A Parallel and Distributed Approach to the Analysis of Time Series on Remote Sensing Big Data

Sávio S. T. de Oliveira, Luiz M. L. Pascoal, Marcelo de C. Cardoso,
Elivelton F. Bueno, Vagner J. S. Rodrigues, Wellington S. Martins

Universidade Federal de Goiás, Goiânia-GO, Brazil
{savioteles,luizmlpascoal}@gmail.com, {marcelo.cardoso,elivelton.bueno,vagner}@gogeo.io,
wellington@inf.ufg.br

Abstract. The era of Remote Sensing Big Data has arrived. Indeed, massive amounts of remotely sensed data have been collected by different countries from a large number of Earth observation spaceborne and airborne sensors. They allow us to identify meaningful changes in the Earth's surface that may affect whole ecological systems and be a threat to biodiversity. Crucial to that end is time series analysis of remote sensing images, for which the Time-Weighted Dynamic Time Warping (TWDTW) algorithm stands out as one of the most used approaches found in the literature so far. However, the computational complexity of the TWDTW algorithm makes it rather inefficient for Remote Sensing Big Data. Also, the huge volume of high spatial-temporal resolution remote sensing data cannot be handled by a single computing node. To overcome that drawback, this work proposes a parallel algorithm, named SP-TWDTW (Spatial Parallel TWDTW), that allows for the analysis of large scale time series using Manycore architectures (GPU). In order to process massive time series of remote sensing data in a cluster of computers, an approach for distributing the TWDTW processing is introduced in this paper.

Categories and Subject Descriptors: D.1.3 [**Software**]: Concurrent Programming; H.2.4 [**Database Management**]: Miscellaneous; C.2.4 [**Computer-communication networks**]: Distributed Systems—*Distributed databases*; *Distributed applications*

Keywords: Big Data, Parallel Programming, Remote Sensing, Time Series Analysis

1. INTRODUCTION

Never before in the current era has the Earth's surface changed so fast. While urban and agricultural areas greedily expand into the surrounding natural space, whole forest ecosystems are diminishing at an alarming speed. To identify those changes and highlight their dynamics, the analysis of remotely sensed image time series has become an essential technique. In fact, time series analysis is now being embedded in many systems used by the global community addressing questions on sustainable environment [Bégué et al. 2018]. The ever-increasing volume of satellite remote-sensing data shows significant potential for scientists to explore it through time series analysis [Huang et al. 2017]. Remote sensing data are clearly showing the characteristics of Big Data. We are now living in the age of Remote Sensing Big Data [Chi et al. 2016].

Dynamic time warping (DTW) has been widely used as a distance measure for time series classification [Bagnall et al. 2017]. However, it is not well suited for time series analysis of land use and land cover data because it disregards the temporal range when finding the best alignment between two time series [Maus et al. 2019]. The *Time-Weighted Dynamic Time Warping (TWDTW)* algorithm performed well in identifying land use and land cover with better results than other DTW variations for searching occurrences of patterns in time series of remote sensing data [Maus et al. 2019].

The TWDTW algorithm has a high computational cost, with time complexity of $O(n^2)$, where n is the length of the longest time series being analyzed, making its use unfeasible in Remote Sensing Big Data [Xiao et al. 2013]. In this scenario it is necessary to parallelize the TWDTW algorithm. However, the data dependence of it is high and is a challenge to exploit the fine-grained parallelism. Besides that, the huge volume of high spatial-temporal resolution remote sensing data cannot be handled by a single computing node [Huang et al. 2017].

This paper proposes a new massive parallel algorithm, named *Spatial Parallel TWDTW (SP-TWDTW)*, based on TWDTW, that explores the Manycore (GPU) architectures addressing the TWDTW data dependency. In order to support time series analysis in Remote Sensing Big Data, a new architecture is introduced for processing the time series analysis algorithm in a cluster of computers, where the memory spaces are automatically managed so that they do not exceed capacity. Unlike TWDTW, SP-TWDTW takes into account the temporal axis and the spatial autocorrelation to determine the land use mapping in a given region. Techniques which ignore spatial autocorrelation typically perform poorly in the presence of spatial data [Vatsavai 2008].

The main contributions of this work are:

—A new massive parallel algorithm for spatio-temporal analysis of remote sensing images.
—The inclusion of the spatial dimension in time series classification.
—An automatic system to manage the memory usage between CPU and GPU spaces.
—An architecture for processing time series analysis in a distributed environment.
—Experimental work with real and synthetic data

This paper is an extended version of [de Oliveira et al. 2018], presented in the XIX Brazilian Symposium on GeoInformatics (GEOINFO 2018). In particular, the present work introduces a new architecture for processing TWDTW in a cluster of computers to support the time series analysis for Remote Sensing Big Data. This paper is organized as follows. Section 2 summarizes the background, related work and existing systems that support parallel processing of time series analysis. Section 3 describes the TWDTW algorithm used as the basis for this work. The new algorithm proposed in this paper is presented in Section 4 for parallel and distributed processing of time series analysis. Section 5 validates the parallel and distributed solution proposed in this work. Finally, Section 6 presents some conclusions and future work.

2. BACKGROUND AND RELATED WORK

In this section, we summarize the background, related work and existing systems that support parallel processing of time series analysis. In Section 2.1, we review methods for the time series analysis of remote sensing data using the DTW algorithm. In Section 2.2 we elaborate on the current work for processing time series analysis with parallel architectures. Section 2.3 introduces some algorithms to improve the time series analysis through spatial interpolation.

2.1 Time Series Analysis

Time series analysis comprises methods for extracting important statistics and characteristics from time series data. The DTW allows the alignment between two time series, even if they have different lengths or they are not aligned on the time axis. Given the time series A and B, the distance between them is computed as

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

where $w_k = (i, j)$ represents the association between the i-th and j-th observations, a_i and b_j, respectively time series A and B, which are equivalents according to the Euclidean distance, $d(i, j) = \sqrt{(a_i - b_j)^2}$. The sequence $w_1, w_2, ..., w_k$ represents the association between pairs of observations from the two given time series, denoted by the adjustment path. Equation 1 is subject to the following conditions: i) The first observation of one series must match the first observation of the other series, $w_1 = (1, 1)$, and the last observation of one series must match the last observation of the other, $w_k = (m, n)$; ii) Given $w_k = (i, j)$ and $w_{k+1} = (i', j')$ then $0 \leq i' - i \leq 1$ and $0 \leq j' - j \leq 1$.

Several techniques of time series analysis have been previously proposed [Cressie and Wikle 2015]. Some methods process each image independently and compare the results for different time instances [Gómez et al. 2011; Lu et al. 2016]. The technique presented in [Costa et al. 2018] builds a time series of each pixel and processes them independently. In the end, the algorithm chooses some seed pixels in the image and calculates the distance between the time series of these seeds to their neighbors using the DTW method, grouping similar neighbors. Some papers [Petitjean et al. 2012; Petitjean and Weber 2014] proposed non-parallel methods using DTW to analyze time series of satellite images. Petitjean et al. [2012] and Petitjean and Weber [2014] used a maximum time delay to avoid time distortions based on the date of the satellite images.

2.2 Parallel and Distributed Approaches for Time Series Analysis

The rapid growth of Multicore/Manycore processors has attracted the attention of many researchers. Verbesselt et al. [2010] and Jamali et al. [2015] proposed algorithms for supporting parallel execution in Multicore architectures using the library *foreach*[1] from R language. The TWDTW (described in Section 3) method [Maus et al. 2016] also uses *foreach* package to parallelize the execution. The papers [Xiao et al. 2013; João Jr et al. 2017; Zhu et al. 2018] presented parallel solutions to analyze time series using Manycore architectures (GPUs), but time series analysis in remote sensing was not addressed.

The unprecedented proliferation of data has posed significant challenges in managing, processing and interpreting this large volume of data. Due to the emergence of high-performance computing clusters, the Apache Spark [Zaharia et al. 2010] cluster implementation has been chosen to compute distributed DTW efficiently, achieving nearly linear speedup with DTW [Shabib et al. 2015]. The Apache Hadoop framework [White 2009] is also used for the distributed processing of DTW for fast similarity search based on DTW with the MRDTW (Map Reduce based DTW) platform [Yin et al. 2015]. For remote sensing time series analysis, a map/reduce architecture was proposed [Camara et al. 2016] for distributed processing of the TWDTW algorithm in the Hadoop ecosystem.

Great efforts have been made towards the incorporation of novel ideas for searching and mining massive time series under Dynamic Time Warping (DTW) [Rakthanmanon et al. 2013]. Xu et al. [2016] designed, implemented, and evaluated a time series analysis approach that is able to decompose large scale mobile traffic into regularity and randomness components. Huang et al. [2017] introduced a novel in-memory computing framework using Apache Spark on Hadoop Yarn model with a focus on parallel processing of massive remote sensing data, but not handling time series. Some papers [Chebbi et al. 2018; Ma et al. 2015; Chi et al. 2016; Liu et al. 2018] describe the most challenging issues in managing, processing, and efficient exploitation of big data for remote sensing problems. Chebbi et al. [2018] focus on the comparison of two well-known platforms of Remote Sensing Big Data namely Hadoop and Spark. Liu et al. [2018] makes an attempt to introduce the latest theory, methods and applications to manage, exploit and analyze remote sensing big data. Ma et al. [2015] and Chi et al. [2016] give an overview of the Big Data problems, including the analysis of Remote Sensing Big Data challenges, current techniques and works for processing remote sensing big data. Many platforms have been proposed to deal with big data in remote sensing, but no solution was

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

found for time series processing on Remote Sensing Big Data using distributed and massively parallel (GPU) architectures.

2.3 Spatial Interpolation

Some papers perform the time series analysis through spatial interpolation [Li and Heap 2014], which is the process of using points with known values to estimate values of other unknown points. Methods for time series analysis which ignore spatial autocorrelation typically perform poorly in the presence of spatial data [Vatsavai 2008]. Several methods have been used for it, of which the most important are:

(1) **Nearest Neighbor:** the value of each point is determined by the nearest points [Mitas and Mitasova 1999];
(2) **IDW:** it weights the points closer to the prediction location greater than those farther away [Shepard 1968];
(3) **Kriging:** assumes that the distance or direction between sample points reflects a spatial correlation that can be used to explain variation in the surface [Stein 2012].

3. TIME-WEIGHTED DYNAMIC TIME WARPING (TWDTW)

The DTW algorithm does not perform well for time series analysis of remote sensing images because it disregards the temporal range when finding the best alignment between two time series classifications [Maus et al. 2016]. So, the TWDTW [Maus et al. 2016] was proposed as a variation of the DTW algorithm. The TWDTW is sensitive to seasonal changes of natural and cultivated vegetation types and considers inter-annual climatic and seasonal variability. The TWDTW method computes the cost matrix $\Psi_{n,m}$ given the pattern $U = (u_1, ..., u_n)$ and time series $V = (v_1, ..., v_m)$. The elements $\psi_{i,j}$ of $\Psi_{n,m}$ are computed by adding the temporal cost ω, becoming $\psi_{i,j} = |u_i - v_j| + \omega_{i,j}$, which $u_i \in U$ $\forall\ i = 1, ..., n$ and $v_j \in V\ \forall\ j = 1, ..., m$. To calculate the time cost, the logistic model is used with a midpoint β and a bias α presented in Equation 2.

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

in which $g(t_i, t_j)$ is the elapsed time in days between dates t_i for the patterns U and t_j in the time series V. From the cost matrix Ψ an accumulated cost matrix is calculated, named D by using a recursive sum of the minimum distances, as shown in equation 3

$$d_{i,j} = \psi_{i,j} + min\{d_{i-1,j}, d_{i-1,j-1}, d_{i,j-1}\}, \tag{3}$$

which is subject to the following conditions:

$$d_{ij} = \begin{cases} \psi_{i,j} & i = 1, j = 1 \\ \sum_{k=1}^{i} \psi_{k,j} & 1 < i \leq n, j = 1 \\ \sum_{k=1}^{j} \psi_{i,k} & i = 1, 1 < j \leq m \end{cases} \tag{4}$$

The k-th lowest cost path in D produces an alignment between the pattern and a subsequence of V with associated distance δ_k, in which a_k is the first element and b_k the last element of k. Each minimum point in the last row of the cost matrix is accumulated, i.e. $d_{n,j}\ \forall\ j = 1, ..., m$, produces an alignment, with $b_k = argmin_k(d_{n,j}), k = 1, ..., K$ and $\delta_k = d_{n,b_k}$, in which K is the minimum number of points in the last row of D.

A reverse algorithm, Equation 5, maps the path $P_k = (p_1, ..., p_L)$ along the k-th "valley" to the lowest cost in D. The algorithm starts in $p_{l=L} = (i = n, j = b_k)$ and ends with $i = 1$, i.e. $p_{l=1} = (i =$

$1, j = a_k$), in which L denotes the last point of alignment. The path P_k contains the elements that have been matched between the series.

$$p_{l-1} = \begin{cases} (i, a_k = j) & \text{se } i = 1 \\ (i-1, j) & \text{se } j = 1 \\ argmin(d_{i-1,j}, d_{i-1,j-1}, d_{i,j-1}) & \text{otherwise} \end{cases} \quad (5)$$

The land cover mapping with TWDTW is performed in two steps. In the first step, the DTW algorithm is applied to each pattern in $U \in Q$ and each time series $V \in S$. This step provides information on how many patterns match with time series intervals. In the second step, the best matching pattern found by the DTW algorithm is used for land cover mapping.

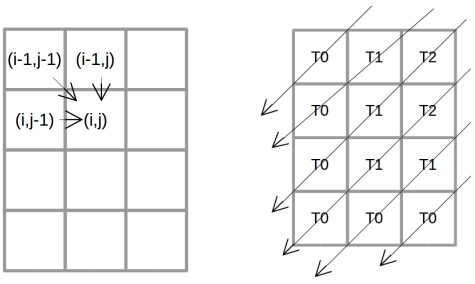
4. SPATIAL-TIME SERIES ANALYSIS OF REMOTE SENSING IMAGES WITH A PARALLEL ARCHITECTURE

TWDTW is a pattern-matching algorithm based on dynamic programming with time complexity $O(n^2)$. This section presents the solution proposed in this work for the parallel processing of spatial time series analysis. This solution, named SP-TWDTW (Spatial Parallel TWDTW), parallelizes the TWDTW analyzing the temporal axis of the time series, as well as the spatial axis of the neighboring pixels to classify each time series.

The accumulated cost matrix D is computed from the cost matrix Ψ using the recursive sum of the minimum distances, as shown in equation 3. The construction of D cannot be trivially paralleled since the computation of each element (i, j) of the matrix depends on the previous elements $(i-1, j)$, $(i, j-1)$ and $(i-1, j-1)$. This dependency can be seen in Figure 1a. The idea behind the SP-TWDTW algorithm is presented in Figure 1b. Each diagonal is computed in parallel, with each thread being responsible for a diagonal cell. Since the elements are not dependent on each other within the diagonal, the calculation of the accumulated cost does not lead to an inconsistent matrix. The details of the SP-TWDTW matrix computation are presented in Algorithm 1 in Section 4.1. Section 4.2 introduces a new architecture for processing time series analysis in a distributed environment.

4.1 Spatial Parallel Time-Weighted Dynamic Time Warping (SP-TWDTW)

Algorithm 1 describes the SP-TWDTW, which has as input the set of patterns Q and the set of time series S and calculates the final alignment cost matrix between each $U \in Q$ and $V \in S$. Since Q



(a) The computation of each element in D depends on the values of previous elements.

(b) SP-TWDTW: Parallel processing of D

Fig. 1. Computation of the accumulated cost matrix D

and S can be larger than available memory, the input of this algorithm admits that these sets are stored on the disk. The SP-TWDTW manages the loading of blocks of Q and S to CPU memory and subsequently to the GPU memory so that it does not exceed their limits. The SP-TWDTW also receives as input the maximum size of the sets Q and S that fill in the GPU memory (bQ and bS respectively) and CPU memory (max_bQ and max_bS respectively).

The algorithm starts in lines 2 and 3 by reading the blocks of the patterns into the $queueQ$ and blocks of the time series into the $queueS$. This task is performed by a CPU thread that manages the input queue size and the CPU memory available size. So, it is not necessary to wait for this step to finish before starting to load the blocks into the GPU global memory. Between lines 4 and 28, the Algorithm 1 loads the blocks into the GPU memory and computes the matrix D. This is done while there are blocks to be processed.

From line 5 until line 8, bQ patterns U and bS time series V are loaded into the GPU global memory, joining the patterns in a single $bigQ$ sequence and the time series in a single $bigS$ sequence. This union allows the GPU to perform block processing using all its computational power. In line 9, the cost matrix is constructed from $bigQ$ and $bigS$, with each GPU thread being responsible for computing an element of the array.

The calculation of matrix D is performed following the idea presented in Figure 1b, where each thread computes the cost of each element in the diagonal. Since each element of the diagonal depends only on the two diagonals, they can be calculated independently. Given that $bigQ$ and $bigS$ have several patterns U and time series V, $bQ * bS$ threads are launched in the GPU, with each block of threads being responsible for calculating the cumulative cost between a pattern U and a time series V. Each block in the GPU has $min(sizeU, sizeV)$ threads that is the maximum size of a diagonal when comparing U and V, so that each thread performs the calculation of one diagonal element.

From line 14 to 20, the costs of the first $sizeU$ elements from the upper diagonals above the secondary diagonal are computed. In a square matrix, these first $sizeU$ diagonals are the upper triangular matrix. The Algorithm 1 assumes that the index starts at position 0. To identify each diagonal, the row index in the upper matrix is computed as $si - tid$ and the column index is determined by the thread id. The matrix D is updated for each diagonal element using the function $update\_accumulated\_cost\_matrix$.

The elements of the next $sizeV - 1$ diagonals are computed between lines 21 and 26. In a square matrix, for example, these $sizeV - 1$ diagonals are the lower triangular matrix. To identify each diagonal, the row index in the upper matrix is calculated as $sizeU - tid - 1$ and the column index is set to $sizeV - sj - tid - 1$. The matrix D is then updated for each element of the diagonal.

$$D[index_{i,j}] = min(D[index_{i-1,j}], D[index_{i-1,j-1}], D[index_{i,j-1}) + \Psi[index_{i,j}] \tag{6}$$

$$index = (i + blockIdx.x * sizeU) * sizeV * gridDim.y + (j + blockIdx.y * sizeV) \tag{7}$$

The function $update\_accumulated\_cost\_matrix$ (Algorithm 2) updates each element $D_{i,j}$ of the accumulated cost matrix D. The calculation of $D_{i,j}$ follows the equation 3, which is the smallest value between $D_{i-1,j}$, $D_{i-1,j-1}$ and $D_{i,j-1}$ plus the cost $\Psi_{i,j}$. The index in the matrices Ψ and D related to i, j of the matrices U and V must be computed, since the matrices Ψ and D are sent as vectors to the GPU and the series U and V in blocks of size bQ and bS respectively. So, it is necessary to find the pair U and V inside D and Ψ and then find the correct position in the matrix D related to U and V. In the GPU $bQ * bS$ blocks of threads are launched with bQ on the x-axis and bS on the y-axis. So, each block of threads handles one block in Ψ and D.

The matrix D is computed following equation 6 and the global index of Ψ and D follows the equation 7, in which $(i + blockIdx.x * sizeU) * sizeV * gridDim.y$ finds the correct line in Ψ and D, wherein $blockIdx.x$ the id of the block of threads in the x-axis and $gridDim.y$ the number of blocks in the

y-axis. The part of the equation $(j + blockIdx.y * sizeV)$ finds within the matrix line the correct position of the element (i, j), being $blockIdx.y$ the id of the block of threads in the y-axis.

Algorithm 1: sp-twdtw(Q, bQ, max_bQ, S, bS, max_bS, $temporal\_weight$, $spatial\_weight$)

Input: Q: set of patterns U
bQ: number of patterns U in the GPU memory
max_bQ: max patterns U in the CPU memory
S: set of time series V
bS: number of time series V in the GPU memory
max_bS: max time series V in the CPU memory
$temporal\_weight$: temporal axis weight
$spatial\_weight$: spatial axis weight
Output: R: final alignment cost matrix between each U and V

```
1  while There are patterns U and time series V on disk do
2      queueQ ← load max_bQ patterns U to CPU memory
3      queueS ← load max_bS timeseries V to CPU memory
4      while There are patterns in queueQ and time series in queueS do
5          gpu_queueQ ← load bQ patterns U to GPU global memory
6          gpu_queueS ← load bS time series V to GPU global memory
7          bigQ ← merge all patterns U in gpu_queueQ
8          bigS ← merge all time series V in gpu_queueS
9          Ψ ← compute the cost matrix between bigQ and bigS
10         sizeU ← compute the pattern size of each U in bigQ
11         sizeV ← compute the time series size of each V in bigS
12         tid ← thread id
13         if tid < sizeU then
14             for si ← 0 to sizeU - 1 do
15                 if tid ≤ min(si, sizeV − 1) then
16                     i ← si − tid
17                     j ← tid
18                     update_accumulated_cost_matrix(Ψ, D, i, j, sizeU, sizeV)
19                 end
20             end
21             for sj ← sizeV − 2 to 0 do
22                 if tid ≤ min(sj, sizeU − 1) then
23                     i ← sizeU − tid − 1
24                     j ← sizeV − sj − tid − 1
25                     update_accumulated_cost_matrix(Ψ, D, i, j, sizeU, sizeV)
26                 end
27             end
28         end
29         for Each U in gpu_queueQ and V in gpu_queueS do
30             compute_dtw_path(R, D, sizeU, sizeV)
31         end
32     end
33 end
34 for Each line i in R do
35     for Each column j in R do
36         I_{i,j} ← compute_spatial_interpolation(R_{i,j})
37         R_{i,j} ← R_{i,j} * temporal_weight + I_{i,j} * spatial_weight
38     end
39 end
```

Algorithm 2: update_accumulated_cost_matrix(Ψ, D, i, j, $sizeU$, $sizeV$)

Input: Ψ: input cost matrix
D: input accumulated cost matrix
i: pattern index in the cost matrix
j: time series index in the cost matrix
$sizeU$: patter length
$sizeV$: time series length

1 $index_{i,j} \leftarrow$ computes the global index in Ψ from indices i and j
2 $index_{i-1,j-1} \leftarrow$ computes the global index in Ψ from indices $i-1$ and $j-1$
3 $index_{i-1,j} \leftarrow$ computes the global index in Ψ from indices $i-1$ and j
4 $index_{i,j-1} \leftarrow$ computes the global index in Ψ from indices i and $j-1$
5 $D[index_{i,j}] \leftarrow min(D[index_{i-1,j}], D[index_{i-1,j-1}], D[index_{i,j-1}) + \Psi[index_{i,j}]$

The algorithm computes the final alignment between each U and V following the equation 5 in the function *compute_dtw_path* between lines 29 and 31. This cost is stored in the matrix R, each row of which represents a pattern U and each column a time series V to be sorted.

Between lines 34 to 39, the SP-TWDTW analyzes the spatial axis, performing a spatial interpolation on line 36 for each alignment between U and V, that is, each element of R. We already have the cost of the alignment between U and V stored in R_{ij}, but in line 36, the method *compute_spatial_interpolation*($R_{i,j}$) searches for the cost of other spatially close time series to V related to the pattern U, estimates the value of R_{ij} and stores it in the matrix I at I_{ij}. There are several factors that impact on the quality of this spatial prediction [Li and Heap 2014] and, therefore, a mechanism has been made where it is possible to give weights for the temporal and spatial axis in line 37 of the algorithm. This weighted cost is stored in the output R matrix.

The drawback of the diagonal based method is that the sizes of the diagonals vary, which causes a waste of GPU resources. When the diagonal size is lower than the number of block threads, some of these threads become idle. But, the performance gain in parallelizing the computation of the diagonal is better than sequential computation.

4.2 Distributed Processing of Spatial-Time Series Analysis

In order to explore the potential of Big Data, a solution with a parallel architecture and distributed memory must be used consisting of three main components [Ranjan 2014]: i) data ingestion: collects data from multiple sources, ii) data processing: exploitation of available computing resources, such as a cluster of computers with CPUs and GPUs and iii) data storage: maintaining and retrieving the system data. This Section presents an architecture for parallel and distributed processing of spatial time series in the context of Remote Sensing Big Data. Because of the volume of data, this solution explores a cluster of computers and the Manycore architecture of the GPUs in each server. So, it is possible to store, index, and query this large volume of data efficiently.

At a high level, it keeps the spatial and relational indexes in host memory, handling data ingestion using CPUs and data recovery via disks. At query time, the data are moved from host memory to GPU memory for parallel processing on GPU. As shown in Figure 2, the architecture consists of a Core Executor, a Query Executor, an ETL module, and a disk store. Clients ingest data and run queries through the HTTP API.

4.2.1 *Core Executor.* The Core Executor is responsible for running the jobs for time series analysis on the local processing unit on each server in the cluster. If the GPU is available and the algorithm is ready to run on it, Core Executor chooses the GPU to process the time series analysis. Otherwise, the CPU cores are chosen. The challenge, in this case, lies in how to gather the best from each processing

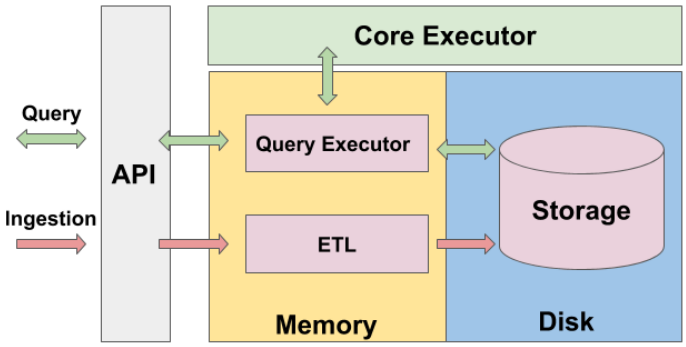


Fig. 2. The architecture for time series analysis in Remote Sensing Big Data.

unit, since CPU cores support fewer threads and do not require additional data communication, while GPUs expect a large number of concurrent threads, and require explicit communication between the standard machine memory and the GPU.

The SP-TWDTW runs on the CPU and is responsible for sending the data to the GPU if it is available on the machine. For algorithms that do not have this mechanism, the algorithm is invoked according to the availability of the GPU on the server, with Core Executor being responsible for controlling CPU and GPU memory. If the server does not have a GPU available, the algorithm runs on the CPU.

4.2.2 *Query Executor.* The Query Executor uses Apache Spark [Zaharia et al. 2010] to perform temporal and relational filtering on the data achieving high performance, using a state-of-the-art DAG (Directed Acyclic Graph) scheduler, a query optimizer, and a physical execution engine. Spark offers over 80 high-level operators that make it easy to build parallel apps. The native Spark ecosystem does not provide support for spatial data and operations. So, the Query Executor included the Geospark system [Yu et al. 2018], which extends the core engine of Apache Spark and SparkSQL to support spatial data types, indexes, and geometrical operations at scale.

The Query Executor utilizes prefilters to cheaply filter archived data before sending them to Core Executor for parallel processing. After prefiltering, only those satisfying filter condition values need to be pushed to the Core Executor. Input data is fed to the Core Executor and executed there one batch at a time. The Query Executor is responsible for managing the CPU memory, so that the time series data is retrieved from the disk and does not exceed the memory capacity. For the SP-TWDTW algorithm, the data is sent to CPU memory, since the SP-TWDTW has an internal memory control mechanism.

4.2.3 *Extract, Transform and Load (ETL).* The ETL process (Extracting-Transforming-Loading) is responsible for extracting data from heterogeneous sources, transforming and finally loading them into the storage system. The remote sensing image data acquired from either aircraft or spacecraft platforms is readily available in digital format and made available like raster data. The ingester API is ready to work with raster image data on GeoTIFF, an extension of the Tagged Image File Format (TIFF). GeoTIFF includes a standard definition of geolocation information and it is one of the most popular formats for earth observing remote sensing data [Qu et al. 2006].

The architecture allows several remote sensing images to be inserted in a distributed and parallel manner. The ingestion of data begins with sending the time series of remote sensing data through the API. The ETL module divides input data streams into batches and stores them in Hadoop Distributed

File System (HDFS) [2]. HDFS provides scalable, highly available and fault-tolerant storage at low-cost [Olson 2010]. HDFS also detects and compensates hardware issues, including disk problems and server failure.

The ETL module converts the input TIFF remote sensing image to CSV file to allow the spatial-temporal filtering on Query Executor because GeoSpark inside it cannot handle remote sensing TIFF image files natively. Once data is available into an ETL HDFS instance, it is then replicated to other servers following a replication factor that can be configured to ensure fault tolerance, load balance, and reliability.

5. EXPERIMENTS AND DISCUSSION

The following section describes the test scenario consisting of the data and hardware that is used to obtain the running times and other characteristics of the SP-TWDTW algorithm. The results are presented in Section 5.1 for parallel processing on Manycore architectures. The size-up, scale-up and speedup of the architecture proposed for distributed processing of the SP-TWDTW is evaluated in Section 5.2. In Section 5.3 we analyze the results of the solution proposed in this work for distributed and parallel time series analysis in Remote Sensing Big Data context.

5.1 SP-TWDTW

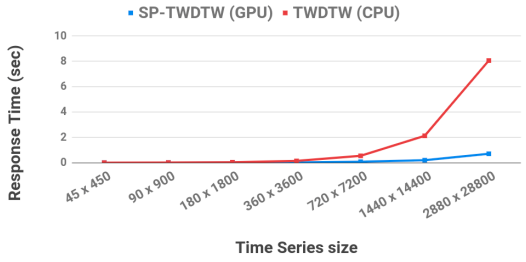
This section aims to evaluate the performance of the TWDTW and SP-TWDTW algorithms executed on CPU and GPU, respectively. In the CPU tests, a machine with Intel Xeon E5-2686 v4 (Broadwell) processors and 61 GB of RAM was used. The GPU tests were performed on an NVIDIA Tesla V100 card with 16 GB of available memory and 5120 CUDA cores with a clock of 1455 MHz. The TWDTW was implemented in C++ [3] and SP-TWDTW was implemented on the GPU using the NVIDIA CUDA language. To compare the response time between the SP-TWDTW and the TWDTW, the time series V and the patterns U were obtained from real data in Brazil [Maus et al. 2016] from MODIS sensor, specifically the Porto dos Gauchos municipality, that covers approximately 7,000 km^2 and is located in the state of Mato Grosso, Brazil, inside of the Amazon Biome. Each test was performed ten times and the response time was obtained from the average of them.

The temporal and spatial resolution of remote sensing systems has increased in the last years being a great challenge for the remote sensing field [Battude et al. 2016]. The results regarding the response time for high temporal resolution are presented in Figure 3a, by comparing the TWDTW and SP-TWDTW over a pattern U and a time series V, varying their size. The Planet's [4] and Jilin-1 [5] constellation of satellites are capable of revisiting the same location on earth twice a day. In 20 years, nearly 14600 observations have been captured and this number tends to grow.

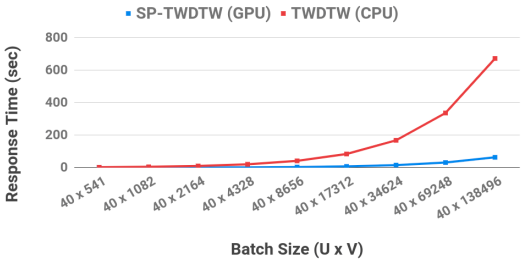
The TWDTW algorithm works better for smaller time series due to the fact that SP-TWDTW uses the GPU for processing. In this case, the transfer time of U and V from the CPU memory to the GPU memory overcomes the final response time of the algorithm. The response time increases considerably on TWDTW algorithm as the size of the pattern U and the time series V increases. The SP-TWDTW algorithm performed better as the time series size grew reaching a response more than 11 times lower than TWDTW because it was able to take advantage of the processing cores available in the GPU when demand was higher.

The results regarding the high spatial resolution are shown in Figure 3b by comparing the response

[2] `https://hadoop.apache.org/docs/current1/hdfs_design.html`
[3] The original version of TWDTW was developed in R, but, in this work, we implemented in C++ to be able to compare fairly with SP-TWDTW, since the C++ language allows for better performance.
[4] https://www.planet.com/products/planet-imagery/
[5] https://www.cgsatellite.com/imagery/static-imagery/



(a) Comparison between the TWDTW (CPU) and the SP-TWDTW (GPU) algorithms using only one pattern U and one time series V varying their size. The x axis contains the size of U and V following the format: $sizeof U$ x $sizeof V$.



(b) Comparison between the TWDTW (CPU) and the SP-TWDTW (GPU) algorithms with several patterns U and time series V. The x axis follows the format: number of patterns U x number of time series V. Each pattern has size 45 and time series size 23.

Fig. 3. Comparing the response time between TWDTW (CPU) and SP-TWDTW (GPU). Both algorithms were implemented in C++.

time of SP-TWDTW and TWDTW using small batches of time series U and V with a size of 45 and 23 respectively. The pattern batch size was fixed at 40 to keep a realistic scenario for remote sensing. The SP-TWDTW used a batch size equal to 40x138496, running several batches to calculate the alignment between each U and V. In this scenario, the SP-TWDTW obtained a response time almost 11 times lower than the TWDTW, since it exploited the GPU architecture.

To compare the accuracy of the SP-TWDTW algorithm and TWDTW algorithm, we selected the Porto dos Gauchos municipality, located in the state of Mato Grosso, Brazil, inside of the Amazon Biome. The most important classes for that area were selected: cotton-fallow, forest, soybean-cotton, soybean-maize, and soybean-millet. Data from this study area were extracted from the MODIS product MOD13Q1 and include vegetation indices "NDVI", "EVI", and original bands "NIR", "RED", "BLUE", and "MIR". This product has 250 x 250 m spatial resolution and a 16 day maximum-value composite (MVC) for each pixel location.

The input has a total of 603 samples divided into five classes: 68 "Cotton-fallow", 138 "Forest", 79 "Soybean-cotton", 134 "Soybean-maize", and 184 "Soybean-millet". To test the effectiveness of SP-TWDTW, we used the cross-validation technique, running the test 100 times splitting the complete data into random training and test sets in each execution. The logistic weight of TWDTW has midpoint $\beta = 100$ days and steepness $\alpha = 0.1$. The Kriging, IDW and Nearest Neighbor(NN) spatial interpolation methods were used, with their parameters being optimized through k-folds cross-validation with k = 5.

Table I presents the overall accuracy of SP-TWDTW with the Kriging, IDW, and Nearest Neighbor(NN) spatial interpolation methods. Overall accuracy (OA) is simply the proportion of the area mapped correctly. It provides the user of the map with the probability that a randomly selected location on the map is correctly classified [Olofsson et al. 2013]. An analysis was performed by varying the spatial and temporal axis weights and the interpolation methods. The SP-TWDTW algorithm with weight 0 for the spatial axis and 1 for temporal shows the OA of the TWDTW algorithm. It is noteworthy that, in this case, the SP-TWDTW algorithm has the same accuracy as the TWDTW, but does gain in response time. The original TWDTW algorithm implemented in R [Maus et al. 2019] obtained a response time of 9851ms while the SP-TWDTW on the GPU took in 40ms, that is, 246 times faster than the CPU TWDTW implementation in R

Regardless of the interpolation method, for our case study dataset, when the spatial axis is set with a higher value of weight than the time axis, the SP-TWDTW presented lower accuracy than the TWDTW. This is explained by spatial closeness between the "Soybean-maize", "Soybean-cotton" and

Table I. Overall accuracy with the confidence interval (95% confidence level) of the SP-TWDTW method varying the spatial and temporal weights and the interpolation methods.

Weights		Interpolation Methods OA (%)		
Spatial	**Temporal**	**Kriging**	**NN**	**IDW**
0	1	98.01±0.17	97.72±0.25	97.72±0.24
0.1	0.9	97.78±0.27	97.94±0.19	97.94±0.17
0.2	0.8	97.99±0.17	98.11±0.12	**98.11±0.10**
0.3	0.7	98.01±0.15	97.94±0.18	97.92±0.18
0.4	0.6	97.63±0.18	97.78±0.15	97.75±0.16
0.5	0.5	97.11±0.18	97.09±0.21	97.08±0.21
0.6	0.4	92.11±0.33	92.24±0.35	92.17±0.34
0.7	0.3	87.16±0.22	87.18±0.19	86.99±0.20
0.8	0.2	86.71±0.18	86.77±0.22	86.70±0.20
0.9	0.1	86.43±0.19	86.45±0.16	86.41±0.16
1	0	86.14±0.22	86.05±0.19	86.09±0.20

Table II. User's and Producer's accuracy evaluation for each land usage type according to the TWDTW and SP-TWDTW algorithms with the confidence interval (95% confidence level)..

Method	Cotton-fallow		Forest		Soybean-cotton		Soybean-maize		Soybean-millet	
	UA (%)	PA (%)	UA (%)	PA (%)	UA (%)	PA (%)	UA (%)	PA (%)	UA (%)	PA (%)
TWDTW	96.01±0.25	99.91±0.07	100±0.0	100±0.0	99.73±0.14	88.80±0.28	93.77±0.70	99.64±0.14	99.68±0.12	98.13±0.78
SP-TWDTW	96.16±0.24	99.77±0.14	100±0.0	100±0.0	99.57±0.24	88.35±0.45	94.41±0.41	99.75±0.11	99.79±0.07	99.07±0.33

"Cotton-fallow" crops. In addition, the time series pattern of these three classes is very similar, which ends up generating confusion in the time series analysis by TWDTW and SP-TWDTW. Concerning the interpolation methods, they performed similarly due to the uniform input dataset distribution. For sparse data, the Kriging method would probably present better accuracy than the IDW [Li and Heap 2014].

Table II presents the user's and producer's accuracy assessment for each land use type from the TWDTW and SP-TWDTW time series analysis methods. User's accuracy is the proportion of the area mapped as a particular category that is actually that category "on the ground" where the reference classification is the best assessment of ground condition [Olofsson et al. 2013]. Producer's accuracy is the proportion of the area that is a particular category on the ground that is also mapped as that category [Olofsson et al. 2013].

The IDW spatial interpolation method was chosen fixing the spatial and temporal weights in 0.2 and 0.8 respectively. For "Forest" land usage, the SP-TWDTW did not perform worse because the data in these land usages are spatially clustered and independent from other land classes. The accuracy for SP-TWDTW was equal or better for "Soybean-maize" and "Soybean-millet", improving the TWDTW accuracy in a region that has already presented great results. Since the "Soybean-cotton" and "Cotton-fallow" crops are spatially closed and mixed, the spatial weights impacted on the analysis with a lower accuracy than TWDTW.

5.2 Evaluation of the Distributed Processing of Time Series Analysis

A horizontal scalability test was executed to measure the performance of the distributed architecture proposed in Section 4.2 to run the TWDTW (CPU) algorithm when adding servers to the cluster. Ideally, the system should have linear horizontal scalability, which is not always possible since: a) the tasks distributed in the cluster have to be synchronized and b) the devices are subject to physical constraints as in the case of those used for networking. The experimental cluster consists of 4 physical machines. The configuration of each machine consisted of an Intel Core i7 3.4 GHz, with 16 GB of RAM and 1 TB hard drives. The machines were connected to a 1 Gbit/s Ethernet network and a

6248P Dell PowerConnect switch.

The cluster was created with Apache Spark 2.3.3 and the algorithms were evaluated on Hadoop Yet Another Resource Negotiator (YARN) platform [6] version 2.8.2. When Spark applications run on YARN, resource management, scheduling, and security are controlled by YARN. To optimize the execution of Spark applications on a Hadoop YARN model, parameters were tuned according to Cloudera's Guide [7]. Specifically, we preserved the necessary memory space for OS and its daemon services to limit the amount of physical memory that can be allocated for containers.

The dataset consists of 4 million up to 128 million synthetic time series with about 23 points, each one varying in size from 640MB to 20GB. Six synthetic databases were generated with 4 million(4M), 8 million(8M), 16 million(16M), 32 million(32M), 64 million(64M) and 128 million(128M) time series allowing an evaluation with an increasing dataset size. The data was stored in HDFS with a replication factor of 3 and a block size of 128 MB.

Figure 4 shows the result of running the TWDTW (CPU) algorithm in the cluster. We measured statistics for speedup and scale-up. For speedup, we fixed the input dataset and increased the number of nodes in the clusters varying from 2 nodes to 4 nodes. The response time was measured with 1, 2, and 4 servers to help in evaluating the platform scalability. Similarly, for scale-up, we increased both the input dataset and the cluster size. Scale-up helps in understanding the behavior of the platform if both data and cluster size increase.

Figure 4a shows the response time of running the TWDTW (CPU) algorithm in the distributed environment. As shown in Figure 4a, in this experiment, we can see that the cluster size can influence the response time and the system proposed for distributed processing achieves an almost linear scale up to the number of servers and the input dataset size. With the increase of the cluster, the response time has a decreasing trend. This scalable behavior was made possible by distributing the time series data among the servers in the cluster allowing a distributed and parallel processing of the TWDTW (CPU).

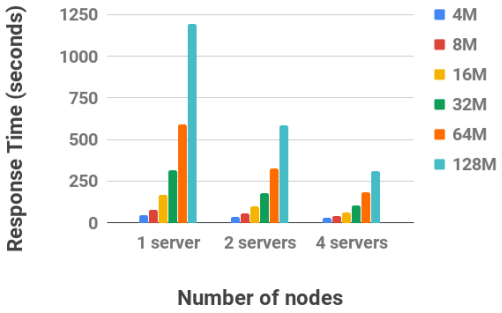
Figure 4b shows the speedup values indicating that the algorithm speedup performance was good on running the TWDTW (CPU) algorithm. Speedup is better for larger datasets, since the task scheduling and the cost of communication between machines, inherent in distributed processing, ends up having a big impact on the total response time. With 128 million time series, the speedup almost reaches the linear scale because the processing cost becomes dominant over the communication cost and task scheduling.

Figure 5 displays the size-up test for the TWDTW (CPU) algorithm in a distributed environment. The size-up shows how the running time increases with the input size for a fixed number of cores [Japkowicz and Stefanowski 2016]. For the size-up test, the computational time was taken by keeping the number of servers unchanged and varying the dataset size from 4 million to 512 million of time series. The running time increased linearly with the input size, which is ideal for size-up test scenarios [Japkowicz and Stefanowski 2016].

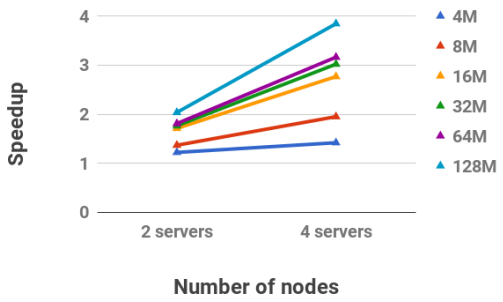
5.3 Discussion

The SP-TWDTW GPU implementation proved to be a promising solution for processing high temporal resolution data, with a speedup of 10 times over the CPU TWDTW implementation and almost 11 times faster than it for high spatial resolution data. The TWDTW was implemented in C++ to be able to compare fairly with SP-TWDTW since the C++ language allows for better performance. SP-TWDTW execution time was 246 times faster than the CPU TWDTW in R, making it a good

[6] https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
[7] `https://www.cloudera.com/documentation/enterprise/latest/topics/admin_spark_tuning.html`



(a) Response time increasing the cluster size from 1 to 4 servers and the number of time series from 4 millions (4M) to 128 millions (128M).



(b) Speedup varying the number of time series from 4 millions (4M) to 128 millions (128M).

Fig. 4. Scalability running the TWDTW (CPU) algorithm and increasing the number of servers and the number of time series.

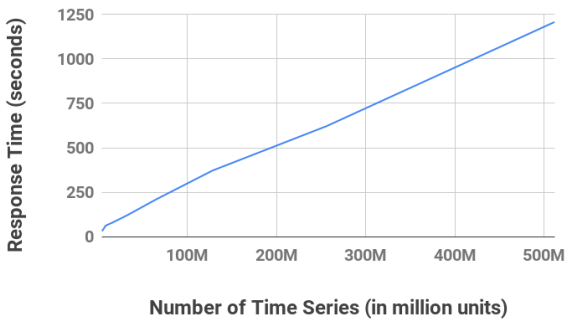


Fig. 5. Size-up performance evaluation running the TWDTW (CPU) algorithm with 4 servers and increasing the number of time series.

alternative to the time series analysis in the Remote Sensing Big Data era. The SP-TWDTW also presented better overall accuracy than the traditional TWDTW.

In general, the overall accuracy difference between TWDTW and SP-TWDTW was not so large because the TWDTW already has great accuracy with the data of this geographic region. Perhaps, in other geographic areas and other classes of land use the results may be different. This analysis will be done in future work. However, in general, SP-TWDTW had greater accuracy than TWDTW even in this region with these land classes. Regarding the spatial interpolation methods, increasing the spatial weight for values higher than 0.4 brings on lower accuracy to the SP-TWDTW.

The scale-up, speed up, and size-up tests showed that the solution proposed in this paper was able to perform distributed and parallel time series analysis in Remote Sensing Big Data context. It successfully ran the TWDTW algorithm with up to 512 million time series. For datasets with less than 16 million time series, it didn't achieve a good speedup because communication cost and task synchronization were dominant over the processing cost. Increasing the dataset size the processing cost has become dominant and TWDTW in a distributed environment achieved almost linear speedup.

6. CONCLUSION

In the class of complex computational problems, the time series analysis is one of the problems with increasing demand for computing power [Rakthanmanon et al. 2013]. Due to the complexity of the algorithms and the large volume of data to be processed, new parallel and distributed approaches are necessary. The TWDTW algorithm has been highlighted as one of the best solutions found in the literature to perform the time series analysis of remote sensing images, but it does not explore parallel architectures. Exploiting the fine-grained parallelism on TWDTW is a challenge because the data dependence on it is high.

This work presented a solution to processing large volumes of time series data by exploring massive parallel architectures. The solution, SP-TWDTW (Spatial Parallel TWDTW), uses the Manycore (GPU) architectures and addresses the TWDTW data dependency on fine-grained parallelism, besides managing the CPU and GPU memory space. Moreover, the SP-TWDTW analyzes the spatial and temporal dimensions using interpolation methods. To support the Remote Sensing Big Data scenario, a new architecture was introduced to process time series analysis in a distributed environment.

The GPU SP-TWDTW implementation has a speedup of 11 times over the CPU TWDTW implementation for high temporal resolution data and almost 11 times for high spatial resolution data. Using spatial interpolation methods, SP-TWDTW was able to increase the accuracy of TWDTW for land use mapping in the Amazon region. The solution proposed in this work for distributed and parallel processing of time series analysis in Remote Sensing Big Data context proved to be promising. It achieved almost linear speed up, size-up and scale up for large datasets to run the TWDTW algorithm in a cluster of computers.

In future work, other accuracy tests with larger datasets in other geographic regions using satellite images with higher resolution than MODIS will be performed to compare the TWDTW and SP-TWDTW methods more accurately. In addition, the distributed time series analysis will be evaluated in a cluster of computers, where each server will have a GPU locally to verify the horizontal and vertical scalability of the platform in the execution of the SP-TWDTW algorithm. The ETL module will be evaluated to ensure the performance for ingesting data into the storage. The Query Executor will also be evaluated to verify its ability to efficiently filter spatially and temporally the time series remote sensing data before sending this data to Core Executor. The SP-TWDTW will be applied to other methods for time series classification and analysis.

Acknowledgement

This work was funded by Companhia Paranaense de Energia (Copel), P&D ANEEL-Copel Distribuição, project number 2866-04842017.

REFERENCES

Bagnall, A., Lines, J., Bostrom, A., Large, J., and Keogh, E. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery* 31 (3): 606–660, 2017.

Battude, M., Al Bitar, A., Morin, D., Cros, J., Huc, M., Sicre, C. M., Le Dantec, V., and Demarez, V. Estimating maize biomass and yield over large areas using high spatial and temporal resolution sentinel-2 like remote sensing data. *Remote Sensing of Environment* vol. 184, pp. 668–681, 2016.

Bégué, A., Arvor, D., Bellon, B., Betbeder, J., De Abelleyra, D., PD Ferraz, R., Lebourgeois, V., Lelong, C., Simões, M., and R Verón, S. Remote sensing and cropping practices: A review. *Remote Sensing* 10 (1): 99, 2018.

Camara, G., Assis, L. F., Ribeiro, G., Ferreira, K. R., Llapa, E., and 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*. ACM, New York, NY, USA, pp. 1–6, 2016.

Chebbi, I., Boulila, W., Mellouli, N., Lamolle, M., and Farah, I. A comparison of big remote sensing data processing with hadoop mapreduce and spark. In *2018 4th International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*. IEEE, Sousse, Tunisia, pp. 1–4, 2018.

Chi, M., Plaza, A., Benediktsson, J. A., Sun, Z., Shen, J., and Zhu, Y. Big data for remote sensing: Challenges and opportunities. *Proceedings of the IEEE* 104 (11): 2207–2219, 2016.

Costa, W. S., Fonseca, L. M. G., Korting, T. S., Simões, M., do Nascimento Bendini, H., and Souza, R. C. M. Segmentation of optical remote sensing images for detecting homogeneous regions in space and time. *Revista Brasileira de Cartografia* 70 (5): 1779–1801, 2018.

Cressie, N. and Wikle, C. K. *Statistics for spatio-temporal data*. John Wiley & Sons, 2015.

de Oliveira, S. S. T., M. L. Pascoal, L., Ferreira, L., de Castro Cardoso, M., Bueno, E., Vagner, J., and Martins, W. S. Sp-twdtw: A new parallel algorithm for spatio-temporal analysis of remote sensing images. In *XIX Brazilian Symposium on Geoinformatics*. GeoInfo, Campina Grande, PB, Brasil, pp. 46–57, 2018.

Gómez, C., White, J. C., and Wulder, M. A. Characterizing the state and processes of change in a dynamic forest environment using hierarchical spatio-temporal segmentation. *Remote Sensing of Environment* 115 (7): 1665–1679, 2011.

Huang, W., Meng, L., Zhang, D., and 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.

Jamali, S., Jönsson, P., Eklundh, L., Ardö, J., and Seaquist, J. Detecting changes in vegetation trends using time series segmentation. *Remote Sensing of Environment* vol. 156, pp. 182–195, 2015.

Japkowicz, N. and Stefanowski, J. *Big Data Analysis: New Algorithms for a New Society*. Springer, 2016.

João Jr, M., Sena, A. C., and Rebello, V. E. Implementação e avaliação de técnicas de paralelização no algoritmo de hirschberg para sistemas multicore. In *Anais do XVIII Simpósio em Sistemas Computacionais de Alto Desempenho*. SBC, Porto Alegre, RS, Brasil, 2017.

Li, J. and Heap, A. D. Spatial interpolation methods applied in the environmental sciences: A review. *Environmental Modelling & Software* vol. 53, pp. 173–189, 2014.

Liu, P., Di, L., Du, Q., and Wang, L. Remote sensing big data: theory, methods and applications, 2018.

Lu, M., Chen, J., Tang, H., Rao, Y., Yang, P., and Wu, W. Land cover change detection by integrating object-based data blending model of landsat and modis. *Remote Sensing of Environment* vol. 184, pp. 374–386, 2016.

Ma, Y., Wu, H., Wang, L., Huang, B., Ranjan, R., Zomaya, A., and Jie, W. Remote sensing big data computing: Challenges and opportunities. *Future Generation Computer Systems* vol. 51, pp. 47–60, 2015.

Maus, V., Câmara, G., Cartaxo, R., Sanchez, A., Ramos, F. M., and 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.

Maus, V., Câmara, G., Appel, M., and Pebesma, E. dtwsat: Time-weighted dynamic time warping for satellite image time series analysis in r. *Journal of Statistical Software, Articles* 88 (5): 1–31, 2019.

Mitas, L. and Mitasova, H. Spatial interpolation. *Geographical information systems: principles, techniques, management and applications* vol. 1, pp. 481–492, 1999.

Olofsson, P., Foody, G. M., Stehman, S. V., and Woodcock, C. E. Making better use of accuracy data in land change studies: Estimating accuracy and area and quantifying uncertainty using stratified estimation. *Remote Sensing of Environment* vol. 129, pp. 122–131, 2013.

Olson, M. Hadoop: Scalable, flexible data storage and analysis. *IQT Quart* 1 (3): 14–18, 01, 2010.

Petitjean, F., Inglada, J., and Gançarski, P. Satellite image time series analysis under time warping. *IEEE Transactions on Geoscience and Remote Sensing* 50 (8): 3081–3095, 2012.

Petitjean, F. and Weber, J. Efficient satellite image time series analysis under time warping. *Ieee geoscience and remote sensing letters* 11 (6): 1143–1147, 2014.

Qu, J. J., Gao, W., Kafatos, M., Murphy, R. E., and Salomonson, V. V. *Earth Science Satellite Remote Sensing: Vol. 2: Data, Computational Processing, and Tools*. Springer, 2006.

Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., and Keogh, E. Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 7 (3): 10, 2013.

Ranjan, R. Streaming big data processing in datacenter clouds. *IEEE Cloud Computing* 1 (1): 78–83, 2014.

Shabib, A., Narang, A., Niddodi, C. P., Das, M., Pradeep, R., Shenoy, V., Auradkar, P., Vignesh, T., and Sitaram, D. Parallelization of searching and mining time series data using dynamic time warping. In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, Kochi, India, pp. 343–348, 2015.

Shepard, D. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*. ACM, New York, NY, USA, pp. 517–524, 1968.

Stein, M. L. *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media, 2012.

Vatsavai, R. R. *Machine Learning Algorithms for Spatio-temporal Data Mining.* Ph.D. thesis, University of Minnesota, Minneapolis, MN, USA, 2008. AAI3338985.

Verbesselt, J., Hyndman, R., Newnham, G., and Culvenor, D. Detecting trend and seasonal changes in satellite image time series. *Remote sensing of Environment* 114 (1): 106–115, 2010.

White, T. *Hadoop: The Definitive Guide.* O'Reilly Media, Inc., 2009.

Xiao, L., Zheng, Y., Tang, W., Yao, G., and Ruan, L. Parallelizing dynamic time warping algorithm using prefix computations on gpu. In *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on.* IEEE, Zhangjiajie Shi, China, pp. 294–299, 2013.

Xu, F., Lin, Y., Huang, J., Wu, D., Shi, H., Song, J., and Li, Y. Big data driven mobile traffic understanding and forecasting: A time series approach. *IEEE transactions on services computing* 9 (5): 796–805, 2016.

Yin, H., Yang, S., Ma, S., Liu, F., and Chen, Z. A novel parallel scheme for fast similarity search in large time series. *China Communications* 12 (2): 129–140, 2015.

Yu, J., Zhang, Z., and Sarwat, M. Spatial data management in apache spark: the geospark perspective and beyond. *GeoInformatica* 22 (4): 1–42, 2018.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster computing with working sets. *HotCloud* 10 (10-10): 95, 2010.

Zhu, H., Gu, Z., Zhao, H., Chen, K., Li, C.-T., and He, L. Developing a pattern discovery method in time series data and its gpu acceleration. *Big Data Mining and Analytics* 1 (4): 266–283, 2018.