UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

SÁVIO SALVARINO TELES DE OLIVEIRA

Explorando Paralelismo em Big Data no Processamento de Séries Temporais de Imagens de Sensoriamento Remoto

Goiânia
2019

sistema de bibliotecas ufg

PRPG
PRÓ-REITORIA DE PÓS-GRADUAÇÃO

UFG

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: **[ ] Dissertação** **[x] Tese**

2. Identificação da Tese ou Dissertação:

Nome completo do autor: Sávio Salvarino Teles de Oliveira

Título do trabalho: Explorando Paralelismo em Big Data no Processamento de Séries Temporais de Imagens de Sensoriamento Remoto

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [x] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF da tese ou dissertação.

Sávio Salvarino Teles de Oliveira

Assinatura do(a) autor(a)[2]

Ciente e de acordo:

Wagner Santos Martins

Assinatura do(a) orientador(a)[2] Data: 13 / 09 / 2019

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

[2] A assinatura deve ser escaneada.

SÁVIO SALVARINO TELES DE OLIVEIRA

Explorando Paralelismo em Big Data no Processamento de Séries Temporais de Imagens de Sensoriamento Remoto

Tese defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Doutor em Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Wellington Santos Martins

Co-Orientador: Prof. Dr. Vagner José do Sacramento Rodrigues

Goiânia
2019

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Salvarino Teles de Oliveira, Sávio
Explorando Paralelismo em Big Data no Processamento de Séries Temporais de Imagens de Sensoriamento Remoto [manuscrito] / Sávio Salvarino Teles de Oliveira. - 2019.
128 f.

Orientador: Prof. Dr. Wellington Santos Martins; co-orientador Dr. Vagner José do Sacramento Rodrigues.
Tese (Doutorado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação em rede (UFG/UFMS), Goiânia, 2019.
Bibliografia.
Inclui abreviaturas, gráfico, tabelas, algoritmos, lista de figuras, lista de tabelas.

1. Análise e Classificação de séries temporais. 2. Sensoriamento Remoto. 3. Sistemas Distribuídos. 4. Processamento Paralelo. 5. Big Data. I. Santos Martins, Wellington, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

ATA DE DEFESA DE TESE

Ata Nº **02/2019** da sessão de Defesa de Tese de **Sávio Salvarino Teles de Oliveira** que confere o título de Doutor em Ciência da Computação, na área de concentração em Ciência da Computação.

Aos trinta dias do mês de agosto de dois mil e dezenove, a partir das nove horas e trinta minutos, na sala 151 do Instituto de Informática, realizou-se a sessão pública de Defesa de Tese intitulada "**Explorando Paralelismo em Big Data no Processamento de Séries Temporais de Imagens de Sensoriamento Remoto**". Os trabalhos foram instalados pelo Orientador, Professor Doutor Wellington Santos Martins (INF/UFG) com a participação dos demais membros da Banca Examinadora: Professor Doutor Clodoveu Augusto Davis Junior (DCC - UFMG), membro titular externo, cuja participação ocorreu através de videoconferência; Professor Doutor Nilton Correia da Silva (FGA - UnB), membro titular externo; Professor Doutor Fábio Moreira Costa (INF/UFG), membro titular interno; e Professor Doutor Sérgio Teixeira de Carvalho, membro titular interno. Durante a argüição os membros da banca não fizeram sugestão de alteração do título do trabalho. A Banca Examinadora reuniu-se em sessão secreta a fim de concluir o julgamento da Tese tendo sido o candidato **aprovado** pelos seus membros. Proclamados os resultados pelo Professor Doutor Wellington Santos Martins, Presidente da Banca Examinadora, foram encerrados os trabalhos e, para constar, lavrou-se a presente ata que é assinada pelos Membros da Banca Examinadora, aos trinta dias do mês de agosto de dois mil e dezenove.

TÍTULO SUGERIDO PELA BANCA

Documento assinado eletronicamente por **Wellington Santos Martins**, **Professor do Magistério Superior**, em 30/08/2019, às 12:38, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Fábio Moreira Costa**, **Professor do Magistério Superior**, em 30/08/2019, às 12:39, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Sérgio Teixeira De Carvalho**, **Professor do Magistério Superior**, em 30/08/2019, às 12:39, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Clodoveu Augusto Davis Junior**, **Usuário Externo**, em 30/08/2019, às 12:39, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Nilton Correia da Silva**, **Usuário Externo**, em 30/08/2019, às 12:40, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0792686** e o código CRC **7B1D325A**.

Referência: Processo nº 23070.024709/2019-91 SEI nº 0792686

Sávio Salvarino Teles de Oliveira

Mestre em Ciências da Computação pela Universidade Federal de Goiás (2013). Tem experiência na área de Ciência da Computação, com ênfase em Sistemas Distribuídos, GeoProcessamento e Big Data. Trabalha desde 2008 no desenvolvimento de uma plataforma de processamento de operações espaciais em Big data, desenvolvendo algoritmos espaciais paralelos e distribuídos. Possui conhecimento na área de Recuperação de Informações, desenvolvimento de aplicações Web (Java, Spring, Ajax (dwr), JavaScript, XMPP) e com a modelagem de aplicações distribuídas. Participou de projetos na área de Big Data tendo trabalhado com Hadoop, Spark, Flink, Storm, Kafka, Elasticsearch, Cassandra e MongoDB.

Dedico este trabalho:
Aos meus pais, José Salvarino de Oliveira e Maria Francisca Teles da Cunha.
Ao meu irmão Vinícius Teles de Oliveira.
À minha esposa Silvana Pereira de Souza e ao meu filho Davi Teles de Souza.

Agradecimentos

Este trabalho devo muito a algumas pessoas e instituições, por diversas razões, e gostaria de agradecer especialmente:

Aos meus pais, José Salvarino de Oliveira e Maria Francisca Teles da Cunha, pelo incentivo durante toda a minha formação acadêmica.

À minha esposa Silvana Pereira de Souza por todo apoio e compreensão ao longo da trajetória que me levou à concretização deste trabalho.

Ao meu orientador Professor Dr. Wellington Santos Martins por compartilhar comigo seu tema de pesquisa. Gostaria de agradecê-lo pela orientação e incentivo durante o desenvolvimento deste trabalho. Ao meu co-orientador Vagner José do Sacramento Rodrigues por ajudar a construir o projeto desta tese.

Ao grupo de trabalho da goGeo pela grande ajuda técnica, científica e por todo apoio durante essa jornada. Sem eles, com certeza, este trabalho não teria sido concretizado.

Ao grupo de pesquisa do Laboratório de Computação de Alto Desempenho e Aplicações (LDA) pela troca de conhecimento que tivemos durante este período, que enriqueceram as discussões do trabalho. Agradeço também ao grupo de pesquisa do Laboratório de Processamento de Imagens e Geoprocessamento (Lapig) que ajudaram na validação do problema da tese e fornecimento dos dados e requisitos para o desenvolvimento do trabalho.

A Fundação de Amparo à Pesquisa do Estado de Goiás (FAPEG) pelo suporte durante este trabalho, apoiando com a bolsa formação, nível Doutorado, da Chamada/ano: 03/2016. Agradeço também ao projeto de P&D ANEEL-Copel Distribuição de número 2866-04842017.

Resumo

de Oliveira, Sávio Salvarino Teles. **Explorando Paralelismo em Big Data no Processamento de Séries Temporais de Imagens de Sensoriamento Remoto**. Goiânia, 2019. 123p. Tese de Doutorado Relatório de Graduação. Instituto de Informática, Universidade Federal de Goiás.

A superfície do planeta Terra está mudando a uma taxa sem precedentes e a classificação do tipo de uso e cobertura do solo, utilizando séries temporais de sensoriamento remoto, é hoje imprescindível para a identificação dessas mudanças. O algoritmo TWDTW se destaca nesta tarefa, mas possui complexidade quadrática com alto custo computacional, dificultando o seu uso em grandes volumes de dados (Big Data). Neste trabalho atacamos esses problemas explorando paralelismo tanto em nível vertical (multicore/manycore) quanto horizontal (*cluster* de computadores), de forma integrada para oferecer alto desempenho. Na dimensão vertical, propomos um algoritmo paralelo (P-INDEX) para o cálculo dos índices de sensoriamento remoto, e outro (P-TWDTW) para o cálculo de medidas de similaridade entre séries temporais. O speedup do algoritmo P-INDEX foi de até 9 vezes no processamento de todas as imagens em relação ao algoritmo sequencial, enquanto o P-TWDTW conseguiu ser até 12 vezes mais rápido que sua versão centralizada em C++ e 246 vezes mais rápido que o algoritmo TWDTW original em R. Além de viabilizar o cálculo rápido de uma medida de similaridade mais sofisticada, a exploração de paralelismo no P-TWDTW também contribuiu para que essas medidas fossem usadas como meta-características para métodos de aprendizado de máquina mais robustos. Isso aumentou a acurácia da classificação das séries temporais de 78%, utilizando o TWDTW com o KNN, para quase 94%, utilizando as meta-características obtidas a partir do P-TWDTW com o SVM. Na dimensão horizontal, propomos uma plataforma distribuída (BigSensing) que permite o tratamento eficiente de grandes volumes de dados de sensoriamento remoto. A plataforma inclui um motor inteligente de busca que é capaz de escolher, em tempo real, o melhor sistema para filtrar e recuperar os dados, de acordo com as restrições espaciais e temporais da consulta, tendo uma redução de quase 22% do tempo de resposta em relação ao SciDB na filtragem e recuperação de dados.

Palavras–chave

Séries Temporais, Sensoriamento Remoto, Sistemas Distribuídos, Processamento Paralelo, Big Data

Abstract

de Oliveira, Sávio Salvarino Teles. **Exploring Parallelism in Big Data on Remote Sensing Time Series Processing**. Goiânia, 2019. 123p. PhD. Thesis Relatório de Graduação. Instituto de Informática, Universidade Federal de Goiás.

The surface of planet Earth is changing at an unprecedented rate and the land use and land cover classification using remote sensing time series is now essential for identifying these changes. The TWDTW algorithm stands out in this task, but it has a quadratic complexity and high computational cost, making it difficult to use with Big Data. In this paper we tackle these problems by exploiting parallelism at both the vertical (multicore / manycore) and horizontal (cluster - distributed system) levels, in an integrated way for high performance. In the vertical dimension, we propose a parallel algorithm (P-INDEX) for the calculation of remote sensing indices, and another (P-TWDTW) for the calculation of similarity between time series. The speedup of P-INDEX was up to 9 times relative to the sequential algorithm in processing all images, while P-TWDTW was up to 12 times faster than its C++ centralized version and 246 times faster than the original in R TWDTW algorithm. In addition to enabling the quick calculation of a more sophisticated similarity measure, P-TWDTW also contributed to the generation of meta-characteristics for more robust machine learning methods. This increased the accuracy of the time series classification from 78% using TWDTW with KNN to almost 94% using the meta-characteristics obtained from P-TWDTW with SVM. In the horizontal dimension, we propose a distributed platform (BigSensing) that enables efficient handling of large volumes of remote sensing data. The platform includes a smart query engine that is able to choose, in real time, the best system to filter and retrieve data according to the spatial and temporal constraints of the query, with a nearly 22% reduction in response time over SciDB.

Keywords

Time Series, Remote Sensing, Distributed Systems, Parallel Processing, Big Data

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviaturas e Siglas

STSR	Séries Temporais de Sensoriamento Remoto
DTW	Dynamic Time Warping
TWDTW	Time-Weighted Dynamic Time Warping
NDVI	Normalized Difference Vegetation Index
EVI	Enhanced Vegetation Index
NIR	Near-infrared
MIR	Mid-infrared
SISD	Single Instruction Single Data
SIMD	Single Instruction Multiple Data
MISD	Multiple Instruction Single Data
MIMD	Multiple Instruction Multiple Data
CPU	Central Processing Unit
GPU	Graphics Processing Unit
CUDA	Compute Unified Device Architecture
OpenCL	Open Computing Language
API	Application Programming Interface
WCS	Web Coverage Service
WMS	Web Map Services

CAPÍTULO 1

Introdução

A superfície do planeta Terra está mudando a uma taxa sem precedentes, onde ecossistemas de florestas diminuem em uma velocidade alarmante e áreas urbanas e agrícolas se expandem em volta do espaço natural. O processamento de séries temporais de sensoriamento remoto (STSR), para classificação do tipo de uso e cobertura do solo tem se tornado indispensável na identificação destas mudanças e tem atraído grande interesse no cenário mundial na resolução de problemas de sustentabilidade [8].

Os algoritmos de processamento das STSR têm, geralmente, alto custo computacional e, por isso, torna-se importante utilizar arquiteturas paralelas para explorar o grande volume de dados disponível e aperfeiçoar os métodos de análise. Em vez de aumentar a velocidade de seu único núcleo de processamento, as arquiteturas paralelas tradicionais são hoje majoritariamente formadas por processadores com vários núcleos. Por isso, no cenário atual, os desenvolvedores de software têm explorado as arquiteturas paralelas com o uso coordenado de vários núcleos para a execução de aplicações que demandam alta capacidade de processamento. Atualmente, pode-se falar de duas abordagens paralelas. A primeira, abordagem *multicore* (i.e. CPUs), integra vários núcleos (até dezenas) em um único microprocessador e, a segunda, abordagem *manycore* (i.e. GPUs), utiliza aceleradores com até milhares de núcleos.

Para que os algoritmos de processamento de STSR possam lidar com o grande volume de dados de sensoriamento remoto, eles devem utilizar, além das arquiteturas paralelas *multicore* e *manycore*, arquiteturas de memórias distribuídas que explorem a escalabilidade horizontal de um *cluster* de computadores [109]. O grande número de sensores que fornecem um massivo volume de dados variados e complexos [72] de sensoriamento remoto, mostra o grande potencial de dados a explorar no processamento das STSR. Nesse cenário, pode-se dizer que estamos vivendo a era do Remote Sensing Big Data [64], que está revolucionando a forma como os dados de sensoriamento remoto são processados, analisados e interpretados para obter conhecimento [118], trazendo desafios únicos no processamento de séries temporais [35].

Os algoritmos convencionais de processamento de STSR não foram projetados para processar de forma eficiente e extrair valor desse grande volume de dados [96]. As

soluções atuais têm sido construídas localmente, criando modelos estatísticos a partir de uma amostra do conjunto total de dados, dada a dificuldade de se carregar e processar esse grande volume de dados em uma única máquina [111]. O advento do Big Data ocasionou uma revolução na forma de construir esses modelos, onde novas soluções são propostas utilizando um grande volume de dados para revelar características antes não visíveis [78].

Diante desse cenário, existe um problema em aberto na área de Remote Sensing Big Data [69, 45, 19, 46, 21, 64] : como explorar arquiteturas paralelas e distribuídas no processamento eficiente de um grande volume de dados de séries temporais de sensoriamento remoto?

Dentre os algoritmos convencionais para o processamento das STSR, o TWDTW (Time-Weighted Dynamic Time Warping) se destaca como um dos melhores na classificação do tipo de uso e cobertura do solo, com o objetivo de ajudar a identificar as mudanças da superfície do planeta Terra [120]. O TWDTW é um algoritmo centralizado que realiza o cálculo de medidas de similaridade entre STSR. Essas medidas podem ser utilizadas como entrada para algoritmos de classificação para identificar o tipo de uso e cobertura do solo. Esse algoritmo possui alta demanda computacional, com complexidade de tempo quadrática e, por isso, a paralelização do TWDTW se torna importante para extrair valor dos dados das STSR em um tempo viável.

A paralelização do TWDTW, entretanto, não pode ser realizada de forma trivial entre os núcleos de processamento. Para calcular a medida de similaridade entre duas STSR, o TWDTW constrói uma matriz de custo entre elas. O cálculo de cada elemento da matriz depende do resultado do processamento de elementos anteriores, ou seja, o cálculo do elemento $TWDTW(i,j)$ depende do processamento dos elementos anteriores $TWDTW(i-1,j)$, $TWDTW(i,j-1)$ e $TWDTW(i-1,j-1)$, iniciando em $TWDTW(1,1)$. Esta dependência faz com que o TWDTW seja processado por um único núcleo de processamento de forma serial.

O uso do TWDTW na classificação de STSR atingiu uma acurácia global de 97% em um estudo em uma área de 5300 km^2 no estado de Mato Grosso, Brasil [120]. Porém, em outras regiões do mundo como, por exemplo, na região da Califórnia, nos Estados Unidos, a acurácia da classificação utilizando o TWDTW foi de 80% [25]. Esta acurácia mais baixa é explicada pela alta variabilidade dos dados dentro de cada classe, o que leva à sobreposição de padrões de séries temporais entre as classes e gera confusão na classificação [25]. Por isso, existe a necessidade de reduzir o impacto da variabilidade dos dados dentro de cada classe na classificação utilizando o TWDTW.

Assim, dois problemas principais foram identificados no uso do TWDTW na classificação do tipo de uso e cobertura do solo e que serão abordados nessa tese: i) alto custo computacional do TWDTW, que não pode ser paralelizado de forma trivial; ii) baixa acurácia da classificação utilizando o TWDTW em regiões com alta variabilidade dos

dados dentro de cada classe.

Esta tese propõe um novo algoritmo massivamente paralelo, denominado Parallel TWDTW (P-TWDTW), baseado no TWDTW, que altera o algoritmo original para permitir explorar as arquiteturas paralelas *manycore* no cálculo da medida de similaridade entre as séries temporais. O P-TWDTW reduziu o tempo de resposta em até 12 vezes se comparado com a sua versão centralizada em C++ e 246 vezes se comparado com a implementação original do TWDTW em R.

A baixa acurácia da classificação utilizando o TWDTW foi abordada em um trabalho que estende a abordagem de [96], que utiliza as séries temporais das bandas espectrais e dos índices de sensoriamento remoto como características de entrada para métodos de aprendizado de máquina. Nossa solução gera meta-características com base nas medidas de similaridade das STSR utilizando o P-TWDTW.

Os índices de sensoriamento remoto utilizados como entrada para os classificadores são calculados a partir de combinações das bandas espectrais. O aumento do volume de dados de sensoriamento remoto trouxe um grande desafio no cálculo eficiente destes índices [66, 97]. Assim, este trabalho também apresenta o algoritmo P-INDEX, que utiliza arquiteturas paralelas *manycore* e *multicore* para o cálculo dos índices que compõem os dados de entrada das STSR para o P-TWDTW.

O P-TWDTW e o P-INDEX exploram a escalabilidade vertical de cada máquina, mas não foram projetados para explorar a escalabilidade horizontal de um *cluster* de computadores no processamento de um grande volume de dados de STSR. Não foram encontradas na literatura soluções eficientes para resolver esses problemas no processamento de séries temporais em Remote Sensing Big Data, sendo um dos grandes desafios da área [72]. Dois problemas principais foram identificados para a criação de uma solução de processamento de STSR em Remote Sensing Big Data: i) grande volume de dados de sensoriamento remoto recebidos em uma velocidade sem precedentes para armazenar, indexar, filtrar e recuperar, com o qual as soluções tradicionais não são capazes de lidar; ii) complexidade em distribuir o processamento dos algoritmos de STSR em um *cluster* de computadores explorando a escalabilidade horizontal do *cluster* e vertical com as arquiteturas paralelas em cada máquina.

Para suplantar esses problemas, foi desenvolvida, nesse trabalho, uma nova plataforma Big Data, denominada BigSensing, que explora a escalabilidade horizontal de um *cluster* de computadores, permitindo armazenar, indexar, filtrar e recuperar um grande volume de dados de STSR. A plataforma é capaz de inserir um grande volume de dados gerado por diversas fontes de sensoriamento remoto e é responsável por distribuir a execução dos algoritmos de processamento de séries temporais, tais como o P-TWDTW.

Diversas soluções de plataformas em Remote Sensing Big Data foram propostas na literatura [69, 45, 19, 46, 21, 64] contendo vários sistemas de Big Data para indexação e

gerenciamento dos dados para processamento em lote ou tempo real das STSR, tais como o Hadoop [1], Spark [2], Elasticsearch [3] e SciDB [4]. Experimentos realizados em diversos trabalhos [33, 126] mostram que nenhum sistema de Big Data tem desempenho melhor que os outros em todos os cenários com filtros espaciais e temporais. Fica a cargo do desenvolvedor da solução escolher previamente qual o sistema deseja utilizar para cada cenário de consulta.

A plataforma BigSensing possui um motor inteligente que é capaz de aplicar os filtros espaciais e temporais e recuperar os dados das STSR, buscando escolher entre qualquer sistema de Big Data capaz de aplicar os filtros espaciais e temporais, em tempo real, aquele mais eficiente. Este motor, denominado SmarT (smart Spatial-Temporal query engine), está preparado para remoção ou adição de sistemas na plataforma, bastando ao desenvolvedor treiná-lo novamente com o novo conjunto de sistemas de Big Data.

1.1 Objetivos

Os objetivos principais do nosso trabalho são apresentados abaixo:

1. Criar um novo algoritmo massivamente paralelo que permita explorar as arquiteturas paralelas *manycore* para o processamento de séries temporais.
2. Aumentar a acurácia da classificação utilizando o TWDTW, minimizando o impacto da variabilidade dos dados dentro de cada classe.
3. Propor uma solução para lidar com o grande volume de dados de STSR, permitindo explorar a escalabilidade vertical e horizontal na execução de algoritmos de processamento de STSR em um *cluster* de computadores, explorando, de forma inteligente, os recursos de sistemas de Big Data.

1.2 Contribuições

As principais contribuições deste trabalho são as seguintes:

1. Algoritmo P-TWDTW que explora o paralelismo massivo no processamento de STSR.
2. Criação de meta-características, para algoritmos de aprendizado de máquina, utilizando as métricas de similaridade entre STSR calculadas pelo P-TWDTW.

[1] https://hadoop.apache.org/
[2] https://spark.apache.org
[3] https://www.elastic.co
[4] https://www.paradigm4.com/

3. Algoritmo P-INDEX para processamento massivamente paralelo dos índices de sensoriamento remoto.
4. Plataforma Big Data, denominada BigSensing, para processamento de STSR:
   (a) Criação de um motor inteligente de busca, denominado SmarT, capaz de escolher, entre diversos sistemas de Big Data, o mais eficiente na filtragem e recuperação de dados de acordo com o cenário da consulta, em tempo real.
   (b) Estratégia para ingestão de grande volume de dados de STSR.
   (c) Abordagem de armazenamento e processamento das STSR em um *cluster* de computadores.
5. Avaliação da solução proposta neste trabalho com grande volume de dados reais e sintéticos (até 19 bilhões de STSR).

1.3 Publicações

As publicações decorrentes do trabalho desenvolvido nesta pesquisa são listadas abaixo:

1. RDebug: A New Debugging Technique for Distributed R-Trees. In: XV Brazilian Symposium on Geoinformatics, 2014, Campos do Jordão - SP. p. 49-60 [30].
2. A New Technique for Verifying the Consistency of Distributed R-Trees. Journal of Information and Data Management - JIDM, v. 6, p. 59-70, 2015 [29].
3. PNDVI: Um novo Algoritmo para Processamento Paralelo do índice de vegetação NDVI. In: Escola Regional de Informática de Goiás, 2016, Goiânia. v. 4. p. 235-248 [87].
4. A new Platform for Time-Series Analysis of Remote Sensing Images in a Distributed Computing Environment. In: XVII Brazilian Symposium on GeoInformatics, 2016, Campos do Jordão. p. 128-139 [88].
5. DistSensing: A new Platform for Time Series Processing in a Distributed Computing Environment. RBC. Revista Brasileira de Cartografia (Online), v. 69, p. 881, 2017 [27].
6. DStream: Plataforma de coleta, processamento e armazenamento de streams de dados de sensoriamento remoto. In: Escola Regional de Alto Desempenho do Centro-Oeste (ERAD-CO), 2018, Goiânia [90].
7. P-TWDTW: Parallel Processing of Time Series Remote Sensing Images Using *manycore* Architectures. In: XIX Symposium on High Performance Computing Systems (WSCAD), 2018, São Paulo. p. 252-258 [28].

8. SP-TWDTW: A New Parallel Algorithm for Spatio-Temporal Analysis of Remote Sensing Images In: XIX Brazilian Symposium on Geoinformatics, 2018, Campina Grande, PB, Brazil. p.46 - 5 [89].
9. A Parallel and Distributed Approach to the Analysis of Time Series on Remote Sensing Big Data. Journal of Information and Data Management - JIDM. **Aceito para publicação em 2019**.

1.4 Organização da tese

O presente trabalho está organizado como se segue. O Capítulo 2 descreve os fundamentos conceituais utilizados para o desenvolvimento da pesquisa nesta tese. O Capítulo 3 apresenta uma visão geral sobre o estado da arte das estratégias de processamento de STSR. O Capítulo 4 apresenta a solução proposta nesta tese para o processamento paralelo de séries temporais utilizando arquiteturas paralelas *manycore* e *multicore*. O Capítulo 5 descreve a arquitetura da plataforma proposta neste trabalho, denominada BigSensing, para explorar os recursos computacionais do *cluster* de computadores e o paralelismo massivo em cada servidor. O Capítulo 6 apresenta a avaliação de desempenho da solução proposta neste trabalho no processamento de STSR no contexto de Big Data, com o objetivo de avaliar se a solução foi capaz de resolver os problemas levantados neste trabalho. O Capítulo 7 apresenta as conclusões, limitações e trabalhos futuros desta pesquisa.

CAPÍTULO 2

Conceitos fundamentais

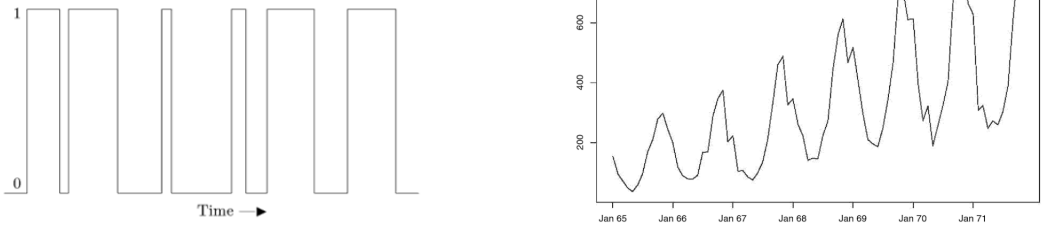
Este capítulo apresenta os fundamentos conceituais para o processamento de STSR utilizando arquiteturas paralelas e distribuídas. Um dos mais promissores usos das STSR é a classificação do tipo de uso e cobertura do solo [96]. A Seção 2.1 introduz os conceitos de séries temporais na área de sensoriamento remoto e descreve como os algoritmos podem ser utilizados para classificar o uso e cobertura do solo. Dentre estes algoritmos, o TWDTW, apresentado na Seção 2.1.1, é um dos mais utilizados em sensoriamento remoto como medida de distância para métodos de aprendizado baseado em distância, como o KNN (K-Nearest Neighbors). Entretanto, em algumas regiões, a acurácia do KNN utilizando as medidas de distância do TWDTW é considerada muito baixa na classificação do uso e cobertura do solo [25]. Por isso, o trabalho de [96] apresenta um novo algoritmo, descrito na Seção 2.1.2, que utiliza as bandas espectrais e índices de vegetação como características dos métodos de aprendizado de máquina. Nosso trabalho propõe uma alteração do trabalho de [96] com a construção de meta-características utilizando as medidas de distância do TWDTW como variáveis para os métodos de aprendizado, além das bandas espectrais e dos índices de vegetação.

A geração dessas meta-características com o TWDTW tem alto custo computacional [77] e, por isso, surge a necessidade de utilizar arquiteturas massivamente paralelas para o cálculo das medidas de distância do TWDTW. O modelo de programação paralela é introduzido na Seção 2.2.1 e explora as arquiteturas Manycore na GPU utilizando principalmente o modelo de programação CUDA, apresentado na Seção 2.2.1. Devido ao grande volume de dados de sensoriamento remoto gerados em uma velocidade sem precedentes, podemos dizer que estamos vivendo a era do Remote Sensing Big Data. Por isso, surge o desafio de armazenar, indexar, extrair e processar de forma eficiente utilizando as arquiteturas distribuídas, descritas na Seção 2.2.2.

2.1 Séries Temporais de Sensoriamento Remoto

Nesta seção, abordarmos conceitos importantes sobre o uso de séries temporais para classificação do tipo de uso e cobertura do solo. Uma série temporal é uma coleção

de observações feitas sequencialmente ao longo do tempo. Uma série temporal é dita ser contínua quando as observações são feitas continuamente ao longo do tempo, como pode ser visto no exemplo da Figura 2.1(a). Nesse exemplo, as observações são tomadas com apenas dois valores, 0 e 1, e simulam um computador que tem um estado de "desligado"ou "ligado"ao longo do tempo. Definindo o conjunto T = $\{t : t_1 < t < t_2\}$, uma série temporal contínua será denotada por $\{X(t) : t \in T\}$. Uma série temporal é dita ser discreta quando as observações são feitas em tempos específicos, em geral, igualmente espaçados. A Figura 2.1(b) apresenta um exemplo de uma série temporal discreta com valores variando durante semanas ou meses, durante vários anos. Definindo o conjunto T = $\{t_1, ..., t_n\}$, a série temporal discreta será denotada por $\{X_t : t \in T\}$.



(a) *Série Temporal Contínua* (b) *Série Temporal Discreta.*

Figura 2.1: *Séries Temporais*

As observações das STSR são continuamente obtidas a partir de sensores espaciais e aéreos. No nosso trabalho, escolhemos as imagens geradas pelo produto MOD13Q1 [32], disponibilizados pelo sensor MODIS/Terra, por serem públicas e de fácil acesso. O sensor MODIS opera com 36 bandas espectrais [1], abrange 2.330 km de largura da faixa de cobertura espacial, está localizado a 705 km de altitude, com diferentes resoluções espaciais (250m, 500m e 1000m), com imageamento global a cada 2 dias, ciclo de repetição de 16 dias, e nenhum custo para obtenção das imagens além da correção atmosférica e georreferenciamento das imagens.

Os dados do produto MOD13Q1 possuem resolução temporal de 16 dias e resolução espacial de 250 metros. A resolução temporal é a medida que temos para medir a capacidade de revisita de um satélite sobre um mesmo local da Terra. Quanto maior for a resolução temporal, maior será a frequência de visita ao mesmo local. A resolução espacial está relacionada com a capacidade de cada sensor de detectar os objetos da superfície terrestre. Dessa forma, quanto maior for a resolução espacial, menor será o objeto distinguível pelo sensor. O produto MOD13Q1 dispõe de dois índices de vegetação,

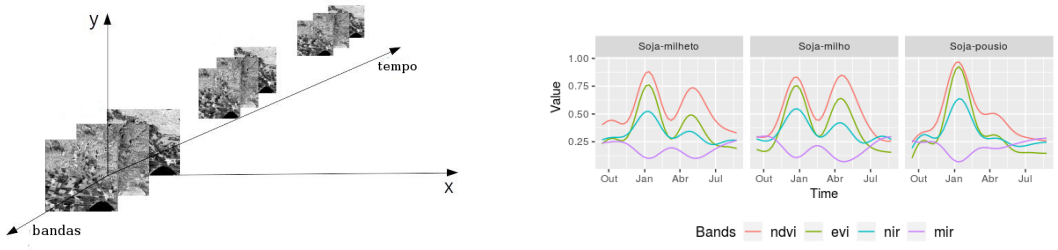
[1] Banda espectral é o intervalo entre dois comprimentos de onda no espectro eletromagnético e têm a capacidade de discriminar e/ou realçar diferentes objetos nas imagens.

NDVI e EVI, e imagens de refletância das bandas RED, BLUE, NIR e MIR. Estes dados disponibilizados pelo sensor MODIS/Terra foram capazes de fornecer informações suficientes para classificação do tipo e uso de solo em diversas pesquisas [38]. No Capítulo 6, de avaliação da nossa proposta, foram geradas bases sintéticas apenas para os cenários em que forem necessários dados com maior resolução espacial e temporal para análise de performance da solução.

A Figura 2.2(a) apresenta as quatro dimensões das STSR. Nesse gráfico, os eixos x e y representam os limites espaciais de cada banda de uma cena, onde o eixo x está associado à longitude das coordenadas geográficas da cena e eixo y está associado à latitude. O eixo "bandas" representa as bandas espectrais e índices de sensoriamento remoto de cada cena, e o eixo "tempo" representa as imagens que foram obtidas em vários pontos do tempo pelo satélite.

A Figura 2.2(b) apresenta um exemplo das séries temporais de três tipos de uso e cobertura do solo: Soja-Milheto, Soja-milho e Soja-pousio. No eixo "Bands", cada observação das séries temporais possui quatro valores: as duas bandas espectrais NIR e MIR e os dois índices de vegetação EVI e NDVI. No eixo "Time", as séries temporais foram coletadas no período de um ano na região do Estado do Mato Grosso (eixos "x" e "y"). Estas séries temporais são utilizadas como entrada para algoritmos de classificação do uso e cobertura do solo.

O problema geral de classificação consiste em classificar observações de uma série temporal $\{X(t), t = 1,2,...,N\}$ em uma observação com k categorias $\pi_1, \pi_2, ..., \pi_k$ com a menor taxa de erro [40]. Ou seja, dado um conjunto de séries temporais, a tarefa da classificação de séries temporais é mapear cada série temporal em uma das classes predefinidas.



(a) *Dimensionalidade da série temporal de imagens de sensoriamento remoto.*

(b) *Séries temporais das bandas NIR e MIR e dos índices NDVI e EVI de três tipos de uso e cobertura do solo: Soja-Milheto, Soja-milho e Soja-pousio.*

Figura 2.2: *Série temporal de imagens de sensoriamento remoto*

Os métodos de classificação utilizando os dados das séries temporais podem ser divididos em três grandes categorias [123]. A primeira categoria, apresentada na Seção 2.1.1, utiliza a classificação baseada na distância, onde a distância é utilizada como

medida de similaridade entre séries temporais. Dentre esses métodos, destaca-se o uso das medidas de similaridade do TWDTW (Seção 2.1.1) como entrada para o algoritmo KNN. A segunda categoria, apresentada na Seção 2.1.2, utiliza a classificação baseada em características, onde uma série temporal é transformada em um vetor de características e então aplicados métodos convencionais de classificação. A Seção 2.1.2 apresenta a descrição do algoritmo de [96] que utiliza as observações das séries temporais das bandas espectrais e índices de vegetação como características para os métodos de aprendizado. Esses dois métodos, TWDTW e a solução de [96], foram importantes para a criação da solução desta tese para classificação do tipo de uso e cobertura do solo, adicionando ao conjunto de variáveis do trabalho de [96] meta-características geradas a partir das medidas de similaridade do TWDTW.

A terceira categoria utiliza classificação baseada em modelos, tais como modelos de cadeias de Markov e outros modelos estatísticos. Essa categoria é baseada em modelos generativos que assumem que séries temporais em uma classe são geradas utilizando um modelo M que modela a distribuição de probabilidade das séries temporais na classe. O modelo generativo mais simples é o classificador *Naive Bayes* [62]. Para modelar a dependência entre séries temporais, os modelos de Markov [106] e modelos ocultos de Markov (HMM) [102] têm sido também bastante utilizados.

2.1.1 Classificação Baseada na Distância entre Séries Temporais

Métodos baseados na distância de séries temporais definem uma função de distância para medir a similaridade entre um par de séries temporais. Uma vez que a função de distância é obtida, podem ser utilizados diversos métodos de classificação existentes, tais como o classificador KNN (K vizinhos mais próximos). Uma das principais funções de distância utilizadas para análise de séries temporais é a distância Euclidiana. Dadas duas séries temporais de valores reais $A = (a_1, a_2, ..., a_m)$ e $B = (b_1, b_2, ..., b_m)$, a distância Euclidiana é definida pela Equação 2-1:

$$L_2(A,B) = \sqrt{\sum_{i=1}^{m} (a_i - b_i)^2} \tag{2-1}$$

Uma desvantagem da distância Euclidiana é a limitação no cálculo da distância apenas entre séries temporais com o mesmo número de observações, ou seja, comprimentos iguais. É possível notar que as séries temporais apresentadas na Figura 2.3 apresentam comportamento similar, mas elas estão fora de fase, ou seja, a série temporal inferior está deslocada no tempo em relação à superior. O uso da distância Euclidiana não permite o reconhecimento dessa similaridade, já que as diferenças entre os pares de observação são calculadas no mesmo instante de tempo.

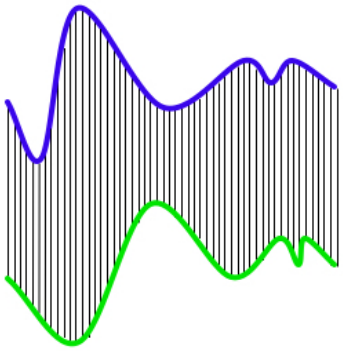


Figura 2.3: *Alinhamento de séries temporais utilizando a distância Euclidiana.*

Uma alternativa para resolver esse problema é utilizar a medida de similaridade *Dynamic Time Warping* (DTW) [9], que permite realizar o alinhamento entre duas séries temporais e, assim, reconhecer a similaridade entre elas, mesmo se apresentarem comprimentos distintos ou não estiverem alinhadas no eixo do tempo. Dadas as séries temporais $A = (a_1, a_2, ..., a_m)$ e $B = (b_1, b_2, ..., b_n)$ com tamanhos m e n respectivamente, o cálculo do DTW é realizado utilizando a Equação 2-2:

$$DTW(A,B) = min \sqrt{\sum_{k=1}^{K} w_k} \tag{2-2}$$

em que $w_k = (i, j)$ representa a associação entre as observações a_i e b_j das séries temporais A e B, sendo equivalentes à distância Euclidiana $d(i, j) = \sqrt{(a_i - b_j)^2}$. A sequência $w_1, w_2, ..., w_k$ representa a sequência de associações entre pares de observação das duas séries, denominada de caminho de ajuste. A Equação 2-2 está sujeita às seguintes restrições: i) a primeira observação de uma série temporal deve ser comparada à primeira observação da outra série, $w_1 = (1, 1)$, e a última observação de uma série deve ser comparada com a última observação da outra série, $w_k = (m, n)$; ii) dado $w_k = (i, j)$ e $w_{k+1} = (i', j')$ então $i' - i \leq 1$ e $j' - j \leq 1$; iii) dado $w_k = (i, j)$ e $w_{k+1} = (i', j')$ então $i' - i > 0$ e $j' - j \leq 0$.

A Figura 2.4 apresenta o resultado da utilização do DTW para calcular a distância entre as séries temporais da Figura 2.3. É possível notar que o DTW alinha as duas séries temporais, de forma a reconhecer a similaridade entre seus comportamentos, mesmo no caso das séries temporais estarem deslocadas no tempo uma em relação à outra.

Time-Weighted Dynamic Time Warping (TWDTW)

Ao contrário do DTW, o método TWDTW é sensível às mudanças sazonais dos tipos de uso e cobertura do solo. O algoritmo DTW não consegue controlar as distorções ao longo do tempo e tornar o alinhamento entre as séries temporais dependente

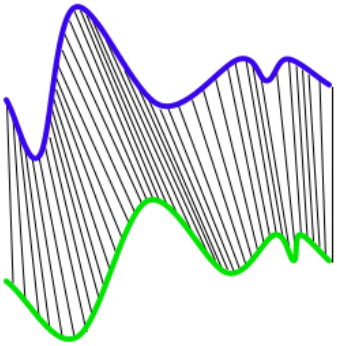


Figura 2.4: *Alinhamento de séries temporais utilizando a DTW.*

apenas das estações do ano. Por isso, foi criado o algoritmo TWDTW [120] para evitar, por exemplo, que um padrão extraído do inverno não case com uma série temporal do verão, já que cada tipo de uso e cobertura do solo, por exemplo na agricultura, pode ter comportamento diferente dependendo da estação do ano. O TWDTW é uma variação do DTW adicionando restrições temporais, ou seja, se existir uma grande diferença sazonal entre o padrão da amostra e as séries temporais que casam com esse padrão, um custo extra é adicionado ao DTW.

O método TWDTW segue as mesmas restrições do DTW: i) $w_1 = (1,1)$ e $w_k = (m,n)$; ii) dado $w_k = (i,j)$ e $w_{k+1} = (i',j')$ então $i' - i \leq 1$ e $j' - j \leq 1$; iii) dado $w_k = (i,j)$ e $w_{k+1} = (i',j')$ então $i' - i > 0$ e $j' - j \leq 0$. O TWDTW calcula uma matriz de custo Ψ com dimensão n por m a partir do padrão $U = (u_1,...,u_n)$ e da série temporal $V = (v_1,...,v_m)$. Os elementos $\psi_{i,j}$ da matriz são calculados adicionando um custo temporal ω, tornando-se $\psi_{i,j} = |u_i - v_j| + \omega_{i,j}$, onde $u_i \in U \ \forall \ i = 1,...,n$ e $v_j \in V \ \forall \ j = 1,...,m$. Para calcular o custo temporal, o modelo logístico é utilizado com um ponto médio β e uma inclinação α da equação 2-3

$$\omega_{i,j} = \frac{1}{1 + e^{-\alpha(g(t_i,t_j)-\beta)}}, \tag{2-3}$$

onde $g(t_i,t_j)$ é o tempo decorrido em dias entre as datas t_i no padrão U e t_j na série temporal V. Da matriz de custo Ψ é calculada a matriz de custo acumulado D utilizando uma soma recursiva das distâncias mínimas, tal como na equação 2-4

$$d_{i,j} = \psi_{i,j} + min\{d_{i-1,j}, d_{i-1,j-1}, d_{i,j-1}\}, \tag{2-4}$$

que está sujeita às seguintes condições:

$$d_{ij} = \begin{cases} \psi_{i,j} & i = 1, j = 1 \\ \sum_{k=1}^{i} \psi_{k,j} & 1 < i \leq n, j = 1 \\ \sum_{k=1}^{j} \psi_{i,k} & i = 1, 1 < j \leq m \end{cases} \tag{2-5}$$

O k-ésimo caminho de menor custo em D produz um alinhamento entre o padrão e uma subsequência de V com a distância associada δ_k, onde a_k é o ponto inicial e b_k é o ponto final da subsequência k. Cada ponto mínimo na última linha da matriz de custo acumulado, i.e., $d_{n,j} \; \forall \; j = 1,...,m$, produz um alinhamento, com $b_k = argmin_k(d_{n,j}), k = 1,...,K$ e $\delta_k = d_{n,b_k}$, onde K é o número mínimo de pontos na última linha da matriz D.

Um algoritmo reverso, Equação 2-6, mapeia o caminho $P_k = (p_1,...,p_L)$ ao longo do k-ésimo "vale" de menor custo em D. O algoritmo inicia em $p_{l=L} = (i = n, j = b_k$ e termina com i = 1, i.e. $p_{l=1} = (i = 1, j = a_k)$, onde L denota o último ponto do alinhamento. O caminho P_k contém os pontos que tiveram casamento entre as séries temporais.

$$p_{l-1} = \begin{cases} (i, a_k = j) & \text{se } i = 1 \\ (i-1, j) & \text{se } j = 1 \\ argmin(d_{i-1,j}, d_{i-1,j-1}, d_{i,j-1}) & \text{caso contrário} \end{cases} \tag{2-6}$$

Os melhores resultados de acurácia utilizando as medidas de similaridade do TWDTW na classificação de STSR têm sido obtidos utilizando a abordagem do KNN com K=1 (1NN) [76, 77, 120]. O KNN é um algoritmo não paramétrico de aprendizado lento que não necessita de dados de treinamento para gerar o modelo, ou seja, todos os dados disponíveis são utilizados na fase de teste. A classificação utilizando o KNN é feita por votação da maioria dos vizinhos do objeto e no caso do 1NN o vizinho mais próximo. A classificação com o 1NN utilizando as medidas de similaridade do TWDTW segue os seguintes passos:

1. Inicializa o valor de K=1
2. Calcula a distância entre cada série temporal V e cada padrão U utilizando o TWDTW.
3. Ordena as distâncias de forma ascendente.
4. Atribui a V a classe do padrão U com menor distância para V.

2.1.2 Classificação Baseada em Características

Métodos de classificação convencionais, tais como árvores de decisão e redes neurais foram projetadas para classificação utilizando vetores de características. Nesses métodos, os valores da série temporal são transformados em um vetor de características, tratando cada observação da série temporal como uma característica.

Nosso trabalho estendeu o método de classificação baseado em características de [96], adicionando, como entrada para os classificadores, meta-características criadas a partir das medidas de similaridade calculadas no TWDTW. A solução proposta por [96]

utiliza todos os valores das séries temporais das bandas espectrais e índices de vegetação como características para os algoritmos de aprendizado de máquina, criando espaços altamente dimensionais. Em um exemplo com séries temporais com 3 observações das bandas MIR e NIR, além do índice de vegetação NDVI, o modelo de dados de entrada para o método de aprendizado de máquina, seguindo a proposta de [96], pode ser visto na Tabela 2.1. Cada tupla de entrada representa um pixel e cada coluna a variável que será utilizada pelo classificador.

O trabalho de [96] propõe utilizar as bandas espectrais NIR e MIR, além dos índices de vegetação NDVI e EVI como variáveis dos métodos de aprendizado de máquina. Ele avaliou três classificadores: Support Vector Machine (SVM), Random Forest (RF) e Linear Discriminant Analysis (LDA). Dos três, o SVM teve maior acurácia em testes realizados em uma área de 5300 km^2 do estado do Mato Grosso.

ID do pixel	mir.1	mir.2	mir.3	nir.1	nir.2	nir.3	ndvi.1	ndvi.2	ndvi.3
1	0.5	0.1	0.8	0.7	0.1	0.15	0.11	0.11	0.05
2	0.8	0.85	0.7	0.2	0.1	0.4	0.3	0.5	-0.5
3	0.6	0	0	0.8	0.3	0.35	0.3	0.3	0.3
4	0.6	1	0.2	0.2	0.3	0.6	0	0	0
5	0.9	0.5	0.8	0.3	0.3	0.3	-1	-1	-1

Tabela 2.1: *Exemplo do modelo de dados utilizando em [96] transformando em características de métodos de aprendizado de máquina, as séries temporais com 3 observações das bandas MIR e NIR e do índice de vegetação NDVI.*

2.2 Processamento Paralelo e Distribuído

Sistemas paralelos e distribuídos são extremamente importantes na construção de uma solução de processamento de grande volume de dados de STSR em Remote Sensing Big Data. Na Seção 2.2.1 são apresentados modelos de programação paralela utilizando arquiteturas Multicore (CPUs) e Manycore (GPUs). Na Seção 2.2.2 são apresentados modelos de computação distribuída para Big Data no processamento de grande volume de dados.

2.2.1 Modelos de Programação Paralela

Um modelo de programação paralela é uma abstração da arquitetura de sistema do computador [75] e, por isso, não está acoplado a nenhum tipo específico de máquina. Nesta seção apresentamos os modelos de programação paralela que exploram arquiteturas Multicore e Manycore para o processamento de algoritmos.

Atualmente, algumas plataformas reúnem em um único sistema uma ou mais CPUs e uma ou mais GPUs (aceleradores), e são classificados como sistemas heterogêneos. As principais características de GPUs (Graphics Processing Unit) são sua alta capacidade de processamento massivo de forma paralela e ótimo desempenho em cálculos sobre um volume grande de dados, resultando em uma alta vazão [73]. Diferentemente do seu uso inicial, GPUs são hoje Unidades de Processamento de Propósito Geral (GPGPU) e atuam como co-processadores (aceleradores) que viabilizam explorar o paralelismo massivo.

Nos primeiros sistemas contendo CPUs e GPUs, foram utilizadas linguagens como Brook [11] e Cg [74]. Para facilitar a interface entre o programador e as aplicações na GPU, a NVIDIA apresentou o CUDA (Compute Unified Device Architecture) [107] em 2006 como o modelo e linguagem padrão para programar suas GPUs. Mais recentemente, a indústria tem trabalhado na construção do padrão OpenCL (Open Computing Language) [81] como um modelo comum para programação heterogênea nas CPUs e GPUs.

CUDA

Ao fornecer abstrações simples com respeito à organização hierárquica de *threads*, memória e sincronização, o modelo de programação CUDA permite aos programadores escreverem programas escaláveis sem a necessidade de aprender a multiplicidade de novos componentes de programação. Em CUDA, um sistema paralelo consiste de um *host* (CPU) e um dispositivo ou recurso de processamento (GPU). A execução de tarefas é feita na GPU, utilizando um conjunto de *threads* rodando em paralelo.

A programação segue o paradigma SPMD (Single Program Multiple Data), onde um único programa define o código das *threads*. A arquitetura de *threads* de uma GPU consiste de uma hierarquia em dois níveis, como pode ser visto na Figura 2.5. Um bloco é um conjunto de *threads* altamente acopladas e identificadas por um ID, ao passo que uma grade é um conjunto de blocos fracamente acoplados com tamanho e dimensão similares. Uma GPU é organizada como uma coleção de multiprocessadores, sendo cada um responsável por tratar um ou mais blocos em uma grade. Um bloco nunca é dividido entre múltiplos multiprocessadores. As *threads* dentro do bloco podem cooperar entre si, compartilhando dados através de uma memória compartilhada e sincronizando suas execuções para coordenar o acesso à memória.

O gerenciamento de *threads* em CUDA é realizado implicitamente, ou seja, os programadores não gerenciam a criação e destruição de *threads*. Eles precisam apenas especificar as dimensões da grade e do bloco necessárias a um processo em uma certa tarefa. Já o particionamento e mapeamento das threads em CUDA são feitos de forma explícita, onde o programador deve definir a carga de trabalho a ser executada em paralelo

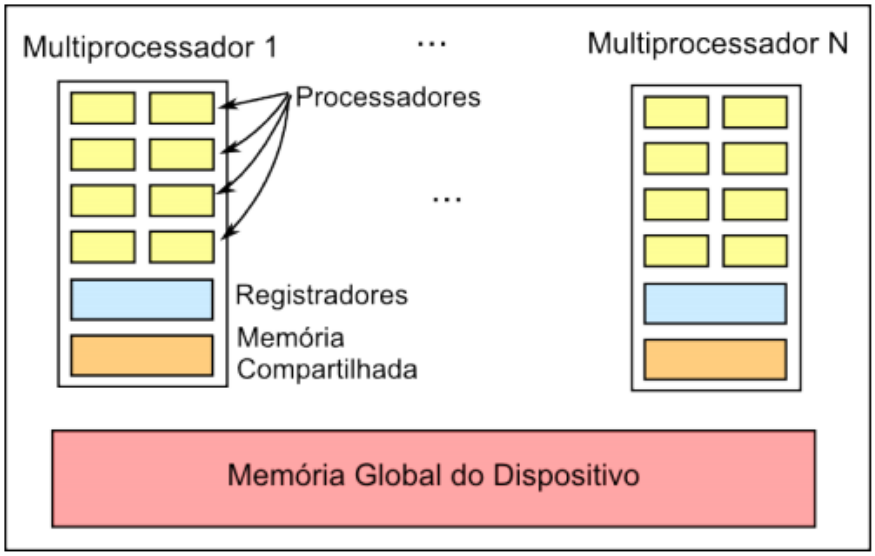


Figura 2.5: *Arquitetura CUDA (Fonte [26])*

utilizando uma função global. O modelo de memória de CUDA é também apresentado na Figura 2.5. Nas GPUs, a memória é um gargalo considerável, sendo que em alguns casos o tempo gasto para transferência de dados da CPU para a GPU é maior que o tempo de processamento. Por esse motivo, deve-se organizar a memória de forma que o acesso possa ser feito através de um único bloco contíguo [58].

Na parte de baixo da Figura 2.5, podemos ver as memórias global e constante, que são os locais onde os programas executados no *host* (no caso CPU) podem escrever e ler dados para a GPU. A memória constante permite apenas o acesso de leitura pelo dispositivo e a memória global permite acesso de escrita e leitura. Dentro de cada bloco, temos a memória compartilhada e os registradores ou memórias locais. A memória compartilhada pode ser acessada por todas as *threads* em um bloco e os registradores são dedicados para cada *thread*. Cada *thread* possui acesso a três níveis diferentes de memória, sendo cada um mais lento que o outro. Em primeiro lugar está a memória privada da *thread*, que não pode ser compartilhada entre *threads* e representa tanto trechos da memória interna de um multiprocessador quanto registradores. Em segundo plano está a memória compartilhada por bloco, onde as *threads* em um mesmo bloco possuem acesso a essa memória que fica no multiprocessador. No terceiro nível está a memória global, que é mais lenta que os outros níveis e sua utilização deve ser feita com cuidado para não ter impactos negativos na performance da aplicação.

Os blocos gerados ao chamar um *kernel* são entregues ao gerenciador dentro de um multiprocessador, sendo distribuídos entre os multiprocessadores pelo gerenciador da GPU para manter o nível de carga igual. Internamente, os blocos ainda são separados em *warps*, que são controlados pelo gerenciador dentro do multiprocessador. A quantidade

de *threads* utilizadas é o produto do número de blocos pelo tamanho de cada bloco. Como é possível criar um número de *threads* muito maior do que o suportado pelo hardware, um controlador de software gerencia e cria as *threads* que serão executadas. Como não se tem controle sobre o gerenciador de *threads*, não é possível determinar qual *thread* executará primeiro e, por isso, deve-se utilizar funções de sincronização dentro do *kernel* somente quando necessário.

2.2.2 Processamento Distribuído em Big Data

Um sistema distribuído, segundo a definição de George Coulouris, é "aquele no qual os componentes localizados em computadores interligados em rede comunicam entre si e coordenam suas ações apenas passando mensagens" [23]. Um sistema distribuído tipicamente satisfaz os seguintes critérios [41]: i) múltiplos processos: o sistema consiste de mais de um processo sequencial, onde cada processo tem uma *thread* de controle independente; ii) comunicação entre processos: processos se comunicam utilizando mensagens através de canais de comunicação; iii) espaço de endereçamento disjuntos: processos tem espaços de endereçamento separados; iv) objetivos coletivos: processos devem interagir uns com os outros para atingirem um objetivo comum.

Tais sistemas distribuídos são primordiais em Big Data para lidar com o grande volume de dados complexos que tem aumentado de forma vertiginosa a cada dia. Big Data pode ser caracterizado por três principais características: volume, velocidade e variedade, definidos como os três Vs pelo Meta Group (agora Gartner) em 2001 [60]. Remote Sensing Big Data também pode ser caracterizado pelos 3 Vs:

- volume: petabytes de dados de sensoriamento remoto estão armazenados ao redor do mundo. Somente a NASA possui milhões de usuários que acessam cerca de 24 PB com uma projeção de chegar a mais de 200 PB em cinco anos [82];
- velocidade: esta característica não está ligada somente à geração dos dados a taxas crescentes, mas também ao processamento eficiente dos dados, ou seja, os dados devem ser analisados em um tempo viável. Este desafio está cada vez maior com a projeção de geração de 50 PB de dados por ano [82];
- variedade: esta característica é inerente aos dados de sensoriamento remoto que são gerados a partir de múltiplas fontes de dados (laser, radar, etc), coletados em diferentes datas, com uma variedade de resoluções espaciais que podem ser utilizados em diferentes domínios de aplicações [19].

Nos últimos anos, sistemas distribuídos tem ganhado importância substancial devido a diversos fatores. Primeiro, existe a necessidade de reduzir o tempo de computação em relação a abordagens utilizando apenas uma máquina, que possuem limites físicos de crescimento de recursos computacionais. Enquanto abordagens MultiCore e ManyCore

aumentam o limite de crescimento introduzindo o nível de paralelismo arquitetural, essas técnicas não escalam de forma satisfatória depois de um determinado nível. Por isso, atribuir processos a máquinas separadas permite explorar um nível adicional de paralelismo. Segundo, existe uma necessidade de compartilhamento de recursos, onde uma máquina pode utilizar recursos computacionais ociosos em outra máquina na rede de computadores. A computação nas nuvens é um exemplo disso, onde os *datacenters* permitem que milhares de clientes compartilhem recursos computacionais através da Internet para computação eficiente a um custo razoável sem a necessidade de se preocuparem com a manutenção das máquinas. Terceiro, mesmo sistemas de computação poderosos construídos em uma única máquina estão propensos a um completo colapso quando esta máquina falha. Sistemas distribuídos tem o potencial de evitar esses problemas utilizando técnicas apropriadas de tolerância a falhas, já que quando um máquina falha restam as outras no sistema para manter a aplicação funcionando corretamente.

As três abordagens mais utilizadas para processamento distribuído no contexto de *Big Data* [17] são descritas nas Seções 2.2.2, 2.2.2 e 2.2.2.

Processamento em Lote

O processamento em lote é uma forma eficiente de processar grandes volumes de dados, onde os dados são coletados, processados e então produzidos os resultados em lote, visando principalmente a alta vazão do sistema, mesmo que em detrimento da latência. O processamento batch tem a característica de não exigir a interação do usuário com a aplicação. Todas as entradas e saídas de dados da aplicação são implementadas por algum tipo de memória secundária, geralmente arquivos em disco.

O modelo de programação MapReduce é o mais utilizado no processamento em lote no contexto de Big Data [17] e permite que grandes volumes de dados sejam processados de forma paralela em um conjunto de máquinas *commodities*. Ele é baseado no fato que a maior parte das informações necessárias às tarefas seguem uma estrutura similar, ou seja, o mesmo processamento é aplicado sobre um grande número de tuplas.

O Apache Hadoop[2] é uma das mais famosas e poderosas ferramentas de processamento em lote. Ele implementa o modelo de programação MapReduce e utiliza o sistema de arquivos distribuído HDFS (*Hadoop Distributed File System*) para armazenar os dados das aplicações. O Spark[3] tem ganhado bastante interesse da comunidade Big Data e foi projetado para reutilizar os dados depois de vários processamentos. Ele utiliza o modelo de programação MapReduce com suporte a processamento na memória através de um modelo utilizando *Resilient Distributed Datasets* (RDDs), que são definidos como

[2] hadoop.apache.org
[3] spark.apache.org

uma coleção distribuída de itens. Os RDDs são estruturas de dados paralelas e tolerante à falhas que permitem que usuários persistam resultados intermediários na memória.

Processamento de *Streams*

O processamento de *streams* de dados deve ser realizado no momento em que os dados chegam ao sistema com uma baixa latência de resposta. Nos sistemas baseados em MapReduce, a fase *reduce* só inicia depois que toda a fase *map* termina o que gera uma alta latência de processamento. Essa alta latência torna quase impossível o processamento de *streams*. O processamento de *streams* simplifica a programação paralela restringindo a computação paralela que pode ser executada. Dado uma sequência de dados (um *stream*), uma série de operações é aplicada a cada elemento no *stream*. O processamento de *streams* é um modelo baseado nos dados, que implica em uma execução mais fácil, rápida e mais eficiente.

O processamento de dados em *streaming* é geralmente composto por duas camadas: uma de armazenamento e outra de processamento. A camada de armazenamento precisa suportar a solicitação de registros e uma forte consistência para permitir leituras e gravações rápidas, de baixo custo e reproduzíveis, de grandes *streams* de dados. A camada de processamento é responsável pelo consumo de dados da camada de armazenamento, realizando cálculos sobre esses dados e, em seguida, enviando notificações para a camada de armazenamento excluir os dados desnecessários. Diversas plataformas foram propostas para o processamento de *streams*, tais como: Apache Storm[4], Spark Streaming[5], Hadoop Streaming[6], Apache Flink[7] e Apache Kafka[8].

Processamento em Tempo Real

Nos últimos anos, sistemas Big Data tentam resolver problemas não apenas de processamento em lote ou de *streams*, mas também de processamento em tempo real ou processamento interativo. O processamento em tempo real permite que o usuário possa executar suas próprias análises em segundos sobre dados estruturados e não estruturados. Dentro desse grupo de soluções, podemos citar o Apache Drill[9] (versão *open source* do Google Dremel).

O Apache Drill é um sistema distribuído projetado para eficientemente explorar grande volume de dados estruturados e não estruturados em um *cluster* com milhares de

[4] http://storm.apache.org
[5] spark.apache.org/streaming
[6] https://hadoop.apache.org/docs/r1.2.1/streaming.html
[7] https://flink.apache.org
[8] https://kafka.apache.org/
[9] http://drill.apache.org/

servidores. Ele utiliza o HDFS para armazenamento e utiliza o MapReduce para análises em lote e possui capacidade de executar consultas sobre grande volumes de dados em segundos combinando árvores de execução multiníveis e dados no formato de colunas.

2.3 Considerações Finais

Este capítulo apresentou diversos conceitos fundamentais importantes para a construção de uma solução para processamento de grandes volumes de STSR. Devido ao custo computacional dos algoritmos de processamento das STSR, é esperado que as soluções para este problema explorem a escalabilidade vertical de cada máquina, com as arquiteturas paralelas Manycore e Multicore (Seção 2.2.1). No cenário de Remote Sensing Big Data, as soluções centralizadas que exploram a escalabilidade vertical de somente uma máquina não são capazes de processar um grande volume de dados, pois esbarram nas limitações de recursos computacionais de um único servidor. Por isso, é necessário explorar também a escalabilidade horizontal de um *cluster* de computadores, com soluções distribuídas que sejam eficientes no processamento em lote, em tempo real e de um grande fluxo de dados (Seção 2.2.2).

CAPÍTULO 3

Trabalhos correlatos

Este Capítulo apresenta uma visão geral do estado da arte das estratégias de processamento de STSR. A Seção 3.1 descreve diversas soluções centralizadas na classificação e detecção de mudanças do uso e cobertura do solo utilizando as STSR. Devido ao alto custo computacional dessas soluções centralizadas, diversos trabalhos apresentados na Seção 3.2 desenvolveram soluções paralelas para processamento das STSR. A limitação de recursos de *hardware* de um computador pode se tornar um gargalo no processamento de um grande volume de STSR. Diversos trabalhos, apresentados na Seção 3.3, foram propostos utilizando arquiteturas distribuídas para lidar com este cenário.

3.1 Processamento de Séries Temporais de Sensoriamento Remoto

Dentre as mais promissoras aplicações das STSR está a classificação e a detecção de mudanças do uso e cobertura do solo [96]. Esta seção apresenta soluções centralizadas para classificação e detecção de mudanças do uso e cobertura do solo utilizando as STSR. A detecção de mudanças é realizada com o objetivo de capturar a variabilidade ao longo do tempo dos diferentes tipos de uso e cobertura do solo [42]. O método LandTrendr realiza essa detecção particionando as séries temporais em segmentos lineares e interpreta componentes individuais [57]. Em [54] é proposto um novo método, *Detecting Breakpoints and Estimating Segments in Trend* (DBEST), para segmentação e análise de mudanças de tendência nas séries temporais das vegetações. O método *Breaks For Additive and Seasonal Trends* (BFAST) [115] possui uma abordagem semelhante ao método DBEST e é bastante utilizado para detectar mudanças abruptas nos componentes sazonais e de tendência, além de caracterizar a mudança gradual e abrupta derivando o tempo, magnitude e direção da mudança dentro do componente de tendência.

Para classificação do tipo de uso e cobertura do solo utilizando as STSR, alguns trabalhos foram propostos utilizando abordagens supervisionadas e não supervisionadas. Em [49], as áreas desmatadas e com queimadas entre 2001 e 2005 são mapeadas e em

[110] foi construído um classificador com redes neurais para definir a probabilidade de mudança de um pixel ao longo do tempo. Em [31] é apresentada uma nova técnica, utilizando um método supervisionado, para detectar transições de cobertura da Terra.

Diversos trabalhos, como [83, 84], propuseram abordagens não supervisionadas com menor intervenção manual. Em [105] é proposto um novo algoritmo para descoberta de padrões relevantes no clima utilizando as STSR. Em [51] é proposta uma nova forma de indexação das imagens utilizando a localização dos *pixels* para particionar as imagens. Em [117] foi proposto um algoritmo de classificação com dados de STSR de imagens de alta resolução.

O algoritmo *Dynamic Time Warping* (DTW) [9] é uns dos mais utilizados para classificação utilizando séries temporais. No entanto, ele não é recomendável para classificação de STSR, pois desconsidera a faixa temporal quando tenta encontrar o melhor alinhamento entre duas séries temporais [76]. Diante disso, alguns trabalhos [94, 95, 120] propuseram métodos que adaptam o algoritmo DTW para classificação de STSR.

Os métodos propostos em [94, 95] utilizam um atraso de tempo máximo para evitar distorções temporais baseados na data das imagens de satélite. Nesses trabalhos, duas séries temporais pertencem à mesma classe se tiverem o mesmo tamanho, onde o primeiro e o último ponto em ambas as séries temporais devem ser iguais. Entretanto, os pedaços de séries temporais que representam ciclos fenológicos podem variar de um ano pro outro, dependendo das condições climáticas e manejo do solo. Assim, uma pastagem identificada em um ano X entre os meses de fevereiro e julho, no ano $X+1$ pode ser identificada em outro período de meses.

Para evitar possíveis inconsistências nestes casos, o método TWDTW [120] (apresentado na Seção 2.1.1), busca encontrar todas as possíveis ocorrências de um determinado padrão dentro de uma série temporal. Por isso, o TWDTW não requer que ambas as séries temporais sejam do mesmo tamanho. Para evitar, por exemplo, que um padrão extraído do inverno não case com um padrão extraído do verão, o TWDTW introduz restrições temporais. Se existir uma grande diferença sazonal entre o padrão e as STSR que casam com esse padrão, um custo extra é adicionado à medida de distância do DTW. Essa restrição controla as distorções ao longo do tempo e torna o alinhamento da série temporal dependente das estações do ano. Esse método é útil para distinguir, por exemplo, regiões com pastagem de regiões com agricultura.

Existe um conjunto de trabalhos que combinam as STSR com métodos de aprendizado de máquina para classificação do tipo de uso e cobertura do solo. Em um problema típico de classificação, temos medidas que capturam os atributos de cada classe. Baseada nessas medidas, conhecidas como dados de treinamento, a tarefa consiste em selecionar um modelo que permita inferir a classe de um conjunto de dados de teste e

validação. Esses métodos de classificação incluem Máquina de Suporte Vetorial (Support Vector Machine ou SVM) [79], Florestas Aleatórias (Random Forest ou RF) [1, 92] e Redes Neurais (Neural Networks ou NN).

A popularidade dos algoritmos de Redes Neurais Profundas (Deep Learning ou DL) tem crescido de forma massiva nos últimos anos na análise e classificação de imagens de sensoriamento remoto. Uma revisão da literatura de 2019 [71] encontrou mais de 200 publicações nesta área, a maior parte publicada nos anos de 2017 e 2018. Este estudo foi conduzido analisando os modelos de DL utilizados, a resolução espacial das imagens, região geográfica de estudo e acurácia atingida na classificação. Uma revisão detalhada foi realizada para descrever e discutir como aplicar os métodos de DL para análise e classificação de imagens de sensoriamento remoto, cobrindo métodos desde o pré-processamento até o mapeamento do tipo de uso e cobertura do solo. Para a classificação utilizando as STSR, a maior parte dos trabalhos utilizam as Redes Neurais Recorrentes (RNN), analisando as observações ao longo do tempo dos *pixels* e determinando a classe de cada *pixel*, já que as RNNs são projetadas para lidar com dados sequenciais. Apesar do potencial das RNNs, o estudo de [71] aponta que ainda existem diversos desafios a ser explorados na classificação das STSR, utilizando os algoritmos de DL.

Uma abordagem alternativa é proposta em [96] (descrita com mais detalhes na Seção 2.1.2) utilizando todos os dados das STSR das bandas espectrais e índices de vegetação para criar características como entrada de métodos de aprendizado de máquina. A ideia é ter o máximo possível de atributos das séries temporais, aumentando a dimensão do espaço de classificação. O trabalho de [96] comparou sua solução com diversas outras, incluindo aquelas reportadas por [77], [93] e [56], tendo maior acurácia na classificação de diversos tipos de uso do solo. O trabalho concluiu que, escolhendo um classificador robusto em relação a ruídos, é possível atingir melhores resultados do que as soluções existentes. O trabalho de [96] avaliou diversos classificadores, incluindo o SVM e RF, e concluiu que o SVM teve a melhor acurácia na classificação do tipo de uso e cobertura do solo.

3.2 Abordagens Paralelas no Processamento de Séries Temporais de Sensoriamento Remoto

Devido ao alto custo computacional, diversos trabalhos desenvolveram soluções paralelas para análise e classificação de séries temporais. Esta seção apresenta algoritmos paralelos (arquiteturas *manycore* e *multicore*) para processamento das STSR permitindo reduzir o tempo de execução e aumentar a escalabilidade de algoritmos sequenciais.

Para trabalhar com dados de geração de mapas de deformação, alguns métodos

paralelos foram propostos [15, 53] utilizando séries temporais coletadas a partir de radares de abertura sintética (SAR). Radar de Abertura Sintética é um sistema imageador ativo de ondas de rádio, em geral microondas, embarcado em aeronaves (aerotransportado) ou satélites (orbital), utilizado para o sensoriamento remoto.

Diversas soluções que utilizam arquiteturas *manycore* (com GPUs) foram propostas para otimizar bancos de dados de processamento de séries temporais. Em [101] é proposto um novo banco de dados utilizando GPU e bancos de dados NoSQL para processar algoritmos de análises de séries temporais e em [100] um algoritmo de compressão para bancos de dados de processamento de séries temporais utilizando vários dispositivos GPUs combinados. Uma solução completa para processamento de consultas em bancos de dados de processamento de séries temporais é proposta em [99] utilizando um ambiente com arquitetura paralela híbrida com GPU e CPU.

A acurácia na estimativa de atraso de tempo dado pares de amostras irregulares de séries temporais possui grande relevância na área de astrofísica [24]. Em [67] é proposto um *framework* na GPU para detecção de mudanças em objetos utilizando STSR e informações espaciais.

3.3 Soluções para Processamento Distribuído de Séries Temporais em Remote Sensing Big Data

A limitação de recursos de *hardware* de um computador pode se tornar um gargalo no processamento de grandes volumes de dados de sensoriamento remoto. Para resolver este problema, novas abordagens têm sido recomendadas por vários trabalhos de revisão da literatura em Remote Sensing Big Data [69, 45, 19, 46, 21, 64] utilizando arquiteturas distribuídas, particularmente para suportar a demanda por processamento em tempo real [72].

O Google Earth Engine [43] é uma das plataformas mais utilizadas no processamento de STSR, utilizando a capacidade de processamento de grande volume de dados do Google, para atender a uma grande gama de cientistas que não têm acesso a *clusters* de computadores. Ele facilita o acesso aos dados através de uma API simples para os desenvolvedores de aplicações em sensoriamento remoto. Por ser um ambiente compartilhado, o Google Earth Engine possui uma cota por usuários ou consultas concorrentes [1] para prevenir o impacto negativo da disponibilidade dos serviços, que pode ser ocasionado pelo uso excessivo de um grupo de usuários.

[1] https://developers.google.com/earth-engine/usage

Um esforço realizado pelo governo da Austrália construiu cubos de dados que permitem que os cientistas extraiam novas informações ricas de um grande volume de STSR, lidando com os desafios de Remote Sensing Big Data [61]. Esses cubos de dados contém as observações de vários *pixels* ao longo tempo, permitindo que mudanças na cobertura do solo possam ser identificadas [122]. Essa iniciativa do governo da Austrália permitiu que outras organizações explorassem o uso de cubos de dados similares em diversos países.

As pesquisas descrevem o uso de Big Data em diversos domínios de sensoriamento remoto. O trabalho de [121], por exemplo, apresenta diversas aplicações de Big Data em Smart Farming [2] para lidar com os desafios de processamento do grande volume de séries temporais geradas através dos dados de agricultura extraídos de sensores, atuadores, GPS, drones e imagens de satélites.

Com o surgimento da computação nas nuvens, os usuários passaram a poder acessar os dados a qualquer momento de qualquer lugar e conduzir análises sobre as séries temporais. Em vista dessa facilidade, vários sistemas de computação distribuída foram construídos para o armazenamento e processamento de grande volume de dados de sensoriamento remoto utilizando os recursos de computação nas nuvens [124, 47]. O projeto Earth Observation Data Centre (EODC) [116], para monitoramento dos recursos hídricos do mundo, foi um dos primeiros que provisionou recursos computacionais e aplicações onde os dados estão armazenados diretamente. O EODC disponibiliza uma variedade de dados no ambiente de nuvem com foco em facilitar o acesso de usuários não especialistas aos dados.

Aliado a computação nas nuvens, o uso do modelo de programação MapReduce causou uma revolução no processamento e gerenciamento de dados de sensoriamento remoto [44]. O MapReduce facilitou a construção de diferentes arquiteturas de computação distribuída para lidar com Big Data. Em [70] foi implementado um algoritmo de clusterização K-means sobre imagens de sensoriamento remoto, utilizando o modelo de programação MapReduce no Hadoop [48]. O modelo MapReduce foi utilizado em [2] para criar uma plataforma de processamento paralelo de imagens de sensoriamento remoto com um sistema de computação nas nuvens. Eles desenvolveram um método de programação eficiente e customizado para o processamento utilizando o *framework* Hadoop, realizando testes em um *cluster* com 112 núcleos.

Para facilitar o desenvolvimento de aplicações Web com o processamento de grande volume de imagens de sensoriamento remoto, em [63] foi proposta e implementada uma arquitetura de uma plataforma de computação nas nuvens baseada nas interfaces

[2] Smart Farming é baseada na integração das aplicações digitais e novas tecnologias de comunicação, introduzindo a Inteligência Artificial na agricultura.

WCS e WMS. As imagens são armazenadas no HDFS, processadas utilizando o Hadoop e os resultados apresentados em *websites*.

Para o processamento de análises de imagens de sensoriamento remoto no contexto *Big Data*, a proposta de [103] apresenta uma arquitetura de uma solução, utilizando o Hadoop, com três componentes principais: 1) unidade de aquisição de grande volume de dados de sensoriamento remoto; 2) unidade de processamento de análise sobre os dados que estão sendo coletados; e 3) unidade de armazenamento e geração dos resultados das análises sobre os dados.

Para processar séries temporais de objetos espaciais, [108] propõe a plataforma Spatiotemporal utilizando computação nas nuvens. A plataforma possui uma aplicação Web que permite facilmente definir o conjunto de análises a serem realizadas sobre imagens de satélite. O ArcGIS é utilizado como servidor para análises espaciais e quando é necessário o processamento de grande volume de dados, o HDFS é utilizado como sistema de arquivos e um serviço de computação baseado no MapReduce é utilizado para análise dos dados espaciais. A plataforma particiona os dados de forma que dados da mesma região geográfica e obtidos no mesmo período de tempo sejam armazenados no mesmo bloco HDFS e, consequentemente, no mesmo conjunto de máquinas. Desta forma, o tráfego de dados na rede é reduzido, pois os algoritmos espaciais sobre dados temporais acessam objetos espacialmente e temporalmente próximos.

Para facilitar pesquisas na área de análises sobre STSR, a plataforma HiTempo é proposta em [114]. As imagens foram armazenadas como um objeto 3D, onde imagens próximas geograficamente e obtidas no mesmo intervalo de tempo são armazenadas na mesma máquina. A HiTempo foi projetada para facilitar a avaliação e comparação de algoritmos na área de sensoriamento remoto e, por isso a filtragem espaço-temporal é realizada apenas antes de inserir as imagens na plataforma, não permitindo filtragens durante as consultas. Por isso, não é possível realizar filtragem espacial e temporal de forma eficiente no momento da consulta sobre o conjunto de imagens armazenadas na plataforma, limitando o usuário da solução.

Para executar análises de STSR em tempo real, [4] propõe uma solução de processamento de *streaming* de dados utilizando o Hadoop Streammig como *framework*. Nesta solução, cada pixel é tratado individualmente como um par <chave, valor>, onde a chave é a localização do pixel na imagem e o valor contém a série temporal dos valores do pixel para uma determinada região.

A evolução da resolução dos dados de sensoriamento remoto aliada ao aumento do número e variedade de satélites, além do aumento de demanda por consumir esses dados do lado da aplicação, levaram ao surgimento dos bancos de dados vetoriais multidimensionais para o armazenamento, gerenciamento, processamento e análise das STSR, tais como o SciDB [10] e o Rasdaman [6]. Os projetos EarthServer e EarthServer-

2 utilizaram o banco de dados Rasdaman para armazenamento e processamento de séries temporais [7], permitindo filtrar, consultar, extrair e processar os dados utilizando os padrões WPS (Web Processing Service) [37] e WCPS (Web Coverage Processing Service) [5] da OGC (Open Geospatial Consortium).

Em [68] são avaliadas as correlações temporais e espaciais do algoritmo BFAST, utilizando como estudo de caso a detecção de mudanças nas áreas de florestas com STSR de imagens MODIS. No trabalho é mostrado como análises de séries temporais baseadas em pixel podem ser estendidas para análises espaço-temporais. Para o processamento de grande volume de imagens, as imagens são armazenadas utilizando um banco de dados multidimensional, denominado SciDB[3], que divide os dados multidimensionais em pedaços iguais e distribui esses pedaços pelo *cluster* de computadores. Como o SciDB não possui interfaces para consultas espaço-temporais, o trabalho utiliza uma extensão [13] com um interpretador de consultas para analisar tipos de dados espaço-temporais.

O trabalho de [12] propõe uma arquitetura amigável para os pesquisadores na área de sensoriamento remoto, que podem executar programas na linguagem R para processar um grande volume de dados em um *cluster* de computadores. Para validar a arquitetura, foi desenvolvido um algoritmo para análise de STSR comparando duas soluções de Big Data: modelo MapReduce com o Hadoop e o banco de dados vetorial SciDB. De forma geral, o SciDB teve melhor desempenho que o Hadoop nessa análise.

3.4 Considerações Finais

Neste capítulo foram apresentadas propostas existentes para o processamento de STSR. A Seção 3.1 descreve várias soluções para a classificação e detecção de mudanças do tipo de uso e cobertura do solo utilizando as STSR. A melhor solução encontrada, foi proposta em [96], que utilizou as observações das STSR das bandas espectrais e índices de vegetação como características de métodos de aprendizado de máquina. Nosso trabalho estendeu esta solução de [96], criando meta-características a partir das medidas de similaridade calculadas no TWDTW e usando como entrada para diversos classificadores, como Deep Learning. Nossa solução, descrita na Seção 4.1.2, teve melhor acurácia que o método apresentado em [96].

Devido ao alto custo computacional, diversas soluções paralelas foram propostas para o processamento de STSR. Na Seção 3.2 foram descritos diversos trabalhos que utilizam arquiteturas *multicore* e *manycore* para processamento de STSR. Alguns trabalhos, como [115, 54, 77], também possuem suporte para execução paralela em arquiteturas

[3] http://www.paradigm4.com

multicore, utilizando a biblioteca *foreach*[4] do R. Estas soluções, entretanto, não foram projetadas para explorar o potencial das arquiteturas paralelas *manycore* no contexto de Big Data. Nosso trabalho propõe um novo algoritmo paralelo, P-TWDTW, descrito na Seção 4.1.1, que explora arquiteturas *manycore* com milhares de núcleo para o cálculo eficiente das medidas de similaridade do TWDTW.

Para o processamento de grande volumes de dados, diversos trabalhos foram apresentados, na Seção 3.3, utilizando arquiteturas distribuídas. Nenhum trabalho encontrado, entretanto, preenche os requisitos levantados na Seção 2.2.2 para o processamento distribuído no cenário de Remote Sensing Big Data. Alguns trabalhos [70, 2, 63, 114, 108] possuem sistemas de armazenamento e processamento em lote eficientes, mas não discutem a respeito de técnicas de indexação espacial e temporal, de forma a filtrar os dados que atendem restrições de consulta. Outros trabalhos [12, 68, 4] apresentaram propostas de processamento de séries temporais através do processamento de fluxo de dados, mas não apresentaram soluções completas para lidar com processamento em lote de grande volume de dados de sensoriamento remoto. Nosso trabalho apresenta, no Capítulo 5, a proposta de uma plataforma Big Data, denominada BigSensing, para o processamento distribuído e paralelo de um grande volume de dados de STSR.

Diversas soluções de plataformas foram propostas na literatura, como pode ser visto em [69, 45, 19, 46, 21, 64], contendo vários sistemas de Big Data para processamento em lote ou tempo real, tais como o Hadoop, Spark, Elasticsearch e SciDB. Experimentos realizados em alguns trabalhos [33, 126] mostram que nenhum sistema tem desempenho melhor que os outros em todos os cenários com filtros espaciais e temporais. Nosso trabalho propõe um motor inteligente na plataforma BigSensing, denominado SmarT (smart Spatial-Temporal query engine), que tem como objetivo aplicar os filtros espaciais e temporais e recuperar os dados das STSR, buscando escolher, entre diversos sistemas em tempo real, aquele mais eficiente.

Os trabalhos de revisão da literatura em Remote Sensing Big Data [69, 45, 19, 46, 21, 64] mostram que não existe uma proposta de uma solução eficiente para explorar, além da escalabilidade horizontal de um *cluster*, o poder de arquiteturas paralelas *manycore* e *multicore* em cada servidor no processamento das STSR para a classificação do tipo de uso e cobertura do solo. Aliando à escalabilidade vertical, explorando arquiteturas *multicore* na CPU e *manycore* na GPU com o P-TWDTW e o P-INDEX, com escalabilidade horizontal, utilizando o poder computacional de um *cluster* de computadores com a plataforma BigSensing, nosso trabalho buscou resolver um problema em aberto na área de Remote Sensing Big Data de permitir o processamento paralelo e distribuído eficiente de um grande volume de dados de STSR.

[4] https://cran.r-project.org/web/packages/foreach/index.html

CAPÍTULO 4

Explorando Arquiteturas Paralelas para o Processamento de Séries Temporais de Sensoriamento Remoto

Este Capítulo apresenta a solução proposta nesta tese para o processamento paralelo de STSR, utilizando arquiteturas *manycore* e *multicore*. A Seção 4.1.1 propõe um novo algoritmo, denominado P-TWDTW, que é capaz de paralelizar de forma eficiente o algoritmo TWDTW no cálculo das medidas de similaridade das STSR. Nosso trabalho propõe criar meta-características a partir dessas medidas de similaridade, na Seção 4.1.2, como entrada de métodos de aprendizado de máquina com o objetivo de aumentar a acurácia da classificação do tipo de uso e cobertura do solo utilizando as STSR. A Seção 4.2 apresenta o algoritmo P-INDEX, que utiliza arquiteturas paralelas no cálculo dos índices de sensoriamento remoto que irão compor os dados de entrada para algoritmos de processamento das STSR, tais como o P-TWDTW.

4.1 Processamento Paralelo de Séries Temporais de Sensoriamento Remoto utilizando GPUs

O TWDTW é implementado por um algoritmo de programação dinâmica com alto custo computacional e complexidade de tempo $O(n^2)$, o que o torna inviável para um grande volume de dados. Esta seção apresenta a descrição da solução proposta neste trabalho para o processamento paralelo das STSR utilizando arquiteturas *manycore* (GPU). Esta solução, denominada P-TWDTW (Parallel TWDTW), possui complexidade de tempo $O(\frac{n^2}{p})$, onde p é o número de núcleos disponíveis.

A matriz de custos acumulados, que denotaremos por D, é calculada a partir da matriz de custo, denominada Ψ. A construção da matriz D não pode ser paralelizada de forma trivial, já que o cálculo de cada elemento (i, j) da matriz depende dos elementos $(i-1, j)$, $(i, j-1)$ e $(i-1, j-1)$. Estes devem, portanto, ser previamente calculados. Essa dependência é ilustrada na Figura 4.1.

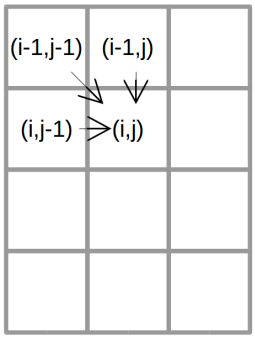


Figura 4.1: *O cálculo de cada elemento (i,j) da matriz D depende dos valores calculados para os elementos anteriores* $(i-1, j)$, $(i, j-1)$ *e* $(i-1, j-1)$.

A ideia do algoritmo P-TWDTW para calcular a matriz D de forma paralela é ilustrada na Figura 4.2. Os elementos de cada diagonal secundária são calculados de forma paralela, onde cada *thread* fica responsável por um de seus elementos. Como tais elementos não são dependentes entre si, o cálculo dos custos acumulados pode ser realizado, em paralelo, sem o risco de gerar inconsistências. Os detalhes do cálculo da matriz D do P-TWDTW são apresentados no Algoritmo 4.1.

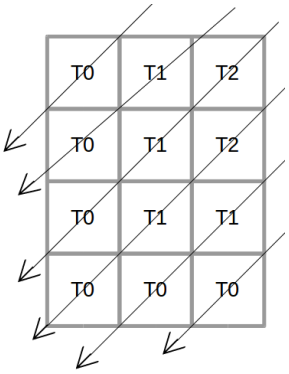


Figura 4.2: *P-TWDTW: ilustração do cálculo de D de forma paralela para cada diagonal secundária.*

4.1.1 P-TWDTW: Parallel Time-Weighted Dynamic Time Warping

O Algoritmo 4.1 descreve o P-TWDTW, que recebe como entrada o conjunto Q de padrões e o conjunto S de séries temporais, para calcular a matriz de custos acumulados D entre cada padrão $U \in Q$ e cada série temporal $V \in S$. O P-TWDTW fica responsável por coordenar o carregamento de blocos de Q e S para a memória RAM e, posteriormente, para a memória da GPU, de forma que não exceda os limites das mesmas. O P-TWDTW também recebe como entrada o número que representa o tamanho máximo dos conjuntos Q e S que cabem na memória da GPU (bQ e bS respectivamente) e na memória RAM

(*max_bQ* e *max_bS* respectivamente). O algoritmo tem como saída várias matrizes D de custos acumulados que serão retornadas como resultado final do processamento.

O algoritmo inicia nas linhas 1 a 3 lendo os blocos dos padrões para a fila *queueQ* e blocos das séries temporais para a fila *queueS*. Esse trabalho é realizado por uma *thread* na CPU, que coordena o tamanho da fila e da memória RAM, de forma que não exceda o seu limite disponível e que as filas não fiquem vazias. Desta forma, não é preciso esperar finalizar esta etapa para começar a carregar os blocos para a memória global da GPU. Entre as linhas 4 e 29, enquanto houver blocos para serem processados, o Algoritmo 4.1 carrega os blocos para a memória da GPU e calcula a matriz D.

Nas linhas 5 a 8, bQ padrões e bS séries temporais são carregados para a memória global, juntando os padrões em um único padrão *bigQ* e as séries temporais em uma só, denominada *bigS*. Essa união permite que a GPU possa realizar o processamento em bloco explorando seu poder computacional. Na linha 9, a matriz de custo Ψ é construída a partir de *bigQ* e *bigS*, com cada *thread* da GPU sendo responsável pelo cálculo de um elemento da matriz.

O cálculo da matriz D é realizado seguindo a ideia apresentada na Figura 4.2, onde cada *thread* fica responsável pelo cálculo do custo de cada elemento da diagonal. Como cada elemento da diagonal depende apenas das duas diagonais anteriores, eles podem ser calculados de forma independente. Como *bigQ* e *bigS* possui vários padrões U e séries temporais V, são lançados $bQ * bS$ blocos de *threads* na GPU, onde cada bloco de *threads* fica responsável pelo cálculo dos custos acumulados entre um padrão U e uma série temporal V. Cada bloco de *threads* na GPU possui $min(sizeU, sizeV)$ *threads* que é o tamanho máximo de uma diagonal na comparação entre U e V, para que cada *thread* realize o cálculo de um elemento da diagonal.

Entre as linhas 14 e 20 são calculados os custos dos elementos das primeiras *sizeU* diagonais superiores à diagonal secundária. Em uma matriz quadrada, estas primeiras *sizeU* diagonais representam a matriz triangular superior. No Algoritmo 4.1 admite-se que o índice inicia na posição 0. Para identificar cada diagonal, o índice da linha na matriz superior é calculado como $si - tid$ (tid é o índice da *thread*) e o índice da coluna é determinado pelo id da *thread*. A matriz D é atualizada para cada elemento da diagonal utilizando o Algoritmo 4.2.

Entre as linhas 21 e 26, o algoritmo calcula os elementos das próximas $sizeV - 1$ diagonais. Em uma matriz quadrada, por exemplo, essas $sizeV - 1$ diagonais representam a matriz triangular inferior. Para identificar cada diagonal, o índice da linha na matriz superior é calculado como $sizeU - tid - 1$ e o índice da coluna é definido como $sizeV - sj - tid - 1$. A matriz D é então atualizada para cada elemento da diagonal.

Algoritmo 4.1: P-TWDTW(Q, bQ, max_bQ, S, bS, max_bS)

Entrada: Q: conjunto de padrões a serem utilizados na consulta
bQ: número de padrões processados ao mesmo tempo
max_bQ: número de padrões na memória
S: conjunto de séries temporais
bS: número de séries temporais processadas ao mesmo tempo
max_bS: número de séries temporais na memória
Saída: Matriz de custos acumulados D

```
1  enquanto Houver novos padrões em Q e novas séries temporais em S faça
2      queueQ ← carregue max_bQ padrões para a memória RAM
3      queueS ← carregue max_bS séries temporais para a memória RAM
4      enquanto Houver novos padrões em queueQ e novas séries temporais em queueS faça
5          gpu_queueQ ← carregue bQ padrões para a memória global da GPU
6          gpu_queueS ← carregue bS séries temporais para a memória global da GPU
7          bigQ ← junte todos os padrões em gpu_queueQ
8          bigS ← junte todos as séries temporais em gpu_queueS
9          Ψ ← calcule a matriz de custo, de forma trivialmente paralela, entre bigQ e bigS
10         sizeU ← calcula o tamanho de cada padrão U dentro de bigQ
11         sizeV ← calcula o tamanho de cada série temporal V dentro de bigS
12         tid ← id da thread
13         se tid < sizeU então
14             para si de 0 até sizeU - 1 faça
15                 se tid ≤ min(si, sizeV − 1) então
16                     i ← si − tid
17                     j ← tid
18                     update_accumulated_cost_matrix (Ψ, D, i, j, sizeU, sizeV)
19                 fim
20             fim
21             para sj de sizeV − 2 até 0 faça
22                 se tid ≤ min(sj, sizeU − 1) então
23                     i ← sizeU − tid − 1
24                     j ← sizeV − sj − tid − 1
25                     update_accumulated_cost_matrix (Ψ, D, i, j, sizeU, sizeV)
26                 fim
27             fim
28         fim
29     fim
30     Retorna a matriz D;
31 fim
```

O Algoritmo 4.2 é responsável por atualizar cada elemento $D_{i,j}$ da matriz D de custos acumulados. O cálculo de $D_{i,j}$ segue a equação 2-4, que é o menor valor entre $D_{i-1,j}$, $D_{i-1,j-1}$ e $D_{i,j-1}$ somado ao custo $\Psi_{i,j}$. O índice nas matrizes Ψ e D relativo a i, j das matrizes U e V são calculados porque as matrizes Ψ e D são enviadas como vetores para a GPU e o conjunto de séries temporais V e padrões U são unidos em blocos de tamanho bQ e bS respectivamente. Ou seja, é preciso encontrar o par U e V dentro de D e Ψ e depois deslocar o índice no vetor de forma a encontrar a posição correta na matriz relativa a U e V. Na GPU são lançados $bQ * bS$ blocos de *threads*, sendo bQ no eixo x e bS no eixo y para que cada bloco de *threads* seja relativo a um bloco em Ψ e D.

Para calcular o índice global das matrizes Ψ e D utilizamos a equação 4-1, onde $(i + blockIdx.x * sizeU) * sizeV * gridDim.y$ encontra a linha correta em Ψ e D, sendo $blockIdx.x$ o id do bloco de *threads* no eixo x e $gridDim.y$ o número de blocos no eixo y. O trecho $(j + blockIdx.y * sizeV)$ encontra dentro da linha a posição correta do elemento i, j, sendo $blockIdx.y$ o id do bloco de *threads* no eixo y.

$$index = (i + blockIdx.x * sizeU) * sizeV * gridDim.y + (j + blockIdx.y * sizeV) \tag{4-1}$$

Algoritmo 4.2: update_accumulated_cost_matrix(Ψ, D, i, j, $sizeU$, $sizeV$)

Entrada: Ψ: matriz de custo de entrada
D: matriz de custos acumulados
i: índice do padrão na matriz de custo
j: índice da série temporal na matriz de custo
$sizeU$: tamanho do padrão
$sizeV$: tamanho da série temporal

1 $index_{i,j} \leftarrow$ calcule o índice global em Ψ que corresponde a i e j
2 $index_{i-1,j-1} \leftarrow$ calcule o índice global em Ψ que corresponde a $i-1$ e $j-1$
3 $index_{i-1,j} \leftarrow$ calcule o índice global em Ψ que corresponde a $i-1$ e j
4 $index_{i,j-1} \leftarrow$ calcule o índice global em Ψ que corresponde a i e $j-1$
5 $D[index_{i,j}] \leftarrow min(D[index_{i-1,j}], D[index_{i-1,j-1}], D[index_{i,j-1}) + \Psi[index_{i,j}]$

Depois de construída a matriz D de custos acumulados, o algoritmo calcula o alinhamento final entre cada padrão U e cada série temporal V, de acordo com a equação 2-6. Para cada alinhamento é calculado o custo total que será utilizado como entrada para os algoritmos de classificação do tipo de uso e cobertura do solo, utilizando as medidas de similaridade das STSR.

4.1.2 Utilizando as Medidas de Similaridade do P-TWDTW como Meta-características dos Algoritmos de Aprendizado de Máquina na Classificação do Tipo de Uso e Cobertura do Solo

No processamento de STSR, cada padrão de uma amostra de séries temporais pertence a uma certa classe que está relacionada a um tipo de uso de solo, como por exemplo, pastagem. O padrão é gerado a partir de uma amostra de séries temporais da classe dada. A Figura 4.3 apresenta um exemplo da geração de um padrão a partir de um conjunto de séries temporais de "Pastagem" utilizando a mediana (linha vermelha). Essa nova série temporal, gerada a partir da mediana, representa nesse caso, o padrão da classe "Pastagem". Existem também outras abordagens na geração de padrões. É o caso, por exemplo, dos métodos aditivos generalizados (GAM) [120].

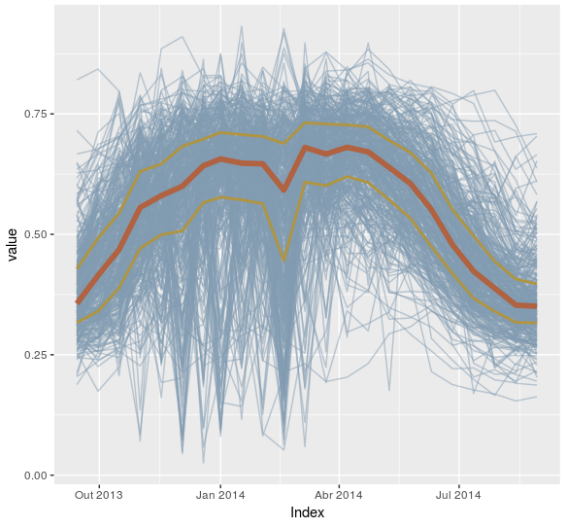


Figura 4.3: *Geração do padrão de uma amostra de séries temporais de uma classe (neste caso, "Pastagem") a partir da mediana dos valores (linha vermelha), p25 e p75 (linhas amarelas).*

Os melhores resultados de acurácia utilizando as medidas de similaridade do TWDTW na classificação das STSR têm sido obtidos utilizando a abordagem do KNN com K=1 (1NN) [76], detalhado na Seção 2.1.1. Entretanto, para regiões geográficas em que as amostras de dados não conseguem capturar o padrão correto de cada classe, o TWDTW tende a não ter uma acurácia muito alta [77]. No exemplo da Figura 4.3 é possível perceber que devido à variabilidade das séries temporais é praticamente impossível definir um único padrão de série temporal para "Pastagem". A solução proposta por [96], descrita na Seção 2.1.2, reduz esse impacto utilizando todos os valores das séries temporais das bandas e índices de vegetação como características para os algoritmos de aprendizado de máquina, criando espaços altamente dimensionais.

Esta tese propõe um novo método de classificação que utiliza, além das bandas e índices de vegetação de [96], as medidas de similaridade das séries temporais para cada padrão, calculadas pelo P-TWDTW, criando meta-características para os algoritmos de aprendizado de máquina. Formalmente, dada uma série temporal V e um conjunto de padrões Q = {U_1, U_2, ..., U_n}, A solução deste trabalho constrói um vetor de meta-características de V, F-TWDTW(V), utilizando o P-TWDTW, da seguinte forma: F-TWDTW(V) = (P-TWDTW(V, U_1), P-TWDTW(V, U_2), ..., P-TWDTW(V, U_n)). Assim, são adicionadas a cada tupla do modelo de dados de entrada do classificador n características, onde n é o número de padrões em Q. Como cada padrão está relacionado a uma classe específica, então n também representa o número de classes à qual cada pixel pode pertencer.

A criação das meta-características com o P-TWDTW para métodos de aprendizado pode ser exemplificada da seguinte forma. Suponha que existam três classes: "Floresta", "Pastagem" e "Desmatamento". Neste caso, teremos 3 padrões Q = {U_1, U_2, U_3}, um para cada classe. Para cada série temporal V (referente a um pixel), é calculada a medida de similaridade entre V e cada $U \in Q$. Supondo que a menor distância de V seja para o padrão U_2 referente a "Pastagem", o método 1NN, utilizando a medida de similaridade do P-TWDTW, iria atribuir a classe "Pastagem" a V. Quando se utiliza as medidas de similaridade do P-TWDTW como entrada para os métodos de aprendizado de máquina, são geradas, neste caso três novas características com as medidas de similaridade de V para cada $U \in Q$. Esse modelo de dados é, então, utilizado como entrada para os algoritmos de aprendizado de máquina.

A justificativa para criar essas meta-características é que, embora tanto o 1NN quanto o F-TWDTW utilizem as medidas de similaridade do P-TWDTW, o 1NN se vale de um número fixo de vizinhos, neste caso 1, para decidir a classe à qual V pertence. Por outro lado, os métodos de aprendizado utilizam funções complexas sobre a entrada de dados para melhorar a acurácia de classificação utilizando um conjunto de dados de treinamento. Por exemplo, esses métodos são capazes de, durante o treinamento, reduzir o peso de uma amostra de treinamento que, isoladamente, seria o vizinho mais próximo. Algumas classes, por exemplo "Plantio de Milho" e "Plantio de Milheto" [96], têm características semelhantes nas suas séries temporais, o que acaba gerando medidas de similaridade muito próximas e, assim, gerando confusão no classificador 1NN durante a escolha do mais próximo. O F-TWDTW pode aprender a identificar esses casos e reduzir o peso dessas amostras na base de treinamento, o que não é possível com o 1NN. O F-TWDTW busca, assim, uma hipótese de classificação mais complexa do que é usada pelo 1NN e, portanto, é esperado que o F-TWDTW funcione melhor que o 1NN, em geral.

O método de aprendizado de máquina é capaz também de aprender, a partir da base de treinamento, as características que são importantes para cada base de dados

e, assim, agregar os pontos fortes do 1NN com P-TWDTW na classificação. Neste trabalho, foi escolhido o método de aprendizado supervisionado Support-Vector Machines (SVM) [22], que é o mesmo utilizado no trabalho de [96], que realizou experimentos com diversos métodos de aprendizado para definir que o SVM teve a melhor acurácia utilizando as bandas espectrais e índices de vegetação como características.

4.2 P-INDEX: Algoritmo para Processamento Paralelo de Índices de Sensoriamento Remoto

Nesta seção é apresentada a descrição da solução de processamento paralelo do cálculo de índices de sensoriamento remoto para grandes volumes de imagens de satélite, denominada P-INDEX. Esses índices são importantes para compor as observações das STSR que serão utilizadas como entrada para os algoritmos de processamento de séries temporais, tais como o P-TWDTW. Geralmente, esses índices não são fornecidos pelas fontes de dados e, por isso, devem ser calculados a partir dos dados das imagens de satélite.

O algoritmo P-INDEX utiliza todo o poder de processamento disponível, lidando com os limites de memória da máquina. Se a máquina tiver alguma GPU disponível, esta será utilizada para calcular o índice de sensoriamento remoto devido ao seu alto poder de processamento e à grande quantidade de núcleos. Caso contrário, os núcleos da CPU serão utilizados para o cálculo. O Algoritmo 4.3 apresenta a descrição do P-INDEX [1] que se inicia com a leitura de um bloco da imagem de entrada. O tamanho desses blocos de imagem devem ser definidos, na entrada dos dados, a partir da memória da CPU disponível em uma máquina, de forma que o bloco não ultrapasse o limite da memória. O algoritmo recebe como entrada um ou mais objetos no formato matricial que representa uma imagem de sensoriamento remoto. Por isso, na linha 1, o algoritmo itera sobre a lista de objetos, processando um objeto por vez.

Na linha 2 do algoritmo, as bandas da imagem necessárias para o cálculo do índice de sensoriamento remoto são lidas a partir do objeto que contém os dados da imagem e armazenadas, cada banda como uma matriz, na lista de matrizes *BANDS*. Esta lista *BANDS* pode ter tamanho variável, já que cada índice de sensoriamento remoto é calculado usando um número diferente de bandas da imagem. Por exemplo, o índice EVI utiliza as bandas RED, BLUE e NIR, enquanto o índice de vegetação NDVI utiliza as bandas NIR e RED. Se a máquina tiver uma GPU disponível, o algoritmo de cálculo do NDVI é processado na GPU (linhas 4 a 11) e, caso contrário, é processado na CPU (linhas

[1] O algoritmo P-INDEX foi implementado utilizando a biblioteca GDAL[2] para ler e escrever blocos de imagem.

14 a 16). O algoritmo de processamento paralelo na GPU é apresentado em detalhes na Seção 4.2.2 e o algoritmo de processamento na CPU é apresentado na Seção 4.2.1.

Algoritmo 4.3: P-INDEX

Entrada: *images_list*: lista de objetos, *bandas*: lista de bandas da imagem utilizadas no cálculo do índice de sensoriamento remoto
Saída: *result_list*: Lista de objetos com os Índices de sensoriamento remoto calculados

```
1  enquanto Tiver mais objetos de entrada em images_list faça
2      BANDS ← lista de matrizes referentes às bandas do objeto que contém a imagem
3      se Existir uma GPU disponível na máquina então
4          V ← converta a lista de matrizes em BANDS em vetor
5          bgpu ← tamanho da memória da GPU
6          enquanto Tiver mais blocos de tamanho bgpu a partir de V faça
7              V' ← próximo bloco de tamanho bgpu a partir de V
8              Transfira o vetor V' para a GPU
9              R ← calcule o índice de sensoriamento remoto na GPU
10             Transfira o vetor R da GPU para a CPU
11             Converta o vetor R na matriz C
12         fim
13     senão
14         B ← n blocos de tamanho bcpu a partir das matrizes BANDS
15         Balanceie os blocos B entre as threads
16         C ← calcule o índice de sensoriamento remoto utilizando os blocos B
             correspondentes
17     fim
18     result_list ← adicione a matriz C
19 fim
20 retorna result_list
```

O algoritmo gera um bloco resultante que é adicionado ao resultado final, *result_list*, na linha 18. Se houver mais objetos, o próximo é lido e o algoritmo, na linha 1, processa o novo objeto. Se não houver mais objetos para serem lidos, o algoritmo é finalizado e *result_list* retornado como resultado.

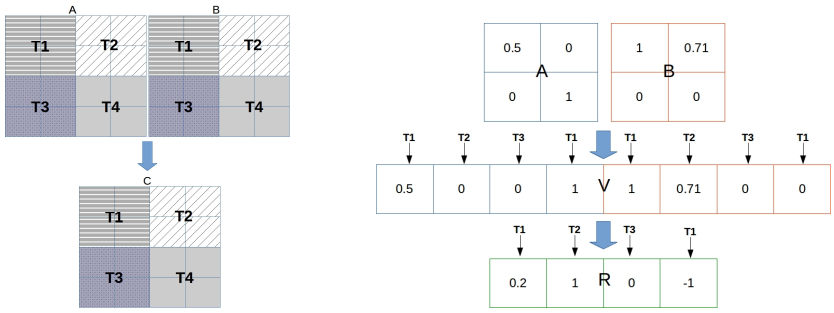
4.2.1 Processamento paralelo do índice de sensoriamento remoto utilizando uma arquitetura *multiCore* (CPU)

O processamento do índice de sensoriamento remoto utilizando uma arquitetura *multicore* consiste em utilizar dois ou mais núcleos da CPU para calcular o índice. O processamento paralelo utilizando a CPU é realizado com cada núcleo da CPU sendo

responsável por um bloco da imagem. No algoritmo P-INDEX na CPU, é criado um número de *threads* igual ao número de núcleos da CPU. Para que seja possível balancear o processamento entre os núcleos da CPU, são geradas partições das matrizes A e B de entrada para distribuir entre estas *threads*, onde cada *thread* fica responsável por uma partição com tamanho igual ao do bloco de entrada.

Na linha 14 do Algoritmo 4.3, as matrizes presentes na lista *BANDS* são divididas em n blocos iguais de tamanho $bcpu$, onde n é o número de núcleos disponíveis na máquina. Na linha 15, esses blocos são distribuídos entre as *threads*, de forma que cada *thread* fique responsável por partições iguais do bloco. Cada *thread* calcula o índice NDVI e armazena o resultado dessa partição da imagem na matriz C (linha 18 do Algoritmo 4.3).

A Figura 4.4(a) apresenta um exemplo do algoritmo P-INDEX na CPU calculando o índice de vegetação NDVI. Cada *thread* fica responsável por processar blocos de mesmo tamanho das matrizes A e B, onde a matriz *A* contém os valores da banda RED e matriz *B* os valores da banda *NIR*. Estas matrizes são inseridas na lista *BANDS* e são retiradas em formato de blocos de tamanho $bcpu$ para serem processadas nos núcleos de processamento da CPU. Cada *thread* lê as partições nas mesmas posições da matriz A e B e escreve o resultado na área correspondente da matriz C.



(a) *Exemplo do algoritmo NDVI na CPU utilizando o P-INDEX*

(b) *Exemplo de processamento do NDVI na GPU utilizando o P-INDEX*

Figura 4.4: *Exemplo do algoritmo P-INDEX na CPU e GPU.*

4.2.2 Processamento paralelo do índice de sensoriamento remoto utilizando a GPU

O algoritmo P-INDEX foi implementado para ser processado na GPU utilizando a linguagem CUDA da NVIDIA. Com ela é possível paralelizar o processamento de algoritmos, utilizando a grande quantidade de núcleos de processamento presentes na GPU. Na linha 4 do Algoritmo 4.3, a lista *BANDS* é convertida em vetor V, transformando cada matriz, dentro da lista *BANDS*, em um vetor e os concatenando em um único vetor V, havendo assim apenas uma transferência da CPU para a GPU.

Como a memória da CPU é geralmente maior que a memória da GPU, o vetor V pode não caber na memória da GPU. Por isso, são gerados blocos V' com tamanho b igual a dois terços da memória da GPU a partir do vetor V, visto que um terço deve estar reservado para armazenar o vetor R resultante do processamento da GPU.

Entre as linhas 7 e 11 do Algoritmo 4.3, os blocos gerados são lidos dentro do vetor V. Na linha 7, o bloco V' é obtido a partir das primeiras b posições do vetor V. Os blocos seguintes são obtidos a partir das posições subsequentes. O vetor V' é transferido para a GPU, na linha 8, e o cálculo do índice de vegetação executado na GPU na linha 9.

Dentro da GPU, o algoritmo inicia calculando o índice global da *thread* no processamento. Para aumentar o desempenho de acesso à memória global da GPU, cada *thread* acessa posições no vetor V' correspondentes ao índice global com *threads* acessando regiões próximas na memória global da GPU. Esse acesso aglutinado das *threads* otimiza o acesso à memória global da GPU, visto que blocos podem ser lidos da memória global e utilizados por várias *threads* ao mesmo tempo. Como o tamanho do vetor V' de entrada pode ser maior que o número de *threads*, cada *thread* é responsável por calcular o índice de várias posições do vetor.

Em cada iteração dentro da GPU, a *thread* obtém o deslocamento que representa a mesma posição de cada vetor referente a cada banda da imagem. O resultado é calculado, utilizando os valores obtidos, e armazenado no vetor de resposta. A resposta é escrita no vetor R com o resultado do cálculo do índice. O vetor R é então transferido da GPU para a CPU na linha 10 do Algoritmo 4.3. Na linha 11, o vetor R é agora convertido na matriz resultante C. Se houver mais blocos da imagem disponíveis no vetor V, o algoritmo lê o próximo bloco e executa novamente tendo este bloco como entrada. Se não houver mais blocos de entrada, a matriz C é retornada como resposta para a CPU.

A Figura 4.4(b) apresenta a ilustração do exemplo do cálculo do índice de vegetação NDVI na GPU. As matrizes A e B representam as bandas *RED* e *NIR*, respectivamente, e são transformadas em vetores que são concatenados em um único vetor V. O vetor V é dividido em blocos, caso seja maior que a memória da GPU ($bgpu$). Nesse exemplo, V é transferido totalmente para a GPU, pois o tamanho de V é menor que a memória disponível na GPU. Para manter o acesso aglutinado na GPU, cada posição das matrizes A e B, no vetor V, é calculada por uma única *thread*, podendo cada *thread* calcular várias posições de A e B. Na Figura 4.4(b), a *thread* $T1$ ficou responsável pelo cálculo do NDVI da primeira e quarta posição das matrizes A e B. No cálculo da primeira posição de A e B, por exemplo, $T1$ deve acessar a primeira (referente a A) e quinta posição de V (referente a B). O resultado final é composto de um vetor R com tamanho igual a A ou B, já que ambas as matrizes devem possuir o mesmo tamanho.

4.3 Considerações Finais

De forma geral, o uso do TWDTW na classificação das STSR apresenta alta demanda de recursos computacionais para calcular as medidas de similaridade [77]. Esse alto custo computacional torna inviável o uso do TWDTW em um cenário com grande volume de dados. Apresentamos a proposta de um novo algoritmo paralelo P-TWDTW, que explora a escalabilidade vertical de uma máquina, utilizando as arquiteturas *many-core*, que torna possível o uso das medidas de similaridade do TWDTW na classificação do tipo de uso e cobertura do solo utilizando as STSR.

O algoritmo TWDTW constrói a matriz de custos acumulados de forma serial utilizando um algoritmo de programação dinâmica, onde o cálculo de cada célula da matriz depende do resultado do cálculo de outras células. O P-TWDTW calcula a matriz de custo caminhando nas diagonais secundárias da matriz e paralelizando o processamento da diagonal entre várias *threads*, onde cada *thread* fica responsável por uma célula da diagonal. O P-TWDTW sempre aloca d *threads* para o processamento da diagonal secundária, onde d é igual ao tamanho da diagonal principal. Entretanto, as outras diagonais paralelas à secundária possuem tamanho menor que d, o que acaba desperdiçando recursos da GPU e deixando d - n *threads* ociosas, onde n é o tamanho da diagonal que está sendo processada. Esse comportamento impacta no desempenho do P-TWDTW, mas, ainda assim, o algoritmo consegue paralelizar, de forma eficiente, o processamento do cálculo das medidas de similaridade. Além disso, é possível paralelizar os cálculos das medidas de similaridade entre várias séries temporais e vários padrões entre os núcleos da GPU, diminuindo assim sua ociosidade.

A paralelização do algoritmo TWDTW, com o P-TWDTW, abriu a possibilidade de criar meta-características a partir das medidas de similaridade do TWDTW como entrada de métodos de aprendizado de máquina, que na solução serial é inviável devido ao alto custo computacional. Nosso trabalho estendeu a abordagem de [96] que utiliza as séries temporais das bandas da imagem e dos índices de sensoriamento remoto como características de métodos de aprendizado de máquina, criando meta-características a partir das medidas de similaridade de cada série temporal V para cada padrão U, de forma a aumentar a acurácia do método de aprendizagem.

Como os índices de sensoriamento remoto geralmente não são fornecidos como dados de entrada e precisam ser calculados a partir das bandas espectrais das imagens de satélite, este trabalho propõe o algoritmo P-INDEX para cálculo destes índices utilizando arquiteturas paralelas *manycore* e *multicore*. O uso do P-TWDTW para calcular as medidas de similaridade entre cada U e V e o uso do P-INDEX para calcular os índices de sensoriamento remoto resultam numa forma eficiente, explorando arquiteturas paralelas, para classificação do tipo de uso e cobertura do solo utilizando as STSR.

CAPÍTULO 5

Plataforma de *Big Data* para Processamento Distribuído de Séries Temporais de Sensoriamento Remoto

Os algoritmos P-TWDTW e P-INDEX exploram a escalabilidade vertical de um servidor utilizando arquiteturas paralelas *manycore* e *multicore*. Para *Big Data*, entretanto, devido ao grande volume de dados, faz-se necessário explorar a escalabilidade horizontal [52], já que o processamento em uma máquina esbarra nos limites de CPU, memória, disco e barramentos de dados. Este capítulo descreve a arquitetura da plataforma de *Big Data* proposta neste trabalho, denominada BigSensing, para explorar os recursos computacionais do *cluster* de computadores e o paralelismo massivo em cada servidor.

Diversas soluções de plataformas foram propostas na literatura [69, 45, 19, 46, 21, 64] contendo vários sistemas de *Big Data* para processamento em lote ou tempo real, tais como o Hadoop, Spark, Elasticsearch e SciDB. Experimentos realizados em alguns trabalhos [33, 126] mostram que nenhum sistema tem desempenho melhor que os outros em todos os cenários com filtros espaciais e temporais.

Por exemplo, o Elasticsearch tem ótimo desempenho na filtragem ou agregação de um grande volume de dados. Entretanto, quando o Elasticsearch precisa recuperar um grande volume de dados, seu desempenho cai drasticamente. Já o SciDB e o Spark são melhores que o Elasticsearch, por exemplo, nos cenários em que há o retorno de um grande volume de dados.

Uma solução em sensoriamento remoto de classificação do tipo de uso e cobertura do solo necessita armazenar e indexar um grande volume de dados, devido a característica de Remote Sensing *Big Data* do problema. Mas, nem sempre é necessário recuperar um grande volume de dados dessa base, principalmente quando um estudo é realizado em pequenas regiões geográficas [92, 77, 80, 85], visto que quanto menor for a região geográfica, menor será o número de *pixels* recuperado para processar as observações das STSR. Nesses casos, o Elasticsearch poderia ser uma boa opção, por ter um ótimo desempenho na filtragem dos dados espaço-temporais. Quando é necessário realizar um processamento

sobre uma grande região geográfica, um grande volume de dados de STSR tem que ser recuperados do sistema de armazenamento e, neste caso, por exemplo, o SciDB ou o Spark seriam melhores soluções que o Elasticsearch.

Entretanto, fica a cargo do usuário da plataforma escolher previamente qual sistema deseja utilizar em cada execução. Essa tarefa é inviável de ser realizada manualmente devido à quantidade de variáveis que precisam ser analisadas para a tomada de decisão. Por exemplo, o usuário tem que analisar os filtros espaciais e temporais, métricas de monitoramento do *cluster*, como uso de CPU, memória, rede e uso de disco e área de swap, além de métricas internas de cada sistema, para decidir qual utilizar em cada requisição.

Nosso trabalho propõe um motor inteligente na plataforma BigSensing, apresentado na Seção 5.2, denominado SmarT (smart Spatial-Temporal query engine), que tem como objetivo aplicar os filtros espaciais e temporais e recuperar os dados das STSR, buscando escolher dentre diversos sistemas de *Big Data*, em tempo real, aquele que potencialmente é o mais eficiente. O SmarT está preparado para remoção ou adição de novos sistemas de *Big Data* na plataforma, sendo necessário ao desenvolvedor treinar o motor novamente com dados desse novo conjunto de sistemas.

O motor de busca SmarT é capaz de distribuir o processamento dos algoritmos de STSR entre os servidores do *cluster*. Já para explorar as arquiteturas paralelas ManyCore e *multicore* disponíveis em cada servidor do *cluster* no processamento dos algoritmos, tais como o P-TWDTW, foi criado o módulo Core Executor, apresentado na Seção 5.3. Para que a plataforma BigSensing seja capaz de receber, indexar e armazenar um grande volume de dados de sensoriamento remoto, foi construído o módulo DStream, apresentado na Seção 5.4, para ingestão do fluxo de dados de sensoriamento remoto. Uma visão geral da integração desses módulos é apresentada na Seção 5.1.

As Seções 5.1, 5.2, 5.3, 5.4 apresentam a proposta arquitetural da plataforma BigSensing sem levar em conta as escolhas de tecnologias para implementação de cada módulo da plataforma. A Seção 5.5 apresenta as ferramentas utilizadas em cada módulo que atendem à arquitetura proposta e a maneira como a plataforma foi implantada em um *cluster* de computadores. Além disso, são apresentadas diversas outras opções de ferramentas que poderiam ser utilizadas na arquitetura.

5.1 Visão geral da Arquitetura

Esta seção apresenta uma visão geral da arquitetura da plataforma BigSensing no processamento de grandes volumes de dados de STSR em um *cluster*. A plataforma BigSensing possui uma arquitetura distribuída, onde os servidores do *cluster* não compartilham CPUs, unidades de discos e memória e a comunicação entre eles é realizada através do mecanismo de troca de mensagens. A plataforma BigSensing é constituída por

uma API de interação com os clientes e os servidores que executam as operações sobre os dados de sensoriamento remoto.

A arquitetura da plataforma BigSensing é apresentada na Figura 5.1 com os módulos DStream, SmarT e Core Executor, além da camada de armazenamento. O módulo SmarT é um motor de busca inteligente capaz de aplicar filtros espaciais e temporais escolhendo, em tempo real, o melhor sistema de processamento para recuperar os dados de STSR. Os dados que atendem às restrições dos filtros da consulta são enviados para o módulo Core Executor que explora as arquiteturas paralelas em cada máquina. O módulo Core Executor pode devolver o resultado da requisição na memória para o SmarT ou escrever o resultado no armazenamento da plataforma para ser buscado posteriormente pelo usuário. O módulo DStream é responsável pela ingestão dos dados de sensoriamento remoto enviados por diversas fontes de dados. O DStream recebe os dados das séries temporais, realiza as transformações necessárias e escreve o conteúdo final em cada sistema, replicando os dados.

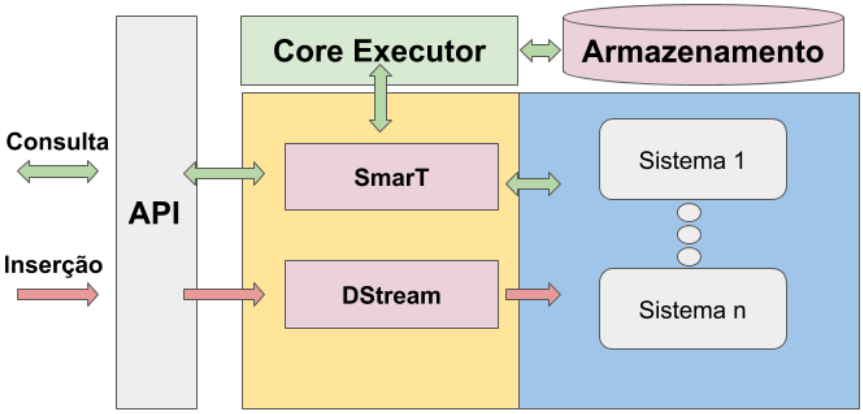


Figura 5.1: *Arquitetura da plataforma BigSensing*

As aplicações clientes comunicam com a plataforma através de uma API cliente, que disponibiliza métodos para atualização e consultas nas bases de dados. Requisições de leitura ou escrita dos clientes podem ser enviadas para qualquer servidor do *cluster*. O servidor que recebe a requisição do cliente torna-se o coordenador da requisição em particular. O coordenador age como um *proxy* entre a aplicação cliente e os servidores responsáveis pelos dados que serão acessados. Para garantir a alta disponibilidade, se o servidor estiver indisponível, outro servidor do *cluster* é escolhido como coordenador da requisição.

A API de consulta da plataforma permite informar as restrições espaciais e temporais com os seguintes parâmetros: *bounding_box*, *start_time*, *end_time*. O *bounding_box* é o retângulo que irá limitar espacialmente a área geográfica de execu-

ção dos algoritmos de processamento de séries temporais. Já os parâmetros *start_time* e *end_time* visam delimitar a data inicial e final dos dados de análise, respectivamente.

5.2 SmarT: Motor de Busca Inteligente para Filtragem e Recuperação de Dados Espaciais e Temporais

A plataforma BigSensing permite que diversos sistemas de *Big Data* para filtragem e recuperação de dados espaço-temporais sejam acoplados à plataforma. Escolher um deles, em tempo real, a cada requisição de processamento de algoritmos de STSR é uma tarefa complicada, pois cada sistema tem características que permitem o melhor desempenho para determinados cenários de filtragem espacial e temporal. Além disso, eles estão em constante evolução e a cada nova versão dos sistemas de *Big Data*, podem surgir otimizações que se sobrepõem aos sistemas concorrentes. Este trabalho apresenta, nesta seção, um motor inteligente para filtragem e recuperação de dados espaço-temporais, denominado SmarT, que busca escolher, em tempo real, o melhor sistema de *Big Data* para processar a requisição.

O SmarT faz a predição do tempo de execução em cada sistema e escolhe aquele de menor duração estimada, tendo como entrada: i) as restrições espaciais e temporais de entrada da consulta; ii) as métricas do *cluster* obtidas no sistema de monitoramento e iii) a quantidade aproximada de resultados da consulta obtida no sistema de indexação da plataforma.

A Figura 5.2 apresenta a arquitetura do SmarT dentro da plataforma BigSensing. As restrições espaciais e temporais são enviadas pelo cliente para a API que repassa ao SmarT. Este requisita ao sistema de monitoramento do *cluster* métricas de uso da rede, da área de swap, da memória e da CPU dos servidores. Ele também realiza uma consulta ao sistema de indexação da plataforma para definir uma quantidade estimada de resultados retornados na consulta. Depois da filtragem e recuperação dos dados no sistema escolhido, apenas as observações das séries temporais que satisfazem as restrições da consulta são enviadas para o módulo Core Executor para a execução dos algoritmos de processamento das STSR.

O SmarT é treinado com o objetivo de minimizar o tempo de resposta na filtragem e recuperação dos dados escolhendo o sistema de *Big Data* mais eficiente. Os dados de treinamento do SmarT são armazenados no formato tabular, onde cada linha contém os dados de entrada e o tempo de resposta de uma consulta. Nossa solução para construção do SmarT, utilizando aprendizado de máquina, é apresentada na Seção 5.2.1.

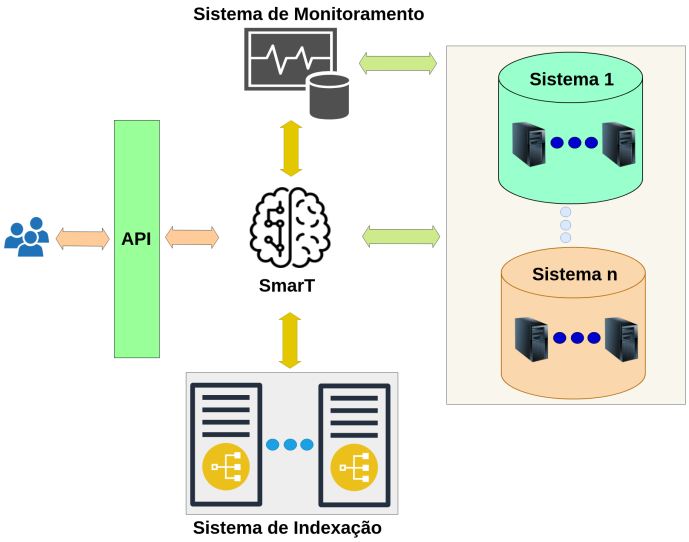


Figura 5.2: *Arquitetura do motor de consulta SmarT na plataforma BigSensing.*

5.2.1 Uso do Aprendizado de Máquina na Seleção em Tempo Real do Sistema de *Big Data* para Consultas com Filtros Espaço-Temporais

Esta seção apresenta nossa solução para reduzir o tempo de resposta na execução de consultas com filtros espaciais e temporais em um ambiente distribuído buscando escolher, em tempo real, o melhor sistema de *Big Data* para o processamento da consulta. Dado que o tempo de resposta é uma variável contínua, o problema de minimizar o tempo de resposta foi mapeado com uma modelagem preditiva de regressão com o objetivo de aproximar a função de mapeamento das características de entrada para a variável de saída contínua que representa o tempo de resposta. O problema foi mapeado como regressão para tentar prever o tempo de resposta de cada sistema e escolher aquele com menor tempo previsto. Mesmo que não seja escolhido o melhor sistema, aumenta a possibilidade de se escolher um dos melhores.

Na nossa proposta de modelagem preditiva de regressão, o tempo de resposta da consulta de cada sistema de *Big Data* é a variável de resposta (dependente) e as variáveis explicativas (independentes) são compostas pelas restrições da consulta, métricas de monitoramento do *cluster* e o número estimado de resultados na consulta. Para cada sistema de *Big Data* é construído um modelo dentro do SmarT, onde a variável dependente representa o tempo de resposta do sistema de *Big Data*. Denotaremos a variável de resposta por y e o conjunto de variáveis explicativas por $x_1, x_2, ..., x_p$, onde p indica o número de variáveis explicativas. O relacionamento entre y e $x_1, x_2, ..., x_p$ é aproximado

pelo modelo de regressão da equação 5-1:

$$y = f(x_1, x_2, ..., x_p) + \varepsilon \tag{5-1}$$

onde ε é adotado como um erro randômico representando a discrepância na aproximação, para levar em conta os possíveis erros de predição do modelo. Assim, a função $f(x_1, x_2, ..., x_p)$ descreve o relacionamento entre y e $x_1, x_2, ..., x_p$.

Existem diversos modelos de regressão [16] que podem ser utilizados para estimar o tempo de resposta de cada sistema. Os métodos *ensemble* têm se destacado como os melhores em várias competições de ciência de dados, onde se tem dados tabulares de entrada [127, 98]. Esses métodos combinam o resultado de um conjunto de métodos treinados individualmente para tomar a decisão final, permitindo reduzir a variância dos dados [91].

Nossa solução utiliza o método *ensemble* XGBoost (Extreme Gradient Boosting) [18]. O XGBoost foi escolhido pelo seu destaque em várias competições de ciência de dados, ganhando a maior parte delas com dados tabulares de entrada (que é exatamente nosso formato de entrada) [86]. O XGBoost se destaca nestas competições em relação aos demais métodos por duas principais razões [86]: i) habilidade de criar uma representação rica dos dados, aproximando funções complexas de relacionamento entre os dados, incluindo interações em espaços altamente dimensionais; ii) versatilidade de se adaptar com diferentes conjuntos de variáveis explicativas executando a seleção de características de forma automática.

Boosting é um método *ensemble* que adiciona novos membros ao conjunto do *ensemble* de forma sequencial. O membro mais novo é adicionado para compensar as instâncias incorretamente classificadas pelos modelos anteriores. O *Gradient boosting* é uma variação do *boosting* que representa o problema de aprendizado como um gradiente descendente utilizando algumas funções de custo arbitrárias que medem a performance do modelo no conjunto de treinamento. Especificamente, são executadas M iterações para aprender uma função $F(x)$, que tem como saída as predições $\hat{y} = F(x)$, minimizando a função de custo $L(y, \hat{y})$. Essa função é responsável por dizer quão longe estamos da predição ideal e, portanto, quantifica o "custo" ou "perda" ao aceitarmos a predição gerada pelos parâmetros atuais do modelo. Em cada iteração, um novo $f(x)$ é adicionado para tentar corrigir a predição de y para cada instância de treinamento

O XGBoost é um algoritmo supervisionado que implementa um processo chamado *boosting* para produzir modelos com maior acurácia. A entrada para o algoritmo são pares de exemplos de treinamento $(\vec{x}_1, y_1)$, $(\vec{x}_2, y_2)$, $(\vec{x}_p, y_p)$, onde $\vec{x}$ é um vetor de características representando as variáveis explicativas e y é o tempo de resposta da consulta. O XGBoost é composto por um conjunto de métodos de aprendizado de máquina baseados em árvores de decisão com o objetivo de otimizar a função objetivo de forma mais

robusta e escalável que as técnicas de árvore atualmente existentes. Árvores de decisão constroem um modelo repetidamente dividindo subconjuntos dos dados de treinamento seguindo algum critério. A construção das árvores de decisão é realizada a partir do conjunto completo de dados de treinamento e da avaliação de diversas formas de criar uma divisão destes dados de treinamento baseado nas características de entrada em $\vec{x}$.

Ao invés de utilizar o vetor de características, o XGBoost constrói as árvores para, inteligentemente, obter o *score* de cada característica, assim indicando a importância de cada característica para o treinamento do modelo. Quanto mais a característica for utilizada nas decisões com as árvores, maior será seu *score*.

O XGBoost inclui um termo de regularização, utilizado para não gerar *overfitting* [50] e suportar funções de custo arbitrárias. A função objetivo apresentada na Equação 5-2 é construída com duas partes, sendo a primeira uma função de custo sobre o conjunto de treinamento e a segunda parte o termo de regularização que penaliza a complexidade do modelo:

$$Obj = \sum_i L(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \tag{5-2}$$

com $L(y_i, \hat{y}_i)$ definida como qualquer função de custo que é capaz de medir a diferença entre a predição e o resultado esperado para cada instância de treinamento. $\Omega(f_k)$ descreve a complexidade da árvore f_k e é definido no algoritmo XGBoost pela equação 5-3:

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda w^2 \tag{5-3}$$

onde T é o número de folhas da árvore f_k e w os pesos das folhas (i.e. os valores preditos armazenados nos nós folhas). Quando $\Omega(f_k)$ é incluído na função objetivo, existe a necessidade de otimizar o modelo de forma a construir árvores menos complexas e ao mesmo tempo minimizar $L(y_i, \hat{y}_i)$. Isso ajuda a reduzir o *overfitting*. Uma penalidade é aplicada, com o γT, para cada nova folha adicional na árvore. Os pesos das folhas com valores muito altos ou muito baixos são penalizados com λw^2. Tanto γ quanto λ são valores configuráveis pelo usuário.

Considerando que o método de *boosting* executa de forma iterativa, é possível definir a função objetivo para a iteração corrente m em termos da predição das iterações anteriores, $\hat{y}_i^{(m-1)}$, ajustada pela árvore de decisão mais recente, f_k, como pode ser visto na equação 5-4:

$$Obj^m = \sum_i L(y_i, \hat{y}_i^{(m-1)} + f_k(x_i)) + \sum_k \Omega(f_k) \tag{5-4}$$

Os dados tabulares de entrada para o XGBoost podem ser descritos como uma matriz onde cada linha representa uma instância e cada coluna uma característica. No

nosso trabalho, para a escolha do melhor sistema na filtragem espaço-temporal dos dados em um ambiente distribuído, foram utilizadas as seguintes características:

- *area*: tamanho da área geográfica da restrição espacial
- *time_interval*: número de dias da restrição temporal
- *count*: número estimado de resultados da consulta
- *swap_free*: média de área de *swap* disponível no *cluster*
- *mem_free*: média de *memória* disponível no *cluster*
- *load_one*: média do *load* dos servidores do *cluster* no último minuto
- *load_five*: média do *load* dos servidores do *cluster* nos últimos 5 minutos
- *load_fifteen*: média do *load* dos servidores do *cluster* nos últimos 15 minutos
- *cpu_user*: média do uso de CPU no espaço de execução do usuário no *cluster*
- *cpu_system*: média do uso de CPU no espaço de execução do *kernel* do Sistema Operacional no *cluster*
- *bytes_in*: média do tráfico de rede de entrada nas máquinas do *cluster*
- *bytes_out*: média do tráfico de rede de saída nas máquinas do *cluster*

Sabendo que o tempo de resposta (*response_time*) representa a variável de resposta y, então para cada par de entrada do treinamento $(\vec{x}_i, y_i)$, a variável de resposta y_i é definida pela equação 5-5 e o vetor de características $\vec{x}_i$ pela equação 5-6:

$$y_i = response\_time_i \tag{5-5}$$

$$\begin{aligned} \vec{x}_i = (&area_i, time\_interval_i, count_i, swap\_free_i, mem\_free_i, \\ &load\_one_i, load\_five_i, load\_fifteen_i, \\ &cpu\_user_i, cpu\_system_i, bytes\_in_i, bytes\_out_i) \end{aligned} \tag{5-6}$$

Os valores das variáveis explicativas *area* e *time_interval* são obtidos a partir dos filtros espacial e temporal enviados na consulta pelo usuário da plataforma. Estas variáveis são importantes porque impactam no custo computacional de filtragem e recuperação dos dados. Elas tem impacto no número de resultados retornados, já que, geralmente, quanto maior for a restrição espacial e temporal maior será o número de resultados retornados. Em alguns casos, entretanto, onde não existem dados que atendam à restrição da consulta, esta afirmação pode não ser verdadeira. Por exemplo, para uma base de dados de todo o Estado de Goiás, consultas englobando todo continente africano não irão retornar algum resultado. Já a filtragem espacial dentro de Goiás, nessa base de dados, irá retornar um conjunto de resultados.

A variável *count* é gerada a partir do sistema de indexação na plataforma BigSensing, que retorna o número estimado de resultados da consulta dada as restrições

espacial e temporal. Combinando as variáveis *area*, *time_interval* e *count* é possível estimar, de forma mais acurada, o custo de recuperação dos dados de resposta da consulta. Esta informação é importante para a decisão do SmarT de qual sistema processar a consulta, pois cada sistema pode ter um desempenho diferente de acordo com o volume de dados a ser recuperado.

As métricas do *cluster*, obtidas no sistema de monitoramento, são importantes porque impactam na performance de um sistema em um ambiente distribuído. São coletadas a média das métricas de todos os servidores, para representar o estado do *cluster* no momento da consulta. Apesar das variáveis *area*, *time_interval* e *count* permitirem estimar o impacto do custo de filtragem e recuperação dos dados da consulta, o uso de memória, disco, CPU e rede também podem alterar a escolha realizada pelo SmarT, já que cada sistema pode ter otimizações internas em que uma ou mais métricas de uso do *cluster* podem impactar na performance do sistema. O Spark, por exemplo, é um sistema de processamento de consultas em memória e, por isso, as métricas *mem_free* e *swap_free* podem afetar o seu desempenho.

Diversas outras variáveis podem ser acopladas ao modelo de aprendizado de máquina do SmarT, mas nosso trabalho definiu um escopo com as variáveis apresentadas acima. O SmarT permite também que outros métodos de aprendizado possam ser acoplados, substituindo o XGBoost atual.

5.3 Core Executor: Abordagem de Integração da Solução Distribuída com Algoritmos de Processamento de Séries Temporais em Arquiteturas Paralelas

Esta seção descreve o módulo Core Executor, que é responsável por executar o processamento das STSR em cada servidor do *cluster*, que recebe uma porção dos dados filtrados e recuperados pelo SmarT. Se a máquina tiver uma GPU disponível e o algoritmo for projetado e implementado para explorá-la, o Core Executor seleciona a GPU como unidade de processamento. Caso contrário, os núcleos da CPU são escolhidos.

A Figura 5.3 apresenta a arquitetura de interação entre o SmarT e o Core Executor. Como os dados já estão localmente filtrados pelo SmarT, o desafio do Core Executor está em como explorar o melhor de cada unidade de processamento, já que os núcleos da CPU suportam poucas *threads*, mas não requerem transferência adicional de dados. Quando o Core Executor executa o algoritmo P-TWDTW, os dados são enviados sempre para a memória da CPU, já que o P-TWDTW possui um mecanismo interno otimizado de envio dos dados em lote para a GPU.

O resultado do processamento do Core Executor pode ser gravado na camada de armazenamento da plataforma BigSensing ou enviado de volta para o SmarT. O usuário envia, na requisição, a forma como deseja recuperar o resultado do processamento das séries temporais. Caso seja na memória, todo o resultado é agrupado e devolvido pelo SmarT. Como não existe um controle de memória neste momento, o volume de dados retornado pode estourar o limite de memória do nó coordenador que age como um *proxy* entre a aplicação cliente e a plataforma. Por isso, por padrão, os dados são gravados no sistema de armazenamento e o usuário, posteriormente, pode recuperar os resultados iterando sobre eles ou realizando outras operações, como gerar estatísticas e mapas com o resultado da classificação, por exemplo. Para o usuário poder interagir com os resultados, a API da plataforma também tem acesso ao sistema de armazenamento.

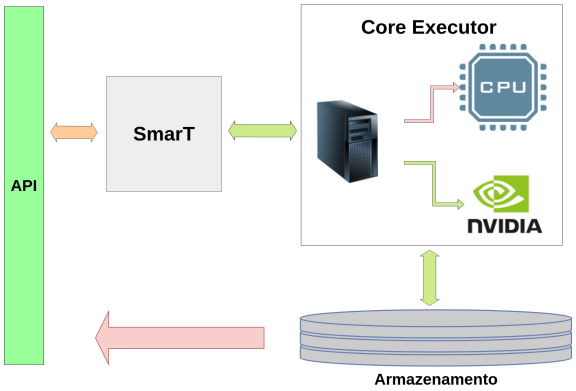


Figura 5.3: *Arquitetura de interação entre os módulos SmarT e Core Executor.*

5.4 DStream: Estratégia para Ingestão de Grande Volume de Dados de Sensoriamento Remoto

Esta seção apresenta o módulo DStream, responsável por receber, indexar e armazenar um grande fluxo de dados de sensoriamento remoto. As observações das séries temporais chegam à plataforma no formato de um objeto multidimensional, contendo informações da localização geográfica de cada pixel da imagem e os dados espectrais da imagem. Cada objeto contém dados de diversos *pixels* em um determinado momento no tempo. Por exemplo, na coleta de dados do Brasil, cada objeto poderia representar um recorte de 100x100 *pixels* de alguma região do país. Junto com o objeto são enviados os metadados contendo: i) *timestamp*: data de coleta dos dados e ii) *source*: fonte de obtenção dos dados das séries temporais (por exemplo MOD13Q1). Desta forma, será possível inserir na plataforma um lote com informações de vários *pixels* ao mesmo tempo, otimizando assim o armazenamento.

A arquitetura dessa solução, apresentada na Figura 5.4, é constituída pelas etapas de coleta, transformação e armazenamento dos dados de séries temporais. Os dados são coletados de diversas fontes de dados de imagens de sensoriamento remoto, tais como: satélites, fotos aéreas, clientes da plataforma e integração com outras fontes de dados. Os dados coletados são enviados para uma fila distribuída que serve como entrada para o processamento dos *streams* de dados.

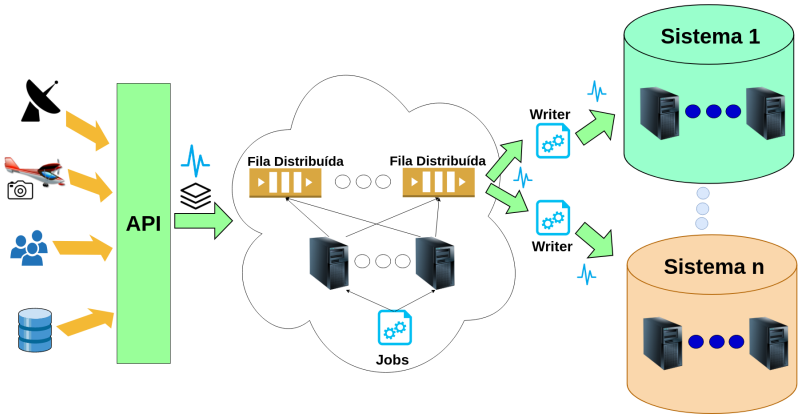


Figura 5.4: *Arquitetura de ingestão de dados com o módulo DStream*

A fila de entrada é distribuída e visível a todos os servidores do *cluster*, onde qualquer servidor é capaz de recuperar um objeto da fila. Cada *job* pode ser executado em qualquer servidor do *cluster*, cabendo à plataforma garantir a tolerância a falhas, o controle de fluxo e o escalonamento dos *jobs* entre os servidores. Cada objeto enviado para a fila distribuída é recuperado por um *job*, que realiza alguma operação sobre esse objeto em uma máquina. Cada *job* pode ser replicado em várias máquinas para que determinada operação possa ser realizada de forma paralela, onde cada réplica fica responsável por processar um objeto da fila. A saída desse *job* é enviada para outra fila distribuída da plataforma. O cliente define, no momento que envia a requisição para a API, quais *jobs* serão executados sobre os dados até serem armazenados e em qual ordem.

Esta arquitetura, onde objetos são recuperados de uma fila de entrada, processados e enviados para uma fila de saída, permite que outros *jobs* sejam integrados facilmente à plataforma, bastando adicionar filas distribuídas e *jobs* para consumir dados destas filas. Por exemplo, suponha que a plataforma receba os dados brutos das imagens e precise construir as séries temporais calculando o índice de vegetação NDVI de cada pixel ao longo do tempo, utilizando o algoritmo P-INDEX apresentado na Seção 4.2. Um dos *jobs* poderia integrar o algoritmo P-INDEX e calcular o índice NDVI explorando arquiteturas paralelas *multicore* e *manycore*, enviando para a próxima etapa os dados de entrada enriquecidos com o NDVI.

Quando os dados chegam à fila de saída da plataforma, vários *jobs* (*writers*) recuperam esses dados e armazenam o resultado nos sistemas da plataforma, como o

Spark, Elasticsearch e SciDB. Cada *job* é responsável por armazenar os dados em um sistema específico, já que ele conhece o método de escrita do sistema em questão e, desta forma, é possível realizar otimizações específicas para cada um. Desta forma, é possível adicionar sistemas na plataforma com a construção de um novo *job* de escrita para ele. Os dados na plataforma BigSensing são replicados em cada sistema, já que ela não possui um barramento de dados único para todos eles. As observações das séries temporais são armazenadas nos sistemas da plataforma no seguinte formato: i) *timestamp*: data de coleta da observação, ii) *source*: fonte de obtenção dos dados, iii) *lat*: latitude do pixel, iv) *lon*: longitude do pixel e v) *values*: dados da observação, como por exemplo os valores das bandas da imagem e índices de sensoriamento remoto.

Como pode ser visto na Figura 5.5, o objeto final também é enviado para um sistema distribuído de indexação espacial e temporal. Para indexar os objetos espaciais, é utilizado o índice espacial *Quad Tree* [36] que está presente em cada servidor do *cluster* e é construído a partir dos objetos espaciais armazenados no servidor. A plataforma é responsável por coordenar a filtragem espacial distribuída entre os vários índices espaciais. O índice *Quad Tree* foi escolhido por ter um bom desempenho na filtragem de dados de sensoriamento remoto [39]. O índice temporal é construído a partir do campo *timestamp* de cada observação e é importante porque permite, por exemplo, realizar análises em intervalos de tempo definidos.

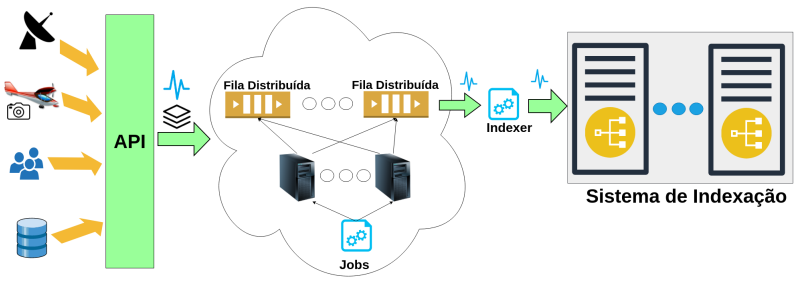


Figura 5.5: *Arquitetura do Sistema Distribuído de Indexação de Dados de Séries Temporais*

5.5 Implementação da plataforma BigSensing

Esta seção apresenta as tecnologias utilizadas na implementação dos módulos da plataforma BigSensing e algumas opções que poderiam ser utilizadas em cada módulo. A plataforma disponibiliza uma API REST implementada utilizando o gerenciador de API Tyk [1]. O Tyk é uma solução de gerenciamento de API que permite controlar quem acessa,

[1] https://github.com/TykTechnologies/tyk

quando acessa e como acessa os *endpoints* da API. Desta forma, resta ao desenvolvedor se preocupar apenas com a implementação dos serviços, ao invés de lidar com autorização, controle de acesso e limites de taxa de uso da API. Existem diversas outras opções ao Tyk, tais como o Apiary [2] e Apigee [3], mas o Tyk foi escolhido por ser *open source*, pela facilidade de configuração, ter excelente performance e apresentar um grande conjunto de funcionalidades para o gerenciamento de API.

No motor inteligente de buscas SmarT, apresentado na Seção 5.2 foram escolhidos como sistemas de filtragem e recuperação dos dados, o Apache Spark [4], SciDB [5] e Elasticsearch [6]. O Apache Spark foi escolhido por ser uma das soluções mais utilizadas no processamento de séries temporais no contexto de *Big Data* [34]. O SciDB foi selecionado por ter sido projetado para lidar com dados multidimensionais, especialmente séries temporais, sendo inclusive utilizado pela NASA (National Aeronautics and Space Administration) [104] e pelo INPE (Instituto Nacional de Pesquisas Espaciais) [12]. O Elasticsearch foi escolhido por ser uma solução com bom desempenho para filtragem de dados espaço-temporais [119]. Existem diversas outras opções de soluções para lidar com séries temporais, tais como Apache Druid [7] e InfluxDB [8], que podem inclusive ser adicionadas na arquitetura expansível do SmarT.

Foi utilizada a implementação do método XGBoost da biblioteca *open source* H2O [9]. Esta biblioteca foi escolhida pela sua facilidade de uso e por estar preparada para execução dos modelos de aprendizado de máquina em um *cluster* de computadores. Existem diversas outras bibliotecas para aprendizado de máquina, em diversas linguagens, se destacando o Scikit-learn [10], o Tensorflow [11] e o Pytorch [12]. O Keras [13] também se destaca no âmbito de Deep Learning, mas ela está integrada ao Tensorflow na versão 2.0.

A plataforma não possui um barramento de dados único que integra os sistema acoplados ao SmarT e, por isso, os dados das STSR são replicados no Elasticsearch, Spark e SciDB. Para uso de propósito geral, como no armazenamento de resultados das consultas pelo Core Executor, a plataforma BigSensing tem o HDFS (Hadoop Distributed File System) como sistema de armazenamento distribuído. O HDFS é um sistema de

[2] https://apiary.io/
[3] https://cloud.google.com/apigee/
[4] https://spark.apache.org/
[5] https://paradigm4.atlassian.net
[6] https://www.elastic.co
[7] https://druid.apache.org
[8] https://www.influxdata.com
[9] https://www.h2o.ai/
[10] https://scikit-learn.org/
[11] https://www.tensorflow.org/
[12] https://pytorch.org/
[13] https://keras.io/

arquivos distribuído, que faz parte do ecossistema do Hadoop, e permite armazenar grandes quantidades de dados estruturados e não-estruturados.

No módulo DStream, descrito na Seção 5.4, as filas distribuídas foram implementadas utilizando o Apache Kafka [14]. Esta escolha foi feita pelo fato do Kafka ter sido criado para lidar com o cenário de *Big Data* com grande volume de fluxos de dados em tempo real de forma escalável em um *cluster* de computadores. O ActiveMQ [15] e RabbitMQ [16] são duas de várias opções que poderiam ser utilizadas [65].

Os *jobs* do módulo DStream foram implementados utilizando o Spark Streaming [17] para gerenciamento do fluxo de dados de entrada e saída das filas do Apache Kafka. O Spark Streaming foi escolhido por ter integração nativa com o Apache Kafka, pela sua facilidade de uso e por já ter um *cluster* Spark configurado no módulo SmarT que facilita a utilização do Spark Streaming. A utilização de outra solução de *streaming* iria gerar a necessidade de manutenção de mais uma tecnologia. Existem várias opções ao Spark Streaming [65], tais como o Apache Flink [18] e Apache Storm [19].

Ao longo da pesquisa, um índice espaço-temporal distribuído foi construído e mantido [30, 29], mas com a evolução da indexação espacial e temporal do Elasticsearch [20], o nosso sistema de indexação foi substituído pelo sistema distribuído de indexação do Elasticsearch, que também atende à arquitetura de indexação definida no nosso trabalho. Além do ganho de performance na filtragem de dados espaciais e temporais, existe um ganho nas diversas outras funções de agregação, filtragem e análise de dados disponíveis no Elasticsearch.

O módulo Core Executor, descrito na Seção 5.3, foi implementado na linguagem JAVA e possui integração com o algoritmo P-TWDTW. A implantação de todo o ecossistema de ferramentas da plataforma BigSensing foi realizada utilizando a conteinerização com o Docker [113]. Existe um grande desafio de se criar um ambiente de *Big Data* com o Docker, integrando todas as ferramentas, que já são complexas por natureza, em um ambiente distribuído. Entretanto, quando este ambiente conteinerizado é criado, fica mais fácil manter, escalar, gerenciar as dependências e atualizar os componentes individuais da plataforma *Big Data*. Desta forma, se torna menos complexo mover a solução de *Big Data* do ciclo de desenvolvimento para o ambiente de produção.

[14] https://kafka.apache.org
[15] https://activemq.apache.org/
[16] https://www.rabbitmq.com/
[17] https://spark.apache.org/streaming/
[18] https://flink.apache.org/
[19] https://storm.apache.org/
[20] https://www.elastic.co/pt/elasticon/tour/2019/san-francisco/geospatial-advancements-in-elasticsearch-and-apache-lucene

5.6 Considerações Finais

O algoritmo P-TWDTW apresentado no Capítulo 4 e a arquitetura da plataforma BigSensing, apresentada neste capítulo, irão possibilitar lidar com o processamento de grande volume de STSR geradas em alta velocidade por diversas fontes de dados. A plataforma permite explorar a escalabilidade horizontal de vários servidores em um *cluster* de computadores, enquanto o algoritmo P-TWDTW explora a escalabilidade vertical de cada máquina utilizando arquiteturas paralelas *manycore*.

A plataforma BigSensing possui um motor inteligente de busca, denominado SmarT, que foi projetado para distribuir o processamento de consultas no *cluster* de computadores, de forma eficiente, escolhendo o melhor sistema para filtragem e recuperação dos dados que serão enviados para o Core Executor. O SmarT é responsável pelo processamento em lote e em tempo real, sendo capaz de escolher, no momento da consulta, qual sistema usar para cada tipo de requisição do usuário. O Spark e SciDB, por exemplo, são eficientes no processamento em lote pelo desempenho de ambos na recuperação de um grande volume de dados. Já o Elasticsearch é eficiente no processamento em tempo real, por ser capaz de filtrar uma grande base de dados, mas retornar um conjunto pequeno de resultados em poucos segundos para os usuários da plataforma. Já o processamento do fluxo de dados é realizado pelo módulo DStream.

Os dados são replicados em todos os sistemas acoplados à plataforma, já que ela não possui um barramento de dados único integrado a todos estes sistemas. Essa replicação aumenta o tempo para inserir os dados e o gasto com armazenamento. Mas, como a inserção é realizada uma só vez e o custo de armazenamento está cada vez menor, a redução do tempo de resposta gerado pelo SmarT se sobrepõe às desvantagens.

Em cada máquina, o Core Executor é responsável por explorar as arquiteturas paralelas *multicore* e *manycore* no processamento dos algoritmos de STSR. Não existe controle de memória em relação ao tamanho do resultado final entre o SmarT e o Core Executor e, por isso, cabe ao usuário decidir se irá receber o resultado na memória ou se irá iterar no resultado inserido na camada de armazenamento da plataforma. Por padrão, esses resultados são armazenados para facilitar análises posteriores sobre esses dados.

A arquitetura proposta neste trabalho foi projetada de forma a facilitar a mudança de tecnologias dentro dos módulos através do desacoplamento entre eles. As tecnologias são, geralmente, escolhidas levando em conta as funcionalidades disponíveis, facilidade de uso e manutenção, desempenho, se é *open source* ou não, e o tamanho da comunidade envolvida. Com a criação de novas tecnologias e evolução cada vez maior delas, esses fatores vão influenciando na escolha de uma em detrimento de outras. Por isso, o desacoplamento dos módulos irá permitir que os desenvolvedores possam acompanhar as evoluções das tecnologias de *Big Data* e escolher aquelas que sejam de sua preferência.

CAPÍTULO 6

Avaliação

Este capítulo apresenta a avaliação de desempenho da solução proposta neste trabalho no processamento das STSR no contexto de Big Data, com o objetivo de avaliar se a solução foi capaz de resolver os seguintes problemas:

- Alta demanda computacional dos algoritmos de processamento das STSR.
- Baixa acurácia do algoritmo TWDTW na classificação do tipo de uso e cobertura do solo em algumas regiões geográficas.
- Grande volume de dados de sensoriamento remoto a serem armazenados e processados.

A Seção 6.1 mostra os resultados da avaliação do algoritmo P-INDEX utilizando arquiteturas paralelas *manycore* e *multicore* para o processamento de índices de sensoriamento remoto que irão compor os dados de entrada para os algoritmos de classificação das STSR. A Seção 6.2 apresenta os resultados da avaliação do P-TWDTW, que apresenta uma abordagem paralela para o algoritmo TWDTW, que possui alta demanda computacional no cálculo de medidas de similaridade das STSR. A Seção 6.3 apresenta a avaliação de desempenho da plataforma Big Sensing para armazenar, indexar, filtrar e recuperar um grande volume de dados das STSR.

6.1 Avaliação de Desempenho do algoritmo P-INDEX

Nesta seção avaliamos o P-INDEX para verificarmos o desempenho do algoritmo explorando as arquiteturas paralelas *multicore* na CPU e *manycore* na GPU. Este algoritmo permite calcular, de forma paralela, os índices de sensoriamento remoto que serão utilizados como observações das STSR. Estes dados serão utilizados como entrada para os algoritmos de processamento de STSR. Os testes realizados nesta seção calculam o índice de vegetação NDVI.

Nos testes em CPU foi utilizada uma máquina com processador Intel Core i7-2670QM (2.2GHz, 6MB Cache, 4 núcleos, 8 Threads) e 6GB de memória RAM. Na

execução dos testes da GPU foi utilizada uma placa NVIDIA GeForce GT525M com 1GB de memória disponível e 96 CUDA *cores* com *clock* máximo de 1200 MHz. Os testes na CPU foram executados com até 8 *threads*. Como a complexidade de I/O da CPU e GPU é a mesma em relação à leitura e escrita do disco, o tempo de leitura e escrita dos blocos do disco foram desprezados do resultado final para analisar a diferença de desempenho entre a CPU e a GPU. O algoritmo implementado para ser processado na CPU foi desenvolvido na linguagem C++ e na GPU utilizando a linguagem CUDA da NVIDIA. Ambos os algoritmos utilizaram a biblioteca GDAL do C++ para ler e escrever os blocos de imagem no disco.

Foram utilizadas quatro imagens de satélite, como entrada, com diferentes características nos testes: i) imagem *A*: cada banda com dimensão de 3944 x 3041 e 95 MB de uso da memória RAM; ii) imagem *B*: cada banda com dimensão de 7834 x 7059 e 442 MB de uso da memória RAM; iii) imagem *C*: cada banda com dimensão de 8907 x 10454 e 744 MB de uso da memória RAM; iv) imagem *D*: cada banda com dimensão de 25831 x 31208 e 6,5 GB de uso da memória RAM. Para cada imagem devem ser alocados o triplo do tamanho de cada banda da imagem. Dois terços deste espaço é utilizado para armazenar as bandas NIR e RED da imagem e um terço para armazenar o resultado do processamento do NDVI.

Para processar a imagem *A* devem ser alocados 285 MB de memória, e como as memórias da GPU e da CPU possuem tamanho maior que 285 MB, então a imagem *A* pode ser totalmente transferida para a GPU sem criar blocos menores. A Figura 6.1 contém o tempo de resposta do cálculo do índice NDVI na CPU e GPU. O processamento na CPU teve redução do tempo de resposta de aproximadamente 352% com o aumento de núcleos da CPU de 1 para 8. O tempo de resposta na GPU foi 2 vezes menor que o tempo de resposta da CPU com 8 núcleos e teve *speedup* de 7 vezes em relação ao tempo de resposta do algoritmo sequencial na CPU.

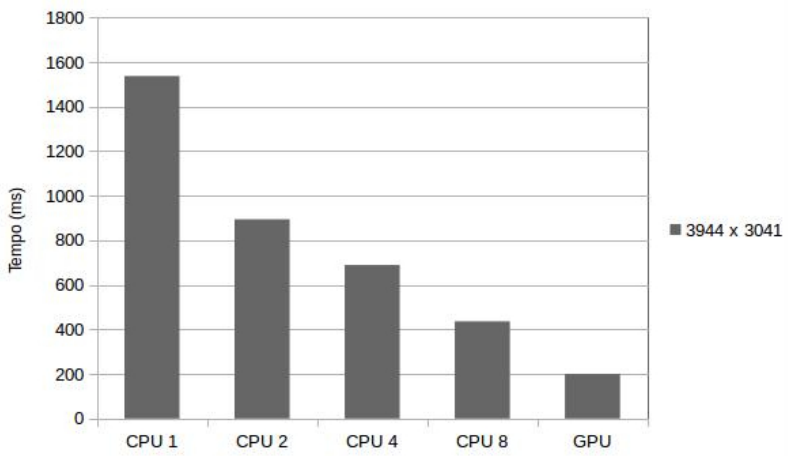
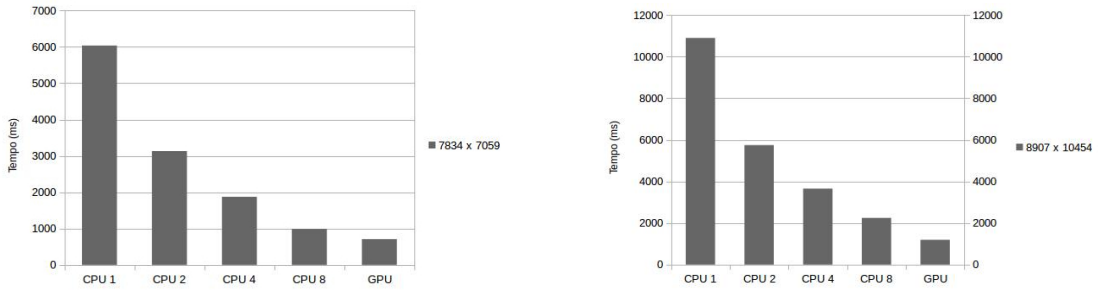


Figura 6.1: *Gráfico com o resultado do cálculo do NDVI com a imagem A.*

A Figura 6.2 apresenta o tempo de resposta do cálculo do NDVI das imagens *B* e *C*. Devem ser alocados 1,3 GB e 2,2 GB para processar as imagens *B* e *C* respectivamente. Como a GPU tem limite de memória de 1GB e a memória RAM limite de 6GB, estas imagens podem ser carregadas totalmente na memória RAM, mas são dividas automaticamente pelo algoritmo P-INDEX em blocos para serem processadas na GPU. Cada bloco de imagem tem tamanho aproximado de 600 MB para armazenar as bandas da imagem. O espaço restante na GPU é utilizado para armazenar o resultado do processamento.

Como pode ser visto na Figura 6.2(a), o processamento da imagem *B* teve escalabilidade de aproximadamente 6 vezes com o aumento de *threads* de 1 para 8. O tempo de resposta na GPU foi 39% menor que o tempo de resposta na CPU com 8 *threads* e teve speedup de 8 vezes em relação ao algoritmo sequencial na CPU. O processamento da imagem *C* teve escalabilidade de aproximadamente 5 vezes com o aumento do número de *threads* de 1 para 8 (Figura 6.2(b)). A GPU teve desempenho 80% melhor que a CPU com 8 *threads* e *speedup* de 9 vezes em relação ao algoritmo sequencial da CPU.



(a) *Resultado do cálculo do NDVI da imagem B* (b) *Resultado do cálculo do NDVI da imagem C*

Figura 6.2: *Tempo de resposta do cálculo do NDVI de imagens que cabem na memória da CPU, mas não cabem na memória da GPU.*

Para testar o caso onde a imagem não cabe na memória da CPU, foi utilizada a imagem *D* que possui resolução 25831 x 31208 com 6,5 GB de uso da RAM para cada banda da imagem e 19,5 GB de uso total da memória para armazenar as bandas da imagem, além do resultado final. Como a memória da CPU possui tamanho de 6 GB é necessário utilizar a abordagem de leitura de blocos de imagens do disco. Assim, são necessárias quatro leituras de blocos do disco, com cada bloco de 3,2 GB de tamanho. O restante da memória é utilizado para armazenar as estruturas intermediárias do algoritmo e o resultado final.

A GPU tem limite de 1 GB de memória e, por isso, foram gerados blocos de tamanho de 600 MB divididos entre as bandas da imagem. O gráfico da Figura 6.3 apresenta o tempo de processamento em milissegundos do algoritmo P-INDEX com a imagem *D*. Houve um ganho de aproximadamente 5 vezes com o aumento do número de *threads* de 1 para 8 na CPU. O tempo de processamento na GPU foi aproximadamente

53% menor que a CPU com 8 *threads*. A GPU teve um *speedup* de 9 vezes em relação ao algoritmo sequencial na CPU.

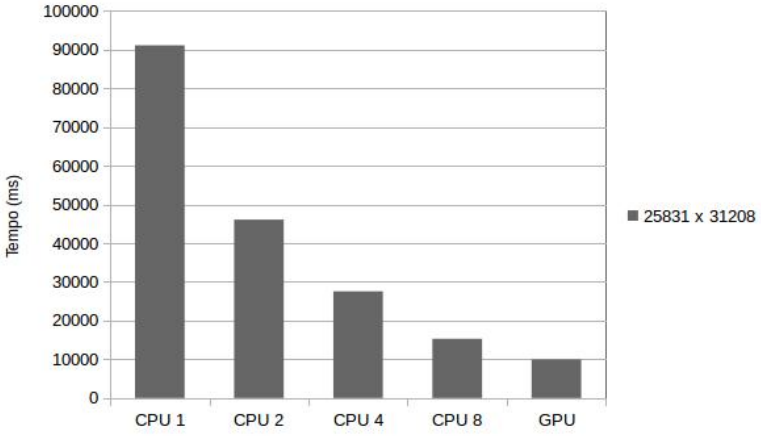


Figura 6.3: *Gráfico com o resultado do cálculo do NDVI da imagem D que não cabe na memória RAM.*

O *speedup* do algoritmo P-INDEX foi de até 9 vezes no processamento de todas as imagens em relação ao algoritmo sequencial. Quanto maior a imagem, maior foi o ganho de *speedup* entre a CPU e a GPU, exceto no processamento da imagem *D* com maior resolução (25831 x 31208) que apresentou um *speedup* igual ao da imagem *C*. Isto ocorreu devido à grande quantidade de transferências de blocos da CPU para a GPU para processar a imagem *D*. Portanto, quanto maior a memória disponível na GPU, melhor será o desempenho do algoritmo.

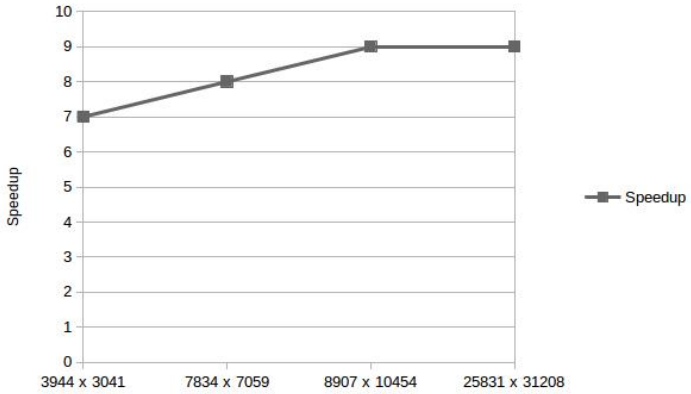


Figura 6.4: *Gráfico com o speedup do algoritmo P-INDEX*

Os testes apresentados nesta pesquisa foram construídos utilizando imagens com diferentes tamanhos para demonstrar o desempenho do algoritmo paralelo P-INDEX em cenários onde existe a necessidade de lidar com os limites de memória da CPU e da GPU. Houve um ganho de desempenho do algoritmo P-INDEX com o aumento do número de *threads* na CPU. A utilização da GPU obteve melhor desempenho que a CPU com 8 *threads* tendo até 9 vezes de *speedup* em relação ao algoritmo sequencial da CPU. Por

isso, o P-INDEX permite que os cálculos dos dados de entrada para os algoritmos de processamento de séries temporais sejam realizados, de forma eficiente, explorando as arquiteturas paralelas *multicore* e *manycore*.

6.2 P-TWDTW

Esta seção apresenta o resultado da avaliação de desempenho do P-TWDTW explorando arquiteturas paralelas *manycore* na Seção 6.2.1. Esta avaliação tem como objetivo mostrar que o P-TWDTW é capaz de paralelizar de forma eficiente o algoritmo TWDTW, reduzindo seu tempo de resposta, que possui complexidade de tempo quadrática com alta demanda de recursos computacionais de processamento. Na Seção 6.2.2 é realizada a avaliação da acurácia da classificação da cobertura e uso de solo, a partir da criação de meta-características utilizando as medidas de similaridade calculadas com o P-TWDTW.

6.2.1 Avaliação da Performance do Algoritmo P-TWDTW

Esta Seção tem como objetivo avaliar a performance dos algoritmos TWDTW e P-TWDTW executados na CPU e GPU, respectivamente. Esta avaliação visa identificar se o P-TWDTW é capaz de reduzir o tempo de resposta do TWDTW, de forma a permitir o processamento de séries temporais no contexto de Remote Sensing Big Data. O P-TWDTW implementa uma estratégia para processar as STSR utilizando os núcleos de arquiteturas paralelas *manycore*.

Ambiente Experimental e Bases de Dados

Os testes do TWDTW na CPU foram realizados em uma máquina com processador Intel Xeon E5-2686 v4 (Broadwell) com *clock* de 2.20 GHz e 61 GB de RAM. O algoritmo P-TWDTW foi executado na placa Tesla V100 da NVIDIA com 16 GB de memória disponível, 5120 núcleos de processamento CUDA e *clock* de 1455 MHz. O algoritmo TWDTW foi implementado em C++ [1] e o P-TWDTW foi implementado na GPU utilizando a linguagem de programação CUDA da NVIDIA.

Para comparar o tempo de resposta de ambos os algoritmos, foram geradas sinteticamente séries temporais V com 45 observações e padrões U com 23 observações. Na avaliação de performance, a utilização de bases sintéticas ou reais com o mesmo

[1] A versão original do TWDTW foi desenvolvido na linguagem de programação R. Neste trabalho, implementamos em C++ para comparar de forma justa com o P-TWDTW, já que a linguagem C++ permite atingir um desempenho melhor.

tamanho não faz diferença no desempenho dos algoritmos TWDTW e P-TWDTW, já que o tempo de resposta de ambos é impactado pelo número de observações das séries temporais e dos padrões, mas não pelo comportamento da série temporal (i.e. tendência, sazonalidade, etc). Esta base sintética foi gerada selecionando, randomicamente, séries temporais reais do Brasil extraídas do produto MOD13Q1. Apesar das séries temporais serem reais, dizemos que a base de dados é sintética porque existe a possibilidade de duplicação de séries temporais na base e ela foi criada, de forma sintética, para os testes de validação de performance.

Esse conjunto de testes visa avaliar o desempenho dos algoritmos com bases de dados de séries temporais de diferentes tamanhos. Neste cenário, foram geradas sinteticamente 40 padrões U de séries temporais, já que é raro em sensoriamento remoto existirem cenários de comparação de séries temporais com muitos padrões. Cada U pode representar, por exemplo, o padrão da série temporal de um determinado tipo de uso de solo, como o plantio de milho.

O número de séries temporais V que são comparadas com cada padrão U depende da resolução espacial dos dados de sensoriamento remoto e da área geográfica de estudo. Neste experimento, foram gerados 11 conjuntos contendo de 541 até 553894 séries temporais V dobrando o tamanho de cada conjunto. Cada um destes 11 conjuntos foi utilizado como entrada junto com o conjunto de 40 padrões para a avaliação do P-TWDTW frente ao TWDTW. Para validar o desempenho em um cenário com pouca quantidade de dados, foram utilizados 7 conjuntos de dados contendo entre 5 e 320 séries temporais e um conjunto de dados com 5 padrões. Em cada experimento, é realizado o cálculo da medida de similaridade de cada V para cada U.

O P-TWDTW junta um conjunto de padrões U formando um novo padrão U' e um conjunto de séries temporais V formando uma nova série temporal V' para calcular as medidas de similaridade entre cada U e V transferindo um conjunto de dados maior para a GPU, minimizando assim o tráfego da memória da CPU para a memória da GPU. Neste experimento, cada U' contém no máximo cinco padrões U e cada V' no máximo 541 séries temporais V.

Os algoritmos P-TWDTW e TWDTW foram comparados também em relação ao número de observações de U e V. Foram gerados 9 padrões com tamanhos entre 45 e 11520 e 9 séries temporais com tamanhos entre 450 e 115200 para avaliar a performance dos algoritmos processando grandes séries temporais. Nestas bases de dados sintéticas, as séries temporais reais, com 23 observações em U e 45 observações em V, foram duplicadas até atingir o número de observações do experimento, criando um comportamento sazonal ao longo do tempo. Este cenário permite avaliar os dois algoritmos com dados de satélites com alta resolução temporal, tais como a constelação

de satélites Planet [2] e o Jilin-1 [3], que são capazes de revisitar o mesmo local na Terra duas vezes ao dia.

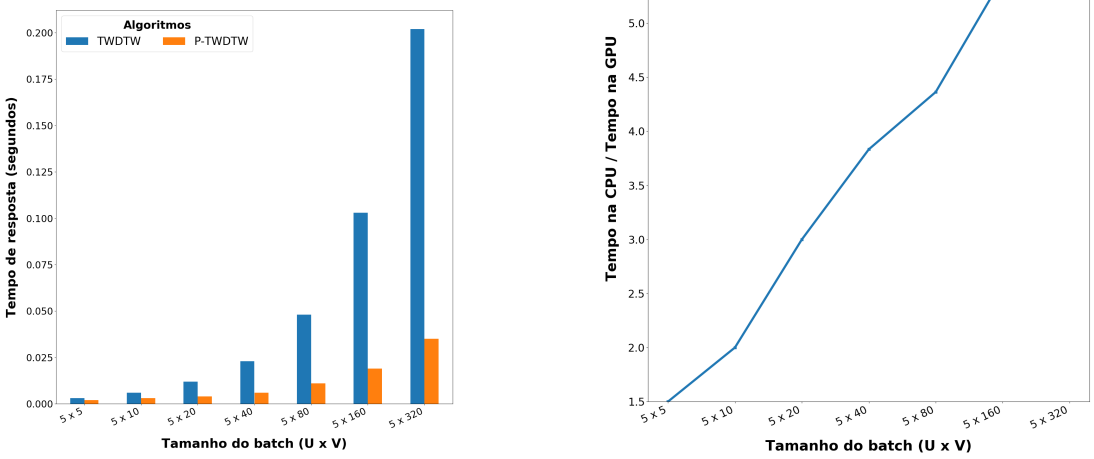
Resultados e Discussões

Esta seção apresenta os resultados e discussões a respeito do desempenho dos algoritmos TWDTW e P-TWDTW no processamento de STSR, comparando o tempo de resposta de ambos em diferentes cenários de avaliação. A Figura 6.5 apresenta a avaliação da performance dos algoritmos P-TWDTW e TWDTW no cálculo da medida de similaridade entre cada série temporal V e cada padrão U. Esta avaliação é realizada sobre um conjunto pequeno de séries temporais V, entre 5 e 320, e um número fixo de 5 padrões U, onde cada U possui 45 observações e cada V 23 observações. O eixo x da Figura 6.5 representa o número de séries temporais versus o número de padrões.

O gráfico da Figura 6.5(a) mostra que ambos os algoritmos tiveram um aumento no tempo de resposta com o aumento do número de séries. Houve um aumento linear no tempo de resposta do algoritmo TWDTW porque este algoritmo foi projetado para ser processado de forma serial e, por isso, os cálculos de medida de similaridade entre cada U e V devem ser escalonados na CPU. Como o P-TWDTW cria um novo padrão U' e uma nova série temporal V' a partir de um conjunto de padrões U e séries temporais V, respectivamente, e envia para a GPU, ele consegue paralelizar o cálculo da medida de similaridade entre cada U e V construindo a matriz de custo entre U' e V' de forma eficiente na GPU. Assim, o dominante neste caso é a transferência de U' e V' da memória da CPU para a GPU e quanto maior for o número de padrões e séries temporais, maior será o grau de utilização do recurso de processamento de arquiteturas *manycore* pelo P-TWDTW. Por isso, como pode ser visto no gráfico da Figura 6.5(b), quanto maior for o número de séries temporais V, maior fica a diferença de tempo de resposta entre o P-TWDTW e o TWDTW, sendo o P-TWDTW quase 6 vezes mais rápido que o TWDTW com 320 séries temporais.

A Figura 6.6 apresenta o tempo de resposta dos algoritmos P-TWDTW e TWDTW com um número fixo de 40 padrões e uma quantidade de séries temporais maior (entre 541 e 553984). Esta avaliação foi feita de forma separada nos gráficos das Figuras 6.6(a) e 6.6(b) para facilitar a visualização dos resultados. Os gráficos mostram que o tempo de resposta de ambos aumentou com o aumento do conjunto de dados. O P-TWDTW se manteve entre 10 a 11 vezes melhor que o TWDTW variando o tamanho das séries temporais entre 541 e 553984. Neste cenário, a diferença de resposta entre o P-TWDTW e o TWDTW não aumentou de forma considerável, já que o uso do processa-

[2] https://www.planet.com/products/planet-imagery/
[3] https://www.cgsatellite.com/imagery/static-imagery/



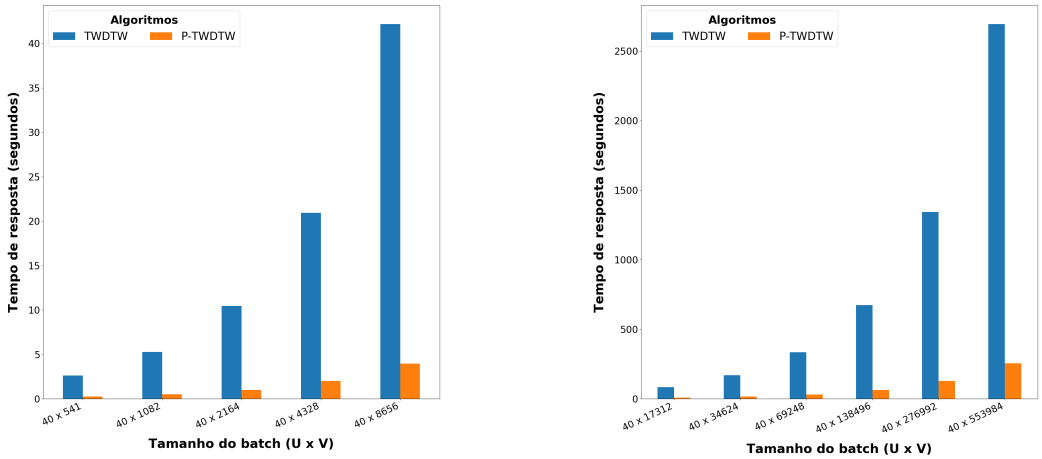
(a) *Gráfico com o tempo de resposta para os algoritmos P-TWDTW e TWDTW com um conjunto pequeno de séries temporais.*

(b) *Comparação do tempo de resposta dos algoritmos P-TWDTW e TWDTW.*

Figura 6.5: *Comparação dos algoritmos TWDTW e P-TWDTW em um cenário com poucas séries temporais. Foram criados padrões U e séries temporais V variando os seus tamanhos. O eixo x representa o número de séries temporais V e o número de padrões U no formato: número de padrões U x número de séries temporais de V. Cada padrão possui 45 observações e cada série temporal 23 observações.*

mento da GPU já estava alto com 541 séries temporais e permaneceu alto com o aumento do número de séries. Além disso, o custo de tráfego de dados entre a memória da CPU e da GPU aumentou de forma proporcional ao aumento da quantidade de séries temporais, impactando negativamente no desempenho do P-TWDTW. Comparando com os resultados da Figura 6.5, o P-TWDTW conseguiu um desempenho ainda melhor em comparação com o TWDTW com o aumento do volume de dados.

Os gráficos da Figura 6.7 apresentam o cenário de variação da resolução temporal dos dados de sensoriamento remoto. O gráfico da Figura 6.7(a) mostra o tempo de resposta dos algoritmos em um cenário de dados coletados de fontes de sensoriamento remoto com baixa ou média resolução temporal e o gráfico da Figura 6.7(b) cenários com alta resolução temporal. O tempo de resposta em ambos os algoritmos aumenta à medida que cresce o tamanho de U e V, mantendo o tempo de resposta do algoritmo P-TWDTW sempre menor que o TWDTW. O TWDTW possui uma estrutura interna em que o cálculo de cada célula da matriz de custo depende do resultado do cálculo de uma célula anterior, o que impede a paralelização. O P-TWDTW consegue paralelizar o processamento da matriz de custo entre os núcleos de processamento reduzindo, assim, o tempo de resposta.



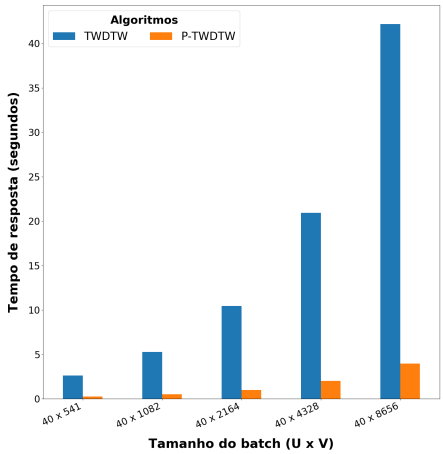
(a) *Tempo de resposta para conjuntos com 541 até 8565 séries temporais, fixando o número de 40 padrões.*

(b) *Tempo de resposta para conjuntos com 17312 até 553984 séries temporais, fixando o número de 40 padrões.*

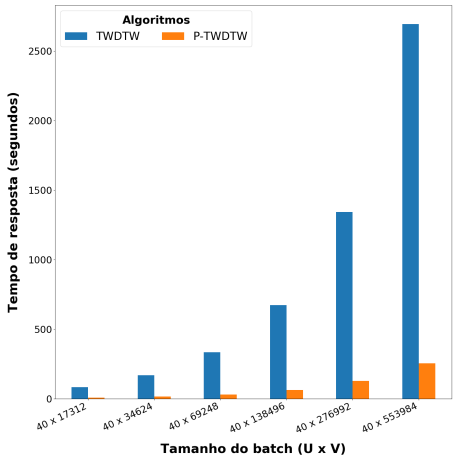
Figura 6.6: *Comparação dos algoritmos TWDTW e P-TWDTW em um cenário com uma grande quantidade de séries temporais. Foram criados padrões U e séries temporais V variando os seus tamanhos. O eixo x representa o tamanho de U e V no formato: tamanho de U x tamanho de V. O eixo y é o tempo de resposta de cada teste.*

O aumento do tamanho de U e V gerou um melhor aproveitamento dos recursos de processamento da GPU pelo P-TWDTW. Este melhor aproveitamento aumentou a diferença do tempo de resposta em relação ao TWDTW, como pode ser visto no gráfico na Figura 6.8. O tempo de resposta do P-TWDTW foi quase 13 vezes menor que o TWDTW com U de tamanho 11520 e V com tamanho 115200. A transferência de dados entre a memória da CPU e a memória da GPU impactou de forma negativa o P-TWDTW, principalmente com tamanhos menores de U e V. A partir de padrões com tamanho 1440 e séries temporais com tamanho 14400, a diferença de tempo entre o P-TWDTW e o TWDTW aumenta de forma mais suave porque os núcleos de processamento da GPU estão menos ociosos.

O P-TWDTW possui um mecanismo de controle da memória da GPU enviando um lote de padrões U e séries temporais V que não excedam sua capacidade. Ele sempre envia pelo menos um padrão U e uma série temporal V para a GPU, e, portanto, nos casos em que o tamanho somado de U e V exceder a capacidade da memória da GPU, o P-TWDTW não irá conseguir processar o cálculo da medida de similaridade. De qualquer forma, para atingir este limite de memória, o número de observações em U e V tem que ser muito alto. Por exemplo, nos experimentos das Figuras 6.7 e 6.8, o padrão U com 11520 observações possui tamanho aproximado de 11 MB, enquanto a série temporal V com 11520 observações possui tamanho de 115 MB.



(a) *Tempo de resposta para os algoritmos P-TWDTW e TWDTW no cálculo da medida de similaridade de padrões com tamanho entre 45 e 360 observações e séries temporais com tamanho entre 450 e 3600 observações.*



(b) *Tempo de resposta para os algoritmos P-TWDTW e TWDTW no cálculo da medida de similaridade de padrões com tamanho entre 720 e 11520 observações e séries temporais com tamanho entre 7200 e 115200 observações.*

Figura 6.7: *Comparação dos algoritmos TWDTW e P-TWDTW no cálculo da medida de similaridade entre um padrão U e uma série temporal V e variando os seus tamanhos. O eixo x representa o tamanho de U e V no formato: tamanho de U x tamanho de V. O eixo y é o tempo de resposta de cada teste.*

A paralelização do P-TWDTW nem sempre aproveita todos os núcleos de processamento das arquiteturas *manycore*. O P-TWDTW calcula a matriz de custo caminhando nas diagonais da matriz e paralelizando o processamento da diagonal entre várias *threads*, onde cada *thread* fica responsável por uma célula da diagonal. O P-TWDTW sempre aloca *d threads* para o processamento da diagonal, onde *d* é igual ao tamanho da diagonal principal. O problema é que as outras diagonais da matriz possuem tamanho menor que *d*, o que acaba desperdiçando recursos da GPU e deixando *n - d threads* ociosas, onde *n* é o tamanho da diagonal que está sendo processada. Este comportamento acabou impactando negativamente os resultados do P-TWDTW e o ganho dele frente ao TWDTW poderia ter sido ainda maior com o aproveitamento destas *threads* ociosas.

Ainda assim, o ganho de performance com o P-TWDTW em relação ao TWDTW foi expressivo, sendo quase 11 vezes mais eficiente que o TWDTW no cenário de alta resolução espacial e acima de 12 vezes mais eficiente no cenário de alta resolução temporal. Além disso, o P-TWDTW foi 246 mais rápido que a implementação original do TWDTW em R, provando ser uma alternativa eficiente para o processamento de STSR em Remote Sensing Big Data.

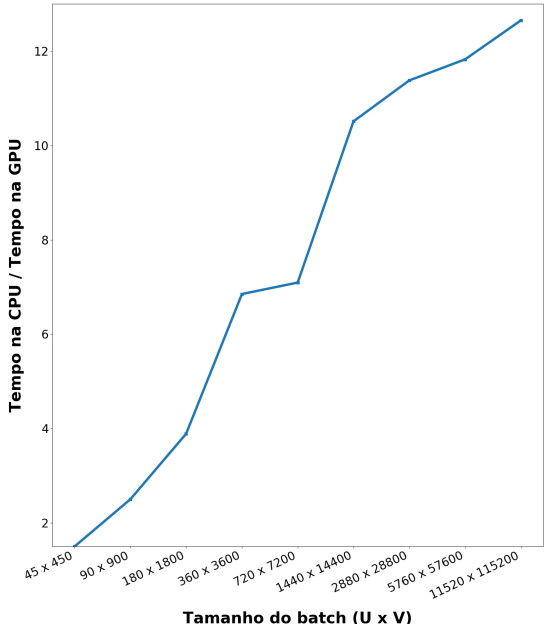


Figura 6.8: *Comparação da diferença dos algoritmos P-TWDTW e TWDTW no cálculo da medida de similaridade entre um padrão U e uma série temporal V, com a variação do tamanho de ambos. O eixo x representa o tamanho de U e V no formato: tamanho de U x tamanho de V. O eixo y é a diferença do tempo de resposta entre P-TWDTW e TWDTW.*

6.2.2 Avaliação da Acurácia da Classificação da cobertura e uso do solo utilizando o P-TWDTW

Experimentos realizados com séries temporais do produto MOD13Q1 de uma área de 5300 km^2 de Mato Grosso comprovam que as medidas de similaridade do P-TWDTW utilizados como variáveis de entrada para o algoritmo KNN possuem grande potencial para classificar diferentes tipos de uso de solo [120]. Esta avaliação foi realizada em [120] com 603 amostras divididas em cinco classes: “Floresta”, “Algodão-pousio”, “Soja-algodão”, “Soja-milho” e “Soja-milheto”. A acurácia global da classificação com intervalo de confiança de 95% foi de 97% a 99%. A acurácia do usuário e do produtor ficaram entre 90% e 100% para todas as classes, exceto para “Soja-algodão” onde a acurácia do usuário ficou entre 59% e 85%.

Entretanto, para regiões em que as amostras de dados não conseguem capturar o padrão correto de cada classe, o KNN com o P-TWDTW tende a não ter uma acurácia muito alta [77]. Por exemplo, na região da Califórnia, nos Estados Unidos, a acurácia da classificação foi de 80% [25]. A solução proposta por [96] reduz este impacto utilizando todos os valores das séries temporais das bandas e índices de vegetação como características para os algoritmos de aprendizado de máquina, criando espaços altamente dimen-

sionais. Nosso trabalho propõe a criação de meta-características a partir das medidas de similaridade entre as séries temporais V para cada padrão U, calculadas pelo P-TWDTW. Estas meta-características, juntamente com as bandas e índices de vegetação do trabalho de [96], são utilizadas como variáveis explicativas para os algoritmos de aprendizado de máquina. Esta seção visa comparar nossa proposta com a solução criada por [96].

Ambiente Experimental e Bases de Dados

A avaliação da acurácia foi realizada em uma máquina com processador AMD FX-8320E com *clock* de 3.2 GHz e 24 GB de RAM. Foram utilizadas 2115 séries temporais coletadas entre 2001 e 2016 do produto MOD13Q1 de uma área de 903357 km^2 do estado do Mato Grosso. Foram definidas nove classes de cobertura do solo, descritas na Tabela 6.1 juntamente com a quantidade de amostras por classe.

Classe	**Número de amostras**
Floresta	138
Cerrado	400
Pastagem	370
Soja-pousio	88
Pousio-algodão	34
Soja-algodão	399
Soja-milho	398
Soja-milheto	235
Soja-girassol	53

Tabela 6.1: *Quantidade de amostras por classe de cobertura de solo utilizadas para o treinamento dos algoritmos de aprendizado de máquina.*

A solução proposta nesta tese será comparada com o método proposto em [96]. O trabalho de [96] utiliza as observações das séries temporais dos índices de vegetação, NDVI e EVI, além das bandas NIR e MIR como características para o método de aprendizado de máquina SVM. Este método proposto em [96] será denominado SVM-Picoli nos experimentos. Nosso trabalho propõe o método que utiliza o vetor de meta-características F-TWDTW, definido na Seção 4.1.2, que utiliza as mesmas características de [96] adicionando as medidas de similaridade do P-TWDTW para cada classe de cobertura de uso de solo. Ou seja, serão adicionadas nove características no treinamento do SVM referentes as medidas de similaridade para cada uma das nove classes. Este método será denominado como SVM-TWDTW nos experimentos. O método KNN com as medidas de similaridade do P-TWDTW será definido como KNN-TWDTW.

Os padrões temporais de cada classe foram gerados utilizando os Modelos Aditivos Generalizados (GAM) para estimar a distribuição associada do conjunto de amostras de cada classe [96]. O GAM utiliza uma função suavizadora que aproxima as

séries temporais. Os padrões gerados para as nove classes podem ser visualizados na Figura 6.9.

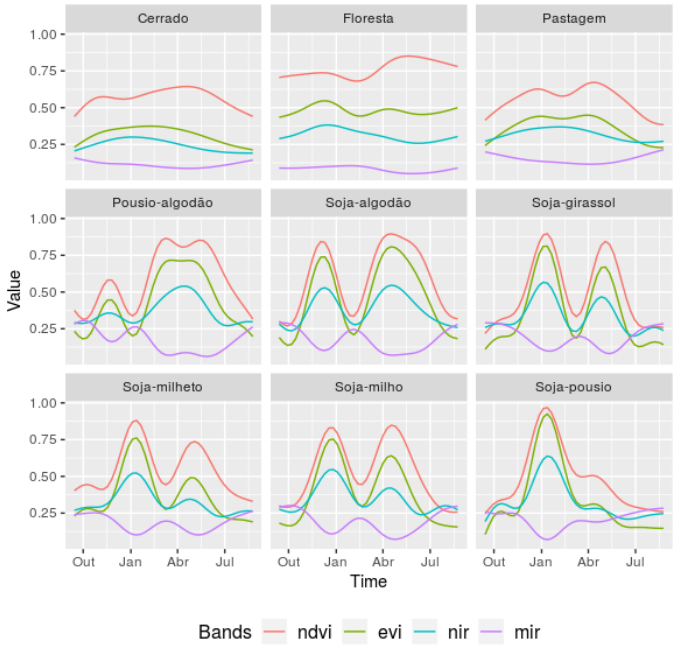


Figura 6.9: *Padrões temporais estimados das banda MIR e NIR e dos índices de vegetação NDVI e EVI, NIR and MIR utilizando o GAM [96].*

A função logística utilizada no P-TWDTW foi configurada com $\alpha = -0.1$ e $\beta = 50$. A acurácia do KNN-TWDTW foi definida utilizando o método de validação cruzada executando a classificação 100 vezes, cada uma formando uma partição de dados com 10% para construção dos padrões. Para estimar a acurácia da classificação utilizando o método SVM (SVM-TWDTW e SVM-Picoli), foi utilizado o método de validação cruzada k-fold com k=10 [59]. A classificação foi executada 100 vezes com 90% dos dados como amostras para treinamento e 10% como dados de teste. A acurácia final de ambas as execuções foi calculada a partir da média das 100 execuções.

O classificador SVM de ambos os métodos, SVM-Picoli e SVM-TWDTW, foi configurado com os mesmos parâmetros para permitir a análise do impacto da adição das medidas de similaridade do P-TWDTW como meta-características. Na implementação do SVM foi utilizada a biblioteca e1071 [4] em R na versão 1.7-2 com os seguintes parâmetros: kernel = "radial", degree = 3, coef0 = 0, cost = 10, tolerance = 0.001, epsilon = 0.1, cross = 0, scale = FALSE, cachesize = 1000.

[4] https://www.rdocumentation.org/packages/e1071/versions/1.7-2/topics/svm

Resultados e Discussões

Esta seção apresenta os resultados da avaliação de acurácia dos métodos KNN-TWDTW, SVM-Picoli e SVM-TWDTW. A descrição dos resultados apresenta os valores da acurácia do usuário (UA) e do produtor (PA) de cada classe de cobertura do solo. Acurácia do usuário são estimativas dos *pixels* corretamente classificados e está associada ao erro de comissão, que é o erro cometido ao atribuir um *pixel* a uma classe, quando este pertence a alguma outra classe. A acurácia do produtor refere-se à probabilidade de determinada classe ter sido corretamente classificada de acordo com os pontos de referência de campo e está associada ao erro de omissão, que ocorre quando deixamos de mapear um pixel de uma classe corretamente. A acurácia do usuário está associada a confiabilidade de cada classe mapeada e a acurácia do produtor está associada à sensibilidade do classificador, ou seja, a capacidade de distinguir corretamente determinada classe das demais. Além destes, será calculada a acurácia global que é a estimativa da proporção de acerto global dos classificadores.

A Tabela 6.2 apresenta a matriz de confusão com o número de amostras classificadas em cada classe e a média da acurácia do produtor e do usuário da avaliação do KNN-TWDTW, com intervalo de confiança de 95% depois de 100 execuções, utilizando o método de validação cruzada. O método teve uma baixa acurácia global (OA) de 78% explicada por conjuntos de padrões que possuem comportamento temporal semelhante. Na Figura 6.9 podemos observar, por exemplo, que Cerrado, Pastagem e Floresta possuem padrões semelhantes de séries temporais. Por isso, o KNN-TWDTW teve boa acurácia na identificação de padrões de Floresta, com PA igual a 100%, mas em contrapartida classificou erroneamente como Floresta amostras de Cerrado e Pastagem o que ocasionou um baixo UA para Floresta. Isto acontece porque para cada série temporal V, o KNN-TWDTW calcula a distância para cada padrão U e atribui a V a classe referente ao padrão com menor distância. Mas, nos casos em que os padrões são semelhantes, a distância entre V e cada U fica muito próxima e acaba gerando erros de classificação. O mesmo comportamento ocorre para o conjunto de padrões Soja-Milho, Soja-milheto e Soja-girassol.

Para diminuir o impacto da similaridade temporal dos padrões, o trabalho de [96] propôs utilizar o SVM como classificador utilizando os índices de vegetação NDVI e EVI, além das bandas RED e MIR como características. O resultado da classificação utilizando este método é apresentado na matriz de confusão da Tabela 6.3, onde o classificador obteve acurácia global de 92,3%, aproximadamente 12,3% maior que o KNN-TWDTW. A matriz mostra que houve confusão entre as classes Soja-milho e Soja-milheto, assim como no KNN-TWDTW, devido as características similares dos padrões das classes Soja-milho e Soja-milheto, mas com uma taxa de erro bem menor. O milho e o milheto são gramíneas com folhas lanceoladas, onde o milho pode atingir até 3,5 metros e o milheto

		1	2	3	4	5	6	7	8	9	UA (%)
1	**Pastagem**	332	3	5	0	0	0	8	0	0	95,4
2	**Soja-milho**	6	257	32	37	1	14	0	0	0	74,1
3	**Soja-milheto**	1	19	155	2	0	1	0	0	3	85,6
4	**Soja-algodão**	1	10	2	314	4	0	0	0	0	94,9
5	**Pousio-algodão**	1	3	2	32	29	0	0	0	0	43,3
6	**Soja-girassol**	0	100	29	12	0	37	0	0	0	20,8
7	**Cerrado**	11	0	0	0	0	0	303	0	0	96,5
8	**Floresta**	12	0	0	0	0	0	89	138	0	57,7
9	**Soja-pousio**	6	6	10	2	0	1	0	0	85	77,3
	PA (%)	89,7	64,6	65,9	78,7	85,3	69,8	75,7	100	96,6	**OA:78%**

Tabela 6.2: *Matriz de confusão de séries temporais de imagens MODIS obtida através da classificação utilizando o KNN-TWDTW com os valores da acurácia do produtor (PA) e acurácia do usuário (UA) para cada classe, além da acurácia global (OA).*

entre 1,5 e 3 metros, podendo chegar a 5 metros e essa similaridade acabou confundindo os métodos KNN-TWDTW e SVM-Picoli.

A Tabela 6.4 mostra a matriz de confusão da classificação utilizando o método SVM-TWDTW. A utilização das medidas de similaridade do P-TWDTW ocasionou um impacto positivo na classificação aumentando em 1,5% a acurácia global em relação ao SVM-Picoli. Este impacto positivo com acurácia global de 93,8% é explicado porque utilizando as medidas de similaridade do P-TWDTW junto com as características NDVI, EVI, NIR e MIR, o SVM-TWDTW consegue combinar os métodos SVM-Picoli e o KNN-TWDTW. Neste experimento, das 2115 amostras utilizadas, o KNN-TWDTW conseguiu classificar corretamente 125 amostras que o método SVM-Picoli classificou de forma errônea, sendo que este número cai para 91 quando comparado com o método SVM-TWDTW. Um exemplo claro deste cenário pode ser visto comparando o PA de Floresta entre o KNN-TWDTW (Tabela 6.2) e o SVM-Picoli (6.3), onde o KNN-TWDTW teve acurácia de 100% contra 97,8% do SVM-Picoli. Na Tabela 6.4 é possível ver que o SVM-TWDTW teve o mesmo resultado de 100% do classificador KNN-TWDTW, sendo assim também melhor que o SVM-Picoli.

Esta combinação do SVM com as medidas de similaridade do TWDTW também possui cenários que geram classificações errôneas. Por exemplo, o PA de Soja-pousio do KNN-TWDTW (96,6%) foi menor que o PA do SVM-Picoli (100%) porque o KNN-TWDTW fez confusão entre as classes Soja-milheto e Soja-pousio. O SVM-TWDTW sofreu este efeito da classificação errada do KNN-TWDTW e também teve um PA exatamente de 96,6%. Já na classificação das amostras de Soja-milheto, o SVM-TWDTW teve acurácia menor que o KNN-TWDTW e o SVM-Picoli, com UA igual a 82,7% contra 86,4% do SVM-Picoli e 85,6% do KNN-TWDTW. O SVM-TWDTW teve um número maior de confusão entre as amostras das classes Soja-milheto e Soja-Milho que os outros

		1	2	3	4	5	6	7	8	9	UA (%)
1	**Pastagem**	355	4	8	0	1	0	2	2	0	95,4
2	**Soja-milho**	4	350	33	24	0	7	0	0	0	83,7
3	**Soja-milheto**	1	25	191	1	0	3	0	0	0	86,4
4	**Soja-algodão**	0	13	2	370	5	4	0	0	0	93,9
5	**Pousio-algodão**	0	1	0	3	28	0	0	0	0	87,5
6	**Soja-girassol**	0	4	1	1	0	39	0	0	0	86,6
7	**Cerrado**	9	1	0	0	0	0	396	1	0	97,3
8	**Floresta**	1	0	0	0	0	0	2	135	0	97,8
9	**Soja-pousio**	0	0	0	0	0	0	0	0	88	100
	PA (%)	95,9	87,9	81,2	92,7	82,3	73,5	99	97,8	100	**OA:92,3%**

Tabela 6.3: *Matriz de confusão de séries temporais de imagens MODIS obtida através da classificação utilizando o método SVM-Picoli com as características NDVI, EVI, NIR e MIR. São apresentados na matriz a comparação de classificação entre as classes e os valores da acurácia do produtor (PA) e acurácia do usuário (UA) para cada classe, além da acurácia global (OA).*

métodos, o que ocasionou a rotulação errada de algumas amostras de Soja-milho como Soja-milheto. Entretanto, o SVM-TWDTW teve maior PA que o SVM-Picoli (85,1% contra 81,2%) e o KNN-TWDTW (85,1% contra 65,9%) conseguindo identificar um número maior de amostras de Soja-milheto.

Em todos os outros cenários, a Tabela 6.4 mostra que as acurácias de usuário e produtor do método SVM-TWDTW foram melhores que as do SVM-Picoli e do KNN-TWDTW. O SVM-Picoli errou a classificação de 145 amostras corretamente rotuladas com o SVM-TWDTW, enquanto o cenário oposto ocorreu 113 vezes. Os resultados mostram que o método SVM-TWDTW trouxe avanços na acurácia de mapeamento do uso do solo com o uso das medidas de similaridade do P-TWDTW como características do SVM. Até então, a maior acurácia global no mapeamento do uso de solo do Mato Grosso foi atingida com o método SVM-Picoli [96].

6.3 Avaliação da plataforma BigSensing

Esta Seção avalia a performance da plataforma BigSensing em explorar arquiteturas de memória distribuída para processar um grande volume de dados de STSR. A Seção 6.3.1 apresenta a avaliação do motor inteligente de busca SmarT que escolhe, em tempo real, entre o Elasticsearch, Apache Spark e SciDB, o melhor sistema para filtrar e recuperar os dados das STSR. A Seção 6.3.2 apresenta os resultados da capacidade da plataforma em distribuir o processamento de algoritmos de STSR em um *cluster* de computadores. A plataforma deve ser capaz de receber, indexar e armazenar um grande volume de dados de STSR, com uma arquitetura distribuída projetada para ingerir um grande

		1	2	3	4	5	6	7	8	9	UA (%)
1	**Pastagem**	359	2	4	1	0	0	3	0	0	97,3
2	**Soja-milho**	1	358	29	22	1	4	0	0	0	86,3
3	**Soja-milheto**	4	27	200	3	0	5	0	0	3	82,7
4	**Soja-algodão**	1	11	2	373	3	0	0	0	0	95,6
5	**Pousio-algodão**	0	0	0	0	30	0	0	0	0	100
6	**Soja-girassol**	0	0	0	0	0	44	0	0	0	100
7	**Cerrado**	5	0	0	0	0	0	397	0	0	98,7
8	**Floresta**	0	0	0	0	0	0	0	138	0	100
9	**Soja-pousio**	0	0	0	0	0	0	0	0	85	100
	PA (%)	97	89,9	85,1	93,4	88,2	83	99,2	100	96,6	**OA:93,8%**

Tabela 6.4: *Matriz de confusão de séries temporais de imagens MODIS obtida através da classificação utilizando o método SVM-TWDTW com as características NDVI, EVI, NIR e MIR, além das medidas de similaridade calculadas no P-TWDTW.*

volume de dados de STSR por segundo. Esta avaliação é apresentada na Seção 6.3.3.

6.3.1 Avaliação do Motor Inteligente de Busca SmarT

Esta seção avalia a capacidade do SmarT de escolher o melhor sistema de Big Data para cada cenário. Três sistemas são avaliados neste contexto: i) Elasticsearch versão 7.0.1, ii) SciDB versão 18.3 e iii) Apache Spark versão 2.3.2. A hipótese deste trabalho é que nenhum destes três sistemas consegue ser eficiente em todos os cenários de consultas com filtros espaciais e temporais sobre a base de dados de STSR.

Ambiente Experimental e Bases de Dados

Os testes foram realizados em um *cluster* de computadores com 8 servidores no ambiente de *Cloud Computing* da Amazon EC2[5]. Os três sistemas (Elasticsearch, Spark e SciDB) foram configurados nas 8 máquinas inserindo 7,2 bilhões de observações. Cada um destes servidores está equipado com processadores Intel Xeon® Platinum 8175 de até 3,1 GHz com 16 vCPUs, 128 GB de memória RAM e 600 GB de SSD. As máquinas foram conectadas por uma rede Ethernet 10 Gbit/segundo com largura de banda dedicada de 3500 Mbps.

A bateria de testes visou medir o tempo de resposta de cada sistema para filtrar e extrair os resultados de cada consulta. Esta análise é importante para verificar se o SmarT é capaz de reduzir o tempo de processamento de filtragem e recuperação dos dados. Para realizar esta medição, foi utilizada uma máquina para enviar as requisições para o *cluster*

[5]https://aws.amazon.com/pt/ec2/.

e coletar as métricas. Esta máquina possui um processador Intel Xeon E5-2676 v3 de 2.4 GHz com 8 núcleos, 32 GB de memória RAM e 1TB de SSD.

Foram armazenadas mais de 7,2 bilhões de observações referentes a 16777216 *pixels* de uma área de 800.824 km^2 que cobre um pedaço do Centro-Oeste e Sudeste do Brasil, como pode ser visto na Figura 6.10. Cada *pixel* possui uma série temporal com 435 observações entre 2000 e 2019 extraídas do produto MOD13Q1 da NASA[6] (National Aeronautics and Space Administration), com uma periodicidade de 16 dias entre observações e resolução espacial de 250 metros.

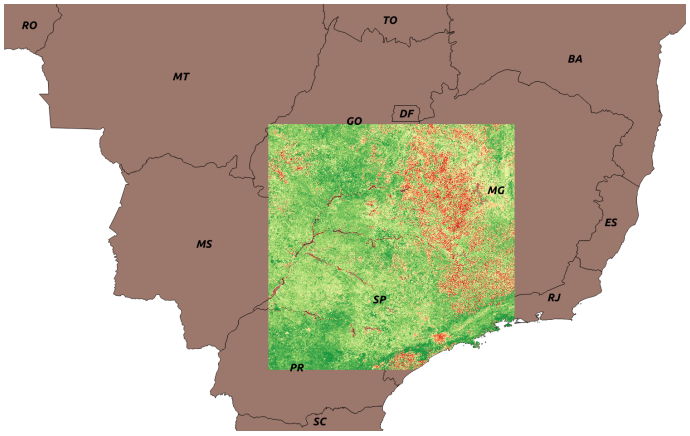


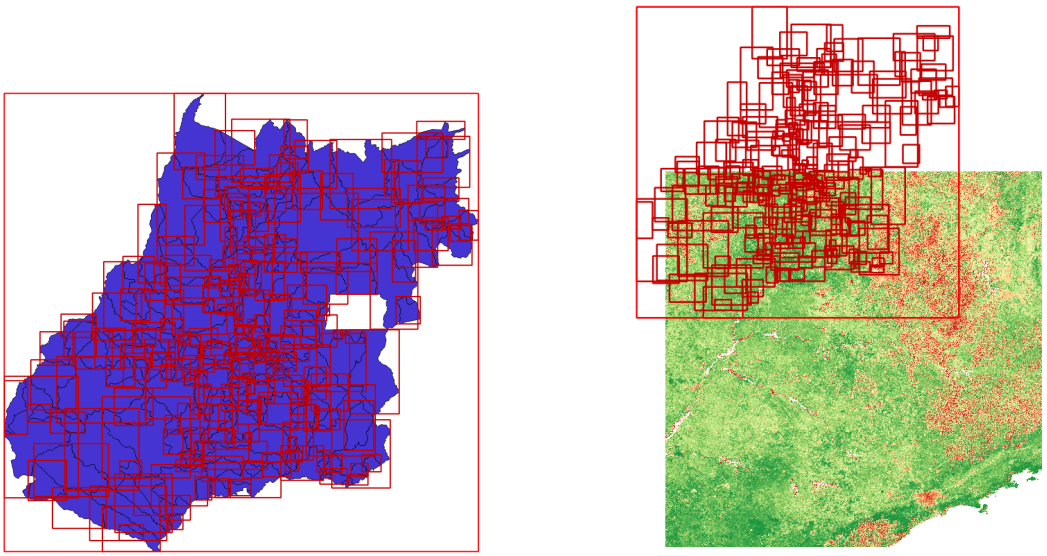
Figura 6.10: *Área de 800824 km^2 de onde foram retirados os dados do índice de vegetação NDVI do produto MOD13Q1. Esta área cobre um pedaço do Centro-Oeste e Sudeste do Brasil.*

Foram executadas consultas com restrições espaciais e temporais como o objetivo de avaliar a performance de cada um dos três sistemas e do SmarT desenvolvido para escolher o melhor de acordo com o cenário da consulta. As restrições espaciais foram retiradas a partir do MBR [7] dos polígonos que delimitam os municípios do Estado de Goiás, como pode ser visto na Figura 6.11(a). Estas áreas do Estado de Goiás foram escolhidas por permitirem avaliar diversos cenários de restrição espacial (Figura 6.11(b)), filtrando desde áreas que não têm resultados na consulta quanto áreas que retornam mais de 2 bilhões de resultados, como no caso do MBR do Estado de Goiás.

As restrições temporais foram criadas para avaliar a performance dos sistemas em aplicar o filtro temporal sobre os dados. A Tabela 6.5 apresenta as faixas de tempos definidas como restrição temporal nas consultas sobre a base de dados e o número de objetos retornados na filtragem feita na base de dados utilizando somente o eixo temporal.

[6]Esta base de dados foi disponibilizada pelo LAPIG (Laboratório de Processamento de Imagens e Geoprocessamento)

[7]Minimum Bounding Rectangle ou Retângulo Envolvente Mínimo. Este retângulo é formado a partir da observação dos limites geométricos mínimo e máximo do contorno do objeto, e é expresso pelas coordenadas dos pontos inferior esquerdo e superior direito [14].



(a) *MBRs de cada município e de todo Estado de Goiás que irão ser utilizados nas restrições espaciais.*

(b) *Os MBRs filtram um pedaço da área de avaliação permitindo testar diversos cenários de teste em relação ao número de objetos retornados na consulta.*

Figura 6.11: *Mapa com os MBRs dos municípios do Estado de Goiás utilizados nas consultas.*

Utilizando os filtros espaciais e temporais ao mesmo tempo, as consultas retornam desde 0 até 333 milhões de resultados.

No total, foram geradas 1983 restrições de consulta a partir de 247 filtros espaciais, 7 filtros temporais e 1729 (247 * 7) filtros espaço-temporais que combinam os 247 filtros espaciais e 7 temporais. Para cada uma das 1983 restrições, foram feitas duas consultas. A primeira consulta visa analisar a performance de cada sistema para filtrar sem recuperar os resultados. A segunda consulta tem como objetivo analisar a performance de cada sistema para filtrar e recuperar os resultados da consulta. Cada consulta foi executada 10 vezes, coletando os tempos de resposta e medidas de CPU, rede, memória e disco. Essas medidas foram coletadas e utilizadas para treinar o SmarT, para escolher o melhor sistema de acordo com o cenário da consulta. Ao todo, foram executadas 118980 consultas, construídas a partir de 1983 restrições, com duas consultas para cada restrição, sendo executadas 10 vezes em cada uma dos três sistemas (1983 * 2 * 10 * 3).

O SciDB e o Apache Spark foram instalados no *cluster* com as configurações padrões, sem nenhuma otimização ou customização. Em relação ao Elasticsearch, foi alterado o valor padrão de 1 GB para memória *heap* da JVM[8] para 48 GB, já que as consultas estavam causando estouro da memória *heap* durante os testes. O restante da

[8]Elasticsearch foi desenvolvido na linguagem Java

Faixa de tempo da restrição temporal	Número aproximado de objetos retornados
2016-01-01 até 2016-05-24	167 milhões
2016-01-01 até 2016-10-31	335 milhões
2016-01-01 até 2017-04-07	503 milhões
2016-01-01 até 2017-09-14	671 milhões
2016-01-01 até 2018-02-18	839 milhões
2016-01-01 até 2018-07-28	1 bilhão
2016-01-01 até 2019-01-01	1,17 bilhão

Tabela 6.5: *Definição das restrições temporais e do número de itens retornados em cada consulta que for feita somente com a filtragem do eixo do tempo.*

memória da máquina ficou alocada para os outros sistemas instalados nos servidores. Os testes no Spark foram executados utilizando o ecossistema Hadoop versão 3.1.1, com o Apache Yarn escalonando as consultas e gerenciando os recursos do *cluster* consumidos pelo Spark, além do HDFS como sistema de armazenamento. Os dados do Spark estão no HDFS no formato Parquet com a compressão Snappy [9] que, comparado ao formato CSV, reduz em até 34 vezes o tempo de filtragem em uma consulta e utiliza até 87% menos espaço em disco[10]. O Spark, SciDB e o Elasticsearch possuem sua própria camada de armazenamento e, por isso, os dados são replicados em cada um dos sistemas.

Resultados e Discussões

Esta seção apresenta os resultados e discussões a respeito do SmarT em escolher o sistema para execução da consulta em cada cenário. Primeiramente, iremos analisar a performance do Elasticsearch, Apache Spark e SciDB na filtragem espacial e temporal sobre as séries temporais dos *pixels* das imagens de sensoriamento remoto. Depois, iremos comparar a performance do SmarT em relação aos sistemas para definir se houve algum impacto positivo no tempo de resposta.

	Apache Spark	Elasticsearch	SciDB
Média do tempo de resposta (ms)	321	4	398
Desvio padrão do tempo de resposta (ms)	697	12	332

Tabela 6.6: *Análise do tempo de resposta (ms) das consultas para filtrar sem recuperar os resultados nos três sistemas*

A Tabela 6.6 apresenta o tempo de resposta das consultas em cada um dos sistemas para filtrar sem recuperar os resultados das consultas. O Elasticsearch teve o

[9] https://github.com/google/snappy
[10] https://dzone.com/articles/how-to-be-a-hero-with-powerful-parquet-google-and

melhor desempenho em relação à média do tempo de resposta das 10 execuções das consultas devido à sua eficiente estrutura de indexação temporal e espacial. O Apache Spark teve na média melhor tempo de resposta do que o SciDB, mas com um desvio padrão mais alto, porque o Spark teve melhor desempenho para consultas que retornam entre 100 e 500 mil resultados, mas um pior desempenho para consultas com mais de 500 mil resultados, como pode ser visto na Figura 6.12.

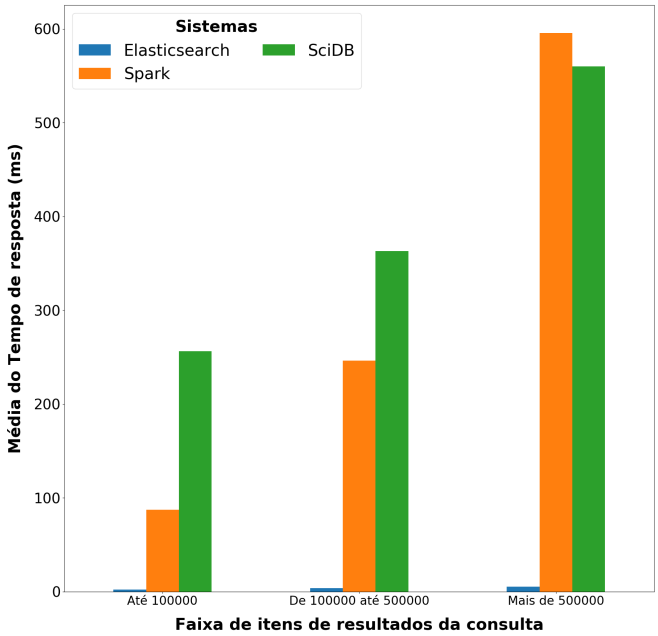


Figura 6.12: *Comparação do tempo médio de resposta (ms) das consultas nos três sistemas filtrando sem recuperar os dados.*

O Spark tem que percorrer todos os arquivos Parquet com os dados para filtrar os resultados, e quando a base de dados é maior que o tamanho da memória, os resultados intermediários precisam ser serializados como objetos Java e armazenados em dispositivos de armazenamento secundários (disco ou SSD) ou recalculados. Este *overhead* de serialização e desserialização é crítico para o Spark. Já o SciDB possui uma otimização para acessos de I/O, minimizando o *overhead* de acesso aos arquivos.

A Tabela 6.7 mostra a média do tempo de resposta para filtrar e recuperar o resultado da consulta. O Elasticsearch teve o pior desempenho entre os três sistemas, sendo mais de 10 vezes pior que o SciDB e mais de 7 vezes pior que o Spark. Esta diferença de desempenho do Elasticsearch, vista nas Tabelas 6.6 e 6.7, é explicada pelo fato de que o Elasticsearch possui um mecanismo interno de indexação muito eficiente, mas sua performance depende do tamanho do conjunto de dados retornados. Quando o resultado a ser recuperado fica maior que um determinado limite[11], o tempo de

[11] Este limite depende muito da configuração do Elasticsearch e da infraestrutura de *hardware* do **cluster**.

resposta aumenta consideravelmente [112], de forma que, o Elasticsearch passa a ter um desempenho pior que os outros dois sistemas.

	Apache Spark	Elasticsearch	SciDB
Média do tempo de resposta (ms)	822	6291	605
Desvio padrão do tempo de resposta (ms)	1842	9295	430
Tempo máximo de resposta (ms)	49986	78574	2946

Tabela 6.7: *Análise do tempo de resposta (ms) das consultas nos três sistemas*

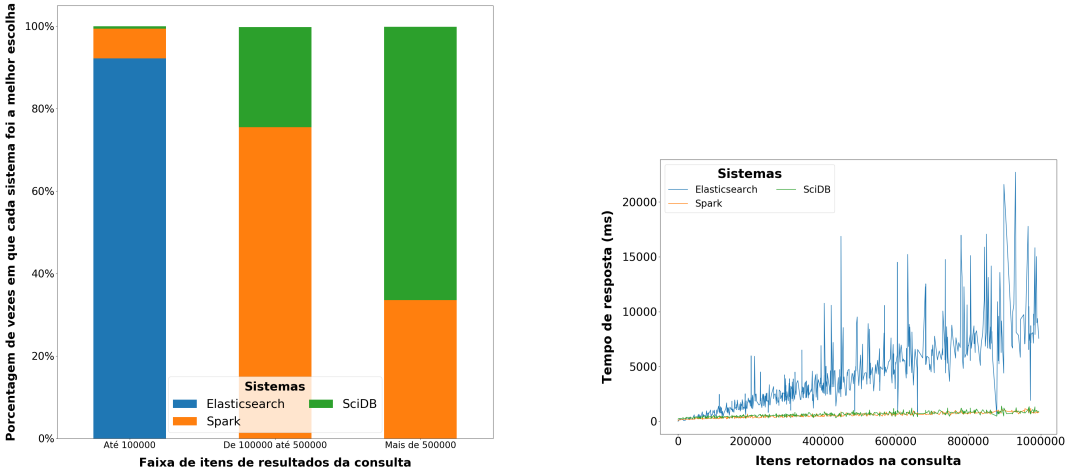
A Figura 6.13 mostra que para um número de resultados pequeno (menos de 100 mil), o Elasticsearch é a melhor opção entre os três sistemas, mas a partir de 100 mil resultados retornados, o SciDB e o Spark se tornam melhores opções, como pode ser visto na Figura 6.13(a). Na Figura 6.13(b) é possível visualizar que, enquanto o tempo de resposta teve ligeiro aumento no SciDB e no Spark com até 1 milhão de resultados, o Elasticsearch aumenta de forma linear conforme aumenta o número de resultados. Este comportamento é explicado devido ao fato do Elasticsearch ter sido projetado para filtrar um grande volume de dados, mas recuperar um conjunto menor de resultados mais relevantes e, por isso, sofre um alto impacto negativo com o aumento do número de dados retornado [12].

O Spark armazena os dados no formato Parquet, que consome menos espaço em disco do que o formato JSON [3] utilizado pelo Elasticsearch e, por isso, consegue recuperar os resultados de forma eficiente. O SciDB possui um sistema interno eficiente de recuperação dos dados do disco que fica mais evidente com resultados maiores que 500 mil, onde seu desempenho é melhor que o Spark. Para um grande número de resultados, o Apache Spark, que é um sistema de processamento em memória, precisa serializar os dados como objetos JAVA e armazenar no disco impactando assim seu desempenho. O SciDB é um sistema projetado para lidar com dados de séries temporais, ao contrário do Spark, que é um motor de processamento geral para grande volume de dados. Este cenário fica claro para um grande volume de dados, onde o SciDB consegue ter um desempenho melhor que o Spark [125] aproveitando seu eficiente armazenamento, indexação e recuperação de grande volume de dados multidimensionais.

As Tabelas 6.6 e 6.7 mostram que a diferença entre o tempo de resposta do SciDB e do Spark para filtrar e recuperar os dados não foram tão significativas quanto o Elasticsearch. O SciDB possui uma estrutura interna otimizada que consegue manter um baixo desvio padrão no tempo de resposta com tempo máximo de 2946 milissegundos de resposta. Já o Spark, com o armazenamento dos dados no formato Parquet, consegue

[12] https://www.elastic.co/guide/en/elasticsearch/reference/current/general-recommendations.html

ser mais otimizado que o Elasticsearch, mas tem um tempo máximo maior (49986 milissegundos).



(a) *Comparação do tempo de resposta nos três sistemas variando o número de objetos retornados.*

(b) *Comparação do aumento do tempo de resposta nos três sistemas com o aumento do número de resultados retornados nas consultas.*

Figura 6.13: *Comparação do tempo médio de resposta (ms) das consultas nos três sistemas filtrando e recuperando os dados. Foram utilizadas três faixas para avaliação da recuperação dos resultados: i) até 100000 resultados, ii) entre 10000 e 500000 e iii) mais de 500000 resultados.*

Para efeitos de comparação dos três sistemas, o Elasticsearch é recomendado para um volume de resultados menor, o Spark para um volume de resultados médio e o SciDB para um volume de dados maior. O desafio aqui, entretanto, é a escolha do sistema, já que a definição de um conjunto de resultados menor, médio ou maior é subjetiva e depende da configuração de *hardware* do *cluster*, dos sistemas disponíveis e dos cenários de filtragem espacial e temporal destes sistemas.

É possível perceber nos gráficos da Figura 6.13, que nenhum sistema domina totalmente os outros, analisando a porcentagem de vezes que cada um teve o menor tempo de resposta em cada faixa de dados retornados na consulta. Este resultado leva a elaboração de uma hipótese que outras variáveis também impactam no tempo de resposta dos sistemas. Este trabalho propõe um motor inteligente de busca que permite a escolha automática do melhor sistema a partir de diversas variáveis disponíveis no momento da consulta: *area*, *time_interval*, *count*, *swap_free*, *mem_free*, *load_one*, *load_five*, *load_fifteen*, *cpu_user*, *cpu_system*, *bytes_in*, *bytes_out*.

As características *area* e *time_interval* são geradas no momento da consulta a partir das restrições espacial e temporal, respectivamente. A variável *count* é gerada a partir de um serviço na plataforma BigSensing que retorna o número estimado de

resultados da consulta dada a restrição espacial e temporal. As características restantes são geradas a partir de medidas coletadas pelo sistema de monitoramento das máquinas da plataforma BigSensing, utilizando o sistema Ambari [13]. Estas variáveis são descritas de forma detalhada na Seção 5.2.

Na implementação, foi utilizado o algoritmo supervisionado XGBoost [18] da plataforma *open source* H2O [14], versão 3.24.0.4. Os experimentos foram executados utilizando o método de validação cruzada denominado k-fold, com k=5, com os valores padrões para os parâmetros do algoritmo XGBoost na plataforma H2O. Foram utilizadas as medidas coletadas nas 118980 consultas como amostras de treinamento, extraindo 75% para a base de treinamento e 25% para a base de teste.

O SmarT obteve uma acurácia geral de 90,1% na escolha do melhor sistema e, mesmo nos casos em que não escolhe o melhor, existe uma diferença pequena de tempo de resposta se tivesse escolhido sempre o melhor sistema para cada consulta. Em 95% dos casos, o sistema escolhido pelo SmarT é o melhor ou tem tempo de resposta até 10% maior do que o tempo do melhor sistema. Como pode ser visto na Tabela 6.8, utilizando o SmarT, o tempo médio de resposta das consultas foi de 496ms contra 482ms se tivesse escolhido sempre o melhor sistema (coluna “Best”).

	SmarT	Best	Apache Spark	Elasticsearch	SciDB
Todos os Sistemas	496	482	822	6291	605
Elasticsearch + SciDB	537	533	∅	6291	605
Elasticsearch + Spark	800	796	822	6291	∅
Spark + SciDB	527	513	822	∅	605

Tabela 6.8: *Comparação do tempo médio de resposta (ms) entre o SmarT e os três sistemas. A coluna “Best” indica o tempo médio de resposta se fosse escolhida sempre a melhor opção.*

O tempo médio de resposta do SmarT é calculado a partir da média dos tempos de resposta do sistema escolhido em cada consulta. Por exemplo, para uma determinada consulta Q, o SmarT fez uma previsão dos tempos de resposta de 434ms para o Elasticsearch, 390ms para o Apache Spark e 1023ms para o SciDB, mas os tempos reais de resposta foram de 455ms, 460ms e 840ms, respectivamente. Neste caso, o SmarT escolheu o Apache Spark para executar a consulta e somou 460ms para o cálculo do tempo médio final. Para a média do melhor motor, somaria o valor de 455ms, pois foi o menor tempo entre os três sistemas. A média é calculada somando o tempo real de resposta em cada consulta dividindo pelo número de consultas. Para cada consulta, é adicionado no tempo final do SmarT, o tempo para a coletar as métricas do *cluster*, calcular a área e

[13] https://ambari.apache.org/
[14] https://www.h2o.ai/

intervalo de tempo e consultar o sistema de indexação para estimar o número de resultados da consultas. A tabela 6.8 apresenta os resultados do SmarT com este *overhead* (em média 3ms) incluso.

Na estimativa do tempo de resposta dos sistemas utilizando o XGBoost, a característica mais importante na tomada de decisão do SmarT foi a variável *count*, como já era esperado pelos resultados apresentados nesta seção. Na estimativa do tempo do Elasticsearch, outras características tiveram impacto, sendo elas por ordem de prioridade: *area*, *time_interval* e *bytes_out*. As variáveis *area* e *time_interval* são ligadas ao número de resultados retornados, já que, geralmente, quanto maior for a restrição espacial e temporal maior será o número de resultados retornados. A característica *bytes_out* teve impacto no Elasticsearch devido ao custo de tráfego dos resultados da consulta, inerente ao Elasticsearch que não possui otimizações neste sentido, e a situação da rede no momento da consulta.

Na estimativa do tempo do Spark, além das características *count*, *area* e *time_interval*, a quantidade de memória livre teve impacto na previsão. Isto acontece porque o Spark faz o processamento das consultas utilizando computação em memória e, por isso, é impactado pela quantidade de memória livre no momento da consulta, tendo, em alguns casos, de usar a área de *swap* do sistema operacional quando não existe memória suficiente.

Na estimativa de tempo do SciDB, além das características *count*, *area* e *time_interval*, as variáveis *load_five*, *load_five* e *cpu_system* tiveram impacto na previsão do tempo de resposta. O SciDB possui um mecanismo interno de indexação e armazenamento otimizado para recuperação de dados do disco, mas com alto custo de processamento. Por isso, a situação do *cluster* em relação ao uso de processamento nas máquinas impacta no tempo de resposta do SciDB.

A Tabela 6.8 também apresenta a comparação do tempo médio de resposta do SmarT em relação ao Spark, Elasticsearch e SciDB. Nesta tabela, também é apresentada uma comparação do tempo de resposta se o SmarT escolher todos os sistemas ou somente dois deles. Como visto na Tabela 6.6, o SciDB teve o menor tempo de resposta seguido pelo Spark e Elasticsearch, respectivamente.

De forma geral, o SmarT teve um tempo médio de resposta próximo ao “Best” com diferença de menos de 3% para todas as combinações de sistemas. Com dois sistemas, a dupla Spark e SciDB teve o melhor desempenho, seguida pela dupla Elasticsearch e SciDB com uma pequena diferença de tempo médio de 10 ms entre as duas. A dupla Elasticsearch e Spark teve o pior desempenho, sendo 33% pior que as outras opções. Como era esperado, pela avaliação da Tabela 6.6, o SciDB é o sistema mais importante dentre os três para reduzir o tempo de resposta. O tempo médio de todas as consultas no SciDB é o mais baixo entre as três porque possui o melhor desempenho quando a con-

sulta retorna um alto número de resultados. Para estes casos, os tempos no Elasticsearch e Spark ficam mais altos, como pode ser visto na Figura 6.13, o que impacta o tempo médio final.

O recomendado é ter os três sistemas para ter o melhor desempenho possível, mas caso não seja possível, olhando para os resultados da Tabela 6.8 e para o gráfico na Figura 6.13, a dupla Elasticsearch e SciDB deve ser utilizada se houver um cenário em que haverá várias consultas com restrições que retornam poucos resultados. Quando não se sabe o cenário das consultas dos clientes, em relação as restrições de filtragem espacial e temporal, ou em cenários com restrições que retornam muitos resultados, o ideal é utilizar a dupla Spark e SciDB.

Na escolha de um sistema, o SciDB provou ter melhor desempenho que o Spark e o Elasticsearch na filtragem e recuperação de dados de STSR. Mas ficou evidenciado na Tabela 6.8 que utilizar o SmarT com os três sistemas é a melhor opção. O SciDB teve um tempo médio de resposta 21,9% maior que o SmarT quando todos os sistemas estão disponíveis. Mesmo em relação as duplas Elasticsearch e SciDB e Spark e SciDB com o SmarT, o SciDB teve tempo médio de resposta 12,6% e 14,8% maior do que as duplas, respectivamente. Assim, o SmarT, com acurácia de 90,1% e redução de quase 22% em relação ao SciDB no tempo de resposta, provou ser uma boa opção quando é possível ter mais de um sistema de Big Data no *cluster*.

6.3.2 Avaliação da Plataforma BigSensing no Processamento Distribuído de Algoritmos de STSR

Esta seção visa avaliar a capacidade da plataforma BigSensing para paralelizar algoritmos de processamento de séries temporais em um *cluster* de computadores. Especificamente, nesta avaliação será distribuído o processamento do algoritmo TWDTW, implementado em C++, com o objetivo de verificar se a plataforma foi capaz de escalar a medida que mais nós são adicionados no *cluster*. Não foi possível distribuir o processamento do P-TWDTW porque não tivemos acesso a um *cluster* de computadores com GPUs. De qualquer forma, a plataforma BigSensing está preparada para executar ambos os algoritmos.

Ambiente Experimental e Bases de Dados

Os testes foram realizados em um *cluster* de computadores com 8 servidores no ambiente de *Cloud Computing* da Amazon EC2[15]. Cada um destes servidores está

[15]https://aws.amazon.com/pt/ec2/. Estes servidores foram criados com base no sistema Nitro, uma coleção de componentes de hardware e software desenvolvidos pela Amazon que oferece altos níveis de

equipado com processadores Intel Xeon® Platinum 8175 de até 3,1 GHz com 16 vCPUs[16], 128 GB de memória RAM e 600 GB de SSD. As máquinas foram conectadas por uma rede Ethernet 10 Gbit/segundo com largura de banda dedicada de 3500 Mbps.

A bateria de testes mediu o tempo para executar o algoritmo TWDTW sobre STSR no *cluster*. Esta análise é importante para verificar se a arquitetura proposta é capaz de paralelizar de forma horizontal os algoritmos de processamento de séries temporais em um *cluster* de computadores de forma eficiente.

Foram geradas treze bases de dados sintéticas de 5 GB até 1,9 TB, cada uma com 23 pontos variando o tamanho como se segue: i) 5 milhões (5M), ii) 10 milhões (10M), iii) 20 milhões (20M), iv) 40 milhões (40M), v) 80 milhões (80M), vi) 160 milhões (160M), vii) 320 milhões (320M), viii) 640 milhões (640M), ix) 1,2 bilhão (1,2B), x) 2,4 bilhões (2,4B), xi) 4,8 bilhões (4,8B), xii) 9,6 bilhões (9,6B) e xiii) 19,2 bilhões (19,2B). Esta base sintética foi gerada selecionando, randomicamente, séries temporais reais do Brasil extraídas do produto MOD13Q1. Apesar das séries temporais serem reais, dizemos que a base de dados é sintética porque ela foi criada de forma sintética, inclusive com a possibilidade de duplicação de séries temporais na base. Como o tempo de resposta da plataforma, na execução dos algoritmos P-TWDTW e TWDTW, é impactado pelo número de observações das séries temporais, a utilização de bases sintéticas ou reais, desde que com tamanhos iguais, não gera diferença no tempo de resposta final. Para os testes de escalabilidade e *speedup* foram utilizadas as bases de dados entre 5 milhões e 1,2 bilhões de STSR variando o número de servidores entre 1, 2, 4 e 8. Para o teste de *sizeup*, foi fixado o número de servidores em 8 variando o tamanho da base de dados entre 5 milhões e 19,2 bilhões de STSR.

Resultados e Discussão

Esta seção apresenta os resultados e discussão dos testes de escalabilidade, *speedup* e *sizeup* na plataforma. O ideal é que o sistema tenha uma escalabilidade horizontal linear, o que nem sempre é possível devido à necessidade de sincronização de tarefas distribuídas no *cluster* e a limitação física de dispositivos como a rede de comunicação.

O gráfico da Figura 6.14 apresenta o resultado da avaliação da escalabilidade da plataforma. Esta avaliação é importante para entender o comportamento da plataforma com o aumento do tamanho da base de dados e do número de máquinas no *cluster*. Como pode ser visto na Figura 6.14, o aumento do tamanho do *cluster* reduziu o tempo de resposta para todas as bases de dados atingindo uma escalabilidade quase

performance, disponibilidade e segurança

[16]Cada vCPU é um *thread* de um núcleo Intel Xeon

linear. Este comportamento escalável foi possível distribuindo o processamento das STSR entre os servidores do *cluster*. Apesar do TWDTW ser um algoritmo centralizado, os vários núcleos de processamento na CPU das máquinas foram utilizados, balanceando o processamento das série temporais entre os processadores, permitindo assim explorar também a escalabilidade vertical em cada servidor.

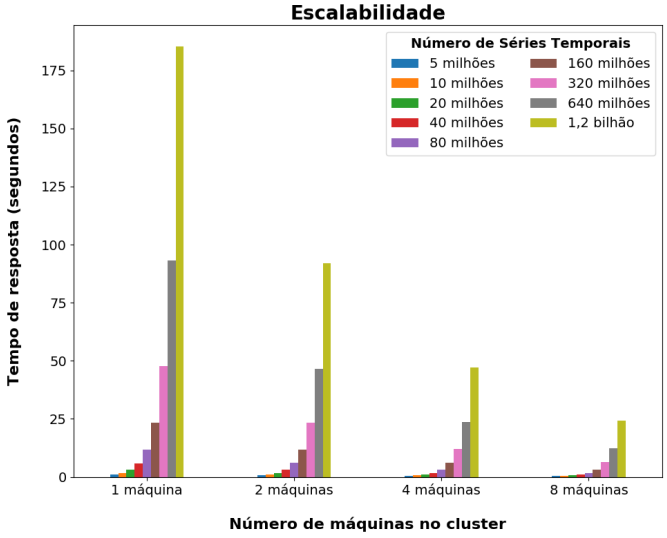


Figura 6.14: *Escalabilidade da plataforma BigSensing medindo o tempo de resposta com o aumento o tamanho do cluster de 1 para 8 servidores e o número de STSR de 5 milhões (5M) até 1,2 bilhões (1,2B).*

O gráfico da Figura 6.15 mostra o resultado da avaliação de *speedup* da plataforma BigSensing. O *speedup* é importante para avaliar o desempenho da plataforma com o aumento de máquinas do *cluster*. De forma geral, a plataforma BigSensing teve bom *speedup* no processamento do algoritmo TWDTW no *cluster*. Como pode ser visto na Figura 6.15, o *speedup* foi melhor com bases de dados maiores, pois nestes cenários é possível explorar todo o poder de processamento das arquiteturas paralelas *multicore*. Com a base de dados com 1,2 bilhões de STSR, o *speedup* alcançou quase uma escala linear porque o custo de processamento neste instante se tornou dominante.

O gráfico da Figura 6.15 mostra o resultado da avaliação de *size-up* da plataforma BigSensing com 8 servidores aumentando o tamanho da base de dados de 5 milhões até 19,2 bilhões. A avaliação de *size-up* é importante por mostrar o comportamento da plataforma em relação ao tempo de resposta com um número fixo de servidores e o aumento da base de dados [55]. O tempo de resposta da plataforma aumentou de forma linear com o aumento da base de dados, o que é o ideal para o cenário de avaliação de *size-up* [55].

A plataforma BigSensing foi capaz de distribuir o processamento do algoritmo TWDTW entre o *cluster* de computadores, tendo escalabilidade, *speedup* e *size-up* quase

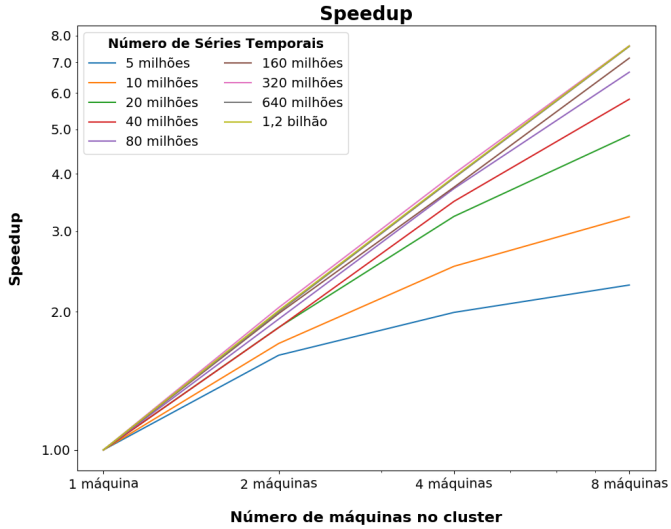


Figura 6.15: *Speedup da plataforma BigSensing aumentando o tamanho do cluster de 1 para 8 servidores e o número de STSR de 5 milhões (5M) até 1,2 bilhões (1,2B).*

linear. Com o aumento do número de máquinas, os recursos computacionais adicionados foram utilizados de forma eficiente, permitindo um comportamento escalável da plataforma. O aumento da base de dados para 19,2 bilhões de STSR com 23 observações cada (i.e. mais de 441 bilhões de observações) foi tratado de forma eficiente pela plataforma com 8 servidores, evidenciando a capacidade da BigSensing de processar algoritmos de STSR sobre um grande volume de dados, explorando os recursos computacionais de um *cluster* de computadores.

6.3.3 Avaliação da Ingestão de dados na plataforma BigSensing

Esta Seção visa avaliar a capacidade da plataforma BigSensing de coletar, transformar e ingerir um grande volume de dados. Nesta avaliação utilizamos o Apache Kafka, versão 2.0.0, como sistema de troca de mensagens e o Spark Streaming, versão 2.3.2, como processador do fluxo de dados. Para indexação e armazenamento dos dados utilizamos os seguintes sistemas com os dados replicados em cada um: i) Elasticsearch versão 7.0.1, ii) SciDB versão 18.3 e iii) Apache Spark versão 2.3.2.

Ambiente Experimental e Bases de Dados

Foram armazenadas nos três sistemas, mais de 7,2 bilhões de observações referentes a 16777216 *pixels* de uma área de 800.824 km^2 que cobre um pedaço do Centro-Oeste e Sudeste do Brasil. Cada *pixel* possui uma série temporal com 435 observações

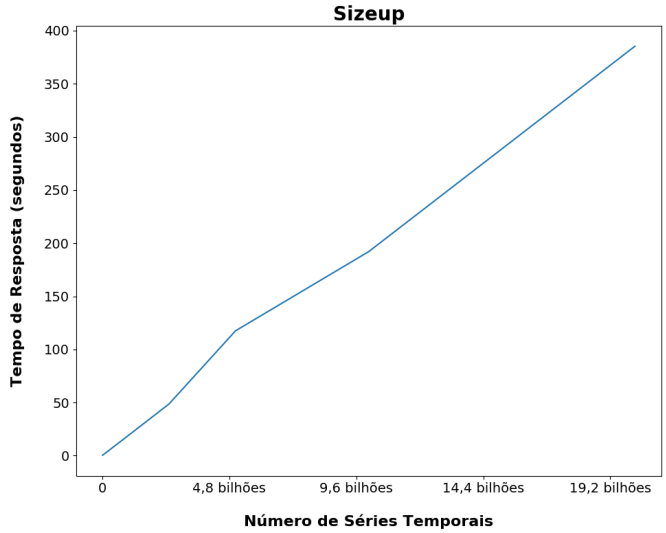


Figura 6.16: *Sizeup da plataforma BigSensing com 8 servidores aumentando o tamanho da base de dados de 5 milhões até 19,2 bilhões.*

entre 2000 e 2019 extraídas do produto MOD13Q1 da NASA[17] (National Aeronautics and Space Administration), com uma periodicidade de 16 dias entre cada coleta de uma observação e resolução espacial de 250 metros.

Cada um dos três sistemas (Elasticsearch, Spark e SciDB) foi implantado e configurado em 8 servidores no ambiente de *Cloud Computing* da Amazon EC2[18]. Cada um destes servidores está equipado com processadores Intel Xeon® Platinum 8175 de até 3,1 GHz com 16 vCPUs, 128 GB de memória RAM e 600 GB de SSD. As máquinas foram conectadas por uma rede Ethernet 10 Gbit/segundo com largura de banda dedicada de 3500 Mbps.

Para receber, transformar e inserir os dados, o módulo DStream foi configurado com 3 servidores no ambiente de *Cloud Computing* da Amazon EC2, sendo 2 máquinas reservadas para o Apache Kafka e uma para o Zookeeper. Cada um destes servidores está equipado com processadores Intel Xeon® Platinum 8175 de até 3,1 GHz com 16 vCPUs[19], 128 GB de memória RAM e 600 GB de SSD. As máquinas foram conectadas por uma rede Ethernet 10 Gbit/segundo com largura de banda dedicada de 3500 Mbps.

Os dados de entrada são enviados como mensagens para o Kafka contendo as observações temporais com os valores das bandas e índices de vegetação do pixel. Cada mensagem que chega na plataforma é convertida para o modelo de dados utilizado internamente e comprimida com o algoritmo lz4 [20], para permitir maior vazão na leitura

[17] Esta base de dados foi disponibilizada pelo LAPIG (Laboratório de Processamento de Imagens e Geoprocessamento)

[18] https://aws.amazon.com/pt/ec2/.

[19] Cada vCPU é um thread de um núcleo Intel Xeon

e escrita das mensagens no Kafka[20].

Para realizar esta avaliação, foi utilizada uma máquina para enviar as requisições de consulta para o *cluster* e coletar as métricas. Esta máquina possui um processador Intel Xeon E5-2676 v3 de 2.4 GHz com 8 núcleos, 32 GB de memória RAM e 1TB de SSD.

Resultados e Discussões

Esta seção apresenta os resultados e discussões a respeito da avaliação da capacidade do plataforma em receber, indexar e armazenar um grande volume de dados de STSR. Foram coletadas métricas da taxa de mensagens que a plataforma é capaz de receber e processar, além de métricas sobre o tempo geral de indexação e armazenamento de dados no Elasticsearch, Spark e SciDB.

A velocidade cada vez maior da geração de novas observações de STSR traz a tona o desafio de construir arquiteturas capazes de lidar com este cenário. O gráfico da Figura 6.17 apresenta o número de mensagens processadas por segundo pela plataforma durante 200 segundos, utilizando o módulo DStream. É possível ver que o sistema foi capaz de processar até 6,2 milhões de mensagens por segundo, mantendo uma média de aproximadamente 5,8 milhões de mensagens processadas por segundo ao longo do tempo. O desvio padrão de aproximadamente 630 mil mensagens processadas por segundo ou cerca de 10% da média mostra que a plataforma foi capaz de manter um taxa de processamento de mensagens durante o tempo.

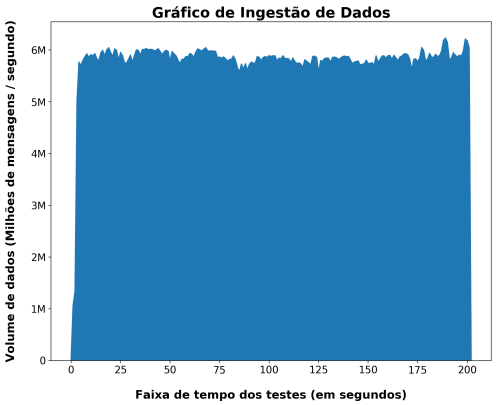


Figura 6.17: *Milhões de mensagens processadas por segundo pela plataforma Big Sensing*

O gráfico da Figura 6.18 apresenta o volume de mensagens em MB por segundo durante 200 segundos que foram processadas pela plataforma. Cada mensagem possui aproximadamente 40 bytes porque foram comprimidas utilizando o algoritmo lz4, antes

[20]http://blog.yaorenjie.com/2015/03/27/Kafka-Compression-Performance-Tests/

de serem enviadas para o Apache Kafka, afim de aumentar a vazão do sistema. O sistema proposto foi capaz de processar até 164 MB por segundo com uma média de 148 MB/segundo. Assim como no gráfico da Figura 6.17, o consumo em MB de mensagens por segundo se manteve constante ao longo do tempo com desvio padrão de 15 MB/segundo.

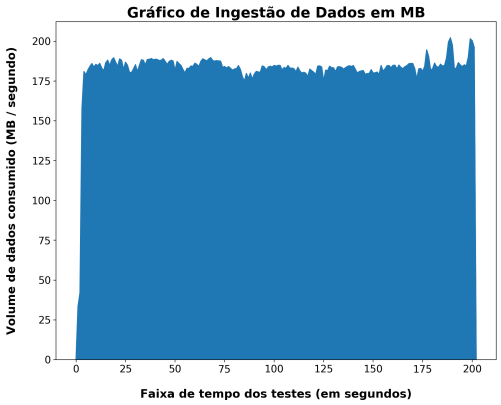


Figura 6.18: *Volume de mensagens (em MB) processadas por segundo pela plataforma BigSensing*

A Tabela 6.9 apresenta o tempo total em segundos para armazenar e indexar as 7,2 bilhões de observações no Elasticsearch, Spark e SciDB. O Apache Spark foi o que levou menos tempo (780 segundos) para armazenar todas as observações, pois além de usar o formato Parquet comprimido e codificado, ele não constrói um sistema de indexação sobre os dados. O Apache Spark teve vazão na inserção quase 39 vezes maior que o Elasticsearch e 43 vezes maior que o SciDB. Além disso, o Spark consumiu apenas 12,7 GB de espaço em disco para indexar e armazenar os dados no formato Parquet, 21 vezes menos que o Elasticsearch e 20 vezes menos que o SciDB.

O Elasticsearch levou 30360 segundos para armazenar e construir os índices espacial e temporal sobre os dados. Esta indexação tem alto custo computacional porque cada pixel tem que ser indexado espacialmente, utilizando a árvore QuadPrefixTree, com os dados de latitude e longitude e, temporalmente, utilizando a data de obtenção de cada observação. A indexação espacial tem menor peso que a temporal porque os 16.777.216 *pixels* são indexados apenas uma vez. Mas, a indexação temporal tem que ser realizada para todas as 7,2 bilhões de observações, o que impactou na vazão do Elasticsearch.

O SciDB armazenou e indexou os dados das séries temporais em 33534 segundos. Ele teve um desempenho pior que o Elasticsearch porque indexa espacialmente e temporalmente cada uma das 7,2 bilhões de observações. No seu sistema de indexação baseado em dimensões, a latitude, longitude e a data de cada observação formam um *array* com três dimensões. Cada uma destas três dimensões devem ser indexadas para permitir depois aplicar os filtros espacial e temporal. O SciDB utilizou 9 GB de espaço em disco

a mais que o Elasticsearch para armazenar os dados. Os dois foram o que mais utilizaram espaço de armazenamento por construírem diversos índices espaciais e temporais e não utilizarem um formato tão eficiente, quanto o Spark com o Parquet, no armazenamento dos dados.

	Apache Spark	Elasticsearch	SciDB
Tempo Total (segundos)	780	30 360	33 534
Tamanho da base de dados em GB no disco	12,7	273	262

Tabela 6.9: *Tabela com o tempo que levou para armazenar os 7,1 bilhões de observações em cada um dos três sistemas.*

6.4 Discussão dos Resultados

Os resultados apresentados neste capítulo mostram que a solução proposta neste trabalho foi capaz de processar, de forma eficiente, as STSR no contexto de Remote Sensing Big Data. Para isto, a solução explorou a escalabilidade vertical das máquinas com as arquiteturas *manycore* e *multicore* e a escalabilidade horizontal de um *cluster* de computadores.

Nossa proposta apresentou o algoritmo P-INDEX para o cálculo paralelo dos índices de sensoriamento remoto, que são importantes na construção das STSR. O P-INDEX teve desempenho até 9 vezes melhor que o algoritmo sequencial na CPU. Diante da alta demanda computacional dos algoritmos de processamento de séries temporais, nosso trabalho propôs um novo algoritmo P-TWDTW, modificando o algoritmo centralizado TWDTW para processar de forma paralela. O ganho de performance com o P-TWDTW em relação ao TWDTW foi expressivo, sendo quase 11 vezes mais eficiente que o TWDTW no cenário de alta resolução espacial e acima de 12 vezes mais eficiente no cenário de alta resolução temporal. Além disso, o P-TWDTW foi 246 mais rápido que a implementação original do TWDTW em R, provando ser uma alternativa eficiente para o processamento de séries temporais em Remote Sensing Big Data, explorando a escalabilidade vertical das máquinas com arquiteturas *manycore*.

Diante do grande volume de dados de sensoriamento remoto a serem armazenados e processados, este trabalho propôs a arquitetura de uma plataforma distribuída, denominada BigSensing, que paraleliza o armazenamento, indexação, filtragem, recuperação e processamento dos algoritmos para análise e classificação de séries temporais em um *cluster* de computadores. Esta plataforma foi capaz de processar um grande volume de dados, além de ser capaz de receber e armazenar uma grande quantidade de dados por segundo.

A plataforma teve alta taxa de 6,2 milhões de mensagens consumidas por segundo, o que permite que o sistema trabalhe com o grande volume de dados no cenário de Remote Sensing Big Data. Foram avaliados três sistemas para o armazenamento e recuperação das observações das séries temporais: Elasticsearch, Spark e SciDB. O Spark teve o melhor desempenho no tempo de armazenamento e indexação dos dados que o Elasticsearch e SciDB, pois além de usar o formato Parquet comprimido e codificado, o Spark não constrói um sistema de indexação sobre os dados.

Dentre os três sistemas avaliados, nenhum deles foi capaz de se sobrepor nos mais variados cenários de consulta com recuperação de resultados com diferentes tamanhos. O SciDB teve um tempo de resposta médio menor que os outros dois porque, para um grande volume de dados retornados, ele teve o melhor desempenho, o que impacta no tempo médio de resposta entre todas as consultas. Mas, no cenário com até 500 mil resultados, o Elasticsearch e o Spark tiveram um desempenho melhor. Por isso, o trabalho propôs um motor inteligente de buscas, o SmarT, que analisa várias variáveis no momento da consulta e tem como objetivo escolher o melhor sistema para executar a consulta. O SmarT teve uma acurácia global de 90,1% na escolha do melhor sistema, além da redução no tempo médio de resposta de mais de 12 vezes em relação ao Elasticsearch, de quase 22% em relação ao SciDB e mais de 60% em relação ao Spark.

Este trabalho também analisou a acurácia do TWDTW e da utilização das medidas de similaridade calculadas pelo TWDTW como características de métodos de aprendizado de máquina. Foi avaliado o método SVM com as características de medida de similaridade do TWDTW que aumentou a acurácia do modelo de 92,2% para 93,8%, sendo melhor na classificação de todos os tipos de uso de solo. Entretanto, a viabilidade do TWDTW na classificação de séries temporais com um grande volume de dados depende de uma solução paralela eficiente do algoritmo TWDTW [77]. Com a solução proposta neste trabalho, o cálculo das medidas de similaridade do TWDTW se torna viável para grande volume de dados utilizando a plataforma BigSensing para distribuir o processamento do algoritmo paralelo P-TWDTW entre as máquinas do *cluster*, explorando assim a escalabilidade vertical com arquiteturas ManyCore e escalabilidade horizontal com os servidores do *cluster*.

CAPÍTULO 7

Conclusões

A plenitude do potencial dos dados de STSR não tem sido explorada completamente devido à falta de soluções eficientes que permitam extrair o valor existente nestes dados. Este trabalho potencializa o avanço das técnicas para melhorar a acurácia e geração de novas aplicações a partir de análises de Big Data no cenário de sensoriamento remoto. Este trabalho apresenta uma proposta para explorar o paralelismo vertical em uma máquina e horizontal em um *cluster* de computadores para processar um grande volume de dados de STSR, de forma eficiente, em Remote Sensing Big Data. A seguir, na Seção 7.1, as contribuições deste trabalho são discutidas bem como as limitações da solução atual, descritas na Seção 7.2, que podem ser investigadas em trabalhos futuros, conforme apresentado na Seção 7.3.

7.1 Contribuições

A principal contribuição deste trabalho foi propor uma solução eficiente para processar um grande volume de dados de STSR em Remote Sensing Big Data. Para isso, exploramos a escalabilidade vertical de uma máquina com o algoritmo P-TWDTW (apresentado no Capítulo 4) utilizando arquiteturas ManyCore. No cenário de Big Data, o P-TWDTW não é capaz de processar um grande volume de dados, pois esbarra nas limitações de memória, disco e barramentos no uso de um único servidor. Por isso, este trabalho apresentou, no Capítulo 5, a proposta da arquitetura de uma plataforma Big Data, denominada BigSensing, para explorar a escalabilidade horizontal de um *cluster* de computadores, permitindo lidar com um grande volume de dados de STSR.

A criação do algoritmo P-TWDTW foi necessária porque a versão centralizada, denominada TWDTW, possui alto custo computacional e complexidade quadrática e não consegue explorar as arquiteturas paralelas de uma máquina. O TWDTW constrói uma matriz de custo entre cada série temporal V e cada padrão U. Na construção desta matriz, o cálculo de cada elemento da matriz depende do resultado do processamento de elementos anteriores e, por isso, não é possível paralelizar o TWDTW de forma trivial. O P-TWDTW modifica a forma de processar a matriz e processa de forma paralela as diagonais da

matriz, mantendo a dependência apenas das diagonais anteriores. Desta forma, as *threads* na GPU podem calcular de forma paralela as células de uma diagonal.

Para otimizar o processamento paralelo com o P-TWDTW propomos também processar vários padrões U e várias séries temporais V ao mesmo tempo. Na GPU, as matrizes devem ser mapeadas como vetor para serem processadas e fica a cargo do programador realizar o controle dos índices das matrizes. O processamento paralelo de várias matrizes aumenta o desafio de controle destes índices. Por isso, este trabalho criou um mecanismo no P-TWDTW para controle dos índices das matrizes de forma a facilitar a programação da paralelização do cálculo das medidas de distância do P-TWDTW.

O P-TWDTW foi capaz de explorar o grande número de núcleos de uma GPU sendo até 12 vezes mais rápido que sua versão centralizada em C++ e 246 mais rápido que a implementação original do TWDTW em R. Nenhuma otimização específica foi feita no código em C++ e esta diferença de tempo entre a implementação em R e C++ se deve ao fato da diferença de desempenho entre as linguagens. Atualmente, os grupos de pesquisa em sensoriamento remoto utilizam a implementação original em R, o que está inviabilizando sua utilização em larga escala. Esta paralelização com o P-TWDTW pode permitir que estes grupos possam adquirir máquinas com arquiteturas paralelas e realmente explorar todo o potencial destas arquiteturas. Por ser 246 vezes mais rápido que a versão centralizada do TWDTW em R, o uso do P-TWDTW consegue reduzir o tempo de resposta de processamentos que antes levavam 1 ano, por exemplo, para serem concluídos com o TWDTW em R e que agora podem ser finalizados em pouco mais de 1 dia com o P-TWDTW na GPU.

A proposta de processamento eficiente com o P-TWDTW permitiu que se explorasse outras características das séries temporais importantes na classificação do tipo de uso e cobertura do solo. Este trabalho estendeu a proposta de [96], que já utiliza as séries temporais das bandas espectrais e índices de vegetação como características para os classificadores, acrescentando meta-características criadas a partir das medidas de similaridade do P-TWDTW. A vantagem destas meta-características é que elas também podem ser utilizadas juntamente com outras características já existentes como entrada para os classificadores. As medidas de similaridade do P-TWDTW foram criadas para serem utilizadas como entrada para o algoritmo KNN, com K=1. Para casos em que as séries temporais apresentam grande variabilidade dos dados dentro de cada classe o algoritmo KNN não é eficiente. Esta proposta utilizando o classificador SVM aumentou de 78% para quase 94% a acurácia de classificação se comparado com o 1NN. Comparamos nossa proposta também com a de [96] e houve um aumento na acurácia de 92% para aproximadamente 94% na classificação do tipo de uso e cobertura do solo. O uso das medidas de similaridade do P-TWDTW permitiu que se revelasse características antes não visíveis dos dados, aumentando a acurácia da classificação. Estas meta-características podem in-

clusive ser utilizadas como entrada para diversos outros métodos de classificação, como Deep Learning.

Os índices de vegetação, utilizados no nosso trabalho, são importantes como entrada para os métodos de classificação e são calculados a partir das bandas espectrais. Este cálculo não possui alta demanda computacional, mas se torna um desafio com um grande volume de dados. Apesar de ser trivialmente paralelizável na GPU, o desafio consiste em utilizar de forma eficiente a memória da GPU reduzindo a latência de transferência entre a memória da CPU e da GPU. Se esta latência não for pequena, o tempo de transferência pode se tornar maior que o tempo de processamento e, por isso, não explorar o grande número de núcleos da GPU. Este trabalho propôs o algoritmo P-INDEX que é capaz de paralelizar o cálculo destes índices, reduzindo a latência de transferência através de um mecanismo de controle de fluxo entre a memória da CPU e GPU, buscando manter o uso dos núcleos da GPU o mais alto possível. A utilização da GPU com o P-INDEX obteve melhor desempenho que a CPU com 8 *threads* tendo até 9 vezes de *speedup* em relação ao algoritmo com 1 *thread* na CPU.

Para lidar com um grande volume de dados, os algoritmos P-TWDTW e P-INDEX enfrentam as limitações inerentes do processamento em uma única máquina. Neste cenário, a utilização dos recursos computacionais de várias máquinas em um *cluster* de computadores se torna necessária. Este trabalho propõe a plataforma Big Data, denominada BigSensing, que provou ser capaz de processar até 19,2 bilhões de STSR, tendo escalabilidade horizontal quase linear com o aumento do número de máquinas e do tamanho da base de dados.

A plataforma BigSensing possui um motor inteligente de buscas, denominado SmarT, capaz de filtrar e recuperar os dados das STSR de forma eficiente. Existem diversos sistemas de Big Data prontos e preparados para armazenar, indexar, filtrar e recuperar os dados das STSR. Este trabalho avaliou três destes sistemas para compor o SmarT: Apache Spark, Elasticsearch e SciDB. Esta avaliação mostrou que nenhum destes sistemas se sobrepôs aos outros, em relação ao tempo de resposta, na filtragem e recuperação dos dados em todos os cenários de consultas. O SmarT tem como objetivo escolher o melhor sistema, entre vários existentes, para filtrar e recuperar os dados da consulta no menor tempo possível. O SmarT utiliza dados de monitoramento do cluster (rede, memória, disco e CPU), as restrições da consulta e uma estimativa do número de resultados da consulta para definir qual sistema escolher. O SmarT teve uma acurácia de mais de 90% na escolha do melhor sistema entre o Apache Spark, Elasticsearch e SciDB. Com isso houve uma redução de 22% no tempo de resposta utilizando o SmarT do que se fosse utilizado apenas um dos três sistemas.

A proposta do SmarT é um passo importante na evolução das pesquisas na área de processamento de STSR, pois permite que a plataforma possa se adaptar ao ambiente

de sistemas disponíveis de forma automática, em tempo real. Seria praticamente inviável realizar essa escolha de forma manual, em cada consulta, pelo pesquisador da área de sensoriamento remoto, devido ao número de combinações possíveis das variáveis utilizadas pelo SmarT. O uso da plataforma BigSensing, com este motor inteligente, permite que os pesquisadores se concentrem na construção de algoritmos para resolver problemas na área de sensoriamento remoto, sem se preocupar com o desafio do processamento eficiente do grande volume de dados.

7.2 Principais Limitações

Na investigação deste trabalho desbravamos os desafios e possíveis soluções para processamento de STSR em Remote Sensing Big Data. Entretanto, as soluções propostas sofrem de algumas limitações quando consideramos diversos outros cenários. As principais limitações são discutidas nesta Seção.

O P-TWDTW sempre aloca *d threads* para o processamento da diagonal, onde *d* é igual ao tamanho da diagonal principal. Entretanto, as outras diagonais da matriz possuem tamanho menor que *d*, o que acaba desperdiçando recursos da GPU e deixando *d - n threads* ociosas, onde *n* é o tamanho da diagonal que está sendo processada. Estas *d - n threads* ociosas divergem na execução do código do restante das *threads*, o que reduz a eficiência do P-TWDTW no uso dos núcleos da GPU.

Tanto o algoritmo P-INDEX quanto o P-TWDTW são capazes de explorar os núcleos de uma GPU. Para ambientes multi-GPU, entretanto, ambos os algoritmos não foram projetados para particionar o trabalho entre as várias GPUs disponíveis. Atualmente para explorar este ambiente multi-GPU, este particionamento deve ser feito de forma manual, o que não é produtivo para grande volumes de dados. Além disso, algoritmos projetados para processamento em multi-GPU devem possuir otimizações que explorem de forma eficiente a comunicação e coordenação do processamento entre as várias GPUs.

Os algoritmos P-INDEX e P-TWDTW possuem sistemas internos eficientes de controle de fluxo de dados do disco para a memória da GPU. Entretanto, este fluxo de dados sempre passa pela memória da CPU, o que gera um passo a mais desnecessário na transferência dos dados.

Apesar de ter um gerenciador do controle de fluxo de dados de várias STSR entre o disco e a memória da GPU, quando uma única série temporal tem tamanho maior que a memória da CPU ou da GPU, o P-TWDTW não trata este cenário e existe a possibilidade de estouro do uso destes espaços de memória. O P-INDEX não sofre deste problema porque cada cálculo é realizado para um único pixel e os valores utilizados para o cálculo do índice de sensoriamento remoto são limitados pelo número de bandas utilizadas.

A plataforma BigSensing possui uma arquitetura eficiente para gerenciamento de grande volume de dados de STSR. O resultado do processamento da plataforma pode ser gravado na camada de armazenamento ou enviado de volta para o usuário final, sendo determinado pelo usuário na requisição. A opção padrão é gravar o resultado no sistema de armazenamento e o usuário, posteriormente, poder recuperar os resultados iterando sobre eles ou realizar outras operações como gerar estatísticas e mapas com o resultado da classificação, por exemplo. Caso o usuário escolha retornar todo o resultado, a plataforma não é capaz de identificar em tempo de execução o tamanho do resultado da consulta, o que pode acarretar no estouro da memória disponível.

As consultas enviadas pelos usuários da plataforma são limitadas por três parâmetros: *bounding_box*, *start_time* e *end_time*. Esta limitação foi imposta neste trabalho para facilitar a tradução dos parâmetros de entrada para as linguagens específicas de cada um dos sistemas internos de filtragem e recuperação de dados (i.e Elasticsearch, Spark e SciDB). Esta limitação impede, por exemplo, de realizar a filtragem dos dados sobre as observações das séries temporais e sobre metadados das STSR.

O treinamento do SmarT é realizado em intervalos de tempo executando um conjunto de consultas sobre os sistemas existentes e coletando as métricas para serem utilizadas como variáveis dos métodos de aprendizado de máquina. Durante a coleta das métricas, não é recomendável que se execute outras consultas no *cluster* para não impactar de forma negativa no tempo de resposta das consultas dos usuários. A coleta de métricas para o SmarT deve ser realizada em momentos do dia em que não tenha historicamente muito uso do *cluster*. Neste caso, o SmarT fica limitado a ser treinado sempre com o mesmo conjunto de consultas e em períodos de tempo pré-determinados. Apesar do conjunto de consultas poder ser modificado, esta alteração tem que ser feita de forma manual pelo desenvolvedor da plataforma. Este conjunto de consultas pode não representar uma amostra real das consultas que estão sendo realizadas pelos usuários em um ambiente de produção e, por isso, o SmarT pode não atingir o seu melhor desempenho na escolha do melhor sistema para executar determinada consulta. Outra limitação da plataforma com uso de vários sistemas é que os dados devem ser replicados entre estes sistemas, pois não existe um barramento eficiente para armazenamento e recuperação dos dados.

Na próxima seção, sugerimos abordagens para lidar com estas limitações da nossa proposta. Além disso, discutimos e sugerimos algumas melhorias identificadas que podem ser realizadas nesta proposta.

7.3 Trabalhos futuros

Nesta seção indicamos trabalhos futuros promissores a serem realizados no desenvolvimento de soluções eficientes para o processamento de STSR em Remote Sensing Big Data. Também identificamos diversas melhorias a serem realizadas na nossa proposta para lidar com as limitações levantadas na Seção 7.2.

O algoritmo P-TWDTW não consegue atingir o máximo de ocupação dos núcleos da GPU devido a ociosidade de algumas *threads* no processamento da diagonal da matriz de forma paralela. Para mitigar esse problema, propomos o processamento de faixas de diagonais, ao invés uma única diagonal, pois desta forma uma *thread* que estiver ociosa no processamento de uma determinada diagonal pode processar células de outras diagonais.

Para explorar o potencial de ambientes multi-gpus, os algoritmos P-TWDTW e P-INDEX podem particionar o conjunto de dados de entrada, de forma automática, entre as várias GPUs. No processamento de uma única série temporal com um grande número de observações, a construção da matriz de custo pode ser paralelizada utilizando as várias GPUs e permitindo com que elas se comuniquem utilizando a tecnologia GPUDirect da NVIDIA. Esta tecnologia possui otimizações internas que permitem a comunicação eficiente entre as GPUs. Além disso, esta tecnologia irá permitir ler os dados diretamente do disco para a memória da GPU, eliminando a etapa de cópia para a memória da CPU, ocasionando uma redução do tempo de transferência dos dados. O uso paralelo das várias GPUs deve aumentar a quantidade de memória disponível e permitir processar séries temporais maiores. Se mesmo assim, a série temporal for maior que a memória disponível, poderá ser criado um fluxo de cópia apenas das diagonais que cabem na memória para serem processadas.

As meta-características geradas neste trabalho a partir das medidas de similaridade calculadas no P-TWDTW foram utilizadas como entrada para o algoritmo de aprendizado de máquina SVM. Apesar do trabalho de [96] ter avaliado diversos outros classificadores, como Redes Neurais e o Florestas Aleatórias, existem diversos outros métodos não avaliados por [96] e que podem se beneficiar destas meta-características, como Deep Learning. Por isso, em trabalhos futuros, é interessante avaliar o impacto da acurácia da classificação do tipo de uso e cobertura do solo utilizando outros métodos além do SVM. Além disso, é importante avaliar nossa proposta em outras regiões geográficas e dados de outros satélites que podem gerar desafios ainda não revelados nos cenários de avaliação deste trabalho.

A plataforma BigSensing pode ser melhorada para lidar com um grande volume de dados retornados para o usuário que podem estourar a memória do nó coordenador da requisição. Para isto, pode ser criado um módulo para tratar o retorno da resposta,

identificando a quantidade de memória disponível e utilizando o disco para armazenar o conjunto de dados de resposta que não couber na memória. O usuário recebe uma mensagem de resposta indicando que o conjunto de dados de resposta extrapolou o limite da plataforma, mas que a resposta pode ser recuperada através de um iterador utilizando um id como identificador do conjunto de respostas. Além disso, pode ser criada uma linguagem de consulta na API da plataforma para permitir que o usuário tenha mais opções de filtros sobre outros campos, como metadados, além dos campos espaciais e temporais.

Para gerar um conjunto de treinamento que represente melhor os cenários de consultas feitas pelos usuários, podemos modificar o SmarT para que ele possa gerar o conjunto de treinamento a partir de consultas reais dos usuários e coletar métricas ao longo do dia, utilizando este conjunto de treinamento, em períodos de ociosidade do *cluster* sem a necessidade de intervenção manual do desenvolvedor da plataforma. Além disso, métricas específicas podem ser coletadas de cada sistema que possam impactar na escolha do melhor no momento da consulta. Um exemplo são métricas do tamanho da fila de espera por execução em cada sistema. Pode acontecer de chegar consultas que suas restrições e as métricas de monitoramento do *cluster* indiquem a escolha de um único sistema. Apesar deste sistema poder ser realmente o melhor no tratamento destas consultas, sua fila de requisições pode começar a ficar cheia e o tempo de resposta aumentar. Por isso, neste cenário pode ser melhor escolher outro sistema para tratar a requisição.

Em trabalhos futuros, é necessário realizar experimentos de integração da plataforma BigSensing com os algoritmos P-TWDTW e P-INDEX em um *cluster* de computadores com cada máquina possuindo uma GPU. Nesta tese não foi possível executar tais avaliações devido a indisponibilidade de acesso a um *cluster* com GPUs.

Seria interessante criar um barramento de dados eficiente que seja capaz de integrar com os diversos sistemas de Big Data. Um desafio deste barramento é ter o mínimo de impacto possível na latência com a substituição da camada nativa de armazenamento de cada sistema. Outro desafio é construir um barramento de dados que facilite a integração com outros sistemas que venham a ser adicionados na plataforma.

Referências Bibliográficas

[1] **Random forest in remote sensing: A review of applications and future directions**. *ISPRS Journal of Photogrammetry and Remote Sensing*, 114:24 – 31, 2016.

[2] ALMEER, M. H. **Cloud hadoop map reduce for remote sensing image analysis**. *Journal of Emerging Trends in Computing and Information Sciences*, 3(4):637–644, 2012.

[3] ALONSO ISLA, Á. **Hdfs file formats: Study and performance comparison**. 2018.

[4] ASSIS, L. F. F. G.; RIBEIRO, G.; GOMES, K. R. F.; VINHAS, L.; LLAPA, E.; SANCHEZ, A.; MAUS, V. W.; CÂMARA, G. **Big data streaming for remote sensing time series analytics using mapreduce**. In: *Anais...* Brazilian Symposium on GeoInformatics, 17. (GEOINFO), 2016.

[5] BAUMANN, P. **The ogc web coverage processing service (wcps) standard**. *Geoinformatica*, 14(4):447–479, 2010.

[6] BAUMANN, P.; FURTADO, P.; RITSCH, R.; WIDMANN, N. **The rasdaman approach to multidimensional database management**. In: *Symposium on Applied Computing: Proceedings of the 1997 ACM symposium on Applied computing*, volume 1997, p. 166–173, 1997.

[7] BAUMANN, P.; MAZZETTI, P.; UNGAR, J.; BARBERA, R.; BARBONI, D.; BECCATI, A.; BIGAGLI, L.; BOLDRINI, E.; BRUNO, R.; CALANDUCCI, A.; OTHERS. **Big data analytics for earth sciences: the earthserver approach**. *International Journal of Digital Earth*, 9(1):3–29, 2016.

[8] BÉGUÉ, A.; ARVOR, D.; BELLON, B.; BETBEDER, J.; DE ABELLEYRA, D.; PD FERRAZ, R.; LEBOURGEOIS, V.; LELONG, C.; SIMÕES, M.; R VERÓN, S. **Remote sensing and cropping practices: A review**. *Remote Sensing*, 10(1):99, 2018.

[9] BERNDT, D. J.; CLIFFORD, J. **Using dynamic time warping to find patterns in time series.** *KDD workshop*, 10(16):359–370, 1994.

[10] BROWN, P. G. **Overview of scidb: large scale array storage, processing and analysis**. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, p. 963–968. ACM, 2010.

[11] BUCK, I.; FOLEY, T.; HORN, D.; SUGERMAN, J.; FATAHALIAN, K.; HOUSTON, M.; HANRAHAN, P. **Brook for gpus: stream computing on graphics hardware**. *ACM Transactions on Graphics (TOG)*, 23(3):777–786, 2004.

[12] CAMARA, G.; ASSIS, L. F.; RIBEIRO, G.; FERREIRA, K. R.; LLAPA, E.; VINHAS, L. **Big earth observation data analytics: matching requirements to system architectures**. In: *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, p. 1–6. ACM, 2016.

[13] CAMARA, G.; EGENHOFER, M. J.; FERREIRA, K.; ANDRADE, P.; QUEIROZ, G.; SANCHEZ, A.; JONES, J.; VINHAS, L. **Fields as a generic data type for big spatial data**. In: *International Conference on Geographic Information Science*, p. 159–172. Springer, 2014.

[14] CASANOVA, M. A.; CÂMARA, G.; DAVIS, C.; VINHAS, L.; QUEIROZ, G. R. **Banco de dados geográficos**. MundoGEO Curitiba, 2005.

[15] CASU, F.; ELEFANTE, S.; IMPERATORE, P.; ZINNO, I.; MANUNTA, M.; DE LUCA, C.; LANARI, R. **Sbas-dinsar parallel processing for deformation time-series computation**. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(8):3285–3296, 2014.

[16] CHATTERJEE, S.; HADI, A. S. **Regression analysis by example**. John Wiley & Sons, 2015.

[17] CHEN, C. P.; ZHANG, C.-Y. **Data-intensive applications, challenges, techniques and technologies: A survey on big data**. *Information Sciences*, 275:314–347, 2014.

[18] CHEN, T.; GUESTRIN, C. **Xgboost: A scalable tree boosting system**. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, p. 785–794. ACM, 2016.

[19] CHI, M.; PLAZA, A.; BENEDIKTSSON, J. A.; SUN, Z.; SHEN, J.; ZHU, Y. **Big data for remote sensing: Challenges and opportunities**. *Proceedings of the IEEE*, 104(11):2207–2219, 2016.

[20] COLLET, Y.; OTHERS. **Lz4: Extremely fast compression algorithm**. *code. google. com*, 2013.

[21] COMBER, A.; WULDER, M. **Considering spatiotemporal processes in big data analysis: Insights from remote sensing of land cover and land use**. *Transactions in GIS*, 2019.

[22] CORTES, C.; VAPNIK, V. **Support-vector networks**. *Machine learning*, 20(3):273–297, 1995.

[23] COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T.; BLAIR, G. **Sistemas Distribuídos-: Conceitos e Projeto**. Bookman Editora, 2013.

[24] CUEVAS-TELLO, J.; GONZÁLEZ-GRIMALDO, R.; RODRÍGUEZ-GONZÁLEZ, O.; PÉREZ-GONZÁLEZ, H.; VITAL-OCHOA, O. **Parallel approach for time series analysis with general regression neural networks**. *Journal of applied research and technology*, 10(2):162–179, 2012.

[25] DADI, M. M. **Assessing the transferability of random forest and time-weighted dynamic time warping for agriculture mapping**. 2019.

[26] DE CAMARGO, R. Y. **Simulação de neurônios biologicamente realistas em gpus**. In: *2009 Symposium on High Performance Computing Systems (WSCAD)*, 2009.

[27] DE OLIVEIRA, S. S. T.; DE CASTRO CARDOSO, M.; DOS SANTOS, W.; COSTA, P.; DO SACRAMENTO RODRIGUES, V. J.; MARTINS, W. S. **Distsensing: A new platform for time series processing in a distributed computing environment**. *Revista Brasileira de Cartografia*, 69(5).

[28] DE OLIVEIRA, S. S. T.; DO SACRAMENTO RODRIGUES, V. J.; FERREIRA, L. G.; MARTINS, W. S. **P-twdtw: Parallel processing of time series remote sensing images using manycore architectures**. In: *2018 Symposium on High Performance Computing Systems (WSCAD)*, p. 252–258. IEEE, 2018.

[29] DE OLIVEIRA, S. S.; DE S FILHO, J. F.; VAGNER, J.; CARDOSO, M. D. C.; CARVALHO, S. T. **A new technique for verifying the consistency of distributed r-trees**. *Journal of Information and Data Management*, 6(1):59, 2015.

[30] DE OLIVEIRA, S. S.; FILHO, J. F. D. S.; VAGNER, J.; CARDOSO, M. D. C.; DE CARVALHO, S. T.; PALMEIRAS, A.; QUADRA, D.; SAMAMBAIA, C. **Rdebug: A new debugging technique for distributed r-trees**. In: *XV Brazilian Symposium on Geoinformatics (GEOINFO)*, 2014.

[31] DEMIR, B.; BOVOLO, F.; BRUZZONE, L. **Detection of land-cover transitions in multitemporal remote sensing images with active-learning-based compound classification**. *Geoscience and Remote Sensing, IEEE Transactions on*, 50(5):1930–1941, 2012.

[32] DIDAN, K. **Mod13q1 modis/terra vegetation indices 16-day l3 global 250m sin grid v006**. *NASA EOSDIS Land Processes DAAC*, 2015.

[33] DOAN, K.; OLOSO, A. O.; KUO, K.-S.; CLUNE, T. L.; YU, H.; NELSON, B.; ZHANG, J. **Evaluating the impact of data placement to spark and scidb with an earth science use case**. In: *2016 IEEE International Conference on Big Data (Big Data)*, p. 341–346. IEEE, 2016.

[34] DUNNING, T.; FRIEDMAN, E. **Time series databases**. *New Ways to Store and Access data*, 2015.

[35] FAN, J.; HAN, F.; LIU, H. **Challenges of big data analysis**. *National science review*, 1(2):293–314, 2014.

[36] FINKEL, R. A.; BENTLEY, J. L. **Quad trees a data structure for retrieval on composite keys**. *Acta informatica*, 4(1):1–9, 1974.

[37] FOERSTER, T.; STOTER, J. **Establishing an ogc web processing service for generalization processes**. In: *ICA workshop on Generalization and Multiple Representation*, 2006.

[38] FRIEDL, M. A.; MCIVER, D. K.; HODGES, J. C.; ZHANG, X. Y.; MUCHONEY, D.; STRAHLER, A. H.; WOODCOCK, C. E.; GOPAL, S.; SCHNEIDER, A.; COOPER, A.; OTHERS. **Global land cover mapping from modis: algorithms and early results**. *Remote Sensing of Environment*, 83(1-2):287–302, 2002.

[39] FU, G.; ZHAO, H.; LI, C.; SHI, L. **Segmentation for high-resolution optical remote sensing imagery using improved quadtree and region adjacency graph technique**. *Remote sensing*, 5(7):3259–3279, 2013.

[40] GABR, M.; FATEHY, L. **Time series classification**. *Journal of Statistics Applications & Probability*, 2(2):123–133, 2013.

[41] GHOSH, S. **Distributed systems: an algorithmic approach**. CRC press, 2014.

[42] GÓMEZ, C.; WHITE, J. C.; WULDER, M. A. **Optical remotely sensed time series data for land cover classification: A review**. *ISPRS Journal of Photogrammetry and Remote Sensing*, 116:55–72, 2016.

[43] GORELICK, N.; HANCHER, M.; DIXON, M.; ILYUSHCHENKO, S.; THAU, D.; MOORE, R. **Google earth engine: Planetary-scale geospatial analysis for everyone**. *Remote Sensing of Environment*, 202:18–27, 2017.

[44] GUO, H.; LIU, Z.; JIANG, H.; WANG, C.; LIU, J.; LIANG, D. **Big earth data: a new challenge and opportunity for digital earth's development**. *International Journal of Digital Earth*, 10(1):1–12, 2017.

[45] GUO, H.; WANG, L.; CHEN, F.; LIANG, D. **Scientific big data and digital earth**. *Chinese science bulletin*, 59(35):5066–5073, 2014.

[46] GUO, H.; WANG, L.; LIANG, D. **Big earth data from space: a new engine for earth science**. *Science Bulletin*, 61(7):505–513, 2016.

[47] GUO, W.; GONG, J.; JIANG, W.; LIU, Y.; SHE, B. **Openrs-cloud: A remote sensing image processing platform based on cloud computing environment**. *Science China Technological Sciences*, 53(1):221–230, 2010.

[48] HADOOP, A. **Hadoop**, 2009.

[49] HANSEN, M.; DEFRIES, R.; TOWNSHEND, J.; CARROLL, M.; DIMICELI, C.; SOHLBERG, R. **Global percent tree cover at a spatial resolution of 500 meters: First results of the modis vegetation continuous fields algorithm**. *Earth Interactions*, 7(10):1–15, 2003.

[50] HAWKINS, D. M. **The problem of overfitting**. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[51] HONG, Z.; YU, C.; XIA, R.; XIAO, J.; WANG, J.; SUN, J.; CUI, C. **Aquadex: A highly efficient indexing and retrieving method for astronomical big data of time series images**. In: *Algorithms and Architectures for Parallel Processing*, p. 92–105. Springer, 2015.

[52] HUANG, W.; MENG, L.; ZHANG, D.; ZHANG, W. **In-memory parallel processing of massive remotely sensed data using an apache spark on hadoop yarn model**. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(1):3–19, 2017.

[53] IMPERATORE, P.; PEPE, A.; LANARI, R. **Spaceborne synthetic aperture radar data focusing on multicore-based architectures**. *IEEE Transactions on Geoscience and Remote Sensing*, 54(8):4712–4731, 2016.

[54] JAMALI, S.; JÖNSSON, P.; EKLUNDH, L.; ARDÖ, J.; SEAQUIST, J. **Detecting changes in vegetation trends using time series segmentation**. *Remote Sensing of Environment*, 156:182–195, 2015.

[55] JAPKOWICZ, N.; STEFANOWSKI, J. **Big Data Analysis: New Algorithms for a New Society**. Springer, 2016.

[56] KASTENS, J. H.; BROWN, J. C.; COUTINHO, A. C.; BISHOP, C. R.; ESQUERDO, J. C. D. M. **Soy moratorium impacts on soybean and deforestation dynamics in mato grosso, brazil**. *PLOS ONE*, 12(4):1–21, 04 2017.

[57] KENNEDY, R. E.; YANG, Z.; COHEN, W. B. **Detecting trends in forest disturbance and recovery using yearly landsat time series: 1. landtrendr-temporal segmentation algorithms**. *Remote Sensing of Environment*, 114(12):2897–2910, 2010.

[58] KIRK, D. B.; WEN-MEI, W. H. **Programming massively parallel processors: a hands-on approach**. Morgan Kaufmann, 2016.

[59] KOHAVI, R.; OTHERS. **A study of cross-validation and bootstrap for accuracy estimation and model selection**. In: *Ijcai*, volume 14, p. 1137–1145. Montreal, Canada, 1995.

[60] LANEY, D. **3d data management: Controlling data volume, velocity and variety**. *META group research note*, 6(70):1, 2001.

[61] LEWIS, A.; OLIVER, S.; LYMBURNER, L.; EVANS, B.; WYBORN, L.; MUELLER, N.; RAEVKSI, G.; HOOKE, J.; WOODCOCK, R.; SIXSMITH, J.; OTHERS. **The australian geoscience data cube-foundations and lessons learned**. *Remote Sensing of Environment*, 202:276–292, 2017.

[62] LEWIS, D. D. **Naive (bayes) at forty: The independence assumption in information retrieval**. In: *European conference on machine learning*, p. 4–15. Springer, 1998.

[63] LIN, F.-C.; CHUNG, L.-K.; WANG, C.-J.; KU, W.-Y.; CHOU, T.-Y. **Storage and processing of massive remote sensing images using a novel cloud computing platform**. *GIScience & Remote Sensing*, 50(3):322–336, 2013.

[64] LIU, P.; DI, L.; DU, Q.; WANG, L. **Remote sensing big data: theory, methods and applications**, 2018.

[65] LIU, X.; IFTIKHAR, N.; XIE, X. **Survey of real-time processing systems for big data**. In: *Proceedings of the 18th International Database Engineering & Applications Symposium*, p. 356–361. ACM, 2014.

[66] LIU, Y.; CHEN, B.; YU, H.; ZHAO, Y.; HUANG, Z.; FANG, Y. **Applying gpu and posix thread technologies in massive remote sensing image data processing**. In: *Geoinformatics, 2011 19th International Conference on*, p. 1–6. IEEE, 2011.

[67] LÓPEZ-FANDIÑO, J.; HERAS, D. B.; ARGÜELLO, F.; DALLA MURA, M. **Gpu framework for change detection in multitemporal hyperspectral images**. *International Journal of Parallel Programming*, 47(2):272–292, 2019.

[68] LU, M.; PEBESMA, E.; SANCHEZ, A.; VERBESSELT, J. **Spatio-temporal change detection from multidimensional arrays: Detecting deforestation from modis time series**. *ISPRS Journal of Photogrammetry and Remote Sensing*, 117:227–236, 2016.

[69] LÜ, X.; CHENG, C.; GONG, J.; GUAN, L. **Review of data storage and management technologies for massive remote sensing data**. *Science China Technological Sciences*, 54(12):3220–3232, 2011.

[70] LV, Z.; HU, Y.; ZHONG, H.; WU, J.; LI, B.; ZHAO, H. **Parallel k-means clustering of remote sensing images based on mapreduce**. In: *Proceedings of the 2010 International Conference on Web Information Systems and Mining*, WISM'10, p. 162–170, Berlin, Heidelberg, 2010. Springer-Verlag.

[71] MA, L.; LIU, Y.; ZHANG, X.; YE, Y.; YIN, G.; JOHNSON, B. A. **Deep learning in remote sensing applications: A meta-analysis and review**. *ISPRS Journal of Photogrammetry and Remote Sensing*, 152:166 – 177, 2019.

[72] MA, Y.; WU, H.; WANG, L.; HUANG, B.; RANJAN, R.; ZOMAYA, A.; JIE, W. **Remote sensing big data computing: Challenges and opportunities**. *Future Generation Computer Systems*, 51:47–60, 2015.

[73] MACEDONIA, M. **The gpu enters computing's mainstream**. *Computer*, 36(10):106–108, 2003.

[74] MARK, W. R.; GLANVILLE, R. S.; AKELEY, K.; KILGARD, M. J. **Cg: A system for programming graphics hardware in a c-like language**. *ACM Transactions on Graphics (TOG)*, 22(3):896–907, 2003.

[75] MATTSON, T. G.; SANDERS, B.; MASSINGILL, B. **Patterns for parallel programming**. Pearson Education, 2004.

[76] MAUS, V.; CÂMARA, G.; CARTAXO, R.; SANCHEZ, A.; RAMOS, F. M.; DE QUEIROZ, G. R. **A time-weighted dynamic time warping method for land-use and land-cover mapping**. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(8):3729–3739, 2016.

[77] MAUS, V. **Land Use and Land Cover Monitoring Using Remote Sensing Image Time Series**. PhD thesis, PhD thesis, Instituto Nacional de Pesquisas Espaciais, Sao José dos Campos, 2016.

[78] MAYER-SCHÖNBERGER, V.; CUKIER, K. **Big data: A revolution that will transform how we live, work, and think**. Houghton Mifflin Harcourt, 2013.

[79] MOUNTRAKIS, G.; IM, J.; OGOLE, C. **Support vector machines in remote sensing: A review**. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(3):247–259, 2011.

[80] MÜLLER, H.; RUFIN, P.; GRIFFITHS, P.; SIQUEIRA, A. J. B.; HOSTERT, P. **Mining dense landsat time series for separating cropland and pasture in a heterogeneous brazilian savanna landscape**. *Remote Sensing of Environment*, 156:490–499, 2015.

[81] MUNSHI, A. **The opencl specification**. In: *2009 IEEE Hot Chips 21 Symposium (HCS)*, p. 1–314. IEEE, 2009.

[82] MURPHY, K. J. **Nasa's earth science data systems overview and evolution**. In: *AGU Fall Meeting Abstracts*, 2018.

[83] NEAGOE, V.; CIUREA, A.; BRUZZONE, L.; BOVOLO, F. **A novel neural approach for unsupervised change detection using som clustering for pseudo-training set selection followed by csom classifier**. In: *Geoscience and Remote Sensing Symposium (IGARSS), 2014 IEEE International*, p. 1437–1440. IEEE, 2014.

[84] NEAGOE, V.-E.; STOICA, R.-M.; CIUREA, A.-I. **A modular neural network model for change detection in earth observation imagery**. In: *Geoscience and Remote Sensing Symposium (IGARSS), 2013 IEEE International*, p. 3321–3324. IEEE, 2013.

[85] NEVES, A. K.; BENDINI, H. N.; KÖRTING, T. S.; FONSECA, L. M. G. **Combining time series features and data mining to detect land cover patterns: a case study in northern mato grosso state, brazil.** In: *GeoInfo*, p. 174–185, 2015.

[86] NIELSEN, D. **Tree boosting with xgboost-why does xgboost win"every"machine learning competition?** Master's thesis, NTNU, 2016.

[87] OLIVEIRA, S.; ALEIXO, E.; MARTINS, W.; FERREIRA, M. **Pndvi: Um novo algoritmo para processamento paralelo do índice de vegetação ndvi**. In: *IV Escola Regional de Informática de GoiÃ¡s*, p. 235–249. ERI, 2016.

[88] OLIVEIRA, S.; CARDOSO, M.; SANTOS, W.; COSTA, P.; SACRAMENTO, V.; MARTINS, W. **A new platform for time-series analysis of remote sensing images in a distributed computing environment**. In: *XVII Brazilian Symposium on Geoinformatics (GEOINFO)*, p. 128–139, 2016.

[89] OLIVEIRA, S. S.; PASCOAL, L. M. L.; FERREIRA, L.; DE CASTRO CARDOSO, M.; BUENO, E. F.; VAGNER, J.; MARTINS, W. S. **Sp-twdtw: A new parallel algorithm for spatio-temporal analysis of remote sensing images.** In: *GEOINFO*, p. 46–57, 2018.

[90] OLIVEIRA, S. S.; SANTOS, C.; FERREIRA, L.; .; VAGNER, J.; MARTINS, W. S. **Dstream: Plataforma de coleta, processamento e armazenamento de streams de dados de sensoriamento remoto**. In: *VI Escola Regional de InformÃ¡tica de GoiÃ¡s*, p. 357–361, 2018.

[91] OPITZ, D.; MACLIN, R. **Popular ensemble methods: An empirical study**. *Journal of artificial intelligence research*, 11:169–198, 1999.

[92] PARENTE, L.; FERREIRA, L.; FARIA, A.; NOGUEIRA, S.; ARAÃ^{O}JO, F.; TEIXEIRA, L.; HAGEN, S. **Monitoring the brazilian pasturelands: A new mapping approach based on the landsat 8 spectral and temporal domains**. *International Journal of Applied Earth Observation and Geoinformation*, 62:135 – 143, 2017.

[93] PELLETIER, C.; VALERO, S.; INGLADA, J.; CHAMPION, N.; DEDIEU, G. **Assessing the robustness of random forests to map land cover with high resolution satellite image time series over large areas**. *Remote Sensing of Environment*, 187:156 – 168, 2016.

[94] PETITJEAN, F.; INGLADA, J.; GANÇARSKI, P. **Satellite image time series analysis under time warping**. *IEEE Transactions on Geoscience and Remote Sensing*, 50(8):3081–3095, 2012.

[95] PETITJEAN, F.; WEBER, J. **Efficient satellite image time series analysis under time warping**. *Ieee geoscience and remote sensing letters*, 11(6):1143–1147, 2014.

[96] PICOLI, M. C. A.; CAMARA, G.; SANCHES, I.; SIMÕES, R.; CARVALHO, A.; MACIEL, A.; COUTINHO, A.; ESQUERDO, J.; ANTUNES, J.; BEGOTTI, R. A.; OTHERS. **Big earth observation time series analysis for monitoring brazilian agriculture**. *ISPRS journal of photogrammetry and remote sensing*, 145:328–339, 2018.

[97] PLAZA, A. J.; CHANG, C.-I. **High performance computing in remote sensing**. CRC Press, 2007.

[98] POLIKAR, R. **Ensemble based systems in decision making**. *IEEE Circuits and systems magazine*, 6(3):21–45, 2006.

[99] PRZYMUS, P. **Query optimization in heterogeneous CPU/GPU environment for time series databases**. PhD thesis, University of Warsaw. Faculty of Mathematics, Informatics and Mechanics, 2014.

[100] PRZYMUS, P.; KACZMARSKI, K. **Dynamic compression strategy for time series database using gpu**. In: *New Trends in Databases and Information Systems*, p. 235–244. Springer, 2014.

[101] PRZYMUS, P.; KACZMARSKI, K. **Time series queries processing with gpu support**. In: *New Trends in Databases and Information Systems*, p. 53–60. Springer, 2014.

[102] RABINER, L.; JUANG, B. **An introduction to hidden markov models**. *ieee assp magazine*, 3(1):4–16, 1986.

[103] RATHORE, M. M. U.; PAUL, A.; AHMAD, A.; CHEN, B.-W.; HUANG, B.; JI, W. **Real-time big data analytical architecture for remote sensing application**. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(10):4610–4621, 2015.

[104] RIVERS, J. **Scidb: An array-native computational database for heterogeneous, multi-dimensional data sets**. In: *2017 IEEE International Conference on Big Data (Big Data)*, p. 3206–3210, Dec 2017.

[105] ROMANI, L. A.; DE ÁVILA, A. M. H.; ZULLO JR, J.; TRAINA JR, C.; TRAINA, A. J. **Mining climate and remote sensing time series to discover the most relevant climate patterns.** In: *SBBD*, p. 181–195, 2009.

[106] ROSS, S. M. **Introduction to probability models**. Academic press, 2014.

[107] SANDERS, J.; KANDROT, E. **CUDA by example: an introduction to general-purpose GPU programming**. Addison-Wesley Professional, 2010.

[108] SONG, W.; JIN, B.; LI, S.; WEI, X.; LI, D.; HU, F. **Building spatiotemporal cloud platform for supporting gis application**. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1:55–62, 2015.

[109] SOTTILE, M. J.; MATTSON, T. G.; RASMUSSEN, C. E. **Introduction to concurrency in programming languages**. CRC Press, 2009.

[110] STRAHLER, A.; MUCHONEY, D.; BORAK, J.; FRIEDL, M.; GOPAL, S.; LAMBIN, E.; MOODY, A. **Modis land cover and land cover change**. *Algorithm Theoretical Basis Document (ATBD)*, 5, 1999.

[111] SUDMANNS, M.; TIEDE, D.; LANG, S.; BERGSTEDT, H.; TROST, G.; AUGUSTIN, H.; BARALDI, A.; BLASCHKE, T. **Big earth data: disruptive changes in earth observation data management and analysis?** *International Journal of Digital Earth*, 0(0):1–19, 2019.

[112] THACKER, U.; PANDEY, M.; RAUTARAY, S. S. **Performance of elasticsearch in cloud environment with ngram and non-ngram indexing**. In: *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, p. 3624–3628. IEEE, 2016.

[113] TURNBULL, J. **The Docker Book: Containerization is the new virtualization**. James Turnbull, 2014.

[114] VAN DEN BERGH, F.; WESSELS, K. J.; MITEFF, S.; VAN ZYL, T. L.; GAZENDAM, A. D.; BACHOO, A. K. **Hitempo: a platform for time-series analysis of remote-sensing satellite data in a high-performance computing environment**. *International journal of remote sensing*, 33(15):4720–4740, 2012.

[115] VERBESSELT, J.; HYNDMAN, R.; NEWNHAM, G.; CULVENOR, D. **Detecting trend and seasonal changes in satellite image time series**. *Remote sensing of Environment*, 114(1):106–115, 2010.

[116] WAGNER, W.; FRÖHLICH, J.; WOTAWA, G.; STOWASSER, R.; STAUDINGER, M.; HOFFMANN, C.; WALLI, A.; FEDERSPIEL, C.; ASPETSBERGER, M.; ATZBERGER, C.; OTHERS. **Addressing grand challenges in earth observation science: The earth observation data centre for water resources monitoring.** *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 2(7), 2014.

[117] WANG, G.; HE, G.; LIU, J. **A new classification method for high spatial resolution remote sensing image based on mapping mechanism**. In: *Proceedings of the International Conference on Geographic Object-Based Image Analysis (GEOBIA 12)*, p. 186–190, 2012.

[118] WANG, L.; MA, Y.; YAN, J.; CHANG, V.; ZOMAYA, A. Y. **pipscloud: High performance cloud computing for remote sensing big data management and processing**. *Future Generation Computer Systems*, 78:353 – 368, 2018.

[119] WANG, S.; ZHONG, Y.; WANG, E. **An integrated gis platform architecture for spatiotemporal big data**. *Future Generation Computer Systems*, 94:160–172, 2019.

[120] WEGNER MAUS, V.; CÂMARA, G.; APPEL, M.; PEBESMA, E. **dtwsat: Time-weighted dynamic time warping for satellite image time series analysis in r**. *Journal of Statistical Software*, 88(5):1–31, 2019.

[121] WOLFERT, S.; GE, L.; VERDOUW, C.; BOGAARDT, M.-J. **Big data in smart farming-a review**. *Agricultural Systems*, 153:69–80, 2017.

[122] WULDER, M. A.; COOPS, N. C.; ROY, D. P.; WHITE, J. C.; HERMOSILLA, T. **Land cover 2.0**. *International Journal of Remote Sensing*, 39(12):4254–4284, 2018.

[123] XING, Z.; PEI, J.; KEOGH, E. **A brief survey on sequence classification**. *ACM Sigkdd Explorations Newsletter*, 12(1):40–48, 2010.

[124] ZHANG, W.; WANG, L.; LIU, D.; SONG, W.; MA, Y.; LIU, P.; CHEN, D. **Towards building a multi-datacenter infrastructure for massive remote sensing image processing**. *Concurrency and Computation: Practice and Experience*, 25(12):1798–1812, 2013.

[125] ZHANG, X.; KHANAL, U.; ZHAO, X.; FICKLIN, S. **Understanding software platforms for in-memory scientific data analysis: A case study of the spark system**. In: *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, p. 1135–1144. IEEE, 2016.

[126] ZHANG, X.; KHANAL, U.; ZHAO, X.; FICKLIN, S. **Making sense of performance in in-memory computing frameworks for scientific data analysis: A case study of the spark system**. *Journal of Parallel and Distributed Computing*, 120:369–382, 2018.

[127] ZHOU, Z.-H. **Ensemble methods: foundations and algorithms**. Chapman and Hall/CRC, 2012.