que representa a quantidade total de execuções do Cenário 1. Com a aplicação do método de discretização obtido no Cenário 6 foram gerados três grupos com diferentes quantidades de elementos, sendo $Baixo = 83$, $Medio = 59$ e $Alto = 3$. Observa-se pela Figura 6.20 que nesta situação o modelo preditivo iria classificar a maioria dos elementos (100) como *Baixo*. Conforme a matriz de confusão todos os elementos dos grupos *Alto* e 28,8% dos elementos do grupo *Medio* seriam classificados equivocadamente de acordo com o modelo preditivo.

		Predicted			
		1-BAIXO	**2-MEDIO**	**3-ALTO**	**∑**
Actual	**1-BAIXO**	100.0 %	0.0 %	0.0 %	**83**
	2-MEDIO	28.8 %	71.2 %	0.0 %	**59**
	3-ALTO	0.0 %	100.0 %	0.0 %	**3**
	∑	**100**	**45**		**145**

Figura 6.20: Matriz de Confusão Resultante da Aplicação dos dados do Cenário 1 (Actual) no modelo Preditivo gerado a partir do Cenário 6 (Predicted)

Uma vez que o lado esquerdo da árvore de classificação do Cenário 6 só classifica em *Baixo* ou *Medio*, isto se reflete na matriz de confusão, pois o modelo preditivo classifica todas as execuções do Cenário 1 como *Baixo* ou *Medio*.

Na Tabela 6.24 são apresentados os resultados de avaliação do modelo preditivo sobre

os dados de execução do Cenário 1. Nesta avaliação, a predição do *Grupo* foi correta em 84,8% dos casos de acordo com a métrica F1. Pode-se concluir que o modelo preditivo obtido no cenário 6 obtém uma boa probabilidade de acerto na predição dos elementos no Cenário 1.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,862	0,848	0,855	0,862

Tabela 6.24: Métricas de Avaliação: Cenário 6 (Modelo) versus Cenário 1 (Teste)

6.2.3.3 Teste de Validação do Modelo Preditivo do Cenário 6 com os Dados de Execução do Cenário 4

Nesta avaliação o Cenário 6 é utilizado como modelo preditivo e os dados de execução no Cenário 4 são aplicados como teste de validação do modelo preditivo. Os dados de teste do Cenário 4 foram discretizados conforme apontado na subseção 6.2.3.1. Na Figura 6.21 é apresentada a matriz de confusão obtida com o resultado do modelo preditivo (Predicted) e a classificação real do Cenário 4 (Actual). Pode-se observar que o total de elementos é 135, que representa a quantidade total de execuções do Cenário 4. Com a aplicação do método de discretização obtido no Cenário 6 foram gerados três grupos com diferentes quantidades de elementos, sendo $Baixo = 10$, $Medio = 34$ e $Alto = 91$. Observa-se pela Figura 6.21 que nesta situação o modelo preditivo iria classificar a maioria dos elementos (105) como *Alto*. Conforme a matriz de confusão todos os elementos dos grupos *Baixo*, 44,1% dos elementos do grupo *Medio* e 1,1% dos elementos do grupo *Alto* seriam classificados erroneamente de acordo com o modelo preditivo.

		Predicted			
		1-BAIXO	**2-MEDIO**	**3-ALTO**	**∑**
Actual	**1-BAIXO**	0.0 %	100.0 %	0.0 %	**10**
	2-MEDIO	0.0 %	55.9 %	44.1 %	**34**
	3-ALTO	0.0 %	1.1 %	98.9 %	**91**
	∑		**30**	**105**	**135**

Figura 6.21: Matriz de Confusão Resultante da Aplicação dos dados do Cenário 4 (Actual) no modelo Preditivo gerado a partir do Cenário 6 (Predicted)

Uma vez que o lado direito da árvore de classificação do Cenário 6, na Figura 6.19, só classifica em *Alto* ou *Medio*, isto se reflete na matriz de confusão pois o modelo preditivo classifica todas as execuções do Cenário 4 como *Alto* ou *Medio*.

Na Tabela 6.25 são apresentados os resultados de avaliação do modelo preditivo utilizando sobre os dados de execução do Cenário 4. Nesta avaliação, a predição do *Grupo* foi correta em 76,9% dos casos de acordo com a métrica F1. Pode-se concluir que o modelo preditivo obtido Cenário 6 também obtém uma boa probabilidade de acerto na predição dos elementos no Cenário 4.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,869	0,769	0,737	0,807

Tabela 6.25: Métricas de Avaliação: Cenário 6 (Modelo) versus Cenário 4 (Teste)

Nestes experimentos verifica-se que o modelo preditivo agrupado por ambiente é importante para destacar a eficiência dos ambientes computacionais na execução de um determinado *workflow*. Isto foi possível, porque existem parâmetros (Qtd. Nós, CPU, Cores, RAM) que são diferentes em cada ambiente avaliado e consequentemente são utilizados pelo método de classificação na construção do modelo preditivo. Com isso, foi possível atestar que um ambiente com maior poder computacional (Petrus) executa o *workflow* de forma mais rápida do que um ambiente com capacidade inferior (Minicluster). Este resultado já é esperado, mas o modelo preditivo agrupado mostra também os parâmetros de configuração importantes em cada Cenário e os intervalos de valores que cada parâmetro assume, para cada grupo de execução. Esse tipo de conhecimento quantitativo sobre o desempenho e os parâmetros seria muito complexo de ser obtido sem a utilização dos métodos de aprendizado de máquina. Por fim, através dos experimentos pode-se concluir que o modelo preditivo gerado por dois cenários diferentes pode ser aplicado na predição dos mesmos cenários de forma separada.

A Tabela 6.26 apresenta um resumo dos principais aspectos relacionados aos Cenários 1,4 e 6.

Cenário	Workflow	Cluster	CC	CM	Regras
6	SkyMap	Petrus,Minicluster	2	8(GB)	CN>4, EC>1, DN>=256
1	SkyMap	Petrus	25	80(GB)	DE=FRANCE, DN>128
4	SkyMap	Minicluster	2	8(GB)	DN>128, EN>3, DE=FRANCE

Tabela 6.26: Quadro Comparativo entre os Cenários: *Workflow*, *Cluster*, Hardware dos Nós Trabalhadores e Regras de configuração de parâmetros que levam a uma execução classificada como *Baixo*

6.2.4 Um Modelo Preditivo Agrupado por Ambiente de Execução

Nesta seção é realizada a avaliação de um modelo preditivo utilizando os dados de treinamentos obtidos através da execução em um ambiente computacional de duas aplicações diferentes. O objetivo é avaliar se este modelo agrupado é capaz de realizar a predição de execução no ambiente computacional para cada aplicação de forma separada. Para tal, será utilizado como padrão o modelo de predição obtido no Cenário 7, descrito na subseção 6.2.4.1, e é avaliado a sua capacidade preditiva sobre os Cenários 1 e 5. Portanto os dados de execução obtidos com os Cenários 1 e 5 serão utilizados como teste para validação do modelo preditivo do Cenário 7.

6.2.4.1 Cenário 7 - Análise da Árvore de Classificação gerada pela Execução dos *Workflows* SkyMap e Constellation Queries no Cluster Petrus

Nestes experimentos é realizada a aplicação da abordagem proposta na construção de um modelo agrupado por ambiente, em particular, os *workflows* SkyMap e Constellation Queries sobre o *cluster* Petrus.

Treinamento: Os parâmetros do Spark foram avaliados igualmente para os dois *workflows* como apontado na Tabela 6.27. Ainda, conforme a Tabela 6.27, os parâmetros avaliados relacionados aos dados de entrada estão apresentados de forma separada para cada *workflow*. Diversas combinações dos parâmetros na Tabela 6.27 foram avaliadas e para cada combinação do conjunto de parâmetros avaliado foram realizadas cinco execuções. Nos experimentos descritos nesta subseção foram avaliadas 121 combinações dos parâmetros gerando assim um total de 605 execuções.

Spark	Qtd. Executors	EN	6, 12, 18, 42, 66, 137
	Cores/Exec	EC	1, 2, 4, 8, 16
	Mem/Exec (GB)	EM	1, 2, 4, 8, 16, 32, 64
Entrada - SkyMap	Tam. Dataset	DS	24GB
	Estratégia de Particionamento	DE	FRANCE, ADHOC
	Tam. da Partição(MB)	DP	48, 96, 192, 256, 384
	Núm. de Partições	DN	64, 96, 128, 256, 512
Entrada - Const. Queries	Tam. Dataset	DS	1GB
	Estratégia de Particionamento	DE	FRANCE
	Tam. da Partição(MB)	DP	58, 115, 230, 460
	Núm. de Partições	DN	2, 4, 8, 16

Tabela 6.27: Valores dos Parâmetros Avaliados na Execução do Cenário 7

Discretização: Os tempos de execução obtidos foram submetidos ao método *Equal-*

Frequency e foram gerados três *Grupos* de classificação de acordo com o tempo de execução: *Baixo*, *Medio* e *Alto*. Conforme a Figura 6.22 foram classificadas como *Baixo* as execuções que obtiveram tempo abaixo de 194,53 segundos, como *Medio* as execuções que obtiveram tempo entre 194,53 e 645,96 segundos e foram classificadas como *Alto* as execuções que obtiveram tempo acima de 645,96 segundos.

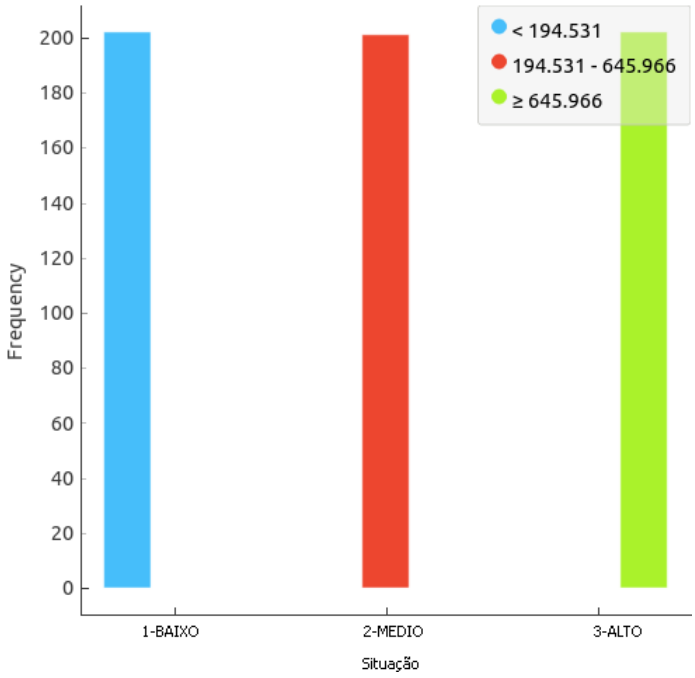

Figura 6.22: Grupos de execuções obtidos com a Discretização do Tempo de Execução do Cenário 7: Baixo, Medio e Alto

Classificação: A Figura 6.23 apresenta a árvore de classificação com três níveis obtida através das execuções dos *workflows* SkyMap e *Constellation Queries*. Nesta análise, pode-se afirmar que o *Grupo Baixo* pode ser definido pelos seguintes parâmetros: e tamanho do dataset (DS) e o tamanho da partição (DP). Conforme a Figura 6.23 pode-se obter as seguintes regras para execuções classificadas como *Baixo*:

(1) $(DS > 1) \wedge (DP \leq 192) \rightarrow Baixo$

Validação: Os resultados de avaliação do modelo preditivo utilizando o método 10-cv estão na Tabela 6.28. Nesta avaliação, a predição do grupo foi correta em 94,2% dos casos.

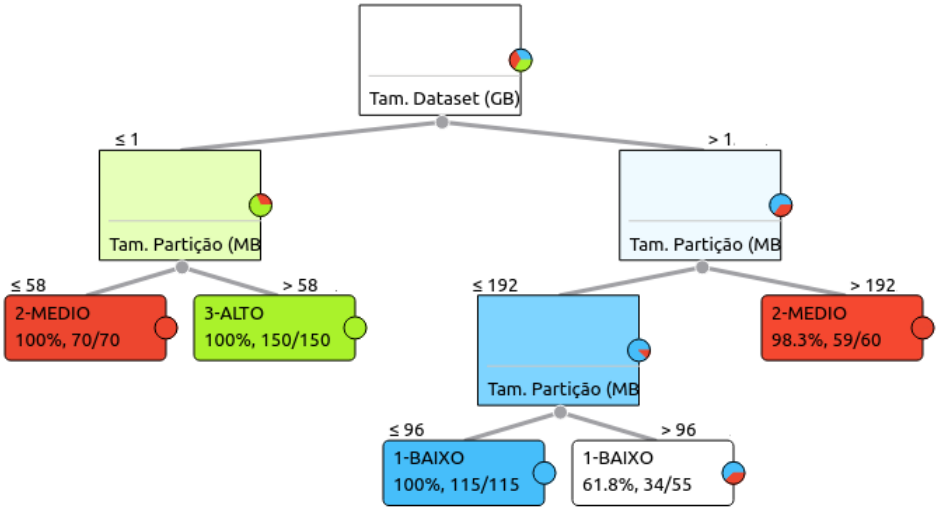

Figura 6.23: Árvore de Classificação do Grupo de execução com 3 níveis para o Cenário 7

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,942	0,942	0,944	0,942

Tabela 6.28: Métricas utilizadas para avaliação do método de classificação

6.2.4.2 Teste de Validação do Modelo Preditivo do Cenário 7 com os Dados de Execução do Cenário 1

Nesta avaliação o Cenário 7 é utilizado como modelo preditivo e os dados de execução no Cenário 1 são aplicados como teste de validação do modelo preditivo. Os dados de teste do Cenário 1 foram discretizados conforme apontado na subseção 6.2.4.1. Na Figura 6.24 é apresentada a matriz de confusão obtida com o resultado do modelo preditivo (Predicted) e a classificação real do Cenário 1 (Actual). Pode-se observar que o total de elementos é 230, que representa a quantidade total de execuções do Cenário 1. Com a aplicação do método de discretização obtido no Cenário 7 foram gerados três grupos com diferentes quantidades de elementos, sendo $Baixo = 150$, $Medio = 80$ e $Alto = 0$. Observa-se pela Figura 6.24 que nesta situação o modelo preditivo iria classificar a maioria dos elementos (170) como *Baixo*. Conforme a matriz de confusão apenas 0,7% dos elementos dos grupos *Baixo* e 26,2% dos elementos do grupo *Medio* seriam classificados equivocadamente de acordo com o modelo preditivo.

Observa-se que o modelo preditivo do Cenário 7 é dividido de acordo com o Cenários 1 e 5. O lado direito da árvore de classificação do Cenário 7, apresentada na Figura 6.23, está relacionado ao Cenário 1 e o esquerdo ao Cenário 5, uma vez que o parâmetro raiz da

		Predicted			
		1-BAIXO	**2-MEDIO**	**3-ALTO**	**∑**
Actual	**1-BAIXO**	99.3 %	0.7 %	0.0 %	**150**
	2-MEDIO	26.2 %	73.8 %	0.0 %	**80**
	3-ALTO	NA	NA	NA	
	∑	**170**	**60**		**230**

Figura 6.24: Matriz de Confusão Resultante da Aplicação dos dados do Cenário 1 (Actual) no modelo Preditivo gerado a partir do Cenário 7 (Predicted)

árvore é o tamanho do *dataset* de entrada. No Cenário 1 as execuções foram realizadas tendo 24GB de dados, enquanto no Cenário 5 as execuções foram realizadas com 1GB. O lado direito da árvore só classifica em *Baixo* e *Medio* e o lado esquerdo classifica em *Alto* e *Medio*. Isto se reflete na matriz de confusão, pois o modelo preditivo classifica todas as execuções do Cenário 1 como *Baixo* e *Medio*. Consequentemente, a matriz de confusão do Cenário 4 classifica todas as execuções como *Alto* e *Medio* como descrito na Seção 6.2.4.3.

Na Tabela 6.29 são apresentados os resultados de avaliação do modelo preditivo utilizando sobre os dados de execução do Cenário 1. Nesta avaliação, a predição do *Grupo* foi correta em 90,1% dos casos de acordo com a métrica F1. Pode-se concluir que o modelo preditivo agrupado por ambiente do Cenário 7 obtém uma boa probabilidade de acerto na predição dos elementos no Cenário 1.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	0,904	0,901	0,914	0,904

Tabela 6.29: Métricas de Avaliação: Cenário 7 (Modelo) versus Cenário 1 (Teste)

6.2.4.3 Teste de Validação do Modelo Preditivo do Cenário 7 com os Dados de Execução do Cenário 5

Nesta avaliação o Cenário 7 é utilizado como modelo preditivo e os dados de execução no Cenário 5 são aplicados como teste de validação do modelo preditivo. Os dados de teste do Cenário 5 foram discretizados conforme apontado na subseção 6.2.4.1. Na Figura 6.25 é apresentada a matriz de confusão obtida com o resultado do modelo preditivo (Predicted) e a classificação real do Cenário 5 (Actual). Pode-se observar que o total de elementos é 220, que representa a quantidade total de execuções do Cenário 5. Com a aplicação do método de discretização obtido no Cenário 7 foram gerados três grupos com diferentes

quantidades de elementos, sendo $Baixo = 0$, $Medio = 70$ e $Alto = 150$. Observa-se pela Figura 6.25 que nesta situação o modelo preditivo iria classificar a maioria dos elementos (150) como *Alto*. Conforme a matriz de confusão todos os elementos dos grupos *Alto* e os elementos do grupo *Medio* seriam classificados corretamente de acordo com o modelo preditivo.

		Predicted			
		1-BAIXO	**2-MEDIO**	**3-ALTO**	**∑**
Actual	**1-BAIXO**	NA	NA	NA	
	2-MEDIO	0.0 %	100.0 %	0.0 %	**70**
	3-ALTO	0.0 %	0.0 %	100.0 %	**150**
	∑		**70**	**150**	**220**

Figura 6.25: Matriz de Confusão Resultante da Aplicação dos dados do Cenário 5 (Actual) no modelo Preditivo gerado a partir do Cenário 7 (Predicted)

Na Tabela 6.29 são apresentados os resultados de avaliação do modelo preditivo utilizando sobre os dados de execução do Cenário 5. Nesta avaliação, a predição do *Grupo* foi correta em 100% dos casos de acordo com a métrica F1. Pode-se concluir que o modelo preditivo obtido Cenário 7 obtém uma ótima probabilidade de acerto na predição dos elementos no Cenário 5.

Método	Accuracy	F1	Precision	Recall
Árvore de Classificação	1,000	1,000	1,000	1,000

Tabela 6.30: Métricas de Avaliação: Cenário 7 (Modelo) versus Cenário 5 (Teste)

O modelo preditivo agrupado por aplicação pode ser usado para classificar o desempenho de aplicações em um determinado ambiente. Isto foi possível, porque existem parâmetros (Tam. Dataset, características do *workflow*) que são diferentes em cada aplicação avaliada e consequentemente são utilizadas pelo método de classificação na construção do modelo preditivo. Nesta seção, verifica-se que a aplicação SkyMap possui um processamento muito rápido para um volume de dados de entrada de 24GB quando comparado com a aplicação *Constellation Queries* que tem um tempo de processamento mais alto com um volume de dados relativamente menor, cerca de 1GB. Este modelo preditivo agrupado mostra também os parâmetros de configuração importantes em cada aplicação e os intervalos de valores que cada parâmetro assume, para cada grupo de execução. Por fim, através dos experimentos pode-se concluir que o modelo preditivo gerado por dois cenários diferentes pode ser aplicado na predição dos mesmos cenários de forma separada.

A Tabela 6.31 apresenta um resumo dos principais aspectos relacionados aos Cenários 1,5 e 7.

Cenário	Workflow	Cluster	CC	CM	Regras
7	SkyMap, Const. Queries	Petrus	25	80(GB)	DS>1, DP<=192
1	SkyMap	Petrus	25	80(GB)	DE=FRANCE, DN>128
5	Const. Queries	Petrus	25	80(GB)	DN>8

Tabela 6.31: Quadro Comparativo entre os Cenários: *Workflow*, *Cluster*, Hardware dos Nós Trabalhadores e Regras que de configuração de parâmetros que levam a uma execução classificada como *Baixo*

6.2.5 Abordagem Proposta para Criação do Modelo Preditivo

Avaliando os modelos gerados pode-se afirmar que um modelo preditivo criado para um determinado Cenário não será válido para todos os outros Cenários, como descrito na Seção 6.2.1. Em contrapartida, pode-se afirmar também que não é necessária a criação de um modelo para cada Cenário, uma vez que os resultados apontaram que um modelo preditivo criado para um determinado Cenário pode ser usado eficientemente para outro Cenário. Em particular, como descrito na Seção 6.2.2, verifica que o modelo preditivo criado para o Cenário 1 é capaz de realizar uma boa predição para os Cenários 2 e 3.

A Figura 6.26 possui o objetivo de apresentar uma estratégia para criação dos modelos preditivos para novos cenários. De acordo com a Figura existe uma base de modelos preditivos para outros cenários que é consultada antes de decidir se deverá ser criado um novo modelo para um novo cenário.

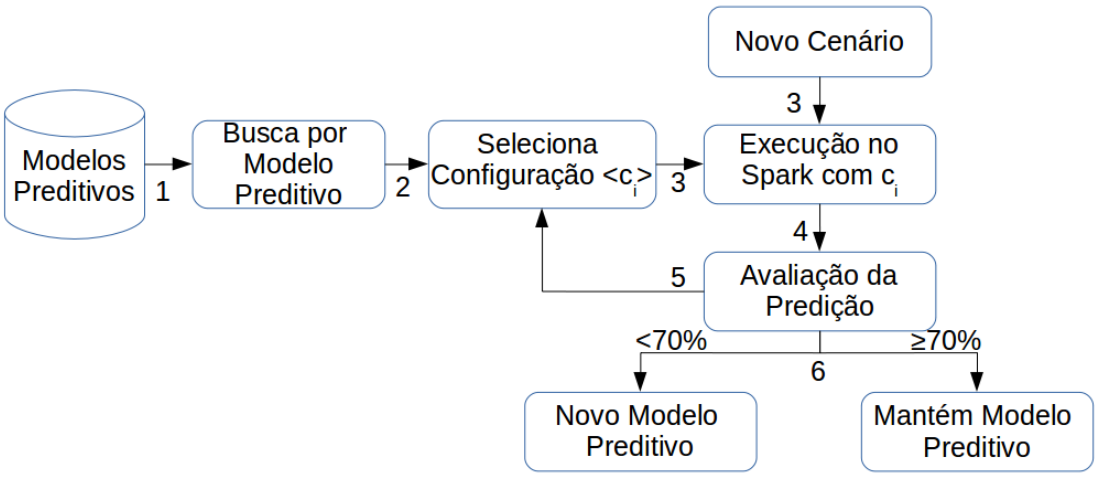

Figura 6.26: Abordagem Proposta para Criação do Modelo Preditivo

1. Primeiramente a base de modelos é consultada para selecionar um modelo preditivo parcialmente diferente, ou seja, que foi obtido para um outro Cenário que possui a mesma aplicação ou ambiente de execução. Se não existir nenhum modelo, então deve ser aplicada a abordagem proposta para geração de um novo modelo descrita na Seção 4.2;

2. Se existir um modelo preditivo para outro Cenário, então deve ser usado o modelo preditivo inicialmente para selecionar um subconjunto de configurações de parâmetros a serem empregadas na execução do novo Cenário para avaliação do modelo preditivo. A seleção do subconjunto de configurações deve abranger igualmente os três grupos de execução: $Baixo$,$Medio$ e $Alto$;

3. Em seguida, o novo Cenário é executado por algumas vezes no Spark utilizando a cada execução uma configuração de parâmetros diferente do subconjunto de configurações que foram selecionadas previamente;

4. Após as execuções deve ser realizado uma avaliação do modelo preditivo afim de verificar se o mesmo alcançou uma taxa de acerto satisfatória;

5. Quando a avaliação da predição é maior ou igual a 70% não é necessário criar um novo modelo preditivo para este novo Cenário. Diante disso é necessário criar um novo modelo preditivo para um determinado Cenário X quando os modelos preditivos existentes obtidos com outros cenários, para a mesma aplicação ou ambiente, não alcançarem uma probabilidade de acerto satisfatória, considerado nesta tese como 70%.

6.3 Resumo do Capítulo

Neste Capítulo foi descrita a aplicação da abordagem proposta de otimização baseada na configuração de parâmetros em diversos cenários. Observou-se que em cada uma delas foi possível obter um subconjunto de parâmetros e intervalos de valores que de fato beneficiam o desempenho do *workflow* em um determinado ambiente de execução, validando assim a abordagem de otimização proposta. Também foi realizada uma análise para definir em quais situações deve ser criado um modelo preditivo para um novo cenário ou usado um modelo já existente.

Capítulo 7

Literatura Relacionada

Os assuntos tratados nesta tese estão relacionados ao problema de otimização da execução paralela de *workflows* MR intensivos de dados em *clusters* computacionais através da exploração da configuração de parâmetros como por exemplo: particionamento dos dados, tamanho das partições, quantidade de memória por processo e de núcleos. Este capítulo é dedicado a identificar e apresentar esforços relacionados aos assuntos que são endereçados por esta tese de doutorado. Diversos trabalhos têm sido desenvolvidos para otimização do desempenho de *workflows* científicos em sistemas MapReduce. É também realizada uma análise crítica de cada um dos trabalhos encontrados, além de apontar sua relevância para a tese.

7.1 Abordagens Existentes para Execução de Aplicações Intensivas de Dados

Sistemas para análise de dados podem ser divididos em duas categorias principais: SGBDs e *workflows*. Tradicionalmente, ferramentas conhecidas de Data Warehouses (DW) construídas sobre SGBDs realizam a análise de grandes volumes de dados. SGBDs paralelos que executam em *clusters*, como Teradata [Teradata, 2012] e Greenplum [Greenplum, 2012], fornecem suporte à análise no contexto DW. Por outro lado, sistemas de execução paralela de *workflows* como MapReduce [Dean and Ghemawat, 2004], Hadoop [White, 2009], Pig [Gates et al., 2009], Hive [Thusoo et al., 2010] e Spark [Zaharia et al., 2010b], surgiram mais recentemente e estão gradativamente se tornando populares tanto no contexto acadêmico e científico quanto em empresas. Na Subseção 7.1.1 são apresentadas as otimizações existentes em SGBDs e MR. Na Subseção 7.1.2 são apresentadas as otimizações realizadas em sistemas MR relacionadas à localidade de dados e escalonamento das tarefas

em diferentes ambientes de execução. Na Subseção 7.1.3 é apresentada a relevância dos trabalhos descritos nesta Seção no contexto geral da tese.

7.1.1 Otimização do MapReduce e SGBDs

Estas duas categorias de sistemas possuem algumas diferenças em áreas básicas como:

Estrutura e Armazenamento dos Dados: A maioria dos SGBDs necessitam que os dados estejam de acordo com um esquema e possuem uma forma específica de armazenamento. Tipicamente, a estrutura de dados segue o modelo relacional apesar de suportar outros modelos como XML e JSON. Esta estrutura bem definida dos dados permite que os SGBDs construam estruturas de índices. Por outro lado, sistemas MR tipicamente processam dados armazenados no sistema de arquivos, permitindo que estes dados estejam em qualquer formato podendo ser estruturados, semi-estruturados ou não-estruturados.

Interface de Acesso: Structured Query Language (SQL) é uma das linguagens declarativas mais utilizadas para acessar, gerenciar e analisar dados em SGBDs relacionais. Usuários podem expressar uma consulta através do SQL e o SGBD irá otimizar e executar a consulta. Por outro lado, o modelo de programação MapReduce é inspirado na programação funcional. Este modelo é mais complexo do que o SQL para ser utilizado em rotinas diárias dos usuários [Olston et al., 2008] [Thusoo et al., 2010]. Como resultado, o *framework* MR evoluiu rapidamente nos últimos anos incluindo linguagens de nível mais alto em sistemas como Hive (SQL-Like) e Pig(com uma interface que possui elementos funcionais e declarativos).

Estratégia de Execução: Uma vez que o paradigma MapReduce foi criado recentemente, este modelo de execução ainda carece de um desenvolvimento de técnicas de otimização quando comparado com SGBDs. Em SGBDs existe um componente denominado de *Query Optimizer* que é responsável por realizar uma execução eficiente das consultas. De forma a alcançar este objetivo, para cada consulta, o *Query Optimizer* produz diferentes planos de execução, estima o tempo de execução para cada plano baseado em históricos, dados estatísticos e configuração de parâmetros, e executa o plano que possuir o menor tempo de execução estimado para realizar a consulta [Selinger et al., 1979]. Com a evolução dos sistemas de armazenamento de dados, o crescente aumento de utilização das *user-defined functions* (UDFs) e a maior complexidade dos dados, o cálculo das estimativas de desempenho foi ficando cada vez mais complexo e impreciso, levando a um desempenho ruim dos planos de execução [Babu et al., 2005]. Diante disso, novas

formas de otimização, mudanças na configuração de parâmetros, atualização de softwares e mudanças no hardware estão entre os vários fatores que enfatizam a necessidade de um ajuste contínuo em SGBDs. Portanto, existe um grande esforço da ciência em fornecer aos usuários de SGBDs as ferramentas para ajustar de forma correta e eficiente um banco de dados. Este ajuste abrange uma grande área de pesquisa que envolve problemas como monitoramento de desempenho [Agrawal et al., 2005] [Dageville et al., 2004] e gerenciamento de estatísticas [Kabra and DeWitt, 1998] [Babu et al., 2005], entre outros. Apesar destas técnicas de otimização da execução de consultas conseguirem solucionar algumas limitações e problemas do *Query Optimizer*, elas não lidam com a necessidade de análise de dados não-estruturados e semi-estruturados, e também com o aumento do uso e complexidade das UDFs em *workflows* científicos.

Existem diversos outros estudos e pesquisas por parte da comunidade científica relacionadas à otimização de desempenho no *framework* MR utilizando conceitos e técnicas aplicadas em SGBDs [Dittrich et al., 2010] [Jahani et al., 2011] [Jiang et al., 2010] [Khoussainova et al., 2012]. MANIMAL [Cafarella and Ré, 2010] é um sistema que procura realizar o processamento de consultas de forma eficiente em *frameworks* MR e utiliza técnicas de análise de programas sobre as UDFs para apontar oportunidades de otimização. Para tratar a diferença de desempenho entre SGBDs paralelos e MR, o Hadoop++ [Dittrich et al., 2010] realiza a inserção de otimizações sobre uma UDF a fim de tornar explícito o processamento de consulta e apresentá-la como um plano de execução de consultas físico. SUDO [Zhang et al., 2012] realiza uma análise das UDFs para identificar propriedades funcionais afim de otimizar o *shuffling* de dados em *frameworks* MR. PerfXplain [Khoussainova et al., 2012] fornece uma ferramenta para os usuários realizarem a otimização de desempenho do MR. Esta ferramenta gera automaticamente uma descrição da execução da consulta, que pode ser usada para ajudar a identificar as razões de uma execução ineficiente ou inesperada. Porém, este trabalho não fornece uma compreensão clara sobre como os parâmetros de configuração devem ser usados. Em [Jiang et al., 2010] é realizado um estudo do desempenho do MR, apontando os fatores de impacto como E/S, indexação, decodificação dos dados e escalonamento em um contexto de banco de dados.

Ainda, no contexto MR existe um crescente esforço em encontrar oportunidades de otimização nos diferentes níveis de atuação do MapReduce. Por exemplo, em um nível mais alto do MR, relacionado a semântica da linguagem, diversas oportunidades inspiradas na otimização de consultas em banco de dados têm sido propostas. Hive [Thusoo et al., 2010] e Pig [Gates et al., 2009][Olston et al., 2008] empregam abordagens baseadas em regras para uma variedade de otimizações como filtros, buscas compartilhadas dos dados de en-

trada [Nykiel et al., 2010], antecipação da projeção e redução do número de tarefas MR em um *workflow* [Zhang et al., 2011]. Em sistemas MR, os usuários especificam o processamento sobre os dados através das funções Map e Reduce. Neste nível, relacionado à execução de *workflows* MR, existem várias tarefas MR. Uma tarefa MR contém funções *black-box map* e *reduce* implementadas em linguagens como Java, Python ou R. Vários usuários do MR, como Facebook e Yahoo! [Olston et al., 2008], têm observado que tarefas MR geralmente possuem funções *black-box* que implementam algoritmos complexos de aprendizado estatísticos a partir de dados não-estruturados. Uma das técnicas de otimização propostas neste nível, implementada por HadoopToSQL [Iu and Zwaenepoel, 2010] e Manimal [Cafarella and Ré, 2010], realiza uma análise do código dos programas MapReduce para extrair partes declarativas como filtros e projeções. Nestas partes podem então ser usadas otimizações de banco de dados tais como antecipação de projeções, compressão baseada em colunas e uso de índices. Apesar disso, o modelo MR ainda tem muito a avançar no que se refere a otimização, que têm sido uma característica chave para o histórico de sucesso dos SGBDs.

7.1.2 Otimização de *Workflows* MR Intensivos de Dados em Diferentes Ambientes de Execução

Tradicionalmente, os *clusters* HPC (*High Performance Computing*) possuem como característica separar os recursos de processamento daqueles que realizam o armazenamento dos dados, utilizando a rede para comunicação e transferência de dados. Com isso, nesta arquitetura distribuída as decisões de escalonamento são baseadas somente na disponibilidade dos recursos computacionais. Por outro lado, o paradigma *MapReduce* possui uma abordagem centralizada nos dados através do conceito de "levar o processamento aos dados". Neste modelo, parte-se do princípio que cada recurso de processamento também possui uma grande disponibilidade de armazenamento local dos dados, e com isso cada recurso realiza tanto o processamento quanto o armazenamento de dados, tornando possível trazer a computação aos dados e consequentemente o que denomina-se localidade de dados.

A localidade de dados se torna ainda mais relevante quando a aplicação realiza o processamento de um grande volume de dados, pois há um maior impacto da redução tanto da transferência de dados entre os recursos quanto do tráfego de dados na rede, consequentemente, maximizando o desempenho da aplicação. Neste contexto, em [Ranganathan and Foster, 2002] é apresentado que a localidade de dados pode favorecer o

processamento de aplicações intensivas de dados em ambientes de *grids* computacionais. Para isto, considerou-se a localidade dos dados no escalonamento de tarefas e foi realizada a replicação dos dados que são mais acessados pelos *sites* do *grid*.

Purlieus [Palanisamy et al., 2011] é um sistema de alocação inteligente de máquinas virtuais (MVs) que executam as tarefas *MapReduce* em nuvens computacionais [Foster et al., 2008]. Este sistema ataca o problema da localidade de dados destacando que a execução de aplicações intensivas de dados na nuvem causam uma sobrecarga na rede de comunicação. Consequentemente, isto impacta tanto a aplicação em questão quanto outras aplicações que compartilham o ambiente de nuvem. Para isso, este sistema instancia de forma inteligente as MVs para obter a melhor localidade possível para as tarefas, acarretando em uma menor transferência de dados. As tarefas *MapReduce* são classificadas em três classes que são: *map-input heavy*, *map-and-reduce-input heavy* e *reduce-input heavy*. Para cada classe são propostas diferentes técnicas de alocação de dados e instanciação das MVs. Ainda no Purlieus este processo de alocação de MVs é realizado somente para tarefas *MapReduce* tradicionais. Na execução de um *workflow* com várias atividades *MapReduce* deve-se reiniciar as máquinas virtuais para cada atividade *MapReduce*, causando um *overhead* no desempenho da aplicação.

Em [Wasi-ur Rahman et al., 2015] é proposta a integração entre as duas arquiteturas, *MapReduce* e HPC, com o objetivo de utilizar as vantagens da rede de alto de desempenho e sistemas de arquivos paralelos, como Lustre [Lustre, 2002], fornecidos pelo *cluster* HPC em conjunto com a facilidade de analisar grandes volumes de dados fornecidos pelo paradigma de processamento *MapReduce*. Para tal, utiliza-se o Lustre ao invés do HDFS para realizar o armazenamento dos dados intermediários gerados entre as etapas *Map* e *Reduce* (no Hadoop, esta etapa é denominada de *shuffle*). Com isso, avalia-se duas estratégias diferentes de execução do *shuffle*. Verificou-se que dependendo do ambiente HPC é alterado a estratégia que fornece o melhor desempenho, portanto foi criado um mecanismo que detecta dinamicamente qual a melhor estratégia de *shuffle* para um determinado ambiente HPC.

Em [Zaharia et al., 2010a] a técnica *delay scheduling* foi utilizada para melhorar a localidade de dados no *MapReduce*. Neste trabalho foi projetado o HFS (*Hadoop Fair Scheduler*) que considera tanto a localidade quanto o *fair sharing*. Se uma tarefa não pode ser escalonada em um nó que tenha os dados, o escalonamento é adiado por um tempo. Este algoritmo possui um bom desempenho quando várias tarefas podem ser concluídas em um curto período de tempo. Uma peculiaridade deste algoritmo é a configuração

do tempo de adiamento (*delay time*). Se este adiamento é muito pequeno, não haverá nenhuma tarefa local a ser atribuída muito rápido. Por outro lado, se o adiamento for muito grande, pode acontecer a estagnação da tarefa e afetar o desempenho do sistema. Para obter a melhor localidade de dados, deve-se observar atentamente a configuração do tempo de adiamento neste algoritmo.

7.1.3 Considerações sobre as Abordagens Existentes para Execução de Aplicações Intensivas de Dados

Semelhante ao SGBD, existem diversas opções que podem ser tomadas na configuração dos parâmetros que podem interferir no desempenho de um *workflow* MR. Porém, sistemas MR não possuem um otimizador que realiza as configurações de forma automática e a escolha dos valores destes parâmetros afim de obter um bom desempenho não é um problema trivial para os usuários [Babu, 2010]. Com isso, nesta tese o alvo é a construção de uma metodologia de otimização da execução de *workflows* MR baseada na coleta das informações do tempo de execução. Para tal, nossa metodologia realiza a predição de desempenho em um contexto de *workflows*, dados, recursos computacionais e configuração de parâmetros.

O problema de otimização de *workflows* intensivos de dados tem sido objeto de estudo pelos cientistas e pesquisadores. Novos desafios têm surgido que envolvem o grande volume de dados a ser processado, mas também novos paradigmas centralizados nos dados, como MapReduce, foram desenvolvidos com o objetivo de tratar estas questões. Neste contexto, destaca-se que o paradigma MapReduce implementa o conceito de localidade de dados, evitando transferência dos mesmos pela rede. Além disso, é observado um constante aprimoramento no funcionamento do paradigma envolvendo questões como escalonamento e integração com sistemas HPC.

Portanto, no contexto de processamento utilizando o paradigma MR, nesta tese é utilizado o *framework* Spark descrito na Seção 2.3.2, uma vez que este *framework* realiza o processamento paralelo de *workflows* MR. Além disso, utiliza a memória principal para o armazenamento dos dados intermediários das atividades de um *workflow* com o objetivo de melhorar ainda mais a eficiência de processamento de *workflows* intensivos de dados.

Finalmente, as soluções descritas nesta Seção não realizam a otimização através do ajuste da configuração de parâmetros. Na Tabela 7.1 é apresentado um resumo das abordagens de otimização da execução de aplicações intensivas de dados. Basicamente as otimizações avaliadas são realizadas através do escalonamento das tarefas e localidade dos

dados. A técnica de escalonamento proposta por [Zaharia et al., 2010a] é implementada e utilizada por padrão pelo Spark, tratando a localidade dos dados de entrada, pois as tarefas geralmente são alocadas nos mesmos recursos onde estão armazenados os dados a serem processados. Isto diminui consideravelmente a transferência de dados pela rede. Adicionalmente, a localidade dos dados intermediários é tratada pelos RDDs do Spark, uma vez que no Spark os dados são mantidos em memória nos RDDs. Isto elimina a necessidade de gravar os dados intermediários em disco. Além disso, a transmissão dos dados pela rede só é necessária quando são realizadas operações de junção e/ou redução. Diante disso são complementares à abordagem proposta nesta tese.

Abordagem	Ambiente	Sistema MR	Método de Otimização
Purlieus	*Nuvem*	Hadoop	Localidade de dados
[Wasi-ur Rahman et al., 2015]	*Cluster*	Hadoop/HPC	Armaz. dos Dados Interm.
[Zaharia et al., 2010a]	*Cluster*	Hadoop	Escalonamento
SPACE	*Cluster*	Spark	Conf. de Parâmetros

Tabela 7.1: Quadro comparativo entre as principais abordagens de otimização da Execução de Aplicações Intensivas de Dados

7.2 Abordagens Existentes para Otimização de *Workflows* MR através da Configuração de Parâmetros

Otimização de desempenho em sistemas MR que processam grandes volumes de dados tem sido um tema muito pesquisado atualmente devido à grande utilização desses sistemas. Porém, relacionada à configuração de parâmetros, foi possível verificar que a maioria das pesquisas são baseadas no sistema Hadoop, e há poucas pesquisas relacionadas à configuração de parâmetros em sistemas Spark. Portanto, na Subseção 7.2.1 são apresentados alguns trabalhos que realizam otimizações em sistemas Hadoop. Na Subseção 7.2.2 são apresentadas otimizações realizadas em sistemas Spark. Na Subseção 7.2.3 são realizados os apontamentos relevantes dos trabalhos descritos nesta Seção para a tese.

7.2.1 Hadoop

Recentemente, existem diversos trabalhos relacionados à otimização através do ajuste da configuração de parâmetros no sistema Hadoop. Starfish [Herodotou et al., 2011a] [Herodotou and Babu, 2011] [Herodotou et al., 2011b] é um sistema que implementa uma abordagem de otimização baseada em custo para ajudar os usuários a identificar bons

valores para a configuração dos parâmetros para execução de aplicações MR. O sistema consiste de três módulos sendo um módulo, denominado *profiler*, que obtém estatísticas de execução do *workflow* e a estimativa de custo. Um segundo módulo, chamado *what-if*, determina o impacto de uma configuração de parâmetros no desempenho baseado nas informações obtidas no primeiro módulo. Por último, o módulo *optimizer* fornece as configurações que realizam uma execução eficiente através de consultas ao módulo *what-if*. A efetividade desta abordagem depende da precisão do módulo *what-if* que usa um modelo baseado nas estimativas obtidas e em simulações.

[Yigitbasi et al., 2013] é adotada uma abordagem baseada em aprendizado de máquina para construir o modelo preditivo de configuração dos parâmetros no Hadoop. Os autores mostram que o modelo *Support Vector Regression* (SVR) realiza uma predição precisa e eficiente. Os resultados demonstraram que essa abordagem pode fornecer um desempenho igual ou melhor que o Starfish com um menor número de parâmetros.

Gunther [Liao et al., 2013] é uma outra abordagem que usa algoritmos genéticos para identificar configurações de parâmetros que realizam uma execução eficiente. Seleciona uma configuração por teste e executa 20-40 testes.

AROMA [Lama and Zhou, 2012] possui o objetivo de automatizar a alocação de recursos e configuração das tarefas para nuvens heterogêneas atendendo os requisitos definidos em um SLA enquanto minimiza o custo monetário. AROMA utiliza uma abordagem baseada em aprendizado de máquina e dividida em duas fases. A primeira fase classifica as tarefas executadas através de algoritmos de clusterização usando as informações relacionadas à CPU, rede e utilização de disco. A segunda fase obtém os dados de utilização dos recursos enquanto as aplicações são executadas. Finalmente, AROMA encontra uma alocação de recursos e configuração de parâmetros baseada no método de otimização por padrões correspondentes.

PREDIcT [Popescu et al., 2013] é um outro sistema que realiza a predição do tempo de execução para algoritmos iterativos intensivos em E/S implementados no sistema Hadoop. Especificamente, o PREDIcT busca prever o número de iterações e o tempo de execução para cada iteração baseado em dados de execuções anteriores. Para alcançar um resultado preditivo satisfatório é necessário um conjunto de dados de treinamento extenso e pode levar a uma predição ruim para aplicações quando não existe o histórico de dados.

7.2.2 Spark

Em [Wang and Khan, 2015] os autores apresentam um modelo de predição obtido através de simulações que pode prever com alta precisão o desempenho de uma aplicação em sistemas Spark. O modelo é usado para predizer o tempo de execução e uso de memória das aplicações Spark com a configuração de parâmetros *default*. Porém, o modelo é muito simples para realizar a predição do tempo de execução utilizando diferentes configurações de parâmetros.

Em [Wang et al., 2016] é proposto um método para otimização de aplicações Spark através da configuração dos parâmetros do Spark baseado em algoritmos e técnicas de aprendizado de máquina. O método proposto estabelece um modelo baseado em classificação binária e multi-classificação. Diversos algoritmos de classificação são explorados afim de obter um modelo preditivo, no qual é possível identificar uma configuração de parâmetros que leva a uma execução eficiente da aplicação Spark. Portanto, o método proposto por [Wang et al., 2016] é baseado nos dados de treinamento obtidos de execuções anteriores para extrair as regras de desempenho, o que o torna mais robusto e flexível.

7.2.3 Considerações sobre as Abordagens Existentes para Otimização de *Workflows* MR através da Configuração de Parâmetros

O método baseado em custo, implementado pelo Starfish [Herodotou et al., 2011a], é uma abordagem onde é necessário um conhecimento aprofundado tanto do funcionamento interno do sistema como também do *workflow*. Este desafio tem motivado a exploração de métodos de desempenho baseados em técnicas de aprendizado de máquina [Wang et al., 2016]. Comparando com as abordagens existentes, métodos de aprendizado de máquina possuem duas vantagens principais. Primeiro, as recomendações para um auto-ajuste dos parâmetros são baseadas nas observações reais de desempenho do sistema na execução de um determinado *workflow* e um determinado *cluster*. Segundo, são mais simples de construir pois não é necessário nenhuma informação detalhada sobre as características internas do sistema e do *workflow*. Com isso, na abordagem baseada em aprendizado de máquina qualquer mudança nas características internas do sistema não alteram as recomendações para o auto-ajuste dos parâmetros.

Apesar de existirem algumas soluções para obtenção de uma configuração dos parâmetros que levem a uma execução paralela eficiente de *workflows* em *frameworks* MR

através dos métodos de aprendizado de máquina [Wang et al., 2016], nenhuma delas leva em consideração todas estas características importantes apresentadas nesta tese. Por exemplo, algumas soluções não abordam *workflows* intensivos de dados e consequentemente os aspectos relacionados ao particionamento dos mesmos como acontece com o Starfish [Herodotou et al., 2011b], o Stubby [Lim et al., 2012] e [Wang et al., 2016].

Em [Wang et al., 2016] são avaliadas quatro aplicações obtidas a partir do BigDataBench, que são: *Sort*, *Wordcount*, *Grep* e *NavieBayes*. O BigDataBench é um benchmark com 14 *datasets* reais e 34 aplicações BigData de código aberto desenvolvido pelo Instituto de Tecnologia Computacional, Academia de Ciências da China [Gao et al., 2013]. Porém as quatro aplicações avaliadas não são intensivas de dados, sendo: I/O intensive (*Sort*), CPU intensive (*Wordcount*), memory-intensive (*Grep*) e iterative-intensive (*NavieBayes*). Com isso, os parâmetros relacionados ao particionamento de dados avaliados nesta tese não fazem parte dos parâmetros avaliados em [Wang et al., 2016]. Isto torna difícil a comparação entre as duas abordagens pois diferentes classes de aplicações são avaliadas e as abordagens de otimização propostas levam em consideração parâmetros relacionados as respectivas classes de aplicações.

Adicionalmente, em [Wang et al., 2016] as execuções com a configuração de parâmetros *default* do *framework* MR são usadas como parte da abordagem de otimização e também são usadas na comparação com as configurações de parâmetros otimizadas. Isto difere, mais uma vez, da abordagem proposta nesta tese uma vez que não é possível executar com a configuração de parâmetros *default* os *workflows* intensivos de dados da astronomia tratados neste trabalho.

Capítulo 8

Conclusões

Este capítulo apresenta as conclusões gerais desta tese bem como suas principais contribuições e limitações. Finalizamos discutindo sobre as principais perspectivas de trabalhos futuros que podem ser desenvolvidos como consequência direta desta tese. A abordagem aqui apresentada se baseou na abordagem adotada em [Wang et al., 2016] e considerando também a estratégia de armazenamento e particionamento dos dados de entrada do *workflow*. Desta forma, fomos capazes de propor uma abordagem que tratasse o problema relacionado à configuração de parâmetros de execução de *workflows* científicos MR em *clusters*, considerando as características do ambiente, *framework* MR e do particionamento dos dados de entrada.

Geralmente, um *workflow* científico MR ao ser executado em paralelo demanda o processamento de grandes volumes de dados. Uma estratégia comum para alcançar o paralelismo é dividir as atividades do *workflow* em tarefas que possam ser executadas em paralelo. Em *frameworks* MR, cada tarefa do *workflow* ao ser executada em paralelo recebe uma parte dos dados, ou partição, a ser processada. Diante disso, a estratégia de particionamento dos dados e o tamanho das partições são aspectos fundamentais para o desempenho das tarefas dos *workflows* científicos.

Adicionalmente, ao executar um *workflow* científico MR, uma configuração de parâmetros do *framework* relacionados a CPU e memória deve ser especificada. Uma estratégia é utilizar a configuração "default" definida pelo administrador do sistema. Porém, a configuração "default" pode não levar a uma execução eficiente, ou até mesmo levar a um erro durante a execução do *workflow*. Portanto, a configuração dos parâmetros de execução deve ser especificada para cada *workflow* científico, levando em consideração as características dos dados de entrada e do ambiente de execução. Porém a configuração de parâmetros não é uma tarefa trivial de ser realizada. Esta tarefa se torna complexa

principalmente devido a dois fatores. O primeiro diz respeito à variedade de parâmetros que podem ser configurados na execução de um *workflow* MR. O segundo fator diz respeito à diversidade de valores que os parâmetros podem assumir. Cada configuração de parâmetros e valores leva a um tempo de execução diferente do *workflow* científico, exigindo assim, uma solução que permita ao cientista conhecer quais parâmetros e valores levam a uma execução eficiente de um determinado *workflow*.

A abordagem proposta para a otimização de desempenho de *workflows* científicos MR em *clusters* é inspirada nas técnicas de aprendizado de máquina [Weiss and Kulikowski, 1991] [Li and Wang, 2002] [Breiman et al., 1984] [Cherkassky and Mulier, 1998] [Duda et al., 2000], e apresenta uma série de etapas que englobam o treinamento, discretização, classificação e validação do modelo preditivo. A abordagem, portanto, possibilita um conhecimento dos valores e relações dos parâmetros do *framework* MR, e também dos valores e parâmetros relacionados aos dados de entrada, que levam a uma execução eficiente de um determinado *workflow* científico em um ambiente de *cluster*.

Diante disso, nessa tese, foi investigada a otimização da execução de *workflows* científicos MR sobre um *cluster*. Foram avaliados dois *workflows* científicos reais da astronomia, SkyMap e Constellation queries, e executamos um conjunto de experimentos extensivos afim de identificar as principais oportunidades de otimização de desempenho. O objetivo é fornecer aos cientistas um método que pode ser aplicado para levar a uma execução eficiente sem que estes necessitem obter conhecimento aprofundado dos parâmetros de configuração do *framework* MR. Neste contexto, esta tese foca na especificação do: (i) número de *cores*, (ii) quantidade de memória alocada; e (iii) estratégia de particionamento dos dados.

A abordagem proposta realiza o treinamento da execução do *workflow* a ser otimizado e adota o método árvore de decisão que gera o modelo preditivo usado para conhecer os valores e parâmetros que levam a uma execução eficiente do *workflow* em um ambiente de *cluster*. Por exemplo, a abordagem proposta aponta que a execução do Cenário 1 é principalmente impactado pela estratégia de particionamento. Portanto, neste cenário, a árvore de decisão reflete a importância de usar a estratégia de particionamento dos dados *equi-depth* para alcançar uma execução eficiente. Adicionalmente, aponta que partições menores beneficiam o tempo de execução das tarefas. Neste cenário, o tempo de execução de uma configuração classificada como *Baixo* é $4x$ mais rápida do que uma configuração de parâmetros classificada como *Alto*, ambas executando o mesmo *workflow* e conjunto de dados sobre o mesmo *cluster*. Ainda, todos os demais cenários avaliados mostraram

resultados similares usando a abordagem proposta, com a métrica de avaliação do modelo ($F1$) acima de 70%, indicando que o modelo realiza uma boa predição. Finalmente, a abordagem proposta pode capturar variações no tamanho do conjunto de dados de entrada do *workflow*, uma vez que este tamanho é parte do conjunto de parâmetros considerado pelo modelo.

Além disso, em casos específicos, os experimentos demonstraram que um modelo preditivo pode ser aplicado a mais de um cenário, com uma boa probabilidade de sucesso. Porém, em outros casos, devido à grande diferença entre dois ambientes de execução foi verificado que não é possível gerar um modelo preditivo baseado em um cenário que atenda satisfatoriamente a predição para todos os outros cenários possíveis, por que nestes casos a precisão da predição não alcança os limites mínimos necessários. Nestes cenários, dever ser realizado um novo processo de treinamento envolvendo o *workflow*, dados e o ambiente de execução. Apesar o custo inicial deste treinamento, o modelo preditivo obtido pode ser usado em diversas execuções futuras do *workflow*.

Neste contexto, a abordagem apresentada nesta tese possibilita a otimização de desempenho de *workflows* científicos Spark executado em *clusters* computacionais por meio de algoritmos de aprendizado de máquina, permitindo ao cientista conhecer os parâmetros e valores, relacionados ao Spark e aos dados, que levam a uma execução eficiente do *workflow*. Nesta tese foi especificado um modelo de custo baseado no tempo de execução do *workflow* de forma que cada configuração de parâmetros seja avaliada de acordo com este modelo pelo cientista.

8.1 Contribuições

Esta tese apresentou os resultados de uma pesquisa desenvolvida ao longo dos cinco últimos anos e suas principais contribuições são:

i. A definição de uma abordagem de otimização que pode ser aplicada no ajuste da configuração dos parâmetros de execução de *workflows* Spark em *clusters* computacionais;

ii. A especificação e o desenvolvimento de um protótipo (SPACE) que implementa a abordagem de otimização proposta;

iii. A avaliação desta abordagem, por meio da execução de dois *workflows* reais da astronomia em dois *clusters* computacionais reais, que mostrou a capacidade da abordagem

proposta de obter configurações de parâmetros que levam a uma execução eficiente dos *workflows*;

Os resultados experimentais demonstraram que é possível construir um modelo de execução eficiente para *workflows* MR utilizando métodos de "caixa preta", ou seja, usando somente as configurações do sistema e sem alterar a implementação do *workflow*.

Esta tese se baseou em duas publicações associadas à execução de *workflows* científicos MR executados em *clusters* e realizadas durante o doutorado (ordenadas cronologicamente):

i. de Oliveira, D. E. M.; Boeres, C.; Porto, F. Análise de estratégias de acesso a grandes volumes de dados. In XXIX Simpósio Brasileiro de Banco de Dados, SBBD 2014, Curitiba, Paraná, Brasil, October 6-9, 2014. (2014), pp. 27–36.

ii. de Oliveira, D. E. M.; Boeres, C.; Fausti, A.; Porto, F. Avaliação da localidade de dados intermediários na execução paralela de workflows bigdata. In XXX Simpósio Brasileiro de Banco de Dados, Petrópolis, Rio de Janeiro, Brasil, October 13-16, 2015. (2015), pp. 29–40.

8.2 Limitações

Algumas limitações foram identificadas, principalmente relacionadas à abordagem de otimização proposta:

i. Os *workflows* científicos utilizados, como estudo de caso, na avaliação da abordagem possuem somente operações *map* e *reduce*. Idealmente, deve ocorrer uma avaliação com *workflows* que possuem uma maior variedade de operações, como por exemplo: *join* e *filter*;

ii. Além da limitação de operações, os *workflows* avaliados são classificados como *pipeline*, que é a representação utilizada por muitos *workflows* científicos. Porém, existem vários *workflows* científicos que são classificados como iterativos. Portanto, o ideal seria validar a abordagem nesta importante classe de *workflows* científicos;

iii. Quanto à abordagem, os dados de treinamento são coletados separadamente para cada aplicação e *cluster*, e consequentemente, este processo de coleta pode levar um tempo relativamente alto. Porém, em uma aplicação real, espera-se que este processo

de treinamento seja realizado somente uma vez em um momento inicial, e as aplicações executadas diversas vezes pelo cientista beneficiando-se das análises obtidas com os dados coletados. Os dados de entrada do *workflow* podem ser frequentemente modificados e atualizados, e o modelo de desempenho proposto pode capturar estas mudanças uma vez que o tamanho dos dados de entrada faz parte do conjunto de parâmetros modelado. Além disso, foram realizados experimentos que demonstraram que o modelo preditivo gerado para um cenário alcança uma boa precisão e desempenho na predição dos valores de configuração dos parâmetros de outros cenários, e com isso não seria necessário realizar a etapa de treinamento destes últimos.

Apesar das limitações apresentadas, a abordagem apresentada nesta tese dá um passo importante em direção à otimização da execução de *workflows* científicos MR em um ambiente de *cluster*.

8.3 Trabalhos Futuros

Alcançar uma configuração de parâmetros que leve a uma execução eficiente de *workflows* científicos MR em um *cluster* computacional é uma tarefa complexa e que demanda tempo. A abordagem proposta nesta tese representa apenas os passos iniciais para o desenvolvimento de uma abordagem definitiva para a otimização da execução paralela de *workflows*, através da configuração de parâmetros e do particionamento de dados. Com isso, a presente tese abriu uma linha de novas oportunidades de pesquisa. Considerando o estágio atual do desenvolvimento e as necessidades dos cientistas, algumas das perspectivas de trabalhos futuros são apresentadas a seguir.

A seleção de parâmetros para a construção um modelo de desempenho baseado em aprendizado de máquina não é uma questão simples mas extremamente importante para validação do resultado, e não pode depender somente do conhecimento obtido com experimentos e observações. Nesta tese foram selecionados um subconjunto de parâmetros básicos do Spark, relacionados somente a CPU e memória RAM. Portanto, deve ser realizada uma pesquisa sobre a seleção de parâmetros considerando outros aspectos que afetam o desempenho, como por exemplo o compartilhamento de recursos entre as tarefas e tráfego de dados na rede, afim de tornar o modelo preditivo mais robusto.

Quanto à geração do modelo preditivo, foi utilizado nesta tese e no SPACE o método de árvore de classificação. Porém existem outros métodos de classificação como *Naive Bayes* [Duda et al., 2001], *Random Forests* [Breiman, 2001], *Support Vector Ma-*

chine (SVM) [Hearst, 1998] e *k-Nearest Neighbor* (k-NN) [Cover and Hart, 2006]. Esses métodos representam diferentes paradigmas e são estratégias tradicionalmente indicadas por apresentar bom desempenho em diferentes domínios de aplicação [Lessmann et al., 2008]. Portanto, são métodos que podem ser aplicados afim de verificar o desempenho do modelo preditivo criado por eles.

Finalmente, de acordo com o conhecimento obtido durante a execução dos experimentos foi possível observar que as funções do *framework* MR que compõem o *workflow* podem ser alteradas de acordo com o particionamento dos dados, ou seja, quando é conhecido que os dados estão particionados de acordo com algum critério específico pode-se realizar algumas modificações no código do *workflow* para aproveitar isto. Diante dessas mudanças, a execução do *workflow* se torna ainda mais eficiente e otimizada. Esta é uma oportunidade de trabalho futuro envolvendo o otimizador Spark com o critério de particionamento de dados.

Referências

[Agrawal et al., 2005] Agrawal, S., Chaudhuri, S., Kollar, L., Marathe, A., Narasayya, V., and Syamala, M. (2005). Database tuning advisor for microsoft sql server 2005: Demo. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 930–932, New York, NY, USA. ACM.

[Altintas et al., 2004] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., and Mock, S. (2004). Kepler: Towards a Grid-Enabled System for Scientific Workflows. *the Workflow in Grid Systems Workshop in GGF10-The Tenth Global Grid Forum, Berlin, Germany, March.*

[Babu, 2010] Babu, S. (2010). Towards automatic optimization of mapreduce programs. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, pages 137–142, New York, NY, USA. ACM.

[Babu et al., 2005] Babu, S., Bizarro, P., and DeWitt, D. (2005). Proactive re-optimization. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 107–118, New York, NY, USA. ACM.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

[Breiman et al., 1984] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA.

[Cafarella and Ré, 2010] Cafarella, M. J. and Ré, C. (2010). Manimal: Relational optimization for data-intensive programs. In *Procceedings of the 13th International Workshop on the Web and Databases*, pages 10:1–10:6, New York, NY, USA. ACM.

[Cherkassky and Mulier, 1998] Cherkassky, V. S. and Mulier, F. (1998). *Learning from Data: Concepts, Theory, and Methods.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

[Cover and Hart, 2006] Cover, T. and Hart, P. (2006). Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27.

[Dageville et al., 2004] Dageville, B., Das, D., Dias, K., Yagoub, K., Zait, M., and Ziauddin, M. (2004). Automatic sql tuning in oracle 10g. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, pages 1098–1109. VLDB Endowment.

[Daniel Gaspar, 2014] Daniel Gaspar, F. P. (2014). A multi-dimensional equi-depth partitioning strategy for astronomy catalog data.

[Dantas, 2005] Dantas, M. (2005). *Computação distribuída de alto desempenho: redes, clusters e grids computacionais.* Axcel Books.

[de Oliveira, 2012] de Oliveira, D. C. M. (2012). *UMA ABORDAGEM DE APOIO A EXECUCAO PARALELA DE WORKFLOWS CIENTIFICOS EM NUVENS DE COMPUTADORES*. PhD thesis, Programa de Engenharia de Sistemas e Computacao, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

[de Oliveira et al., 2015] de Oliveira, D. E. M., Boeres, C., Fausti, A., and Porto, F. (2015). Avaliação da localidade de dados intermediários na execução paralela de workflows bigdata. In *XXX Simpósio Brasileiro de Banco de Dados*, pages 29–40.

[de Oliveira et al., 2014] de Oliveira, D. E. M., Boeres, C., and Porto, F. (2014). Análise de estratégias de acesso a grandes volumes de dados. In *XXIX Simpósio Brasileiro de Banco de Dados, SBBD 2014*, pages 27–36.

[Dean and Ghemawat, 2004] Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pages 10–10, Berkeley, CA, USA. USENIX Association.

[Deelman et al., 2009] Deelman, E., Gannon, D., Shields, M., and Taylor, I. (2009). Workflows and e-science: An overview of workflow system features and capabilities. *Future Gener. Comput. Syst.*, 25(5):528–540.

[Deelman et al., 2005] Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., Laity, A., Jacob, J. C., and Katz, D. S. (2005). Pegasus: A Framework for Mapping Complex Scientific Workflows Onto Distributed Systems. *Sci. Program.*, 13(3):219–237.

[Deelman et al., 2015] Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P. J., Mayani, R., Chen, W., Ferreira da Silva, R., Livny, M., and Wenger, K. (2015). Pegasus, a Workflow Management System for Science Automation. *Future Generation Computer Systems*, 46:17–35.

[Demsar et al., 2013] Demsar, J., Curk, T., Erjavec, A., Gorup, C., Hocevar, T., Milutinovic, M., Mozina, M., Polajnar, M., Toplak, M., Staric, A., Stajdohar, M., Umek, L., Zagar, L., Zbontar, J., Zitnik, M., and Zupan, B. (2013). Orange: Data mining toolbox in python. *Journal of Machine Learning Research*, 14:2349–2353.

[Dias et al., 2013] Dias, J., Ogasawara, E., de Oliveira, D., Porto, F., Valduriez, P., and Mattoso, M. (2013). Algebraic dataflows for big data analysis. In *Big Data, 2013 IEEE International Conference*, pages 150–155.

[Dittrich et al., 2010] Dittrich, J., Quiané-Ruiz, J.-A., Jindal, A., Kargin, Y., Setty, V., and Schad, J. (2010). Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing). *Proceedings of the Very Large Database Endowment*, 3(1-2):515–529.

[Duda et al., 2000] Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification (2Nd Edition)*. Wiley-Interscience.

[Duda et al., 2001] Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification (2nd Ed)*. Wiley.

[Fayyad and Irani, 1993] Fayyad, U. M. and Irani, K. B. (1993). Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In *13th International Joint Conference on Uncertainly in Artificial Intelligence*, pages 1022–1029.

[Foster and Kesselman, 2004] Foster, I. and Kesselman, C., editors (2004). *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Foster et al., 2008] Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop*, pages 1–10.

[FREIRE, 2016] FREIRE, V. P. D. M. (2016). *NACLUSTER: RESOLVENDO ENTIDADES EM LARGA ESCALA A PARTIR DE MÚLTIPLOS CATÁLOGOS DE ASTRONOMIA.* PhD thesis, Tese (Ciência da Computação) - Universidade Federal do Ceará, Departamento de Computação, Fortaleza, CE, Brasil.

[Gao et al., 2013] Gao, W., Zhu, Y., Jia, Z., Luo, C., Wang, L., Li, Z., Zhan, J., Qi, Y., He, Y., Gong, S., Li, X., Zhang, S., and Qiu, B. (2013). Bigdatabench: a big data benchmark suite from web search engines. *Computing Research Repository, CoRR.*

[Garcia et al., 2013] Garcia, S., Luengo, J., Saez, J. A., Lopez, V., and Herrera, F. (2013). A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Trans. on Knowl. and Data Eng.*, 25(4):734–750.

[Gates et al., 2009] Gates, A. F., Natkovich, O., Chopra, S., Kamath, P., Narayanamurthy, S. M., Olston, C., Reed, B., Srinivasan, S., and Srivastava, U. (2009). Building a High-Level Dataflow System on top of Map-Reduce: The Pig Experience. In *PROCEEDINGS OF THE VLDB ENDOWMENT*, pages 1414–1425.

[George, 2011] George, L. (2011). *HBase - The Definitive Guide: Random Access to Your Planet-Size Data.* O'Reilly.

[Ghemawat et al., 2003] Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The Google File System. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, New York, NY, USA. ACM.

[Gorton et al., 2008] Gorton, I., Greenfield, P., Szalay, A., and Williams, R. (2008). Data-intensive computing in the 21st century. *Computer*, 41(4):30–32.

[Gray et al., 2005] Gray, J., Liu, D. T., Nieto-Santisteban, M., Szalay, A., DeWitt, D. J., and Heber, G. (2005). Scientific data management in the coming decade. *SIGMOD Record*, 34(4):34–41.

[Greenplum, 2012] Greenplum (2012). Greenplum, http://www.greenplum.com.

[Gu and Li, 2013] Gu, L. and Li, H. (2013). Memory or Time: Performance Evaluation for Iterative Operation on Hadoop and Spark. In *10th IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, Zhangjiajie, China, November 13-15*, pages 721–727.

[Gu and Grossman, 2009] Gu, Y. and Grossman, R. L. (2009). Sector and Sphere: the design and implementation of a high-performance data cloud. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 367(1897):2429–2445.

[Hadoop,] Hadoop, A. http://hadoop.apache.org/.

[Han et al., 2011] Han, J., Kamber, M., and Pei, J. (2011). *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition.

[HDFS, 2012] HDFS (2012). http://hadoop.apache.org/docs/stable/hdfs-design.html.

[He et al., 2008] He, B., Fang, W., Luo, Q., Govindaraju, N. K., and Wang, T. (2008). Mars: A MapReduce Framework on Graphics Processors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pages 260–269, New York, NY, USA. ACM.

[Hearst, 1998] Hearst, M. A. (1998). Support vector machines. *IEEE Intelligent Systems*, 13(4):18–28.

[Herodotou, 2012] Herodotou, H. (2012). *Automatic Tuning of Data-Intensive Analytical Workloads*. PhD thesis, Duke University, Durham, NC, USA.

[Herodotou and Babu, 2011] Herodotou, H. and Babu, S. (2011). Profiling, what-if analysis, and cost-based optimization of mapreduce programs. *Proceedings of the Very Large Database Endowment*, 4(11):1111–1122.

[Herodotou et al., 2011a] Herodotou, H., Dong, F., and Babu, S. (2011a). Mapreduce programming and cost-based optimization? crossing this chasm with starfish. *Proceedings of the Very Large Database Endowment*.

[Herodotou et al., 2011b] Herodotou, H., Lim, H., Luo, G., Borisov, N., Dong, L., Cetin, F. B., and Babu, S. (2011b). Starfish: A self-tuning system for big data analytics. In *Conference on Innovative Data Systems Research*, pages 261–272. www.cidrdb.org.

[Hey et al., 2009] Hey, T., Tansley, S., and Tolle, K., editors (2009). *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Redmond, Washington.

[Hindman et al., 2011] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., Shenker, S., and Stoica, I. (2011). Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, pages 295–308, Berkeley, CA, USA. USENIX Association.

[Isard et al., 2007] Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. (2007). Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 59–72, New York, NY, USA. ACM.

[Iu and Zwaenepoel, 2010] Iu, M.-Y. and Zwaenepoel, W. (2010). Hadooptosql: a mapreduce query optimizer. In Morin, C. and Muller, G., editors, *EuroSys*, pages 251–264. ACM.

[Jahani et al., 2011] Jahani, E., Cafarella, M. J., and Ré, C. (2011). Automatic optimization for mapreduce programs. *Proceedings of the Very Large Database Endowment*, 4(6):385–396.

[Jiang et al., 2010] Jiang, D., Ooi, B. C., Shi, L., and Wu, S. (2010). The performance of mapreduce: An in-depth study. *Proceedings of the Very Large Database Endowment*, 3(1-2):472–483.

[Kabra and DeWitt, 1998] Kabra, N. and DeWitt, D. J. (1998). Efficient mid-query re-optimization of sub-optimal query execution plans. *SIGMOD Record*, 27(2):106–117.

[Kerber, 1992] Kerber, R. (1992). Chimerge: Discretization of numeric attributes. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 123–128. AAAI Press.

[Khatibi et al., 2017] Khatibi, A., Porto, F., Rittmeyer, J. G., Ogasawara, E., Valduriez, P., and Shasha, D. (2017). *Pre-processing and Indexing Techniques for Constellation Queries in Big Data*, pages 164–172. Springer International Publishing, Cham.

[Khoussainova et al., 2012] Khoussainova, N., Balazinska, M., and Suciu, D. (2012). Perfxplain: Debugging mapreduce job performance. *Proceedings of the Very Large Database Endowment*, 5(7):598–609.

[Lama and Zhou, 2012] Lama, P. and Zhou, X. (2012). Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In *Proceedings of the 9th International Conference on Autonomic Computing*, pages 63–72, New York, NY, USA. ACM.

[Lessmann et al., 2008] Lessmann, S., Baesens, B., Mues, C., and Pietsch, S. (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Trans. Softw. Eng.*, 34(4):485–496.

[Li and Wang, 2002] Li, R.-P. and Wang, Z.-O. (2002). An entropy-based discretization method for classification rules with inconsistency checking. In *Proceedings. International Conference on Machine Learning and Cybernetics*, volume 1, pages 243–246 vol.1.

[Liao et al., 2013] Liao, G., Datta, K., and Willke, T. L. (2013). Gunther: Search-based auto-tuning of mapreduce. In Wolf, F., Mohr, B., and an Mey, D., editors, *Euro-Par*, volume 8097 of *Lecture Notes in Computer Science*, pages 406–419. Springer.

[Lim et al., 2012] Lim, H., Herodotou, H., and Babu, S. (2012). Stubby: A Transformation-based Optimizer for MapReduce Workflows. *Proceedings of the Very Large Database Endowment*, 5(11):1196–1207.

[Lipcon, 2009] Lipcon, T. (2009). Cloudera:7 tips for improving mapreduce performance, http://blog.cloudera.com/blog/2009/12/7-tips-for-improving-mapreduce-performance.

[Liu et al., 2015] Liu, J., Pacitti, E., Valduriez, P., and Mattoso, M. (2015). A Survey of Data-Intensive Scientific Workflow Management. *Journal of Grid Computing*, pages 1–37.

[Lustre, 2002] Lustre (2002). http://wiki.lustre.org/index.php/main_page.

[Mitchell, 1982] Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18(2):203–226.

[Monard and Baranauskas, 2003] Monard, M. C. and Baranauskas, J. A. (2003). Conceitos sobre aprendizado de máquina. In *Sistemas Inteligentes Fundamentos e Aplicações*, pages 89–114. Manole Ltda, Barueri-SP, 1 edition.

[Muralikrishna and DeWitt, 1988] Muralikrishna, M. and DeWitt, D. J. (1988). Equi-depth multidimensional histograms. *SIGMOD Record*, 17(3):28–36.

[Nykiel et al., 2010] Nykiel, T., Potamias, M., Mishra, C., Kollios, G., and Koudas, N. (2010). Mrshare: Sharing across multiple queries in mapreduce. *Proceedings of the Very Large Database Endowment*, 3(1-2):494–505.

[Ogasawara et al., 2011] Ogasawara, E. S., de Oliveira, D., Valduriez, P., Dias, J., Porto, F., and Mattoso, M. (2011). An algebraic approach for data-centric scientific workflows. *Proceedings of the Very Large Database Endowment*, 4(12):1328–1339.

[Oinn et al., 2004] Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M. R., Wipat, A., and Li, P. (2004). Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics (Oxford, England)*, 20(17):3045–54.

[Olston et al., 2008] Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. (2008). Pig Latin: A Not-so-foreign Language for Data Processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1099–1110, New York, NY, USA. ACM.

[Özsu and Valduriez, 2011] Özsu, M. T. and Valduriez, P., editors (2011). *Principles of Distributed Database Systems*. Springer, New York, 3 edition.

[Palanisamy et al., 2011] Palanisamy, B., Singh, A., Liu, L., and Jain, B. (2011). Purlieus: Locality-aware Resource Allocation for MapReduce in a Cloud. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 58:1–58:11, New York, NY, USA. ACM.

[Pautasso and Alonso, 2006] Pautasso, C. and Alonso, G. (2006). Parallel computing patterns for Grid workflows. In *Workflows in Support of Large-Scale Science, 2006. WORKS '06. Workshop on*, pages 1–10.

[Pig,] Pig. Pig: High-level dataflow system for hadoop http://pig.apache.org/.

[Popescu et al., 2013] Popescu, A. D., Balmin, A., Ercegovac, V., and Ailamaki, A. (2013). Predict: Towards predicting the runtime of large scale iterative analytics. *Proceedings of the Very Large Database Endowment*, 6(14):1678–1689.

[Porto et al., 2017] Porto, F., Khatibi, A., Nobre, J. R., Ogasawara, E. S., Valduriez, P., and Shasha, D. E. (2017). Constellation queries over big data. *CoRR*, abs/1703.02638.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Ranganathan and Foster, 2002] Ranganathan, K. and Foster, I. (2002). Decoupling computation and data scheduling in distributed data-intensive applications. In *Proceedings 11th IEEE International Symposium on High Performance Distributed Computing*, pages 352–358. IEEE Comput. Soc.

[Ranger et al., 2007] Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., and Kozyrakis, C. (2007). Evaluating MapReduce for Multi-core and Multiprocessor Systems. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pages 13–24, Washington, DC, USA. IEEE Computer Society.

[Romano, 2008] Romano, P. (2008). Automation of in-silico data analysis processes through workflow management systems. *Brief Bioinform*, 9(1):57–68.

[Ruggieri, 2004] Ruggieri, S. (2004). Yadt: yet another decision tree builder. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 260–265.

[Samet, 1984] Samet, H. (1984). The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260.

[Selinger et al., 1979] Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., and Price, T. G. (1979). Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, pages 23–34, New York, NY, USA. ACM.

[Shasha and Bonnet, 2002] Shasha, D. and Bonnet, P. (2002). Database Tuning: Principles, Experiments, and Troubleshooting Techniques. In *Proceedings of the Very Large Database Endowment*. Morgan Kaufmann.

[Spark,] Spark, A. http://spark.apache.org/.

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

[Storm,] Storm, A. http://storm.apache.org/.

[Teradata, 2012] Teradata (2012). Teradata, http://www.teradata.com.

[Thusoo et al., 2010] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., Antony, S., Liu, H., and Murthy, R. (2010). Hive - a petabyte scale data warehouse using Hadoop. In *International Conference on Data Engineering*, pages 996–1005. IEEE.

[Tutorial, 2016] Tutorial, H. (2016). Hadoop mapreduce tutorial, https://hadoop.apache.org/docs/stable.

[Tutorial, 2017] Tutorial, S. (2017). Spark tutorial, https://spark.apache.org/docs/latest/configuration.html.

[Vaquero et al., 2009] Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2009). A break in the clouds: towards a cloud definition. *SIGCOMM Computer Communication Review*, 39(1):50–55.

[Vavilapalli et al., 2013] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., O'Malley, O., Radia, S., Reed, B., and Baldeschwieler, E. (2013). Apache hadoop yarn: Yet another resource negotiator. In *Annual Symposium on Cloud Computing*, pages 5:1–5:16, New York, NY, USA. ACM.

[Wang et al., 2016] Wang, G., Xu, J., and He, B. (2016). A novel method for tuning configuration parameters of spark based on machine learning. In *IEEE International Conference on High Performance Computing and Communications; IEEE International Conference on Smart City; IEEE International Conference on Data Science and Systems, Sydney, Australia, December 12-14*, pages 586–593.

[Wang and Khan, 2015] Wang, K. and Khan, M. M. H. (2015). Performance prediction for apache spark platform. In *HPCC/CSS/ICESS*, pages 166–173. IEEE.

[Wasi-ur Rahman et al., 2015] Wasi-ur Rahman, M., Lu, X., Islam, N. S., Rajachandrasekar, R., and Panda, D. K. (2015). High-Performance Design of YARN MapReduce on Modern HPC Clusters with Lustre and RDMA. In *Parallel and Distributed Processing Symposium*, pages 291–300. IEEE.

[Weiss and Kulikowski, 1991] Weiss, S. M. and Kulikowski, C. A. (1991). *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[White, 2009] White, T. (2009). *Hadoop: The Definitive Guide.* O'Reilly Media, Inc., 1st edition.

[Yang et al., 2005] Yang, Y., Webb, G. I., and Wu, X. (2005). Discretization methods. In Maimon, O. and Rokach, L., editors, *The Data Mining and Knowledge Discovery Handbook*, pages 113–130. Springer.

[Yigitbasi et al., 2013] Yigitbasi, N., Willke, T. L., Liao, G., and Epema, D. (2013). Towards machine learning-based auto-tuning of mapreduce. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, IEEE 21st International Symposium*, pages 11–20. IEEE.

[Zaharia et al., 2010a] Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., and Stoica, I. (2010a). Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *European Conference on Computer Systems*, pages 265–278, New York, NY, USA. ACM.

[Zaharia et al., 2012] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, pages 2–2, Berkeley, CA, USA. USENIX Association.

[Zaharia et al., 2010b] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010b). Spark: Cluster Computing with Working Sets. In *USENIX Conference on Hot Topics in Cloud Computing*, pages 10–10, Berkeley, CA, USA. USENIX Association.

[Zäschke et al., 2014] Zäschke, T., Zimmerli, C., and Norrie, M. C. (2014). The ph-tree: A space-efficient storage structure and multi-dimensional index. In *ACM SIGMOD International Conference on Management of Data*, pages 397–408, New York, NY, USA. ACM.

[Zhang et al., 2012] Zhang, J., Zhou, H., Chen, R., Fan, X., Guo, Z., Lin, H., Li, J. Y., Lin, W., Zhou, J., and Zhou, L. (2012). Optimizing data shuffling in data-parallel computation by understanding user-defined functions. In *USENIX Symposium on Networked Systems Design and Implementation*, pages 295–308, San Jose, CA. USENIX.

[Zhang et al., 2011] Zhang, X., Lee, R., Huai, Y., He, Y., Luo, T., and Wang, F. (2011). Ysmart: Yet another sql-to-mapreduce translator. *International Conference on Distributed Computing Systems*, pages 25–36.

[Zhao et al., 2007] Zhao, Y., Hategan, M., Clifford, B., Foster, I. T., von Laszewski, G., Nefedova, V., Raicu, I., Stef-Praun, T., and Wilde, M. (2007). Swift: Fast, Reliable, Loosely Coupled Parallel Computation. In *IEEE Congress on Services*, pages 199–206. IEEE Computer Society.

[Zhao et al., 2008] Zhao, Y., Raicu, I., and Foster, I. T. (2008). Scientific Workflow Systems for 21st Century e-Science, New Bottle or New Wine? *CoRR*, abs/0808.3545.