Retrospective Satellite Data in the Cloud: An Array DBMS Approach

Ramon Antonio Rodriges Zalipynis(✉), Anton Bryukhov, and Evgeniy Pozdeev

National Research University Higher School of Economics, Moscow, Russia
rodriges@wikience.org, asbryukhov@gmail.com, jonnypozdeev@gmail.com

Abstract. Earth remote sensing has always been a source of "big" data. Satellite data have inspired the development of "*array*" DBMS. An array DBMS processes N-dimensional (N-d) arrays utilizing a declarative query style to simplify raster data management and processing. However, raster data are traditionally stored in files, not in databases. Respective command line tools have long been developed to process these files. Most tools are feature-rich and free but optimized for a single machine. The approach of partially delegating in situ raster data processing to such tools has been recently proposed. The approach includes a new formal N-d array data model to abstract from the files and the tools as well as new distributed algorithms based on the model. This paper extends the approach with a new algorithm for the reshaping (tiling) of N-d arrays. The algorithm physically reorganizes the storage layout of N-d arrays to obtain an order of magnitude speedup. The extended approach outperforms SciDB up to 28× on retrospective Landsat data – one of the most typical and popular kind of satellite imagery. SciDB is the only freely available distributed array DBMS to date. Experiments were carried out on an 8-nodes cluster in Microsoft Azure Cloud.

Keywords: ChronosServer · SciDB · Raster Data · Cloud Computing · Remote Sensing · Array DBMS · Command Line Tools · Landsat

1 Introduction

Earth remote sensing is increasingly becoming a data-rich, practically important and commercially attractive domain. The most prominent example is the Landsat Program – the longest continuous space-based record of Earth's land in existence. The Program lasts from 1972 onwards and has accumulated over 6.8×10^6 scenes mostly in GeoTIFF files ($\approx$ 6 PB in total) [8]. Landsat data are so popular that Amazon and Google provide Landsat scenes via commercial clouds [5]. The number of practical Landsat applications is rapidly growing [7]. A retrospective time series of Landsat scenes for a particular area is of great importance since it makes it possible to track area changes that were happening over the past decades.

The file-centric model of raster data storage resulted in a broad set of highly optimized raster file formats. For example, GeoTIFF represents an effort by over

160 different companies and organizations to establish interchange format for georeferenced raster imagery [6]. Decades of development and feedback resulted in numerous feature-rich, elaborate, free and quality-assured tools for processing raster files. For example, NCO (NetCDF Operators) are are under development since about 1995 [10], GDAL (Geospatial Data Abstraction Library) has over one million lines of code made by hundreds contributors [4].

An array DBMS is one of the tools to streamline raster data processing. The idea of partially delegating raster data processing to existing command line tools was first presented and proved to outperform SciDB on NetCDF data 3× to 193× on a single machine [16] and up to 1000× running both SciDB and ChronosServer on a computer cluster (Microsoft Azure Cloud) [17]. ChronosServer is the system into which the delegation ability is being integrated [15].

The formal array model and formal distributed algorithms are given in [17]. The new two-level data model was designed to uniformly represent diverse raster data types and formats, take into account the distributed context, and be independent of the underlying raster file formats at the same time [17].

The main goal of this paper is to advance the proposed delegation approach and to show its exceptional suitability for satellite data processing. ChronosServer outperforms SciDB on raw Landsat scenes. To obtain an order of magnitude speedup, a physical reorganization of the storage layout of 2-d Landsat scenes is carried out by cutting and joining them into 3-d arrays. This case is generalized and a generic reshaping algorithm is proposed to transform a set of N-d arrays with arbitrary shapes to a set of M-d arrays with a fixed shape, where $N - M \in \mathbb{Z}$. The new algorithm is useful on a "data cooking" stage to spend some time to prepare data and make further algorithms to run much faster.

In summary, the major contributions of this paper are (i) the generic N-d reshaping algorithm and (ii) an experimental evaluation of ChronosServer and SciDB on retrospective Landsat 8 data in the Cloud.

The rest of the paper is organized as follows. For the sake of completeness, section 2 describes the array model, dataset model, and ChronosServer architecture [17]. Section 3 gives generic distributed algorithms for in situ processing of arbitrary N-d arrays. The algorithms are refined in order to treat NetCDF and GeoTIFF formats and delegate portions of work to NCO and GDAL tools. Section 4 presents the N-d reshaping algorithm. Section 5 gives the performance evaluation. Section 6 reviews the related work. Section 7 concludes the paper.

2 ChronosServer

2.1 ChronosServer Multidimensional Array Model

In this paper, an N-dimensional array (N-d array) is the mapping $A : D_1 \times D_2 \times \cdots \times D_N \mapsto \mathbb{T}$, where $N > 0$, $D_i = [0, l_i) \subset \mathbb{Z}$, $0 < l_i$ is a finite integer, and $\mathbb{T}$ is a numeric type (to be specific about value ranges, size in bytes, precision, etc., a C++ type according to ISO/IEC 14882 can be taken). The l_i is said to be the *size* or *length* of ith dimension (in this paper, $i \in [1, N] \subset \mathbb{Z}$).

Let us denote the N-d array by

$$A\langle l_1, l_2, \ldots, l_N\rangle : \mathbb{T} \tag{1}$$

By $l_1 \times l_2 \times \cdots \times l_N$ denote the *shape* of A, by $|A|$ denote the *size* of A such that $|A| = \prod_i l_i$. A *cell* or *element* value of A with integer indexes $(x_1, x_2, \ldots, x_N)$ is referred to as $A[x_1, x_2, \ldots, x_N]$, where $x_i \in D_i$. Each cell value of A is of type $\mathbb{T}$. The array may be initialized after its definition by enumerating the values of the cells. For example, the following defines and initializes a 2-d array of integers: $A\langle 2, 2\rangle : \text{int} = \{\{1, 2\}, \{3, 4\}\}$. In this example, $A[0, 0] = 1$, $A[1, 0] = 3$, $|A| = 4$, and the shape of A is 2×2.

Indexes x_i are optionally mapped to specific values of ith dimension by *coordinate* arrays $A.d_i\langle l_i\rangle : \mathbb{T}_i$, where $\mathbb{T}_i$ is a totally ordered set, and $d_i[j] < d_i[j+1]$ for all $j \in D_i$. In this case, A is defined as

$$A(d_1, d_2, \ldots, d_N) : \mathbb{T} \tag{2}$$

A *hyperslab* $A' \sqsubseteq A$ is an N-d subarray of A. The hyperslab A' is defined by the notation

$$A[b_1 : e_1, \ldots, b_N : e_N] = A'(d'_1, \ldots, d'_N) \tag{3}$$

where $b_i, e_i \in \mathbb{Z}$, $0 \leqslant b_i \leqslant e_i < l_i$, $d'_i = d_i[b_i : e_i]$, $|d'_i| = e_i - b_i + 1$, and for all $y_i \in [0, e_i - b_i]$ the following holds

$$A'[y_1, \ldots, y_N] = A[y_1 + b_1, \ldots, y_N + b_N] \tag{4a}$$

$$d'_i[y_i] = d_i[y_i + b_i] \tag{4b}$$

Equations (4a) and (4b) state that A and A' have a common coordinate subspace over which cell values of A and A' coincide.

2.2 ChronosServer Datasets

A *dataset* $\mathbb{D} = (A, M, P)$ contains a *user-* or *higher-level* array $A(d_1, \ldots, d_N) : \mathbb{T}$ and the set of *system-* or *lower-level* arrays $P = \{(A_k, B_k, E_k, M_k, node_k)\}$, where $A_k \sqsubseteq A$, $k \in \mathbb{N}$, $node_k$ is an identifier of the cluster node storing array A_k, M_k is metadata for A_k, $B\langle N\rangle : \text{int} = \{b_1, b_2, \ldots, b_N\}$, $E\langle N\rangle : \text{int} = \{e_1, e_2, \ldots, e_N\}$ such that $A_k = A[b_1 : e_1, \ldots, b_N : e_N]$. A user-level array is never materialized and stored explicitly: an operation with A is mapped to a sequence of operations with respective arrays A_k. Let us call a user-level array and a system-level array an array and a subarray respectively for short. A dataset also contains metadata $M = \{(key, val)\}$, where key is a string and val is a string or a number. Dataset metadata includes two types of information: general dataset properties (name, description, contacts, etc.) and metadata valid for all $p \in P$ (array data type $\mathbb{T}$, storage format, etc.). For example, $M = \{(name = \text{“Landsat 8 Band 1”}), (type = \text{int16}), (format = \text{GeoTIFF})\}$. Let us refer to an element in a tuple $p = (A_k, B_k, \ldots) \in P$ as $p.A$ for A_k, $p.B$ for B_k, etc. Example of a subarray metadata $p.M = \{(key, val)\}$ is $p.M = \{(date = \text{“2016-Aug-08”}, bounding\_box = \text{“WKT}(\ldots)\text{”}, projection = \text{“EPSG:32637”})\}$.

2.3 ChronosServer Architecture

ChronosServer runs on a computer cluster of commodity hardware. Files are distributed among cluster nodes without changing their names and formats. A file is always stored entirely on a node in contrast to parallel or distributed file systems. Workers run on each node and are responsible for data processing. One Gate at a dedicated node receives client queries and coordinates workers. A file may be replicated on several nodes for fault tolerance and load balancing.

Gate stores metadata for all datasets and subarrays. Consider a dataset $\mathbb{D} = (A, M, P)$. Arrays $A.d_i$ and elements of $\forall p \in P$ except $p.A$ are stored on Gate. In practice, array axes usually have coordinates such that $A.d_i[j] = start + j \times step$, where $j \in [0, |A.d_i|) \subset \mathbb{N}$, $start, step \in \mathbb{R}$. Only $|A.d_i|$, $start$, and $step$ values have to be usually stored. ChronosServer array model merit is that it has been designed to be generic as much as possible but allowing to establish 1:1 mapping of a $p \in P$ to a physical dataset file at the same time.

Upon startup workers connect to Gate and receive a list of all available datasets and file naming rules. Workers scan their local filesystems to discover datasets and create $p.M$, $p.B$, $p.E$ by parsing file names or reading file metadata. Workers transmit to Gate the described information.

A user-level array may have a *virtual* dimension. Values for virtual dimensions are taken from the subarrays metadata. For example, Landsat files are 2-d arrays $A(lat, lon)$ without a temporal axis. Virtual axis "time" in $A(time, lat, lon)$ may contain scenes acquisition dates extracted from the file names. This makes it possible to treat a set of Landsat scenes as a 3-d array.

3 Array Operations

3.1 Aggregation

The aggregate of an N-d array $A(d_1, d_2, \ldots, d_N) : \mathbb{T}$ over axis d_1 is the $(N-1)$-d array $A_{aggr}(d_2, \ldots, d_N) : \mathbb{T}$ such that $A_{aggr}[x_2, \ldots, x_N] = f_{aggr}(cells(A[0 : |d_1| - 1, x_2, \ldots, x_N]))$, where $x_2, \ldots, x_N$ are valid integer indexes, $f_{aggr} : T \mapsto w$ is an aggregation function, T is a multiset of values from $\mathbb{T}$, $w \in \mathbb{T}$, $cells : A' \mapsto T$ is the multiset of all cell values of an array $A' \sqsubseteq A$.

Algorithm 1 performs aggregation of system-level arrays.

Algorithm 1 Distributed in situ array aggregation with delegation to an external command line tool (procedure AGGREGATE is executed on workers).

Input: wid is the identifier of the worker performing final aggregation

```
1: procedure AGGREGATE(𝔻, f_aggr, wid)          ▷ 𝔻 is a dataset, see section 2.2
2:     aggregate all p ∈ P residing on this worker into p'          ▷ DELEGATION
3:     if the id of this worker equals to wid then
4:         accept subarrays from other workers: P_aggr
5:         aggregate p' and all p ∈ P_aggr into p_aggr
6:         report success to Gate
7:     else send p' to worker with id = wid
```

The generic aggregation algorithm 1 is based on the dataset model from section 2.2 and takes into account that system-level arrays may overlap and cover the N-d space irregularly (e.g., scenes following a riverbed). Also, Landsat scenes for the same path and row may be shifted relatively to each other (it is hard to capture precisely the same area each time).

Algorithm 1 is designed for $f_{aggr} \in \{max, min, sum\}$. The basic idea is that workers aggregate in parallel all subarrays residing locally into one subarray and send it to a worker calculating the final result. Calculating *mean* is reduced to calculating the *sum* and dividing the result onto the number of participating subarrays. The Gate is responsible for calculating this number and sending it to the worker performing final aggregation (not shown in the algorithm).

In section 5, we always set *wid* to the largest possible. Array p' on line 2 may grow large in volume and require splitting. We leave this case for future work. Line 2 is highlighted with light gray to accent the work being delegated to an external tool: `gdal_calc.py` for GeoTIFF format and an NCO tool (`ncra`, `ncwa`, or `ncap2` depending on file structure) for NetCDF format.

3.2 Chunking

Chunking is the process of partitioning original array onto a set of smaller subarrays called chunks. Chunks are autonomous, possibly compressed subarrays (hyperslabs) with contiguous storage layout. Given chunk shape $c_1 \times \cdots \times c_N$ and an N-d array $A(d_1, \ldots, d_N) : \mathbb{T}$, the *exact chunking* operation reorganizes cells in array A such that all cells of A with coordinates $(x_1, \ldots, x_N)$ and $(y_1, \ldots, y_N)$ belong to the same chunk if $x_i \operatorname{div} c_i = y_i \operatorname{div} c_i$ for all i. Due to space constraints, please, find the illustration and benefits of chunking in [16].

The exact chunking of an array may lead to data movement between files and cluster nodes. However, in practice the condition $c_i \ll |A.d_i|$ usually holds. This translates to $c_i \ll |p.A.d_i|$ for $\forall p \in P$ (in practice, raster data are already shipped in wisely cut files satisfying this condition). For example, in climate modeling it is common to split a time series with hourly time step onto files storing yearly or monthly data.

A good practical approach is to do inexact user-level array chunking and exact independent chunking of its subarrays. More chunks will smaller shapes than the given one will appear. However, the fraction of such small chunks will be negligible and they will not influence significantly the I/O performance. Note that if $|A.d_i| \bmod c_i \neq 0$, then even the exact chunking of a user-level array is not possible leading to a certain amount of chunks with smaller shapes.

In practice, inexact chunking is even more desirable in many cases: it is much faster and more consistent than the exact chunking. Recall that files under ChronosServer control are directly accessible by a user and any other software. Consider the climate modeling example given above. In this case, it is inconsistent to have a perfectly chunked file named "2015.nc" and supposed to store data for year 2015 but with extra grids from the next and/or previous years.

Chunking is delegated to `gdal_translate` (GDAL) and `ncks` (NCO) for GeoTIFF and NetCDF file formats respectively.

4 Generic Reshaping Algorithm

This section presents an algorithm to reshape system-level arrays. The algorithm is useful on a "data cooking" stage to spend some time to reshape the subarrays to speedup further raster operations.

It is costly to aggregate and/or hyperslab large number of files. For example, aggregating a time series of 2-d scenes makes the hidden asymptotic constants quite noticeable. Changing position of a virtual axis during reshaping requires complex data moves between files (reshaping operation is defined in [17]). Chunking along a virtual axis $A.v_i$ could be implemented as co-locating $p_1, p_2 \in P$ on a single machine such that $p_1.v_i[x]$ and $p_2.v_i[y]$ are in the same chunk. This "virtual" chunking will not speedup the I/O as chunking of a single physical file.

The reshaping algorithm overcomes these limitations. For example, initial shape of raw Landsat files (system-level arrays) is $1 \times L_1 \times L_2$ ($time \times lat \times lon$), where $L_1 \approx L_2 \approx 8000$. Reshaping subarrays, say, to $5 \times L_1/4 \times L_2/4$ will accelerate aggregation and hyperslabbing (hyperslabbing is defined in [17]). In this case, virtual *time* dimension will become a regular physical dimension and will explicitly present in dataset files. This will make it possible to delegate dimension permutation of Landsat 8 scenes (please, refer to [17] for details on dimension permutation) as well as chunking (section 3.2) to an external tool.

Algorithm 2 takes as input a set of N-d subarrays P with arbitrary shapes and produces a set of subarrays P' such that $\forall p' \in P'$ has shape $s_1 \times s_2 \times \cdots \times s_N$ (except border cases) and sides of p' are parallel to the coordinate axes.

Given $M \neq N$, algorithm 2 reshapes subarrays from N-d to M-d form using virtual axes supported by ChronosServer data model. A virtual axis can be made a physical one to get subarrays with greater physical dimensionality $M > N$ (the case with Landsat scenes described above). An axis can be made virtual and deleted from the files if the axis has a unit length in all resulting subarrays. This will produce subarrays with lower physical dimensionality, i.e. $M < N$.

The basic idea is to cut each $p \in P$ onto smaller pieces $P' = \{p' : p' \sqsubseteq p\}$, assign each piece a key, and merge all pieces with the same key into a single, new system-level array. For $x \in \mathbb{N}$, $lag(x)$ is defined below.

$$lag(x) = \begin{cases} 0, & \text{if } x = 0 \\ x - 1, & \text{if } x \geqslant 1 \end{cases} \tag{5}$$

The Reshape function of algorithm 2 implements the idea outlined above. Each subarray is cut independently by Cut-One procedure, line 3. Set $\mathbb{K}$ collects N-tuples which are N-d keys of cut pieces collected in $\mathbb{C}$. Pieces with the same key are merged into a single subarray on lines 4–7. Line 6 is highlighted with light gray since merging of files is possible to delegate to an external tool.

Algorithm 2 is best illustrated on a 2-d case. Consider a 2-d array $A(lat, lon)$, fig. 1a. Array A has shape 10×15 and consists of 6 subarrays separated by thick blue lines. The reshaping produces 2-d subarrays with shape 3×3, $s_1 = s_2 = 3$. Resulting subarrays P' are separated with dashed red lines. The hatched area marks one of the resulting subarrays $A[3\!:\!5, 3\!:\!5]$.

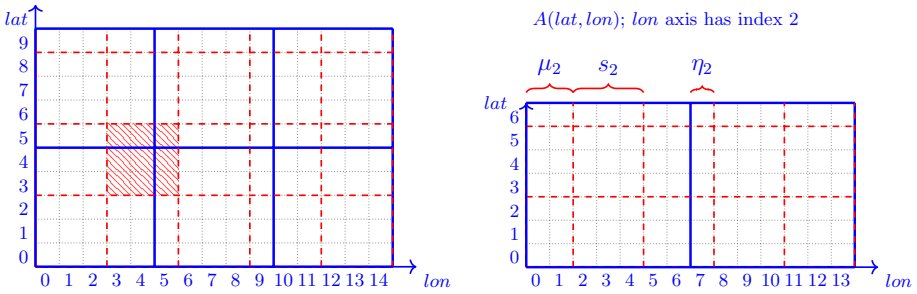

Fig. 1. Reshaping system-level arrays.

Algorithm 2 Generic Reshaping.

Input: $\mathbb{D} = (A, M, P)$ ▷ Dataset, section 2.2
$S = (s_1, s_2, \ldots, s_N)$ ▷ Target shape for arrays from P is $s_1 \times s_2 \times \cdots \times s_N$
$\mu = (\mu_1, \mu_2, \ldots, \mu_N)$ ▷ Shift, fig. 1b
Output: $\mathbb{D}' = (A, M, P')$ ▷ A is the same, $\forall p' \in P'$ shape is $s_1 \times s_2 \times \cdots \times s_N$
▷ (except border cases)
Require: $s_i \in [1, \Theta_{axis}] \subset \mathbb{N}$, ▷ Resulting pieces are not too large
$\prod_{i=1}^{N} s_i \leqslant \Theta_{shape}$, $\mu_i \in [0, s_i - 1] \subset \mathbb{N}$

function RESHAPE($\mathbb{D}, S, \mu$)
  $\mathbb{C} \leftarrow \{\}$ and $\mathbb{K} \leftarrow \{\}$ ▷ $\mathbb{C}$: line 22, $\mathbb{K}$: line 23 of **procedure** CUT-ONE
  for each $p \in P$ **do** CUT-ONE($\mathbb{C}, \mathbb{K}, p, S, \mu$)
  for each $key \in \mathbb{K}$ **do**
    $C \leftarrow \{a \in \mathbb{C} : a.key = key\}$
    $p_{new} \leftarrow$ merge all $a \in C$ given $a.B_{new}$ and $a.E_{new}$ ▷ DELEGATION
    $P' \leftarrow P' \cup \{p_{new}\}$
  return $\mathbb{D}' = (A, M, P')$

procedure CUT-ONE($\mathbb{C}, \mathbb{K}, p, S, \mu$)
  if $b_i \geqslant \mu_i$ **then** ▷ $b_i = p.B[i], e_i = p.E[i]$
    $\eta_i \leftarrow s_i - ((b_i - \mu_i) \bmod s_i)$
  else
    $\eta_i \leftarrow \mu_i - b_i$
  if $\eta_i = 0$ **then** $\eta_i \leftarrow s_i$
  if $\eta_i > e_i - b_i$ **then**
    $x_i \leftarrow 1$
  else
    $x_i \leftarrow (e_i - b_i + 1 - \eta_i) \operatorname{div} s_i + sgn((e_i - b_i + 1 - \eta_i) \bmod s_i) + 1$
  for each $y_i \in [0, x_i - 1] \subset \mathbb{N}$ **do**
    $b'_i \leftarrow s_i \times lag(y_i) + \eta_i \times sgn(y_i)$ ▷ $0 \leqslant b'_i \leqslant e'_i < |p.d_i|$
    $e'_i \leftarrow min(s_i \times y_i + \eta_i - 1, e_i - b_i)$ ▷ b'_i, e'_i are local indexes within p
    $p' \leftarrow p[b'_1 : e'_1, b'_2 : e'_2, \ldots, b'_N : e'_N]$ ▷ DELEGATION
    if $b'_i + b_i > \mu_i$ **then**
      $k_i \leftarrow (b'_i + b_i - \mu_i) \operatorname{div} s_i + sgn(\mu_i)$
    else
      $k_i \leftarrow 0$ ▷ $k_i \geqslant 0$
    $key \leftarrow (k_1, k_2, \ldots, k_N)$ ▷ $key \subset \mathbb{Z}^N_{\geqslant 0}$
    $B_{new}\langle N \rangle : \text{int} = \{b_1 + b'_1, b_2 + b'_2, \ldots, b_N + b'_N\}$ ▷ global indexes for p'
    $E_{new}\langle N \rangle : \text{int} = \{e_1 + e'_1, e_2 + e'_2, \ldots, e_N + e'_N\}$ ▷ within $A.d_i$
    $\mathbb{C} \leftarrow \mathbb{C} \cup \{(p', key, B_{new}, E_{new})\}$ ▷ $\mathbb{C}$ is the set of cuts from all $p \in P$
    $\mathbb{K} \leftarrow \mathbb{K} \cup \{key\}$ ▷ $\mathbb{K}$ is the set of all generated merge keys

Subarray $A[5:9, 5:9]$ will be cut on 9 pieces separated by the red lines: $A[5, 5]$, $A[5, 6:8]$, $A[5, 9]$ and so on. Each of them will be assigned a 2-d key. Resulting subarray $A[3:5, 3:5]$ will be assembled from 4 pieces cut from $A[5:9, 0:4]$, $A[5:9, 5:9]$, $A[0:4, 0:4]$, and $A[0:4, 5:9]$. These 4 pieces are $A[5, 5]$, $A[5, 3:4]$, $A[3:4, 3:4]$, and $A[3:4, 5]$. All 4 pieces will have key $(1, 1)$.

Algorithm 2 accepts a shift μ_i for the ith axis in order to start cutting with an "indent", fig. 1b. Resulting subarrays containing a cell with a zero coordinate will have shape $s'_1 \times s'_2 \times \cdots \times s'_N$, where $s'_i = \mu_i$ if $\mu_i \neq 0$; $s'_i = s_i$ otherwise.

Lines 2–6 of the Cut-One procedure calculate η_i which is a "local indent" within the current subarray p along the ith axis: η_i cells from p along the ith axis must go to the resulting subarray contained in p and other source subarrays bordering with p. Then, x_i is found which is the number of pieces to be cut along the ith axis. Thus, $\prod_i x_i$ is the total number of pieces to be cut from p. Loop on lines 11–23 cuts one piece at a time. Lines 12–13 find indexes within p to cut a piece on line 14 by the delegation to an external tool. The piece is assigned the key which is an N-d 0-based index of the resulting subarray to which the piece belongs. The ith tuple element of an N-d key is the index along the ith axis.

5 Performance Evaluation

Microsoft Azure Cloud was taken for the experiments. Azure cluster creation, scaling up and down with given network parameters, number of virtual machines, etc. was fully automated using Java Azure SDK [17]. The latest version of Ubuntu Linux on which SciDB 16.9 runs is 14.04 LTS. We rented standard D2 v2 machines with 2 CPU cores (Intel Xeon E5-2673 v3 (Haswell) 2.4 GHz), 7 GB RAM, 100 GB local SSD drive (4 virtual data disks), max 4×500 IOPS. Although Azure states the disk to be SSD, after the creation of such a disk Azure displays the disk to be a standard HDD disk backed by a magnetic drive.

We selected band 1 from nine Landsat 8 scenes for path 190 and row 31 (≈585 MB in total) such that the cloud cover percent for the majority of scenes is less than 20%. We evaluated the latest SciDB version 16.9 released in November, 2016. We have written a Java program that converts GeoTIFF files to CSV files to feed the latter to SciDB. To date, this is the only way to import an external file into SciDB 16.9. We aligned all scenes in UTM coordinates since they are slightly shifted relatively to each other and imported the scenes into a SciDB array with shape $9 \times 7971 \times 7941$. We filled the cells with `NULL` values for areas in some scenes that appeared in the result of extension of that scenes during their alignment. Import time of one Landsat 8 scene into SciDB takes about **40 minutes** on a cloud node, not local machine. Therefore, we imported all 9 scenes on a local computer, exported the resulting SciDB array into a file of proprietary SciDB binary format, and copied that file in the Cloud when needed (SciDB imports data from its proprietary format much faster).

cluster in order to deploy it on a cluster. SciDB is mostly written in C++, parameters used: 0 redundancy, 2 instances per machine, 5 execution and prefetch threads, 1 prefetch queue size, 1 operator threads, 1024 MB array cache, etc.

ChronosServer has 100% Java code, ran one worker per node, OracleJDK 1.8.0_-111 64 bit, max heap size 978 MB (-Xmx). We used NCO and GDAL tools available from the standard Ubuntu 14.04 repository. NCO v4.4.2, last modified 2014/02/17. GDAL v1.10.1, released 2013/08/26.

We have evaluated cold query runs (a query is executed for the first time). Every runtime reported is the average of 3 runtimes of the same query. Respective OS commands were issued to free `pagecache`, `dentries` and `inodes` each time before executing a cold query to prevent data caching at various OS levels. ChronosServer benefits from native OS caching and is much faster during hot runs when the same query is executed for the second time on the same data. There is no significant runtime difference between cold and hot SciDB runs.

To increase the data volume and to avoid waiting for loading more scenes, we attached SciDB array to itself to get the time dimension of size 18. We could not attach the resulting array to itself again. We tried in many ways including array import with different chunk shapes but SciDB had been always failing with `not enough memory` error. As of 29-May-2017, we did not receive any feedback from SciDB developers on this issue [11]. The same errors prevented us to measure SciDB chunking performance (section 3.2). Chunking is one of the slowest SciDB queries even on small arrays [16]. We replicated 9 scenes to get 18 scenes and placed them by 2-3 on each node for ChronosServer.

Table 1 summarizes ChronosServer performance on raw and preprocessed Landsat scenes as well as SciDB performance with automatically chosen chunk shape for the SciDB array. Given array $A(time, lat, lon)$, "cut $m \times n$" means extracting a hyperslab $A[0 : |time| - 1, x_1 : x_1 + m, x_2 : x_2 + m]$, where x_1, x_2 are random indexes for array A. Line "Time series" reports hyperslabbing a time series for a single point $A[0 : |time| - 1, x_1, x_2]$. Hyperslabbing is an extraction of a hyperslab from an array. "Chunk" lines report chunking of A (for raw data, *time* is a virtual axis and its chunk size equals to 1).

Table 1. Performance for 18 scenes, 8 cluster nodes

Operation	Time, sec.			Ratio, SciDB/ Chronos	
	ChronosServer (raw data)	ChronosServer ("cooked" data)	SciDB		
Average	38.36	8.12	230.74	6.02	28.42
Maximum	38.83	4.56	127.71	3.29	28.00
Minimum	38.98	4.63	125.70	3.22	27.15
Cut 512×512	1.79	1.01	1.98	1.11	1.96
Cut 1024×1024	3.34	2.14	3.41	1.02	1.59
Time series	0.53	0.31	0.84	1.58	2.71
Chunk $1 \times 64 \times 64$	22.37	—	—	—	
Chunk $1 \times 128 \times 128$	22.49	—	—	—	

Table 2 shows the time for "cooking" Landsat 8 scenes for further speedup of the queries. Algorithm from section 4 is implemented in a serial mode: all 18 scenes were processed on a single node. Future work includes assigning a set of keys to a node which will merge all cut subarrays having the given keys.

Table 2. Preprocessing Landsat data (section 4): 18 scenes, 1 cluster node

Target Shape	Time, sec.	Target Shape	Time, sec.
$4 \times 512 \times 512$	410.25	$9 \times 1024 \times 1024$	216.62
$9 \times 512 \times 512$	376.32	$4 \times 4096 \times 4096$	56.87
$4 \times 1024 \times 1024$	187.41	$9 \times 4096 \times 4096$	55.45

6 Related Work

Numerous techniques exist for remote sensing data processing. This work is novel because it is in the context of array DBMS research field. Four modern raster data management trends are relevant to this paper: industrial raster data models, formal array models and algebras, in situ data processing algorithms, and raster (array) DBMS. Good survey of the algorithms is in [3]. A recent survey of array DBMS and similar systems is in [16]. It is worth mentioning SciDB [18], Oracle Spatial [12], ArcGIS IS [1], RasDaMan [14], Intel TileDB [19], and PostGIS [13].

A recent survey on the array models and algebras as well as industry standard data models is in [17]. Work [17] outlines the peculiar features and merits of ChronosServer data model. It is shown that the most popular array models and algebras can be mapped to Array Algebra [2]. Industry data models are also mappable to each other [9]. SciDB does not have a formal description of its data model. SciDB neither allows array dimensions to be of temporal or spatial types making it difficult or sometimes impossible to process many real-world datasets.

7 Conclusions

ChronosServer delegates portions of raster data processing to feature-rich and highly optimized command line tools. This makes ChronosServer to run much faster than SciDB. ChronosServer is up to 6× faster on raw Landsat 8 scenes than SciDB on its native storage (the same Landsat 8 scenes imported into SciDB). ChronosServer is up to 28× faster than SciDB after preprocessing the scenes which takes 105× to 780× less time than SciDB import.

Future work includes designing a distributed version of the reshaping algorithm proposed in this paper. It could be also beneficial to incorporate fault-tolerance during the reshaping once it will be parallelized.

Acknowledgments. This work was partially supported by Russian Foundation for Basic Research (grant №16-37-00416). We also thank anonymous reviewers for their helpful and inspiring comments.

Contributions. Rodriges: all text, figures, algorithms, ChronosServer, its data model, Azure management code, SciDB import code, experimental setup. Pozdeev: SciDB cluster deployment. Bryukhov: partial implementation of the reshaping algorithm for one machine, adapted SciDB import code to Landsat data. All authors: experiments.

References

1. ArcGIS for server — Image Extension. http://www.esri.com/software/arcgis/arcgisserver/extensions/image-extension
2. Baumann, P., Holsten, S.: A comparative analysis of array models for databases. Int. J. Database Theory Appl. 5(1), 89–120 (2012)
3. Blanas, S., Wu, K., Byna, S., Dong, B., Shoshani, A.: Parallel data analysis directly on scientific file formats. In: ACM SIGMOD 2014. pp. 385–396
4. Coverity scan: GDAL. https://scan.coverity.com/projects/gdal
5. Earth on AWS. https://aws.amazon.com/earth/
6. GeoTIFF. http://trac.osgeo.org/geotiff/
7. Landsat apps. https://aws.amazon.com/blogs/aws/start-using-landsat-on-aws/
8. Landsat project statistics. https://landsat.usgs.gov/landsat-project-statistics
9. Nativi, S., Caron, J., Domenico, B., Bigagli, L.: Unidata's common data model mapping to the ISO 19123 data model. Earth Sci. Inform. 1, 59–78 (2008)
10. NCO homepage. http://nco.sourceforge.net/
11. Not enough memory error – SciDB forum. http://forum.paradigm4.com/t/problem-with-memory-while-stacking-array/1838
12. Oracle spatial and graph. http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/index.html
13. PostGIS raster data management. http://postgis.net/docs/manual-2.2/using_raster_dataman.html
14. RasDaMan homepage. http://rasdaman.org/
15. Rodriges Zalipynis, R.A.: Chronosserver: real-time access to "native" multi-terabyte retrospective data warehouse by thousands of concurrent clients. Inform., Cybern. Comput. Eng. 14(188), 151–161 (2011)
16. Rodriges Zalipynis, R.A.: ChronosServer: Fast in situ processing of large multidimensional arrays with command line tools. In: Voevodin, V., Sobolev, S. (eds.) Supercomputing: Second Russian Supercomputing Days, RuSCDays 2016, Moscow, Russia, September 26–27, 2016, Revised Selected Papers. Communications in Computer and Information Science, vol. 687, pp. 27–40. Springer International Publishing, Cham (2016), http://dx.doi.org/10.1007/978-3-319-55669-7_3
17. Rodriges Zalipynis, R.A.: Distributed in situ processing of big raster data in the cloud. In: Perspectives of System Informatics – 11th International Andrei Ershov Informatics Conference, PSI 2017, Moscow, Russia, June 27–29, 2017, Revised Selected Papers. Lecture Notes in Computer Science, Springer (2017), in press
18. SciDB homepage. http://www.paradigm4.com/
19. TileDB. http://istc-bigdata.org/tiledb/index.html