BitFun: Fast Answers to Queries with Tunable Functions in Geospatial Array DBMS

http://bitfun.gis.land/vldb2020/

Ramon Antonio Rodriges Zalipynis
National Research University Higher School of Economics, Moscow, Russia
rodriges@gis.land

ABSTRACT

Geospatial array DBMSs handle big georeferenced arrays. Due to the geospatial data peculiarities, many queries have tunable parameters with values not known in advance: users gradually tune them until they get a satisfactory result. This generates a series of queries with slightly different structures and very similar outputs. Modern array DBMSs spend the same efforts to answer each such query. BitFun provides novel bitmap indexing strategies to continuously re-index arrays during queries with similar mathematical functions. It can be up to 8× faster than computing the results from scratch. We describe BitFun and offer lessons on real-world geospatial data, related to real practical tasks. A lesson involves tuning a math function parameter while the rich web GUI details the indexing process and query execution. Conference attendees will appreciate BitFun approaches, its performance, and learn its internals via fascinating lessons.

PVLDB Reference Format:
R.A. Rodriges Zalipynis. BitFun: Fast Answers to Queries with Tunable Functions in Geospatial Array DBMS. *PVLDB*, 13(12): 2909-2912, 2020.
DOI: https://doi.org/10.14778/3415478.3415506

1. INTRODUCTION

Geospatial array DBMSs are experiencing the surge of R&D due to the rapid growth of geospatial array volumes. For example, DigitalGlobe, a commercial satellite imagery vendor, collects about 80 TB/day and maintains an archive of over 100 PB of satellite imagery in AWS [5]. Dozens of parameterized math functions are used daily for vital tasks including urban planning, agriculture monitoring, forestry control, and rapid-response for disaster relief [2, 14], fig. 1.

As a typical example, consider Soil-Adjusted Vegetation Index (savi, fig. 1) which aims to minimize soil brightness influence. L is a soil fudge factor varying from 0 to 1 depending on the soil [14]. The user may tune L many times to find appropriate savi values for a given area of interest. nir and r are 2-d arrays with intensities of reflected solar radiation in the near-infrared and visible red spectra respectively.

Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3415478.3415506

All state-of-the-art array DBMS techniques will treat two subsequent queries of computing savi with slightly different L values (e.g. 0.7 and 0.8) as two distinct queries. This will trigger computing savi for the new L value from scratch despite the fact that usually only a small fraction of the resulting array will differ significantly from the previous result. We waste I/O and compute resources by reading two arrays completely from storage and recomputing each resulting cell.

Often savi and similar functions are used for classification. For example, the range of possible savi values is split into the intervals, e.g. $[0, 0.25)$, $[0.25, 0.5)$, $[0.5, 1)$, ... and each interval is assigned a label. If a savi value fits into a predefined interval, its label will be assigned to the resulting array cell instead of the savi value, providing additional space for optimizations: if the user slightly tunes L, the majority of the resulting cells will remain the same.

BitFun is a ChronosDB [12, 11] component which provides novel bitmap indexing techniques for queries with tunable math functions and novel, space-efficient hierarchical bitmap index structure to support tunable indexing.

BitFun tackles an important class of queries not explicitly considered before in the context of array DBMS: tunable queries. BitFun explicitly focuses on the fact of tunability. Moreover, none of the modern array DBMS is equipped with capabilities of indexing array cells: SciDB [4], TileDB [8], RasDaMan [10], PostGIS [9], and Oracle Spatial [7]. Emerging array indexing techniques do not take into account tunable scenarios [3, 13]. As indexing is a relatively new topic for array DBMSs, existing array index structures do not fit well to tunable array DBMS workloads [13].

We invite conference attendees to take interactive lessons related to real-world practical tasks on multispectral satellite imagery. To succeed, the user should tune a math function parameter to get an appropriate 2-d map. The user will interactively receive hints during a lesson. The audience will use a rich Web GUI to live BitFun/ChronosDB deployment in the Cloud.

The web interface features (1) the editor with syntax highlight to compose and submit queries, (2) 2-d and 3-d charts displaying the indexing process, index properties, and query answering details, (3) visual components to facilitate expression tuning, (4) visual components for lesson guidance, (5) interactive map with input data and query results.

Attendees will (a) appreciate the speed of BitFun query answering due to novel indexing techniques, (b) investigate the hierarchical index structure, contents, and learn indexing process insights, (c) extend their knowledge of array DBMSs and real-world geospatial arrays.

$$\text{SAVI} = \frac{\text{NIR} - \text{R}}{\text{NIR} + \text{R} + \boxed{L}} \times (1 + \boxed{L}) \quad \text{ARVI}^* = \frac{\text{NIR} - (\text{R} - \boxed{\gamma}(\text{B} - \text{R}))}{\text{NIR} + (\text{R} - \boxed{\gamma}(\text{B} - \text{R}))} \quad \text{AVI} = tan^{-1}\left[\frac{\boxed{\lambda_3} - \boxed{\lambda_2}}{\boxed{\lambda_2}} \frac{1}{(\text{NIR} - \text{R})}\right] + tan^{-1}\left[\frac{\boxed{\lambda_2} - \boxed{\lambda_1}}{\boxed{\lambda_2}} \frac{1}{(\text{G} - \text{R})}\right]$$

$$\text{GARI}^* = \frac{\text{NIR} - [\text{G} - \boxed{\gamma}(\text{B} - \text{R})]}{\text{NIR} + [\text{G} - \boxed{\gamma}(\text{B} - \text{R})]} \quad \text{TSAVI} = \frac{\boxed{a}(\text{NIR} - \boxed{a}\text{R} - \text{B})}{\text{R} + \boxed{a}\text{NIR} - \boxed{a}\boxed{b}} \quad \text{TVI} = \sqrt{\frac{\text{NIR} - \text{R}}{\text{NIR} + \text{R}} + 0.5\boxed{L}} \quad \text{PVI} = \frac{cos(\boxed{a}) \times \text{NIR}}{-sin(\boxed{a}) \times \text{R}} \quad \text{WDRVI} = \frac{\boxed{a}\text{NIR} - \text{R}}{\boxed{a}\text{NIR} + \text{R}}$$

Figure 1: Examples of Typical Math Functions with Tunable Parameters in Earth Remote Sensing Domain [14]

2. BITFUN OVERVIEW

BitFun is a ChronosDB [12, 11] component and is written in Java. Java lacks symbolic computing libraries, so BitFun interacts with SymPy to find derivatives, solve equations, simplify expressions, etc. Jython and similar tools provide limited SymPy functionality, so we developed a Web server in Python to submit formulas, related parameters and get the required SymPy output via RESTful API. BitFun has to evaluate expressions hundreds of millions of times. Hence, BitFun ingests the SymPy output into Java expressions and runs a compiler to get Java bytecode, serving as part of indexing routines. It is soon compiled into machine code, running significantly faster compared to evaluating expressions in symbolic form. BitFun is summarized in fig. 3.

2.1 Tunable Function Indexing Techniques

Definitions. Let τ be a tunable parameter, $f(\tau)$ be a differentiable function (possibly non-linear). BitFun currently supports 3 types of tunable queries that involve $f(\tau)$: (1) computing $f(\tau)$ values, (2) classification of $f(\tau)$, (3) inequality evaluation $f(\tau) < const$, fig. 3b. We focus on (1) due to space constraints, but the BitFun approach should be clear in general. Let us take SAVI as a running example.

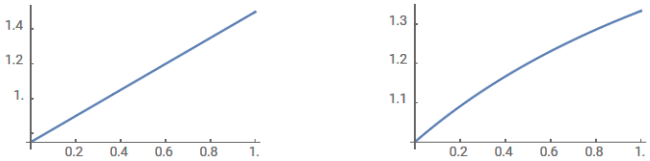
The reasoning in this section is often applicable to any differentiable function, even outside the geospatial domain.

Parameter valid range $[\tau_{min}, \tau_{max}]$ is often fixed and known in advance, e.g. $L \in [0, 1]$ for SAVI. We ask users to specify valid ranges for all tunable parameters as `--param L:0:1:0.01` (`0.01` is the tune step), fig. 4. This greatly facilitates making assumptions about the function behavior.

Computing $f(\tau)$ values. An important observation is that, if we fix NIR and R values, SAVI becomes a function of only one variable, L. This means that each resulting SAVI array cell is a function of one variable. Three most important questions, in the context of tuning L, arise: (1) What is the form of each function (linear, quadratic, etc.)? (2) How similar are the functions for all cells? (3) Among functions for millions of (NIR, R) cell pairs, how many distinct functions are there?

If we look closer at the SAVI function, we notice that, in the vast majority of cases (for most combinations of NIR and R values), it is a line, fig. 2a. Hence, SAVI can be quite closely approximated by a much simpler expression: linear fit $aL + b$. A notable fact is that the new expression does not depend on the original NIR and R values and can be computed only based on the L value.

If the number of all possible linear approximations is relatively small, we may efficiently *index* the data: create an *index array* with the shape fitting the output SAVI array, where each cell is the `ID` of the respective linear fit. The index array will take much less space compared to NIR and R arrays, since the number of `ID`s will be small. Hence, once the index is ready, we will save I/O time during the next query by reading a small index vs. two large arrays.



(a) NIR = 700, R = 100 (b) NIR = 2, R = 0

Figure 2: SAVI values for the whole range of $L \in [0, 1]$

In general, SAVI is not an affine function. Indeed, there is a tiny fraction of NIR and R values for which SAVI slightly exhibits non-linear behavior: when NIR and R values are very close to the range of L, and its contribution is noticeable in the denominator, fig. 2b. However, even in this case, it is possible to fit a rather good linear approximation.

We ask the user to specify a precision for calculating all function values using the syntax `--precision` Υ. For instance, if $\Upsilon = 0.01$ and $|f(\tau_1) - f(\tau_2)| \leqslant 0.01$, we assume $f(\tau_1) = f(\tau_2)$. For all examples in fig. 1, 0.01 is a very high precision, sufficient for many real-world applications.

A linear fit may not suit well for ARVI or similar functions. We try polynomial regression of degree d as follows. We try a linear fit first. If the largest vertical distance between the line and the function, within the `range`, exceeds Υ, we try a quadratic fit and check the distance again. We try until $d = 3$ or $d/1.5$ exceeds the number of input arrays (otherwise it is unlikely to index this cell in a space-efficient way). We mark cells as `UN` if we did not find a good fit, section 2.2.

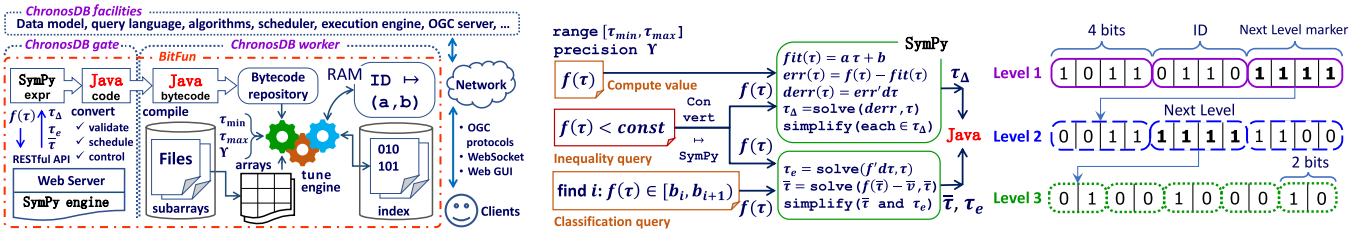
To construct a linear fit, we need two pairs of values of L and SAVI to compute a and b. We must already have one of the pairs since we need to compute SAVI for each array cell. The second pair could be obtained by picking any L in its value range. To find the maximum distance (`error`), we should solve $[\text{SAVI}(L) - (aL + b)]'dL = 0$. After solving and simplifying the solution, we obtain several roots: $L_{1,2} = \pm(\sqrt{a(\text{NIR} - \text{R})(\text{NIR} + \text{R} - 1)} \mp a(\text{NIR} + \text{R}))/a$.

Hence, for each cell, we need to compute L_k and find `error`$=\max_\tau |\text{SAVI}(\tau) - (a\tau + b)|$ for $\tau \in \{\tau_{min}, \tau_{max}\} \cup L_R$, where $L_R = \{L_k : L_k \in [\tau_{min}, \tau_{max}]\}$, $k \in \{1, 2\}$, $\tau_{min} = 0$, $\tau_{max} = 1$ for SAVI. Similarly, we find `error` for other differentiable functions and higher order polynomials.

We assign a unique `ID` $= (\bar{a}, \bar{b})$ to $\forall f = aL + b$ by converting a and b from a floating point to integer representation as $\bar{a}, \bar{b} = a\times, b \times 10^{|\lg(\Upsilon)|}$. We index {(`ID`, Freq)} with our bitmap structure (Freq is the frequency of `ID` occurrence). We keep `ID`s in RAM waiting for the next query.

Functions may take more than one tunable parameter. Often the user tunes only one parameter at a time. In this case we treat such functions as if it is a function of one variable. We do not index functions in other scenarios.

*Input reflectances can be corrected for the molecular effects



(a) BitFun Architecture (b) Index Preparation Workflow (c) Bitmap index example

Figure 3: BitFun architecture, code generation workflow, and hierarchical index examples

2.2 Hierarchical Bitmap Index

Motivation. Real-world geospatial arrays do not typically contain random data. SAVI, for a typical Landsat 8 satellite scene of $8000 \times 8000 = 64 \times 10^6$ cells, with rather diverse contents, including urban areas, forests, farmlands, and water, can be indexed only by ≈ 650 unique linear fits. Moreover, only a half of the lines may approximate 99% of cells and exhibit exponential distribution of usage frequency. Given these facts, we designed a novel, space-efficient hierarchical bitmap index which is fast to create and read.

Index Structure. Let $E = \{e_1, e_2, \ldots, e_n\}$ be the set of objects (e.g. functions), $A\langle l_1, l_2\rangle : E$ is a 2-d array to be indexed (the array notation is from [12]), and $\text{Freq}(e_j)$ is the count of e_j in A. The index is hierarchical with at most K levels, fig. 3c. Level i (L_i) is a 1-d array where each cell is a fixed-length code of m_i bits. The length of level 1 array (the number of m_1 bit cells) equals to the $l_1 \times l_2$. We can always perform N-d↔1-d coordinate conversion between A and level 1 arrays since we know l_1 and l_2. Hence, we can get the level 1 code that corresponds to $A[x_1, x_2]$ in $O(1)$.

L_i may hold $2^{m_i} - 1$ unique codes since the code $2^{m_i} - 1$ (1 in each bit of m_i-bit value) indicates the lookup a level down. The length of L_{i+1} is the number of cells in L_i equal to $2^{m_i} - 1$. The index uses additional codes to support two special values: `NA` for A cells with missing values and `UN` to mark A cells which we did not index (we should refer to the source data to compute the respective A cell).

For example, the structure in fig. 3c indexes A as follows. The first 4 bits 1011 at level 1 map to e_{12} ($1011_2 = 11_{10}$) assuming that $\text{Freq}(e_j) \geqslant \text{Freq}(e_{j+1})$. Bits 1111 take us to level 2 to find out the object index ($1111_2 = 2^{m_1} - 1$). This is the first such combination at level 1, so we should retrieve the first m_2 bits 00 01 10 11 from level 2 to continue. These bits index possibly less frequent object e_{16} ($2^{m_1} - 2 + 1 = 15$).

We arrive at a lower level from level 1 which length equals the size of A. Hence, we can calculate the position of e_{16} in A in $O(1)$. Object e_{16} is indexed by $m_1 + m_2 = 8$ bits 1111 00 01 10 11 as it is associated with level 2. The object index on the third level is coded by $4 + 4 + 2 = 10$ bits: 1111 11 11 11 ?? (the first $m_1 + m_2 = 8$ bits are 1).

BitFun stores each index level as a set of GeoTIFF files. This allows BitFun to keep the georeference data inside the index and enables interoperability: we can use software tools and libraries supporting GeoTIFF to work with the index.

Index Advantages. The index uses less than $l_1 \times l_2 \times \log_2 |E|$ bits to code all possible objects. In practice, it is possible to build an index which takes 4× less space than the input arrays. This considerably saves I/O, especially with limited IOPS (e.g. in the Cloud). However, unlike existing variable-length encoding techniques, the index enables BitFun to efficiently perform array operations directly on the index, often without reading the source data (here we discuss only random access due to space constraints).

Random access. We can get a code from level 1 in $O(1)$ for (x_1, x_2) as noted earlier since codes are fixed-sized. We have to perform a single sequential scan to count the number of "Next Level" markers (`NL`) at each level. Let S_i be a 1-d array such that $|S_i| = |L_i|$, $S_i[y]$ equals the position of `NL` in L_{i-1} for $L_i[y]$. We expect $|L_{i+1}| \ll |L_i|$ due to exponential distribution of $\text{Freq}(e_j)$. Hence, temporary S_i arrays yield a small storage footprint and accesses to lower S_i are very rare. Index random access time is $O(1 + \sum_{i=2}^{K} \log_2 |S_i|)$, which is small in practice due to the described data properties.

Index Optimality. Suppose that Algorithm 1 builds the index. Due to space constraints, we omit Algorithm 1 and the proof of Theorem 1. In practice, it takes about 75 ms for Algorithm 1 to complete for $M = \{2, 4, 8, 16\}$ and $K = 4$, and 10 ms when the Java Virtual Machine is warmed up.

THEOREM 1 (SPACE OPTIMALITY). *Algorithm 1 builds an index structure with K levels and guarantees the minimal average codeword length among all symbol-to-symbol codes for alphabet $E = \{e_1, e_2, \ldots, e_n\}$, probabilities $\Pr(e_j)$, and $m_1, m_2, \ldots, m_K$ bits per symbol code at levels $1, 2, \ldots, K$.*

2.3 BitFun Web Interface

Please, refer to the Introduction for the interface overview.

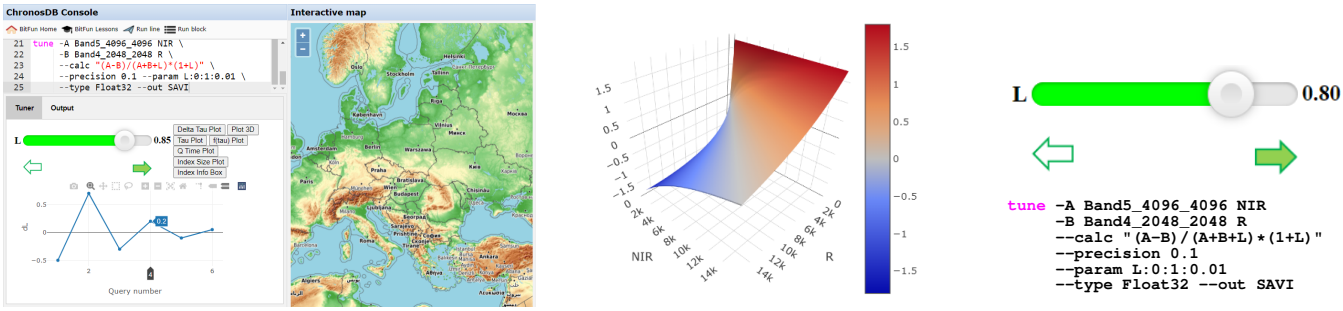
BitFun lessons are described on the BitFun homepage http://bitfun.gis.land. We designed scripts with the respective preparatory and `tune` commands for each lesson, fig. 4a. Attendees can edit scripts using ChronosDB commands. Users can open a lesson script from the BitFun homepage.

Tune slider facilitates easy τ tuning, fig. 4c. Let Q be a query number. Each new value of τ increases Q by 1, triggers submitting the query with new τ and updating the map and other components that depend on τ and Q.

Tune hints. BitFun analyses query results and gives tips on how to tune τ to successfully solve the lesson task: decrease/increase τ relatively to the current τ value. BitFun highlights the left/right arrow to reflect its hint, fig. 4c.

Plots. Let us denote a 2-d plot with abscissa x and ordinate y by $(x \mapsto y)$. Plots are located under the tune slider. To avoid clutter, the user can show/hide a plot with a respective button. All plots are interactive: the user can pan & zoom a plot, download it as PNG, rotate (3-d plots), watch values under the mouse cursor during its move.

Plot $(Q \mapsto \tau_\Delta)$, where $\tau_\Delta \in [\tau_{min} - \tau_{max}, \tau_{max} - \tau_{min}]$, tracks the history of tuning τ. Positive/negative τ_Δ values indicate that the user increased/decreased τ for query Q compared to query $Q - 1$. This plot is useful for tracking



(a) BitFun Web GUI (b) SAVI 3D plot ($L = 0.8$) (c) Tune Slider & `tune` command

Figure 4: BitFun Web GUI and its Components

the pattern of gradually finding the appropriate value of τ: τ_Δ will be large for the first queries and become tiny for the last, fig. 4a. Similarly, plot $(Q \mapsto \tau)$ tracks the value of τ.

If f has 2 input arrays A and B, $(A, B \mapsto f)$ plots f in 3-d for all possible (A, B) value pairs and current τ, fig. 4b. This is useful for estimating tune results and (A, B) by f.

"Q Time Plot" shows $(Q \mapsto T)$, where T is the time spent for answering query № Q. The plot illustrates significant performance improvement, using the proposed indexing techniques, compared to computing results from scratch.

Index info box shows the number of approximating models $|E|$, the number of levels K selected by Algorithm 1, bits per level (m_i), percent of models coded by level i, index volume. The index is treated as an ordinary ChronosDB array: the user can access any index portion via WMTS and investigate it in any software that supports GeoTIFF files.

Interactive map is multilayer, with pan&zoom enabled. ChronosDB delivers layer data via the standard WMTS protocol. Attendees can view layers in the BitFun Web GUI and any other software supporting WMTS, e.g. QGIS. BitFun plots $(\tau \mapsto \overline{f})$ for the map point which the user clicked on with the mouse. Fast plotting illustrates lightning-fast random access for the novel index structure.

3. BITFUN LESSONS

We have described in the Introduction and section 2.3 how the audience will experience BitFun. Here we briefly outline the lessons detailed in the BitFun Web GUI.

Illustrative Data. Landsat Program is the longest continuous space-based record of Earth's land running from 1972 onwards. We will provide 2-d arrays (NIR, R, B, etc.) ingested beforehand from the respective Landsat 8 scenes.

Water Lesson *River flood mapping*. **Goal**: illustrate fast evaluation of $f(\tau) < const$ due to novel indexing techniques. **Area**: Arkansas river basin. Remote sensing data is widely used in practice to forecast river floods, assess the damage caused, identify flood prone areas, select places for protective dams [2]. NDVI values close to zero or negative represent zones with the presence of water. **The task** is to quickly create an accurate water mask by tuning τ in $f(\tau) = \text{NDVI} - \tau < 0$. As a reference, the user will see the RGB map, resulting mask, and the set of points of two colors (ground truth) located in flooded and non-flooded areas [6].

Agriculture Lesson *Crop yield prediction*. **Goal**: illustrate fast computing of $f(\tau)$ values due to novel indexing techniques. We will feed $f(\tau)$ values directly to a crop yield model. **Area**: Saudi Arabia. SAVI is used for arid regions with sparse vegetation and exposed soil surfaces since NDVI is very sensitive to soil brightness [14]. **The task** is to quickly estimate crop yield by tuning L (by finding its appropriate value), fig. 1. As a reference, the user will see the crop yield map (tons per hectare), the RGB map, and the actual yield for a land parcel/irrigation pivot (ground truth) [1].

Takeaway insights: (1) real-world geospatial data is not random, so tunable math functions on the data can be efficiently (and sometimes surprisingly) indexed by specialized techniques and data structures, (2) BitFun is optimized for evaluating queries containing tunable math functions, (3) BitFun significantly accelerates query evaluation compared to computing results from scratch, (4) BitFun utilizes a novel bitmap indexing data structure which can often be used alone to answer queries with tunable math functions.

4. REFERENCES

[1] Al-Gaadi et al. Prediction of potato crop yield using precision agriculture techniques. *Plos one*, 11(9), 2016.

[2] https://learn.arcgis.com/en/arcgis-imagery-book/.

[3] S. Blanas et al. Parallel data analysis directly on scientific file formats. In *ACM SIGMOD 2014*.

[4] P. Cudré-Mauroux, H. Kimura, K.-T. Lim, et al. A demonstration of SciDB: A science-oriented DBMS. *PVLDB*, 2(2):1534–1537, 2009.

[5] DigitalGlobe 80 TB/day. https://youtu.be/mkKkSRIxU8M.

[6] NASA EO. https://earthobservatory.nasa.gov/images/145108/floods-in-the-arkansas-river-watershed.

[7] Oracle spatial and graph. https://www.oracle.com/database/technologies/spatialandgraph.html.

[8] S. Papadopoulos et al. The TileDB array data storage manager. *PVLDB*, 10(4):349–360, 2016.

[9] PostGIS home. http://postgis.net/.

[10] RasDaMan home. http://rasdaman.org/.

[11] R. A. Rodriges Zalipynis. ChronosDB in action: Manage, process, and visualize big geospatial arrays in the Cloud. In *ACM SIGMOD 2019*.

[12] R. A. Rodriges Zalipynis. ChronosDB: Distributed, file based, geospatial array DBMS. *PVLDB*, 11(10):1247–1261, 2018.

[13] H. Xing et al. Accelerating array joining with integrated value-index. In *SSDBM 2019*.

[14] J. Xue and B. Su. Significant remote sensing vegetation indices: A review of developments and applications. *Journal of Sensors*, 2017.