National Research University
Higher School of Economics
(HSE University)

as a manuscript

Ramon Antonio Rodriges Zalipynis

Array (Tensor) DBMS: Theoretical Foundations, Software, and Applications

Dissertation Summary

for the purpose of obtaining academic degree
Doctor of Science in Computer Science

Moscow — 2024

Abstract

Array (Tensor) DBMSs strive to be the best systems for managing, processing, and even visualizing large multidimensional arrays (tensors). It is a young, fast-evolving, and inherently inter-disciplinary area: many core data types in diverse domains are naturally modeled by arrays (tensors). This dissertation presents fundamental theoretical, systems, and practical contributions to the area of Array (Tensor) DBMSs. Namely, we introduce two novel Research & Development directions: physical world simulations and tunable queries in Array (Tensor) DBMSs. We also propose a new formal Array (Tensor) DBMS data model, novel algorithms, approaches, architectural and implementation aspects for operating on multidimensional arrays (tensors), which can exceed the speed of state-of-the-art approaches and technologies by orders of magnitude. The significance of contributions is demonstrated across a wide variety of practical applications and real-world data. This dissertation is based on the results presented at premier international conferences in computer science: VLDB and SIGMOD.

Contents

Dissertation Title and Topic 5
Dissertation & Array (Tensor) DBMS State-of-the-Art 9
Array (Tensor) DBMSs: The Beauty and Impact 11

1 Introduction **14**
1.1 Relevance of the Dissertation Topic 14
1.2 Objectives and Goals of this Dissertation 15
1.3 Main Results 15
1.4 Publications and Probation of the Work 21
1.5 Source Code of the Software 28

2 Theoretical Foundations **29**
2.1 A New Formal Array (Tensor) DBMS Data Model 29
2.1.1 Motivation for a New Data Model 29
2.1.2 Tensors or Multidimensional Arrays 30
2.1.3 Multilevel, Distributed Datasets 32
2.2 New Distributed & In Situ Tensor Algorithms 34
2.2.1 Distributed N-d Retiling 35
2.2.2 Distributed K-Way Array (Tensor) Join 36
2.2.3 Aggregation, Chunking & Other Operations 37
2.3 Tunable Queries, Indexing & Data Structure 43
2.3.1 Introducing a New R&D Direction: Tunable Queries 43
2.3.2 Novel Tunable Function Indexing Techniques 44
2.3.3 A New and Fast Hierarchical Data Structure 45
2.4 New R&D: Simulations in Array (Tensor) DBMSs 46
2.4.1 Rationale, Shortcomings & Benefits 46
2.4.2 New Traffic Cellular Automaton (TCA) 47
2.4.3 Challenges & New Enabling Components 49
2.5 New Scalable Data Science Techniques 50
2.5.1 Array (Tensor) Mosaicking Challenges 50
2.5.2 Scaling MAD & IR-MAD 51
2.5.3 Scaling Canonical Correlation Analysis (CCA) . . . 52

3 Software: Architectural & Implementation Aspects 53
3.1 ChronosDB: An Innovative Array (Tensor) DBMS 54
3.1.1 ChronosDB Architecture & Components 54
3.1.2 Novel Tensor Management Approaches 55
3.1.3 New & Efficient Query Execution Techniques . . . 57
3.2 BitFun: Fast Answers to Tunable Queries 58
3.2.1 BitFun Architecture 58
3.2.2 Interactive User Interface 59
3.3 SimDB: First Simulations in Array (Tensor) DBMSs . . . 60
3.3.1 Novel Array (Tensor) DBMS Convolution Operator 60
3.3.2 The First Native UDF Language for Tensor DBMSs 61
3.3.3 New Scheduling, Versioning & Locking Mechanisms 62
3.4 The First Array (Tensor) DBMS Entirely in a Web Browser 64
3.4.1 Time to Operate on Tensors in Web Browsers . . . 64
3.4.2 WebArrayDB Organization 64
3.4.3 ArrayGIS: WebGIS Components 66
3.5 FastMosaic: A Novel & Scalable Mosaic Operator 67
3.5.1 End-To-End Mosaicking Workflow 67
3.5.2 Rich and Interactive GUI 68

4 Applications: Real-World Data & Use-Cases Revisited 69
4.1 Earth & Climate Data: Manage, Process & Visualize . . . 70
4.1.1 High-Performance Tensor Management & Processing 70
4.1.2 GUI & DWMTS for Array (Tensor) DBMS 73
4.2 Interactive Data Science: Quick Tensor Recomputing . . . 74
4.2.1 Water Management & Flood Mapping 74
4.2.2 Food Security & Crop Yield Prediction 75
4.2.3 Accelerated Web-Based Processing & Visualization 76
4.3 Road Traffic Simulations: A New End-To-End Approach . 78
4.3.1 Simulation Initialization & Plan Investigation . . . 78
4.3.2 Interactive Visualization & Animation 80
4.3.3 Experiencing Interoperability 80
4.4 Fast & Seamless Tensor Mosaicking: Step-By-Step 81
4.4.1 Creating a Mosaic Plan 81
4.4.2 Sampling, Execution & Heatmaps 82
4.4.3 Transformation (Normalization) 84

5 Conclusion 85

Dissertation Title and Topic

R.A. Rodriges Zalipynis is the author of ChronosDB Array (Tensor) DBMS presented at VLDB 2018 [1] and SIGMOD 2019 [2], BitFun at VLDB 2020 [3], a novel R&D direction at SIGMOD 2021 [4], a tutorial at VLDB 2021 [5], WebArrayDB & ArrayGIS [6] and SimDB [7] at VLDB 2022, and FastMosaic [8] at VLDB 2023.

These are based on a wealth of theoretical foundations and software mechanisms applicable to the area of Array (Tensor) DBMSs in general. This is because the theoretical and practical contributions of this Dissertation span a wide range of Array (Tensor) DBMS aspects and applications that originate from diverse practically important domains, including storage, management, processing, exchange, and visualization of large tensors.

Moreover, new R&D directions were first identified and tackled in this Dissertation: tunable queries and physical world simulations; this is explicitly stated in the respective publications. Finally, the publications demonstrate how the presented theoretical foundations & software mechanisms successfully address many significant challenges, including considering industrial experience, user interaction, and interoperability, as well as open numerous promising R&D opportunities.

The Dissertation Title consists of several parts related to Array (Tensor) DBMSs that are sequentially covered in the Dissertation which briefly summarizes key ideas, presented in respective articles and papers, in an easy-to-read manner. Of course, the reader can find very detailed materials in the publications, as well as high-quality videos and project homepages that usually accompany the publications. Let us elaborate on the formulation of the Dissertation Title and its reflection on the Dissertation Structure. Chapters 1 and 5 – Introduction and Conclusion, respectively. The roles of Chapters 2, 3, and 4 are outlined below.

Theoretical Foundations

This chapter **establishes** novel theoretical foundations in the field of Array (Tensor) DBMSs. We start with a new Array (Tensor) DBMS data model that serves as the basis for all other contributions. Next, we introduce new R&D directions that we identified and tackled: tunable queries and physical world simulations. Finally, we describe the core ideas behind our new and efficient distributed tensor algorithms, including multi-dimensional retiling, multi-way join of arrays (tensors), and scalable data science techniques: Canonical Correlation Analysis (CCA), Multivariate Alteration Detection (MAD), and Iteratively Re-weighted MAD.

Software

The chapter is devoted to architectural and implementation **aspects** of managing and processing multidimensional arrays (tensors) of innovative Array (Tensor) DBMSs and their components (ChronosDB, BitFun, SimDB, WebArrayDB, ArrayGIS, and FastMosaic) that make it possible to outperform state-of-the-art systems by orders of magnitude, accelerate interactive data science, run simulation models completely inside an Array (Tensor) DBMS, and perform tensor-related operations entirely inside a Web browser.

Applications

Finally, this chapter **demonstrates** the significance of our contributions across a wide range of real-world data and practical applications. This chapter also presents additional architectural and implementation aspects. Our algorithms and approaches make it possible to quickly manage, process, and visualize Climate & Earth remote sensing data. Algorithms and approaches also target fast recomputing (updates) of tensors for food security tasks and rapid response in emergency scenarios. In addition, it is possible to quickly build array (tensor) mosaics. For the first time using an Array (Tensor) DBMS, we also demonstrate the simulation of road traffic using DBMS-style array (tensor) management and interoperable data exchange.

Array (Tensor)

To date, the R&D area of Array (Tensor) DBMSs is at the stage of forming its terminological dictionary. Moreover, it is a relatively young R&D area and no commonly accepted standards have been established for array (tensor) schema, query languages, the set of supported operations (operators), and many other Array (Tensor) DBMS aspects [9, 52, 66].

As we stated earlier, Array (Tensor) DBMSs operate on multidimensional arrays (tensors): the formal definition is in section 2.1. However, here we additionally elaborate on the naming of this class of DBMSs: why do we use the word combination "Array (Tensor)"?

The history begins from TITAN [12] and PARADISE [17], one of the first database systems that specifically focused on array operations. They targeted Earth remote sensing data, as newly launched satellites challenged the data management community by generating massive amounts of data, mostly 2-dimensional and 3-dimensional arrays. At the time, this data was new to the DBMSs and fundamentally different from the other supported data types.

It was quickly realized that many core data types in numerous other domains are naturally modeled by multidimensional arrays (tensors). As 2-dimensional arrays were most common, even one of the earliest systems was called RASDAMAN, which stands for "Raster Data Manager". However, it was clear that an array database management system goes far beyond rasters. That was reflected in the names of subsequent systems, e.g., "A Multidimensional Array DBMS" [71] or "A query language for multidimensional arrays" [30].

Although the word "array" does not clearly reflect that a system can work with an array with more than 2 dimensions, "multidimensional array" becomes a too lengthy term. Even worse, it is hard to translate "Array DBMS" in an awkward-free manner into other languages. For at least these two solid reasons, the term "Array DBMS" should be reconsidered.

Today, we believe that "Tensor DBMS" best reflects the essence of a database system that manages multidimensional arrays. The trend towards using the word "tensor" is strongly supported not only by the data management community, but also across a wider research environment [45, 66]. For example, "tensors are natural multidimensional generalizations of matrices" and "by tensor we mean only an array with d indices" [45].

However, we are experiencing an intermediate period of the gradual transition to the name "Tensor DBMS". Hence, in this Dissertation, we

still use the terms "Array (Tensor) DBMS" and "array (tensor)" for clarity as to which systems and objects we refer to and to foster the transition.

The word "tensor" is increasingly used not only for an array with over two dimensions, but even for matrices ("2-d arrays" or "2-d tensors") [1]. Technically and semantically, there is little or often no difference for a state-of-the-art Array (Tensor) DBMS on how to operate on a 1-dimensional, 2-dimensional, or an N-dimensional array where $N \in \mathbb{Z}$ and $N > 2$ [66]. Therefore, we use the word combination "array (tensor)" or rarely one of these two words in our Dissertation.

Note that in our data model, a tensor is more than just an array with d indices, as it supports modeling of a wide variety of data types, including meshes, irregular grids, and others, section 2.1.

It is also worth mentioning that some researchers use the term "data cube" [10]. However, it is mostly understood as an object that can be obtained by issuing respective queries to an Array (Tensor) DBMS [66].

Regardless of the current and possible future variations in the naming of database systems that manage diverse types of multidimensional arrays, and the naming of these arrays (rasters, tensors, data cubes, etc.), the word "tensor" perfectly reflects that an array can be multidimensional, is an international term, and is widely used in the research community directly for the purpose of referring to multidimensional arrays.

Dissertation & Array (Tensor) DBMS State-of-the-Art

The history begins from TITAN [12], PARADISE [17], and RASDAMAN [48], as we have already mentioned. However, R&D in this area had been stalling until the big array (tensor) data avalanche. Consequently, advanced research on array (tensor) management has only recently started to emerge. This is why we previously noted that Array (Tensor) DBMS is still a young R&D area [9, 52, 66].

It is possible to categorize array-oriented systems into Array (Tensor) DBMSs, array (tensor) stores, engines, libraries, tools, and national initiatives (which have broader goals, but may have array systems inside), and other classes [52]. An extensive survey of such systems is in [9]. However, only CHRONOSDB [54, 55], SCIDB [15], and RASDAMAN [48] are well-known and full-fledged Array (Tensor) DBMSs [73]. Among them, CHRONOSDB is the only file based Array (Tensor) DBMS: works in situ and leverages the delegation approach, enabling multiple data management benefits, including faster data ingestion and interoperability, section 2.2.

Among Array (Tensor) DBMSs [73], only CHRONOSDB and RASDAMAN data models are formalized, while CHRONOSDB data model has a unique combination of features, section 2.1. While other in situ algorithms exist [52, 54], our new efficient algorithms and approaches, built on top of our new data model, outperform state-of-the-art approaches by orders of magnitude, section 4.1.1.

Indexing is a crucial technique in any DBMS. To date, three types of Array (Tensor) DBMS indexes exist: (1) cell value selection, (2) hyperslabbing, and (3) compute [11, 53, 76, 77]. The first two speed up selecting cells in a given value and index ranges respectively. The latter accelerates computations over arrays (tensors) [52]. The compute index type was first proposed in our work [53] and accelerates queries up to 8×, section 2.3.

Array (Tensor) DBMSs perform array (tensor) storage [26, 28, 46, 53, 66], management [80, 81, 82], processing [59], analysis [13, 14, 25], dissem-

ination [9, 55], visualization [7, 24, 55, 63], and machine learning [44, 62, 72, 73]. We identified and explored another new R&D direction in the area of Array (Tensor) DBMSs: physical world simulations entirely inside an Array (Tensor) DBMS that provides many benefits and promising R&D opportunities [56, 61]. This is explicitly noted in the publication [56].

An expressive query language is of utmost importance: for users, it is an entry point to any DBMS. Operational array (tensor) query languages include AFL, AQL [15], rasQL [9], Command Line [54], GMQL [22], and the first native UDF (User Defined Function) language that we proposed [56].

Array (Tensor) DBMSs mostly work on desktop machines, servers, or computer clusters [15, 48, 54]. We designed WebArrayDB, the first Array (Tensor) DBMS that runs entirely inside a Web browser and can accelerate array (tensor) operations over 2× compared to querying a cloud service alone. To demonstrate its capabilities, we also designed a novel Web GIS (Geographic Information System) based on WebArrayDB [63].

Certain sections also contain state-of-the-art information on Array (Tensor) DBMSs to provide complementary justifications on the novelty and impact of our contributions. It is possible to learn more about Array (Tensor) DBMSs in [9, 52, 66]. All our publications cite related work.

Array (Tensor) DBMSs: The Beauty and Impact

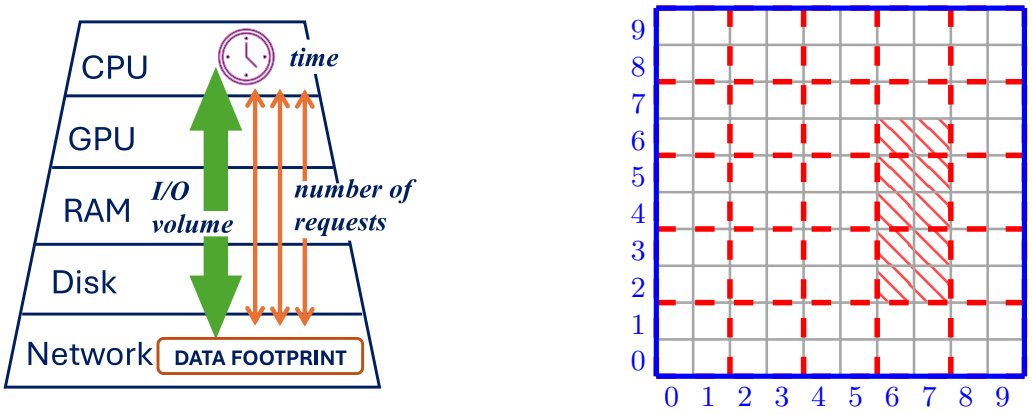
The R&D in Array (Tensor) DBMSs can be broadly categorized into two main classes: qualitative and quantitative [9, 52, 66]. Qualitative and quantitative R&D are interrelated and influence each other.

Qualitative R&D mainly focuses on providing benefits to end users that result from a DBMS-style approach to working with arrays (tensors). For example, Array (Tensor) DBMSs facilitate organizing and streamlining pipelines that involve large array (tensor) management, processing, analyzes, visualization, machine learning, simulation, and other aspects by providing dedicated query languages, data integration, automatic data integrity maintenance, powerful ETL (Extract, Transform, Load) or data ingestion, managing distributed datasets in the Cloud, automatic parallelization, interoperability, and much more in a single system.

Quantitative R&D aims to improve performance (accelerate array operation pipelines), reduce array (tensor) storage volumes, reduce I/O rate (for example, input-output requests per second in the Cloud or latency in network I/O), reduce memory requirements/footprint (operating, persistent or any other type of memory), improve scalability (e.g., process more data in a time frame or with an order of magnitude less runtime, serve more users with the same resources), and many other Array (Tensor) DBMS aspects the success criteria of which are typically expressed numerically (e.g., speed, volume, quantity).

Many types of techniques exist. For example, it may accelerate an array operation by requiring more memory or, on the contrary, provide more compact array storage at the expense of somewhat slower performance. Quantitative techniques consider all levels of the memory hierarchy, fig. 1a.

Array (Tensor) DBMSs can serve as more convenient and seamless tools for accelerating array management in diverse research and practical domains. Users can abstract from array storage, I/O, transmission, exchange,



(a) Quantitative impact (caches are not shown) (b) Array chunking

Figure 1: Illustration of quantitative Array (Tensor) DBMS approaches

and other related problems, rest assured of their efficient solutions, and build other solutions on top of Array (Tensor) DBMSs.

This Dissertation focuses on quantitative R&D. If a quantitative method or technique works inside an Array (Tensor) DBMS, the user will not only benefit from a faster array operation or more compact storage, but also from all other qualitative Array (Tensor) DBMS benefits.

What is the beauty of quantitative Array (Tensor) DBMS approaches? So far, we have listed the overall impact of Array (Tensor) DBMSs on end-users. However, the beauty of these approaches lies in the ways they reorganize array (tensor) storage, I/O, access order, and other aspects of array (tensor) management to reach the aforementioned performance goals. They provide new algorithms, data structures, execution, indexing, caching, and compression techniques, to name a few.

Data structures and indexes can be static or dynamic, built before query execution or re-built adaptively at runtime. Moreover, many Array (Tensor) DBMS aspects are completely different compared to other types of DBMSs, e.g. array joins and indexes are absolutely different from relational table joins and indexes, sections 2.2.2 and 2.3.2.

Consider a simplified example: a matrix is stored on disk in a row-wise layout (row by row). It is easy to fully read it with any programming language into the operating memory, row by row. However, what if the matrix does not fit into the memory and we read only portions of it that were specified by user queries? We also do not know which portion will be requested next, and we would like not only to answer each query quickly, but also to reduce IOPS (Input/Output Requests, or Operations, Per Second) and I/O volumes. If a query requests the dashed matrix area and

the same row-wise layout is used, we can read 5 sub-rows each containing 2 cells with 5 I/O requests, fig. 1b. However, if we chunk (group) matrix cells and treat them as I/O units (chunks or groups are separated by dashed red lines), we will need only 3 I/O requests to answer the same query. Reducing the number of I/O requests is especially important in the Cloud as they are billed separately, more in section 2.2.3

It is possible to construct a theoretically optimal (e.g., that will require the smallest runtime) data structure for some particular (narrow) query cases. However, the problem is that it is often not known a priori what array (tensor) layout is optimal because it depends on data characteristics and workload. Typically, the order, rate, scale, and types of user queries (or even pipelines consisting of a series of queries) are not 100% known and are diverse at runtime (especially in a multi-tenant environment). Many techniques consider certain array peculiarities and workloads to devise very efficient in practice, but rarely theoretically optimal, solutions in such a dynamic area as Array (Tensor) DBMS query execution.

Chapter 1

Introduction

1.1 Relevance of the Dissertation Topic

A multidimensional array is the primary data type in a wide range of domains, including climatology, ecology, and remote sensing, as well as in vital daily tasks related to agriculture, forestry, urban management, and emergencies [4, 52, 55, 60]. All of these are experiencing tremendous growth of data volumes that require efficient management, processing, analysis, and visualization, to name a few.

For example, Maxar (former Digital Globe), a commercial company, alone acquires about 80 TB of satellite imagery per day and accumulated over 100 PB of these data, i.e. arrays (tensors), in the Amazon Cloud [34]. Sentinels is a family of European satellite missions. About 203 TiB of Sentinel products are disseminated daily with total downloads of 80.5 PiB/year [67]. ECMWF (European Centre for Medium-Range Weather Forecasts) has an archive with 360 PB of main data and 363 million files. It increases by 287 TB/day and disseminates 215 TB daily [19].

Array (Tensor) DBMS is a young and fast-evolving area. Array (Tensor) DBMSs are specifically tailored to perform efficient management and other relevant operations on large multidimensional arrays (tensors). Advanced array (tensor) management research is just emerging and many R&D opportunities still "lie on the surface" [52]. Hence, this Dissertation explores various promising R&D opportunities and contributes to the development of more efficient and effective solutions to the aforementioned important practical tasks at the same time.

The common ultimate objective, or the supergoal of R&D in this area is to make Array (Tensor) DBMSs the best systems for managing large multidimensional arrays (tensors). For over 10 years, R.A. Rodriges Zalipynis has deliberately been working in different R&D directions in the

context of Array (Tensor) DBMSs to advance them in diverse directions, as well as to introduce new R&D directions for Array (Tensor) DBMSs.

Degree of Development of the Dissertation Topic

The Array (Tensor) DBMS area can be viewed as young by right: no commonly accepted standards have yet been established, architectures and implementations still to be improved and matured, and many R&D opportunities are attractive and unexplored [9, 52]. Array (Tensor) DBMSs are just expanding their presence in numerous R&D domains [58, 62, 65]. Hence, this Dissertation is a timely contribution to the development of the Array (Tensor) DBMS research and development area.

1.2 Objectives and Goals of this Dissertation

Improve the performance (e.g., reduce runtime, I/O rates, memory requirements) of multidimensional array (tensor) management, processing, analysis, and simulations based on Array (Tensor) DBMSs by developing new approaches, algorithms, and data structures for Array (Tensor) DBMS query execution, indexing, storage re-organization, and other aspects in order to provide efficient array (tensor) operations in combination with Array (Tensor) DBMS benefits (e.g., query language, DBMS-style data management, interoperability) for research and practical domains whose data are modeled by multidimensional arrays (tensors).

1.3 Main Results

Theoretical and Practical Significance of the Dissertation is confirmed by publications in leading computer science conferences and editions (section 1.4), as well as successful applications of the Dissertation results to real-world data and practically important domains (chapter 4).

The following list specifies the novelty and significance of our results. **In summary**, we achieved the following:

- **Established** new theoretical foundations in the area of Array (Tensor) DBMSs by presenting **novel**
  - Array (Tensor) DBMS data model leveraged by our new efficient algorithms, approaches, architectural and implementation

aspects that together outperform existing solutions by tens, hundreds, and thousands of times, section 2.1
  - R&D directions that open a wide range of new R&D opportunities and provide numerous benefits (e.g., DBMS-style data management, visualization, interoperability): tunable queries & simulations entirely inside Array (Tensor) DBMSs, sections 2.3 and 2.4.1
  - type (class, category) of Array (Tensor) DBMS indexes: compute index that accelerates computations over arrays (tensors); previous work speeds up value and range queries, section 2.3.1
  - the first native UDF (User Defined Function) language for Array (Tensor) DBMSs and a new convolution operator to extend their functionality, e.g. enable simulations entirely inside an Array (Tensor) DBMS for the first time, sections 3.3.1 and 3.3.2
  - efficient distributed array (tensor) algorithms that are significantly faster than state-of-the-art techniques: N-d retiling, K-way join, aggregation, resampling, reshaping, and others, section 2.2
  - efficient (in terms of runtime) indexing techniques and fast hierarchical data structure to quickly recompute tensors that are outputs of tunable mathematical functions, sections 2.3.2 and 2.3.3
  - scalable (iterate in linear time, obtain CCA canonical variables & IR-MAD transformation coefficients in the same pass over the input data) data science techniques for Array (Tensor) DBMSs: CCA (Canonical Correlation Analysis), MAD (Multivariate Alteration Detection), and IR-MAD (Iteratively Re-weighted MAD), section 2.5
- **Introduced** new architectural and implementation aspects that, in combination with our theoretical results
  - outperform state-of-the-art systems by orders of magnitude (in terms of runtime), e.g. SciDB, supervised by M. Stonebraker, an ACM Turing Award Recipient ("Nobel Prize of Computing")
  - accelerate interactive data science (e.g., exploratory analysis, tunable function computations, mosaicking) on arrays (tensors) due to accelerating respective operations, section 2.3.2
  - make it possible for the first time to run simulations entirely inside an Array (Tensor) DBMS using our new components: UDF language, convolution operator, scheduling, and others while retaining the Array (Tensor) DBMSs benefits (see below), section 3.3

  - enable running an Array (Tensor) DBMS completely inside a Web browser (for the first time) to reduce increased response times of array (tensor) operations that arise due to excessive client-server communications in state-of-the-art systems, section 3.4

- **Demonstrated** the significance of our contributions on a wide variety of real-world data and practical applications:
  - manage, process, and visualize Climate & Earth remote sensing data with exceptional performance due to our novel techniques: **by up to 1024× faster** compared to SciDB, a state-of-the-art system supervised by M. Stonebraker, an ACM Turing Award Recipient ("Nobel Prize of Computing"), section 4.1
  - recompute tensors **by up to 8× faster** and operate in Web-browsers **over 2× faster** in food safety and rapid response scenarios due to our new and efficient indexing techniques, data structure, architectural and implementation aspects, section 4.2
  - run simulations entirely in an Array (Tensor) DBMS for the first time: a new, complex road traffic model using the aforementioned components with **almost the same performance** as hand-written code combined with **DBMS-style benefits**, including tensor management, processing, and visualization, section 4.3
  - create high-quality seamless array (tensor) mosaics (that reduce the visibility of stitches) with our new scalable approach that can run an **order of magnitude faster** than the popular Python's scikit-learn library for the purpose of array mosaicking, section 4.4

We analyzed, both research and industrial, state-of-the-art array (tensor) data models, storage formats, array-oriented systems, DBMSs, query languages, schemata, algorithms, and approaches to storage, management, processing, visualization, and interoperability, identified their strengths and limitations in order to propose new and more efficient techniques [51, 52, 54, 57, 64].

In order to demonstrate the significance of our contributions, we also

- identified real-world applications that heavily rely on arrays, e.g. Climate & Earth remote sensing R&D utilizes big arrays on a daily basis
- developed auxiliary software to facilitate the ETL process (Extract, Transform, Load) of ingesting the data into software systems

- collected and imported real-world array (tensor) data into Array (Tensor) DBMSs, e.g. climate reanalysis and Earth remote sensing data
- developed software components to automatically deploy computer clusters in the Cloud, scale them as necessary
- conducted experiments on diverse real-world array (tensor) data and computer clusters of different sizes in the Cloud, analyzed, compared, and presented the results
- designed specialized interactive Web GUIs (Graphical User Interfaces) to showcase the application of Array (Tensor) DBMSs and their components to concrete real-world applications and data, sections 3.2.2, 3.4.3, 3.5.2 and 4.1.2

Methodology, Methods of the Dissertation Research. We used both theoretical and experimental methods. Our methods include, but are not limited to: analysis, synthesis, formalization, simulation, experiment, and comparison.

Author's contribution constitutes almost 100%, as R.A. Rodriges Zalipynis is the sole author of almost all publications and projects.

The contributions include, but are not limited to: ideas, vision, data model, approaches, algorithms, system (software) architecture and design, data and application choice, experiments, text, figures, publications, work presentations, home-pages, and videos. In some minor cases, the contributions are explicitly stated in the publications, section 1.4.

Main Results Submitted for Defense

- A novel Array (Tensor) DBMS Data Model with the following **unique combination of features**: (1) the treatment of arrays in multiple files arbitrarily distributed over cluster nodes as a single array, (2) formalized industrial experience to leverage it in the algorithms, (3) a rich set of data types (Gaussian, irregular grids, etc.), (4) subarray mapping to a raster file is almost 1:1 but still independent from a format, section 2.1
- New and Efficient Distributed, In Situ Array (Tensor) Algorithms (K-way join, aggregation, resampling, chunking, and other) that enable significant **performance boost** (**tens, hundreds, and thousands of times**) on a computer cluster and real-world data compared to state-of-the-art approaches, sections 2.2 and 4.1

- New and Efficient Hierarchical Data Structure and Tunable Function Indexing Techniques that provide **up to 8× acceleration** on real-world data for executing tunable Array (Tensor) DBMS queries that were first identified and tackled in this Dissertation, sections 2.3, 4.2.1 and 4.2.2
- Novel architectural and implementation aspects that efficiently **organize and demonstrate** the execution of new tunable function indexing techniques and hierarchical data structure that are **up to 8× faster** on real-world data for recomputing outputs of tunable Array (Tensor) DBMS queries, sections 3.2 and 4.2
- A novel Traffic Cellular Automaton (TCA) that challenges Array (Tensor) DBMS principles: identifies and poses new challenges whose solutions will increase the **efficiency of simulations** entirely inside Array (Tensor) DBMSs (e.g., vehicles have several properties, local transition rules operate on multiple input/output arrays, check for diverse constraints, the need for a new convolution operator, scheduling, locking, and versioning mechanisms), sections 2.4.2 and 4.3.2
- A new, scalable way to perform Canonical Correlation Analysis (CCA), a popular Data Science technique, in linear time and obtain CCA canonical variables & IR-MAD transformation coefficients in the same pass over the input data (the MOSAIC operator) that can run an **order of magnitude faster** than the popular Python's scikit-learn library in the context of the Data Ingestion phase in Array (Tensor) DBMSs, sections 2.5, 3.5 and 4.4
- Novel implementation aspects that make it possible to efficiently **organize and demonstrate** the execution of scalable Data Science techniques on real-world data: Multivariate Alteration Detection (MAD), Iteratively Re-weighted MAD (IR-MAD), and Canonical Correlation Analysis (CCA) that possesses the aforementioned properties (runtime, memory, outputs), sections 3.5 and 4.4
- Novel architectural and implementation aspects of ChronosDB, an innovative Array (Tensor) DBMS, that provide efficient array (tensor) management, processing, and visualization; ChronosDB **outperforms SciDB by up to 75×** on average. ChronosDB is always faster and can **outperform SciDB by up to 1024×**. SciDB is supervised by M. Stonebraker, an ACM Turing Award Recipient ("Nobel Prize of Computing"), sections 3.1.1 and 4.1.2

- Novel Array (Tensor) Management Approaches and Distributed Query Execution Techniques for Array (Tensor) DBMSs that operate with array (tensor) data in situ and organize efficient execution of the algorithms proposed in this Dissertation **by up to 1024× faster** on a computer cluster and real-world data compared to state-of-the-art, sections 3.1.2, 3.1.3 and 4.1.1

- A new convolution operator for Array (Tensor) DBMSs and its implementation aspects that, unlike conventional convolution operators, supplies a kernel several input windows and allows a kernel to modify an arbitrary number of cells throughout multiple output windows **to support efficient simulations** in Array (Tensor) DBMSs, sections 3.3.1 and 4.3.1

- **The First Native** UDF (User Defined Function) Language for Array (Tensor) DBMSs that, unlike DBMS query languages and general-purpose programming languages, makes it possible for the first time to express simulation logic code for Array (Tensor) DBMSs which explicitly enables the use of efficient native UDF execution facilities in Array (Tensor) DBMSs, sections 3.3.2 and 4.3

- New Scheduling, Versioning, and Locking Mechanisms for Array (Tensor) DBMSs that make it possible for the first time to efficiently run simulations in Array (Tensor) DBMSs with **performance competitive to hand-written code**, but with all benefits to users that provides an Array (Tensor) DBMS, sections 3.3.3 and 4.3

- Novel implementation aspects of SimDB and ChronosDB (e.g., workflow, Traffic Cellular Automaton implementation, initialization, proactive simulation plan investigation, interactive animation) that make it possible to **organize and demonstrate** efficient simulations entirely inside an Array (Tensor) DBMS for the first time, section 4.3

- Novel architectural and implementation aspects of WebArrayDB, **the first** Array (Tensor) DBMS that runs **entirely in a Web** browser & ArrayGIS, innovative Web GIS (Geographic Information System) based on WebArrayDB; together they can be **over 2× faster** on real-world data compared to querying only a popular Cloud service for disseminating and processing arrays (tensors), sections 3.4 and 4.2.3

1.4 Publications and Probation of the Work

This dissertation is based on the following publications.

PVLDB – is an open-access **journal ranked Q1**, see Scimago: **top 15** in Computer Science as of May 2023. Each accepted article is offered a presentation slot at the next available VLDB conference: vldb.org/2021

VLDB – is a premier annual international conference for data management and database researchers, vendors, practitioners, application developers, and users; ranked **CORE A*** (the highest) by the CORE rankings

SIGMOD – is a premier international conference on data management, databases, and data structures (**CORE A***); e.g., B-tree, R-tree, and RAID arrays were presented at SIGMOD: https://2021.sigmod.org

According to regulations of the Dissertation Council in Computer Science (HSE University, 06/2022), at least 10 publications, indexed by WOS (Web of Science), Scopus, are listed below. Articles and papers are in:

- PVLDB journal (Q1, WOS, Scopus): [1], [3], [5], [6], [7], and [8]
- SIGMOD proceedings (CORE A*, WOS, Scopus): [2] and [4]
- Lecture Notes in Computer Science, LNCS (Q2, WOS, Scopus): [9], [11], [12], and [13]
- Communications in Computer and Information Science, CCIS (Q3, WOS, Scopus): [14] and [16]
- Conference proceedings (WOS, Scopus): [10] and [15]

The defense must be based on at least 7 of them, here: [1], [2], [3], [4], [6], [7], [8] (1st tier) and [11], [12], [13], [14], [15] (2nd tier).

PVLDB (PROCEEDINGS OF THE VLDB ENDOWMENT) is also included in the HSE University «List A» (Top Journals, ISSN: 2150-8097). In addition, VLDB & SIGMOD are included in the «List A^{CONF}» of the HSE University (Leading Conferences in Computer Science).

Contributions. R.A. Rodriges Zalipynis is the sole author of almost all publications. Otherwise, contributions are stated explicitly.

All listed publications are devoted to Array (Tensor) DBMSs. In addition, all journal articles and conference papers included in this Dissertation were published after the Author earned his Ph.D. degree in 2013.

First-tier publications (all are indexed **both** by WOS & Scopus):

1. **R.A. Rodriges Zalipynis**. ChronosDB: Distributed, File Based, Geospatial Array DBMS. ***PVLDB***, 11(10): 1247–1261, 2018. DOI · PDF · **Article, Q1 Journal**, indexed by WOS & Scopus

   ChronosDB outperforms SciDB by up to **75×** on average. **ChronosDB** is always faster and can outperform SciDB by up to **1034×** **ChronosDB** is a cloud-native DBMS. SciDB is developed by Paradigm4 and M. Stonebraker, an ACM Turing Award Recipient ("Nobel Prize of Computing")

   Homepage: http://chronosdb.gis.land

2. **R.A. Rodriges Zalipynis**. ChronosDB in Action: Manage, Process, and Visualize Big Geospatial Arrays in the Cloud. ***SIGMOD*** 2019, P. 1985–1988. DOI · **CORE A***, WOS & Scopus

   Presents a **new distributed WMTS server** directly inside ChronosDB and **new** **ChronosDB** components enabling users to interact with **ChronosDB** and appreciate its benefits: (1) Web GUI, (2) execution plan explainer (investigate the generated DAG), and (3) dataset visualizer (display ChronosDB datasets on an interactive Web map).

   Homepage: http://chronosdb.gis.land

3. **R.A. Rodriges Zalipynis**. BitFun: Fast Answers to Queries with Tunable Functions in Geospatial Array DBMS. ***PVLDB***, 13(12): 2909–2912, 2020. DOI · PDF · **Article, Q1 Journal**, indexed by WOS & Scopus

   A new class of Array (Tensor) DBMS queries is identified & tackled: tunable queries. **BitFun** provides novel strategies to continuously re-index tensors to efficiently answer queries with similar mathematical functions. It can be up to **8×** faster than computing the results from scratch.

   Homepage: http://bitfun.gis.land
   Video: https://youtu.be/uxGuZU8yEvE (7 min.)

4. **R.A. Rodriges Zalipynis**. Convergence of Array DBMS and Cellular Automata: A Road Traffic Simulation Case. ***SIGMOD*** 2021, P. 2399–2403 · **open access** · DOI · **CORE A***, WOS & Scopus

A novel Research & Development direction in the area of Array (Tensor) DBMS is presented. For the first time, we enabled an Array (Tensor) DBMS to simulate the physical world. The approach brings powerful parallelization, data fusion, array processing, and interoperability to name a few. ChronosDB, for example, simulates a complex road traffic model with multiple lanes, road intersections & traffic lights.

Homepage: http://sigmod2021.gis.gg/
Video: https://youtu.be/3g1m1fNL6P4 (8 min.)

Video 20 min.: dl.acm.org/doi/10.1145/3448016.3458457

5. **R.A. Rodriges Zalipynis**. Array DBMS: Past, Present, and (Near) Future. ***PVLDB***, 14(12): 3186–3189, 2021. DOI · PDF · **Article, Q1 Journal**, indexed by WOS & Scopus

   The **first** comprehensive tutorial on Array (Tensor) DBMS **R&D**. Presents numerous promising **R&D** opportunities.

   Duration: 90 minutes (1.5 hours), included in the main conference program, main conference day. Only 8 tutorials were accepted to VLDB 2021: prooflink

   Homepage: http://vldb2021.gis.gg/
   The homepage provides high-quality video (1.5 hours)

6. **R.A. Rodriges Zalipynis**, N. Terlych. WebArrayDB: A Geospatial Array DBMS in Your Web Browser. ***PVLDB***, 15(12): 3622–3625, 2022. DOI · PDF · **Article, Q1 Journal**, indexed by WOS & Scopus

   The **first** Array (Tensor) DBMS that can run completely inside a Web browser: **WebArrayDB**. The article also presents **ArrayGIS**, a new Web GIS based on **WebArrayDB**. The systems can be over **2×** faster compared to querying only Sentinel-Hub, a popular Cloud service for disseminating Sentinel data (recently acquired by Planet).

   R.A. Rodriges Zalipynis contributions (stated on page №3625 of the article): ideas, approaches, software architecture & design, libraries' choice, the paper.

   Homepage: https://wikience.github.io/webdb2022
   Video: https://youtu.be/NnpNR8GArj0 (5 min.)

Try ArrayGIS and WebArrayDB for free: http://webdb.gis.gg

7. **R.A. Rodriges Zalipynis**. SimDB in Action: Road Traffic Simulations Completely Inside Array DBMS. ***PVLDB***, 15(12): 3742–3745, 2022. DOI · PDF · **Article, Q1 Journal**, indexed by WOS & Scopus

   The **first** Array (Tensor) DBMS running end-to-end simulations completely inside itself: from data preparation to simulation to computing statistics.

   Tensor DBMSs can bring numerous benefits to simulations via a "DBMS approach", e.g., powerful parallelization and interoperability, while simulations expand the Tensor DBMS landscape and open a wide range of R&D opportunities.

   Homepage: https://wikience.github.io/simdb2022
   Video: https://youtu.be/NnpNR8GArj0 (5 min.)

8. **R.A. Rodriges Zalipynis**. FastMosaic in Action: A New Mosaic Operator for Array DBMSs. ***PVLDB***, 16(12): 3938–3941, 2023. DOI · PDF · **Article, Q1 Journal**, indexed by WOS & Scopus

   FastMosaic is a new Array (Tensor) DBMS mosaic operator, equipped with our new, scalable way to perform Canonical Correlation Analysis (CCA) in linear time, deriving CCA canonical variables together with mosaic transformation coefficients in the same pass over the input data.
   The CCA algorithm can run **orders of magnitude faster** for the purpose of array (tensor) mosaicking compared to the popular Python scikit-learn library.

   CCA is a popular tool for finding correlations in multidimensional datasets. CCA is widely used in Data Science for dimensionality reduction and discovering latent variables.

   Homepage: https://wikience.github.io/fastmosaic2023
   Video: https://youtu.be/DXC4r5DCd6k (15 min.)

According to regulations of the Dissertation Council in Computer Science, at least 4 first-tier publications must be without co-authors or the applicant must be the main co-author. Here, all first-tier publications except one (7 of 8) are without co-authors and R.A. Rodriges Zalipynis is the main co-author of one article.

Second-tier publications (all are indexed **both** by WOS & Scopus):

9. R. A. Rodriges Zalipynis (2021) **Towards Machine Learning in Distributed Array DBMS: Networking Considerations**, *Machine Learning for Networking: Third International Conference*, MLN 2020, Paris, France, November 24–26, 2020, Revised Selected Papers, **Lecture Notes in Computer Science (LNCS)**, Vol. 12629, P. 284–304, Springer, 2021. DOI – **WOS, Scopus, Q2**

10. R. A. Rodriges Zalipynis (2019) **Evaluating Array DBMS Compression Techniques for Big Environmental Datasets**, *Proceedings of the 2019 IEEE 10th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, P. 859–863. ***IEEE***, 2019. DOI – **WoS, Scopus**

11. R. A. Rodriges Zalipynis (2018) **Generic Distributed In Situ Aggregation for Earth Remote Sensing Imagery**, *Proceedings of Analysis of Images, Social Networks and Texts – 7th International Conference*, AIST 2018, July 5–7, 2018, Revised Selected Papers. **Lecture Notes in Computer Science (LNCS)**, Vol. 11179, P. 331–342, Berlin: Springer, 2018. DOI – **WOS, Scopus, Q2**

12. R. A. Rodriges Zalipynis (2018) **Distributed In Situ Processing of Big Raster Data in the Cloud**, *Perspectives of System Informatics – 11th International Andrei P. Ershov Informatics Conference*, PSI 2017, June 27–29, 2017, Revised Selected Papers, **Lecture Notes in Computer Science (LNCS)**, Vol. 10742, P. 337–351, Springer, 2018. DOI – **WOS, Scopus, Q2**

13. R.A. Rodriges Zalipynis, E. Pozdeev, A. Bryukhov **Array DBMS and Satellite Imagery: Towards Big Raster Data in the Cloud**, *International Conference on Analysis of Images, Social Networks and Texts* (AIST), Revised Selected Papers. **Lecture Notes in Computer Science (LNCS)**, Vol. 10716, P. 267–279, Springer, 2017. DOI – **WOS, Scopus, Q2**

    Best talk award, certificate: PDF

    R.A. Rodriges Zalipynis contributions (stated on page №277 and on the Springer Web Portal):

    all text, figures, design and implementation of algorithms and ChronosServer, ChronosServer data model, Azure

management code, SciDB import code, experimental setup, experiments.

14. R.A. Rodriges Zalipynis, A. Bryukhov, E. Pozdeev (2017) **Retrospective Satellite Data in the Cloud: An Array DBMS Approach**, Supercomputing. RuSCDays 2017. **Communications in Computer and Information Science (CCIS)**. Revised Selected Papers. Vol. 793, P. 351–362. Springer. DOI – **WOS, Scopus, Q3**

    R.A. Rodriges Zalipynis contributions (stated on page №361 and on the Springer Web Portal):
    all text, figures, design and implementation of algorithms and ChronosServer, ChronosServer data model, Azure management code, SciDB import code, experimental setup, experiments.

15. R. A. Rodriges Zalipynis (2017) **Array DBMS in Environmental Science: Sea Surface Height Data in the Cloud**, *Proceedings of the 2017 IEEE 9th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, P. 1062–1065, ***IEEE***, 2017. DOI – **WOS, Scopus**

16. R. A. Rodriges Zalipynis (2016) **ChronosServer: Fast In Situ Processing of Large Multidimensional Arrays with Command Line Tools**, *Supercomputing. RuSCDays 2016.* Revised Selected Papers. **Communications in Computer and Information Science (CCIS)**. Vol. 687, P. 27–40, Springer, 2016. DOI – **WOS, Scopus, Q3**

Conference presentations:
Conference name, city, presentation title, year:

- The 49th International Conference on Very Large Data Bases (Vancouver, Canada). FastMosaic in Action: A New Mosaic Operator for Array DBMSs, 2023
- The 48th International Conference on Very Large Data Bases (Sydney, Australia). WebArrayDB: A Geospatial Array DBMS in Your Web Browser, 2022
- The 48th International Conference on Very Large Data Bases (Sydney, Australia). SimDB in Action: Road Traffic Simulations Completely Inside Array DBMS, 2022

- The 20th International Conference «Modern Problems of Earth Remote Sensing from Space (Physical foundations, methods and technologies for monitoring the environment, potentially hazardous phenomena and objects)» (Moscow, Russia). ChronosDB: high performance processing of Earth remote sensing data, 2022
- ACM SIGMOD/PODS International Conference on Management of Data (Xi'an, Shaanxi, China). Convergence of Array DBMS and Cellular Automata: A Road Traffic Simulation Case, 2021
- The 47th International Conference on Very Large Data Bases (Copenhagen, Denmark). Array DBMS: Past, Present, and (Near) Future, 2021
- The 46th International Conference on Very Large Data Bases (Tokyo, Japan). BitFun: Fast Answers to Queries with Tunable Functions in Geospatial Array DBMS, 2022
- 3rd International Conference on Machine Learning for Networking (MLN'2020) (Paris, France). Towards Machine Learning in Distributed Array DBMS: Networking Considerations, 2020
- ACM SIGMOD/PODS International Conference on Management of Data (Amsterdam, Netherlands). ChronosDB in Action: Manage, Process, and Visualize Big Geospatial Arrays in the Cloud, 2019
- IEEE 10th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS) (Metz, France). Evaluating Array DBMS Compression Techniques for Big Environmental Datasets, 2019
- The 44th International Conference on Very Large Data Bases (Rio de Janeiro, Brasil). ChronosDB: Distributed, File Based, Geospatial Array DBMS, 2018
- The 7th International Conference on Analysis of Images, Social Networks, and Texts (AIST'2018) (Moscow, Russia). Generic Distributed In Situ Aggregation for Earth Remote Sensing Imagery, 2018
- Perspectives of System Informatics - 11th International Andrei Ershov Informatics Conference, PSI 2017 (Moscow, Russia). Distributed In Situ Processing of Big Raster Data in the Cloud, 2017

- Russian Supercomputing Days (Moscow, Russia). Retrospective Satellite Data in the Cloud: An Array DBMS Approach, 2017
- 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS'2017) (Bucharest, Romania). Array DBMS in Environmental Science: Sea Surface Height Data in the Cloud, 2017
- Analysis of Images, Social Networks and Texts. 6th International Conference, AIST 2017 (Moscow, Russia). Satellite Imagery and Array DBMS: Towards Big Raster Data in the Cloud, 2017
- Russian Supercomputing Days (Moscow, Russia). In-situ processing of big raster data with command line tools, 2016

1.5 Source Code of the Software

The architectural and implementation aspects of the software are among the results of this Dissertation. However, the following software or their source code are not part of or results of this Dissertation: ChronosDB [54, 55], BitFun [53], SimDB [56, 61], WebArrayDB & ArrayGIS [63] and FastMosaic [59].

Chapter 2

Theoretical Foundations

2.1 A New Formal Array (Tensor) DBMS Data Model

We had been improving the model for several years [51, 57, 64]. It is now solid [52, 54] and is a basis for the results described in this work, including new R&D directions, appearing on top of the model without its modifications due to its thoroughness. This formal model makes it possible to put Array (Tensor) DBMSs on a solid theoretical footing and leverage industrial experience. For example, the model enables strict formal definitions of tensor datasets and operations, as well as to work with arrays in situ (in their native file formats). In turn, this enables building other theoretical foundations on top of this model. For example, such strict and formal definitions make it possible to realize various mechanisms that enable efficient simulations entirely inside an Array (Tensor) DBMS, sections 3.3 and 4.3.

2.1.1 Motivation for a New Data Model

Array (Tensor) DBMS data models try to capture and generalize the diversity of large arrays: 1-d time-series, 2-d geospatial grids, tracks, swaths, or other arrays where a dimension represents coordinates (e.g., time, height, model №), 3-d spatio-temporal cubes like astronomical observations or biomedical data, unstructured meshes, and generally any data that can be modeled by multidimensional arrays (tensors).

However, arrays are traditionally stored in files, not databases. Such files have complex naming, diverse coordinate systems, formats, and supported data types, to name a few. It is also important that a dataset is almost always split into multiple files. Moreover, the dramatic increase in array data volumes stimulates the use of computer clusters for distributed array management, processing, analysis, and visualization.

Industrial array (tensor) data models provide a uniform access to an array (tensor) that can be represented in different storage formats or service APIs. The most widely used industry-standard array data models are CDM, GDAL, and ISO, mappable to each other to some extent [39]. These models resulted from decades of considerable practical experience, but share a key drawback: they work only with a single file, not with a set of files as a single array. The most well-known research models and algebras for dense multidimensional, general-purpose arrays are AML [33], AQL [30], and RAM [71]. They are mappable to Array Algebra [8].

The creation of a new formal data model was motivated by several features that are not simultaneously present in the existing data models [54]: (1) the treatment of arrays in multiple files arbitrarily distributed over cluster nodes as a single array, (2) formalized industrial experience to leverage it in the algorithms, (3) a rich set of data types (Gaussian, irregular grids, etc.), (4) subarray mapping to a raster file is almost 1:1 but still independent from a format.

Our model has several levels. A user perceives a large multidimensional array at the logical level as a single object, section 2.1.2. A set of system-level tensors (subarrays) is distributed among cluster nodes and stored as ordinary files in diverse file formats at the second model level, section 2.1.3. An operation with a user-level logical array (tensor) is mapped to a sequence of operations with respective system-level subarrays in our algorithms, section 2.2. More information on the benefits and properties of our new data model are in the introduction, sections 2 and 4 of [54].

Section 2.2 showcases that our new and efficient algorithms can leverage industrial experience by delegating portions of work for a single cluster node to external software. Chapter 4 demonstrates how the operations and algorithms based on our new data model serve for efficient management, processing, visualization, and analyzes of diverse real-world array (tensor) data in numerous real-world applications.

2.1.2 Tensors or Multidimensional Arrays

Now we introduce the logical level of our data model [52, 54], fig. 2.1.

An N-dimensional array (N-d array or tensor) 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[1]. l_i is said to be the *size* or *length* of ith

[1] A C++ type according to ISO/IEC 14882 can be taken to be specific about value ranges, size in bytes, and other properties

dimension[2]. Let us denote the N-d array (tensor) by

$$A\langle l_1, l_2, \ldots, l_N\rangle : \mathbb{T} \tag{2.1}$$

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 $\mathbb{T}$. A missing value is denoted by `NA`.

An array (tensor) may be initialized after its definition by enumerating its cell values. For example, the following defines and initializes a 2-d array (matrix) of integers: $A\langle 2, 2\rangle$: int $= \{\{1, 2\}, \{\texttt{NA}, 4\}\}$. In this example, $A[0, 0] = 1$, $A[1, 0] = \texttt{NA}$, $|A| = 4$, 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, $d_i[j] < d_i[j+1]$, and $d_i[j] \neq \texttt{NA}$ for $\forall j \in D_i$. In this case, A in eq. (2.1) can be also defined as

$$A(d_1, d_2, \ldots, d_N) : \mathbb{T} \tag{2.2}$$

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

$$A[b_1 : e_1, \ldots, b_N : e_N] = A'(d'_1, \ldots, d'_N) \tag{2.3}$$

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:

$$A'[y_1, \ldots, y_N] = A[y_1 + b_1, \ldots, y_N + b_N] \tag{2.4a}$$

$$d'_i[y_i] = d_i[y_i + b_i]. \tag{2.4b}$$

Equations (2.4a) and (2.4b) state that A and A' have a common coordinate subspace over which cell values of A and A' coincide. The dimensionality of A and A' is the same. In hyperslab definitions, eq. (2.3), we will omit "$: e_i$" if $b_i = e_i$ or "$b_i : e_i$" if $b_i = 0$ and $e_i = |d_i| - 1$.

Arrays (tensors) X and Y overlap iff $\exists Q : Q \sqsubseteq X \wedge Q \sqsubseteq Y$. Array (tensor) Q is called the *greatest common hyperslab* of X and Y and denoted by $gch(X, Y)$ iff $\nexists W : (W \sqsubseteq X) \wedge (W \sqsubseteq Y) \wedge (Q \sqsubseteq W) \wedge (Q \neq W)$. An array (tensor) X covers an array (tensor) Y iff $Y \sqsubseteq X$.

We call the array (tensor) depicted in fig. 2.1 as a user-level array (tensor): a logical representation of a possibly very large N-d array (tensor) that may not fit into a single machine, be it operating memory, persistent, or any other storage. The array (tensor) in fig. 2.1 has 3 dimensions and

[2]Here and further on $i \in [1, N] \subset \mathbb{Z}$

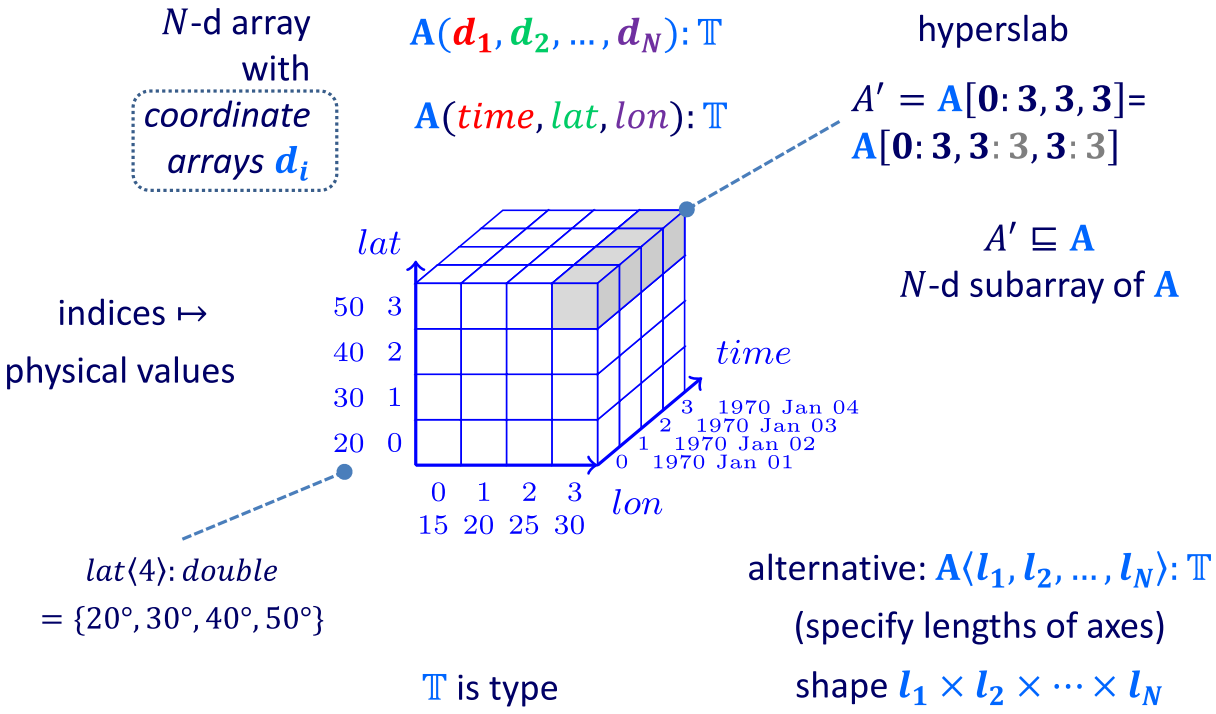


Figure 2.1: An illustration of a user-level array (tensor)

3 coordinate arrays that map dimension indexes into temporal and spatial coordinates. The hyperslab in fig. 2.1 is a time series for a point with coordinates $(50°, 30°)$.

Typically, a DBMS can provide a schema for its object(s). For example, relational DBMSs can produce a schema for their tables. As our model is designed for Array (Tensor) DBMSs, a user-level array (tensor) can also have a schema described in [52, 54] and section 3.1.2. The model is implemented in ChronosDB, whose users avoid learning a new schema notation and inspect arrays (tensors) in a way they are accustomed to.

2.1.3 Multilevel, Distributed Datasets

The second level of our model consists of a set of system-level tensors (subarrays) distributed among cluster nodes and stored as ordinary files in diverse file formats [52, 54], fig. 2.2. A user-level array (tensor) is never materialized and stored explicitly: an operation with a user-level object is mapped to a sequence of operations with respective subarrays.

Datasets can be raw and regular. No shape restrictions on raw system-level arrays (tensors) are imposed making it possible to represent very complex real-world datasets like scattered, overlapping satellite scenes. For regular datasets, among other characteristics, we require that their subarrays meet certain criteria that can be used to design efficient algo-

rithms. For example, subarrays can overlap, but according to a pattern. Raw datasets can be ingested ("cooked") into regular datasets.

Formally, a *raw dataset* $\mathbb{D}^{raw} = (A, P^{raw})$ has a *user-level* array (tensor) $A(d_1, d_2, \ldots, d_N) : \mathbb{T}$, and a set of *system-level* arrays (subarrays) $P^{raw} = \{(A', B, E, wid)\}$, where $A' \sqsubseteq A$, $B\langle N\rangle : \text{int} = \{b_1, b_2, \ldots, b_N\}$, $E\langle N\rangle : \text{int} = \{e_1, e_2, \ldots, e_N\}$ such that $A' = A[b_1 : e_1, b_2 : e_2, \ldots, b_N : e_N]$, wid is an identifier of a cluster node storing A'.

A *regular dataset* $\mathbb{D} = (A, P, S, \rho, r^0)$ has a *user-level* array (tensor) $A(d_1, d_2, \ldots, d_N) : \mathbb{T}$, a set of system-level arrays (subarrays) $P = \{(A', B, E, wid, key)\}$, where A', B, E, wid mean the same as for P^{raw}, $key\langle N\rangle : \text{int} = \{k_1, k_2, \ldots, k_N\}$, $k_i \in \mathbb{Z}$, $A' \sqsubseteq A[h_1^b : h_1^e, \ldots, h_N^b : h_N^e]$, where

$$h_i^b = max(r_i^0 + k_i \times s_i - \rho_i, 0), \tag{2.5a}$$

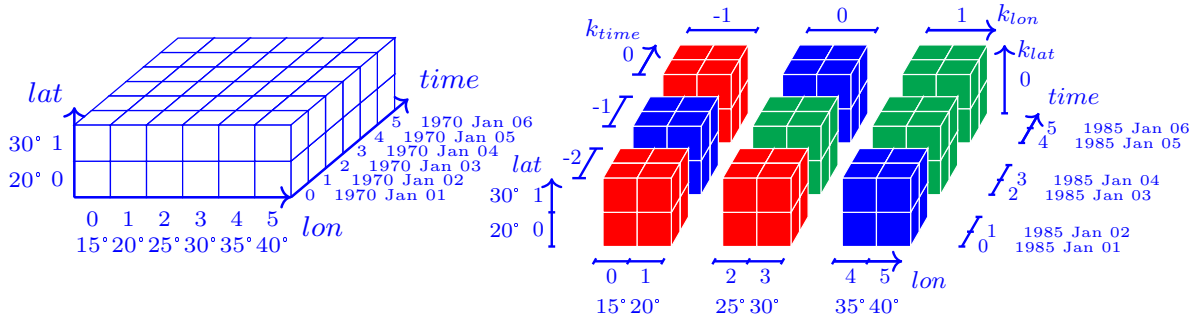
$$h_i^e = min(r_i^0 + (k_i + 1) \times s_i - 1 + \rho_i, |A.d_i| - 1), \tag{2.5b}$$

$S = (s_1, s_2, \ldots, s_N)$ is the largest possible shape for $\forall p \in P$, $\rho = (\rho_1, \rho_2, \ldots, \rho_N)$ is an overlap between subarrays, and $r^0 = (r_1^0, r_2^0, \ldots, r_N^0)$ is a reference index, $s_i, r_i^0 \in \mathbb{Z}$, $s_i > 0$, $\rho_i \in [0, s_i \text{ div } 2) \subset \mathbb{Z}$, and $\nexists p, q \in P : p.key = q.key \wedge p = q$. Note that $\mathbb{D}^{raw}$ and $\mathbb{D}$ share A. Let us refer to subarray A' by key as $\mathbb{D}\langle\langle key\rangle\rangle$ or $\mathbb{D}\langle\langle k_1, k_2, \ldots, k_N\rangle\rangle$.

A regular dataset is shown in fig. 2.2. Array (tensor) $A(time, lat, lon)$ with shape $6 \times 2 \times 6$ is divided by 2-d planes into 9 subarrays (this is very common in practice), where $S = (2, 2, 2)$, $r^0 = (4, 0, 2)$, overlap is not shown. Subarrays with the same color reside on the same cluster node. The values of the coordinate arrays are shown next to each index. The key $\langle\langle -1, 0, 1\rangle\rangle$ refers to the subarray $A[2 : 3, 0 : 1, 4 : 5]$ for Jan 03-04, $lat = 20^\circ \ldots 30^\circ$, $lon = 35^\circ \ldots 40^\circ$.

This means that array (tensor) A is separated by $(N-1)$-d hyperplanes on N-d subspaces. A system-level array may not fully cover the subspace in which it is located. Note that not all subspaces must contain an object. Subarrays can overlap, but only overlapping cells can be inside the same subspace. Finally, all subarray keys are unique. We treat an empty subspace as a subarray with all cells equal to `NA`. This is one of the ways how our model supports large sparse arrays.

The model also has the third level: chunks. This makes the systems based on the model even more flexible in terms of adapting to dynamic workloads due to efficient execution of the "chunking" operation, sections 2.2.3 and 4.1.1.



(a) A User-Level Array (Tensor) (b) Corresponding System-Level Arrays [54]

Figure 2.2: User-Level and System-Level Arrays (Tensors) (best viewed in color)

Note that we omit parts of the model that we do not use in the subsequent sections of this dissertation. The complete formal definition of our model is in [54].

2.2 New Distributed & In Situ Tensor Algorithms

The model presented in section 2.1 makes it possible to formally define tensor operations (operators) and devise distributed algorithms and their efficient execution techniques on computer clusters, possibly in the Cloud.

We formed the set of array (tensor) operations to implement by analyzing their importance regarding providing essential and extended array (tensor) management and processing capabilities. We provide justification for each operation. In other words, the selected operations are building blocks that can be used to construct complex analytic pipelines, see SAVI pipeline as an example (section 3.1.3) with its execution plan (section 3.4 of [54]), as well as other applications based on these operations, chapter 4.

One of the main differences between our new algorithms and other algorithms is that they can manage and process tensors in situ, directly in numerous file formats: subarrays can be stored as ordinary files on cluster nodes without importing into an internal DBMS format. This enables many benefits, including the possibility of partially delegating array (tensor) operations to existing elaborate and quality-assured software (see Introduction of [54]), thus leveraging industrial experience. However, designing an efficient system that works with data in situ and utilizes the delegation approach "goes well beyond invoking separate instances of the same function at every node" as noted about ChronosDB in [66].

The pseudo-codes of the algorithms are quite large [51, 54, 57, 60, 64], so we present here formal definitions of operations and key ideas behind

the algorithms. The pseudo-codes are based on our new formal data model (section 2.1) and have lines highlighted in light gray to indicate the leveraged industrial experience, among other goals. The algorithms can perform a wide variety of distributed operations in a very efficient manner: up to 1034× faster than SciDB (developed by Paradigm4 and M. Stonebraker, an ACM Turing Award Recipient), section 4.1.1.

2.2.1 Distributed N-d Retiling

The distributed N-d retiling is one of the most important tensor operations that appears at the heart of many other vital functions that are key to Array (Tensor) DBMSs. For example, the retiling can be used to "cook" a regular dataset from a raw dataset, to transform a regular dataset, as well as for K-way joins, making it a fundamental array (tensor) operation [54].

As input, the N-d retiling takes raw or regular dataset $\mathbb{D} = (A, P)$ and retiling parameters $S' = (s'_1, s'_2, \ldots, s'_N)$, $\rho' = (\rho'_1, \rho'_2, \ldots, \rho'_N)$, $\bar{r}^0 = (\bar{r}^0_1, \bar{r}^0_2, \ldots, \bar{r}^0_N)$ (target shape, overlap, and reference indexes respectively). As output, the retiling generates a new regular dataset $\mathbb{D}' = (A, P', S', \rho', \bar{r}^0)$. Note that $\mathbb{D}$ and $\mathbb{D}'$ share A.

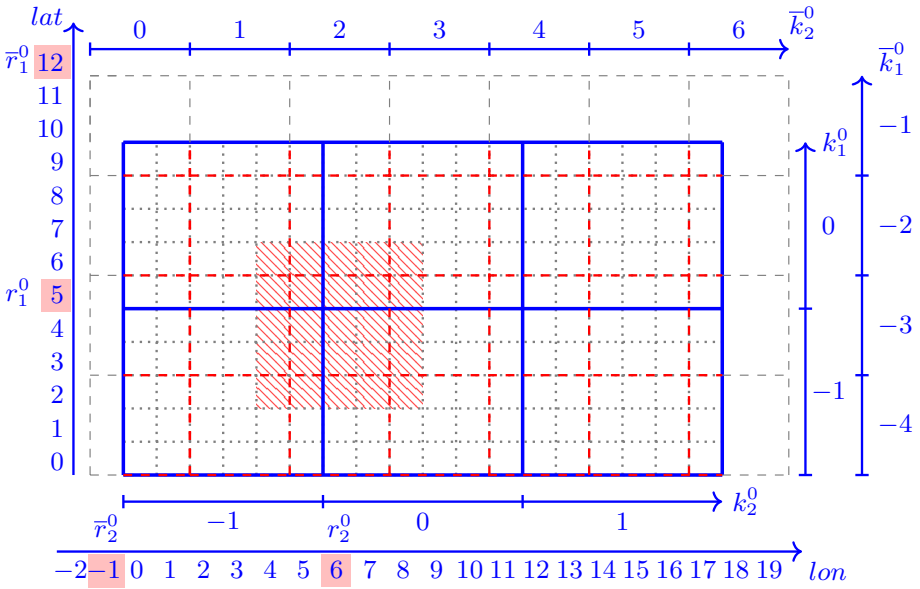


Figure 2.3: Retiling example [54]

The key idea behind the proposed algorithm is as follows. We cut each system-level subarray $p \in P$ onto smaller pieces $P' = \{p' : p' \sqsubseteq p\}$ (sub-sub-arrays) and assign each piece a new key. Next, we merge all pieces with the same key into a single, new system-level tensor (new subarray in a new, output dataset). Each piece, using its key, is associated with a

cluster node. Hence, all pieces with the same key are gathered and merged on the same node in our distributed algorithm which is structured in a way that can delegate hyperslabbing (cutting a piece) and merging (assembling a set of pieces into a subarray) to an external software [54].

Figure 2.3 illustrates the retiling algorithm on a 2-d case. Consider a regular dataset $\mathbb{D} = (A, P, S, \rho, r^0)$, where $A(lat, lon)$ is a 2-d array (matrix), $S = (5, 6)$, $\rho = (0, 0)$, $r^0 = (5, 6)$. The shape of A is 10×18. In addition, A consists of 6 subarrays whose bodies are separated by thick blue lines. Subarrays may be located on different cluster nodes. Gray dotted lines separate matrix cells. The key range for the subarrays is $k_1^0 \in [-1, 0]$, $k_2^0 \in [-1, 1]$.

We retile the dataset $\mathbb{D}$ into $\mathbb{D}' = (A, P', S', \rho', \bar{r}^0)$, where $S' = (3, 3)$, $\rho' = (1, 1)$, $\bar{r}^0 = (12, -1)$. Dashed red lines separate new subarray bodies. The algorithm will yield 28 new subarrays with key range $\bar{k}_1^0 \in [-4, -1]$, $\bar{k}_2^0 \in [0, 6]$. For instance, the hatched area indicates subarray $\mathbb{D}'\langle\langle -3, 2\rangle\rangle = A[2{:}6, 4{:}8]$. Note that not all new system-level arrays will have the shape 5×5 ($5 = s'_1 + \rho'_1 \times 2$). Hence, some subarray will not completely cover their respective subspaces, e.g. $\mathbb{D}'\langle\langle -1, 6\rangle\rangle = A[8 : 9, 16 : 17]$. Reference coordinates r_i^0 and $\bar{r}_i^0$ are highlighted in pink on axes lat and lon . We plot input and output subarray keys on axes k_i^0 and $\bar{k}_i^0$ respectively.

2.2.2 Distributed K-Way Array (Tensor) Join

For $K \in \mathbb{N}$, many K-ary tensor operations require the K-way join [51, 54, 57, 60, 64]. For example, Array (Map) Algebra is widely used in the industry and constitutes a large portion of Array (Tensor) DBMS workloads [52, 66]. It may require the K-way array (tensor) join for 2 or more input arrays. The distributed K-way array (tensor) join is based on the distributed N-d retiling, section 2.2.1.

The following formal definition of the K-way array join is from [54].

The K-way join $\bowtie$: $\odot, \kappa, A_1, A_2, \ldots, A_K \mapsto A_\bowtie$ takes as input a join type $\odot \in \{\text{INNER}, \text{OUTER}\}$, a mapping $\kappa : \mathbb{T}_1, \mathbb{T}_2, \ldots, \mathbb{T}_K \mapsto \mathbb{T}_\bowtie$, and N-d arrays $A_j(d_1^j, d_2^j, \ldots, d_N^j) : \mathbb{T}_j$, $j \in [1, K]$ such that $\exists d_i^\bowtie : d_i^j \sqsubseteq d_i^\bowtie$ for $\forall i, j$.

The K-way join yields the N-d array $A_\bowtie(d_1^\bowtie, d_2^\bowtie, \ldots, d_N^\bowtie) : \mathbb{T}_\bowtie$ such that $A_\bowtie[x_1, x_2, \ldots, x_N] = \kappa'(a_1, a_2, \ldots, a_K)$, where $x_i \in [0, |d_i^\bowtie|)$, $a_j = A_j[y_1^j, y_2^j, \ldots, y_N^j]$ if $\exists y_i^j : d_i^j[y_i^j] = d_i^\bowtie[x_i]$; $a_j = \texttt{NA}$ otherwise.

Operation κ' formally defines the difference between the two join types. If $a_j = \texttt{NA}$ for $\forall j$, κ' returns NA regardless of the $\odot$ value. If $\odot = \text{INNER}$ and $\exists j : a_j = \texttt{NA}$, κ' returns NA. Otherwise κ' returns $\kappa(a_1, a_2, \ldots, a_K)$. For

algebraic calculations where any operation on a set of cells must return `NA` if at least one of the cell values is missing, INNER join is helpful. When at least one non-missing value contributes to the resulting cell, e.g., a K-day cloud-free composite of satellite imagery, OUTER join is useful [54].

A distributed 2-way array (tensor) join is illustrated in fig. 2(e) in [54] that shows detailed information on the 2-d retiling and 2-d join stages, along with key values assigned to each entity that participates in the execution plan. The tuples in the graph contain the keys of subarrays.

Among other things, this join is necessary as part of a complex analytic pipeline that uses Array (Map) Algebra, section 3.1.3. The distributed 2-way array (tensor) join is performed on Band4 and Band5 arrays with different shapes of subarrays. Band5 requires the distributed 2-d retiling during the join (in this case, $K = N$). The algorithm is expressed in such a way that can delegate the calculation of κ on the subarrays with the same key to an external, optimized software [54].

2.2.3 Aggregation, Chunking & Other Operations

Aggregation

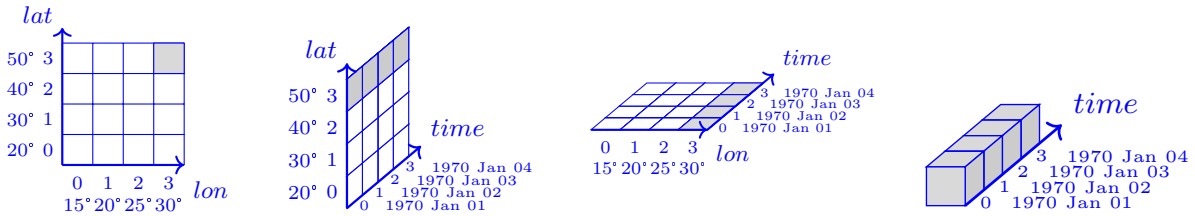
Aggregation frequently occurs during the analysis of real-world array data. For example, given a 4-d array $A(time, elevation, lat, lon)$, aggregate arrays $A'(time, lat, lon)$ and $A''(time, elevation, lat)$ can represent the average over all available vertical levels and a zonal average respectively.

Array aggregation can happen over an arbitrary subset of array axes [60]. The aggregate of an N-d array $A(d_1, d_2, \ldots, d_N) : \mathbb{T}$ over a set of axes $d_{aggr} \subset \{d_1, d_2, \ldots, d_N\}$ is the K-d array $A_{aggr}(d_{y_1}, \ldots, d_{y_K}) : \mathbb{T}$ such that $K = N - |d_{aggr}|$, $y_k < y_{k+1}$, $y_k \in [1, N]$, $d_{y_k} \notin d_{aggr}$, $A_{aggr}[x_{y_1}, \ldots, x_{y_K}] = f_{aggr}(cells(A[g(1), g(2), \ldots, g(N)]))$, where $\forall x_{y_k} \in [0, |d_{y_k}|)$, $g(i) = 0 : |d_i| - 1$ if $d_i \in d_{aggr}$, otherwise $g(i) = x_i$, $cells : A' \mapsto T$ is a multiset of all cell values of an array $A' \sqsubseteq A$, $f_{aggr} : T \mapsto w \in \mathbb{T}$ is an aggregation function. Note that the condition $K > 0$ always holds, otherwise the result is not an array but a scalar.

For example, we can aggregate the array (tensor) in fig. 2.1 over the *time* axis, *lon* axis, *lat* axis, or *lat* and *lon* axes at the same time, fig. 2.4.

It is challenging to perform aggregation (and other operations) in a distributed fashion, especially when subarrays are in complex array (tensor) file formats, including NetCDF, GeoTIFF, HDF, and others [60].

Recall that system-level arrays (tensors) with the same color are located on the same cluster node, fig. 2.2b. The aggregation algorithm consists of



(a) $A_{aggr}(lat, lon)$ (b) $A_{aggr}(time, lat)$ (c) $A_{aggr}(time, lon)$ (d) $A_{aggr}(time)$

Figure 2.4: Aggregation examples for the tensor in fig. 2.1 [60]

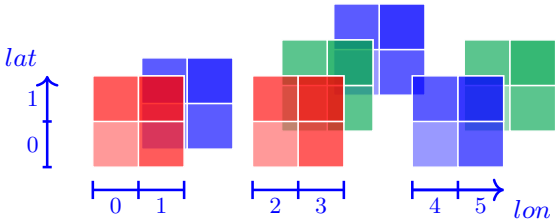
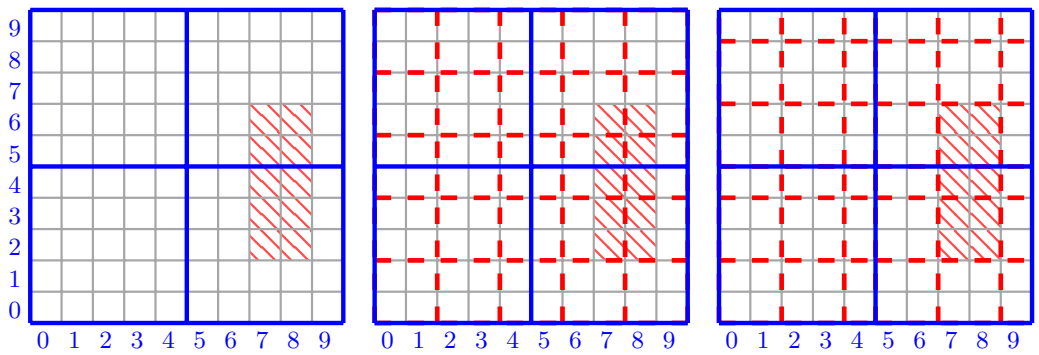


Figure 2.5: Intermediate aggregation of system-level tensors in fig. 2.2b (*time* axis) [60]

two phases. First, system-level tensors (subarrays) are aggregated in parallel on the cluster nodes, creating local intermediate subarrays, fig. 2.5. Intermediate keys for the interim subarrays are also generated appropriately. Next, the intermediate subarrays are distributed among the cluster nodes using a mapping policy and the final result is computed (not a single array, but a set of subarrays distributed over the cluster nodes). Our aggregation algorithms are built in a way that is capable of delegating computing f_{aggr} to an external software [54, 60].

Chunking

Chunking is one of the most important operations, as it allows Array (Tensor) DBMSs to adapt to dynamic workloads. The performance of an



(a) No chunking (b) Exact 2×2 chunking (c) Inexact 2×2 chunking

Figure 2.6: Array (tensor) chunking [54]

operation can differ by orders of magnitude depending on the configuration of chunks, section 4.1.1. System-level arrays (tensors) consist of chunks, I/O units (the third level of the model). Chunking alters the storage layout of chunks and their shapes.

The exact chunking reorganizes array cells such that the cells with indexes $(x_1, \ldots, x_N)$ and $(y_1, \ldots, y_N)$ belong to the same chunk iff x_i div $c_i = y_i$ div c_i for $\forall i$, fig. 2.6(b). Inexact chunking performs exact chunking of system-level arrays (subarrays) [54].

Let us read a 5×2 slice from 2-d array (subarrays are separated by thick blue lines), fig. 2.6. For the row-major storage layout, we can spend 5 I/O requests to read 5 strips sized 1×2, fig. 2.6(a). However, only chunks with the required data are read from a chunked array. The exact chunking operates within the logical, user-level array (tensor) while inexact chunking relies on the system-level indexing, fig. 2.6(b,c).

No single chunk shape is optimal for all access patterns. It is usually not possible to "guess" a priori a good chunk shape for an arbitrary case: chunk shapes are often tuned experimentally. Array (Tensor) DBMSs must be capable to quickly chunk tensors to support the tuning and to adapt to dynamic workloads.

Hyperslabbing

Hyperslabbing is an extraction of a hyperslab from an array (tensor), section 2.1. Its efficient execution is far from straightforward: the hyperslabbing performance can vary by orders of magnitude depending on the subarray shapes and the layouts of their cells which are typically tuned experimentally, by trial and error, section 4.1.1.

The hatched area marks the hyperslab $A' = A[2\!:\!7, 2\!:\!11]$, fig. 2.7. A is a logical 2-d array $A(lat, lon)$ with 15 subarrays separated by thick lines. Subarrays possibly reside on different cluster nodes and can be from a raw or a regular dataset. We reduce hyperslabbing of a logical array (tensor) to hyperslabbing the respective system-level arrays (tensors). First, we filter subarrays that do not overlap with A', e.g. $A[0\!:\!1, 0\!:\!4]$. We also migrate subarrays to a new dataset that are entirely inside A', e.g. $A[4\!:\!5, 5\!:\!7]$. The remaining subarrays are hyperslabbed to complete the operation.

We can delegate hyperslabbing to feature-rich and highly optimized external software tools. The detailed hyperslabbing pseudo-codes (different versions) are in [54, 57]. Chunking and hyperslabbing are extremely efficient in ChronosDB, section 4.1.1. They can be orders of magnitude

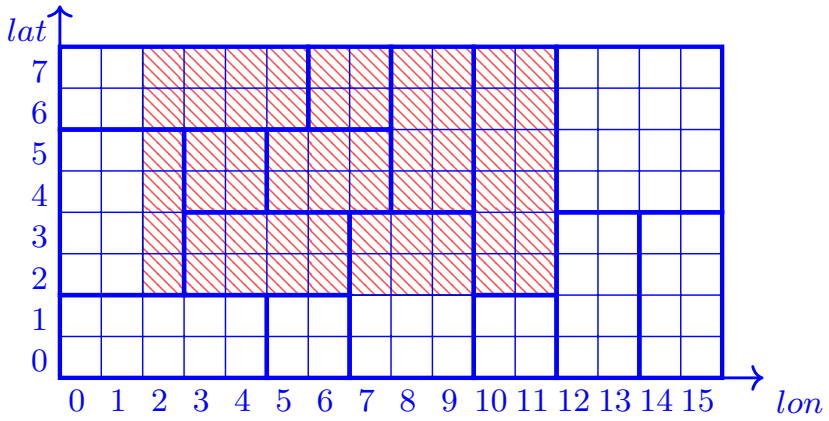
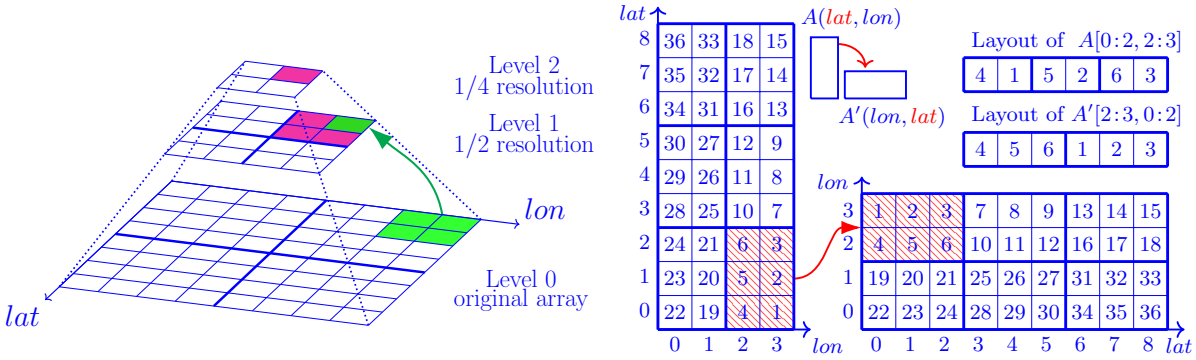


Figure 2.7: Hyperslabbing [57]

faster than SciDB, even when SciDB is appropriately tuned with chunking beforehand [54].



(a) Multiresolution pyramid, 3 levels [51]

(b) Array (Tensor) reshaping [54, 57]

Figure 2.8: Illustration of downsampling and reshaping in Array (Tensor) DBMSs

Resampling

Resampling is a core operation for numerous applications [50]. Upsampling, downsampling, and interpolation are typical representatives of resampling. In turn, Multiresolution Pyramid illustrates downsampling. Figure 2.8a depicts 3 levels of such pyramid.

Usually, large arrays (tensors) are visualized interactively with the help of multiresolution pyramids. Typically, a series of zoom levels is defined, e.g. $Z = \{0, 1, \ldots, 16\}$. A visualizer switches its current zoom level when a user zooms in/out. At zoom level $z \in Z$, the tensor is shown with $2^z\times$ less resolution than the original. A multiresolution pyramid is the stack of arrays (tensors) for all zoom levels, fig. 2.8a. This technique considerably reduces network traffic and system load making this functionality essential for an Array (Tensor) DBMS.

Formally, given a 2-d array $A(d_1, d_2) : \mathbb{T}$, its coarser version is the array $A'(d'_1, d'_2) : \mathbb{T}'$ such that $d'_i\langle\lceil l_i\rceil\rangle = \{d_i[0], d_i[2], \ldots, d_i[\lfloor l_i\rfloor \times 2]\}$, where $l_i = |d_i|/2$ and $A'[x'_1, x'_2] = avg(\{A[x_1, x_2] : x_i \text{ div } 2 = x'_i \wedge A[x_1, x_2] \neq \texttt{NA}\})$ for $\forall x_i$, avg is the average and $avg(\varnothing) = \texttt{NA}$. We assume that d_i stores the smallest border cell coordinates.

Interpolation estimates a value at a coordinate $(y_1, \ldots, y_N)$ for an array $A(d_1, \ldots, d_N)$, where $y_i \in (d_i[j], d_i[j+1]) \subset \mathbb{T}_i$, and $j \in [0, |d_i| - 1) \subset \mathbb{Z}$. For example, array resolution can be doubled by interpolation when $y_i = (d_i[j] + d_i[j+1])/2$ [54].

With our novel data model and in situ approaches, we support many resampling techniques [50, 54]. However, SciDB provides only averaging nearby array (tensor) values.

Reshaping

The reshaping operation changes the order of tensor dimensions, fig. 2.8b. This operation does not just impact the indexing of tensor cells, it has much deeper consequences in terms of Array (Tensor) DBMSs. Reshaping alters the tensor storage layout according to the given order of dimensions.

For example, reshaping helps to achieve the fastest hyperslabbing along a dimension e.g. reading a time series $A[x_1, x_2, 0{:}|time| - 1]$ of a 3-d array $A(lat, lon, time)$ vs. $A(time, lat, lon)$. This is because usually the last dimension of N-d arrays (tensors) varies the fastest: cells along the Nth dimension are stored sequentially in memory: see storage layouts of A and A' in fig. 2.8b. For a row-major layout, we can spend 3 I/O operations to read $A[0:2, 2]$ in contrast to only 1 I/O operation to read the same data from $A'[2, 0:2]$.

Formally, the *reshaping* operation $\Psi : A, \pi \mapsto A'$ takes as input an N-d array $A(d_1, \ldots, d_N) : \mathbb{T}$ and the permutation mapping $\pi : i \mapsto j$, where $i, j \in [1, N] \subset \mathbb{Z}$, $\pi(i) \neq \pi(j)$ for $i \neq j$, and $\bigcup_i \{\pi(i)\} = [1, N]$. The reshaping operation outputs the N-d array $A'(d_{\pi(1)}, \ldots, d_{\pi(N)}) : \mathbb{T}$ such that $A[x_1, \ldots, x_N] = A'[x_{\pi(1)}, \ldots, x_{\pi(N)}]$, where $x_i \in [0, |d_i|) \subset \mathbb{Z}$ for all i.

Reshaping a user-level array can be reduced to reshaping its system-level arrays (subarrays) individually by delegating the reshaping of system-level arrays (subarrays) to external tools.

Convolution

The convolution operation $\Xi : A, K \mapsto A_{conv}$ for a 2-d array $A(d_1, d_2) : \mathbb{T}$ and a kernel $K\langle k_1, k_2\rangle : \mathbb{T}$, where $k_i \leqslant |d_i|$, and $k_i \bmod 2 = 1$ for all

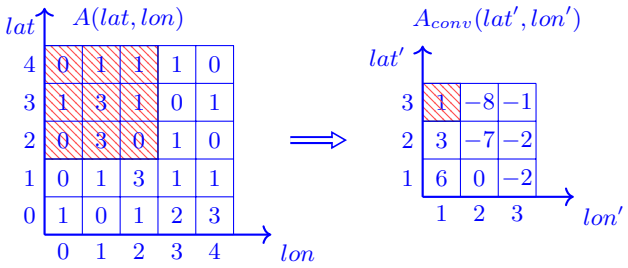


Figure 2.9: An illustration of convolution with K_2 kernel

i, produces the 2-d array $A_{conv}(d'_1, d'_2) : \mathbb{T}$ such that $d'_i = d_i[k_i \text{ div } 2 : |d_i| - k_i \text{ div } 2 - 1]$, and

$$A_{conv}[x_1, x_2] = \sum_{\substack{\forall x'_1 \in [0,k_1) \\ \forall x'_2 \in [0,k_2)}} K[x'_1, x'_2] \times A'[x'_1, x'_2] \tag{2.6}$$

where $A' = A[x_1 - k_1/2 : x_1 + k_1/2, x_2 - k_2/2 : x_2 + k_2/2]$, and $k_i/2 \leqslant x_i < |A.d_i| - k_i/2$ for all i (the division "/" is integer) [64].

Convolution is frequently used for array processing [50]. For example, edge detection with the Sobel Operator $K_1\langle 3,3\rangle = \{\{-1,-2,-1\},\{0,0,0\},\{1,2,1\}\}$ and $K_2\langle 3,3\rangle = \{\{-1,0,1\},\{-2,0,2\},\{-1,0,1\}\}$ happens as follows. First, local gradients are calculated at each array cell along axes d_1 and d_2: $A_{d1} = \Xi(A, K_1)$ and $A_{d2} = \Xi(A, K_2)$, fig. 2.9. Then, array $A_{edges} = \sqrt{A_{d1}^2 + A_{d2}^2}$ will contain higher values for cells classified as edges and lower values for other types of cells [64].

It is possible to run convolution for each 2-d $p \in P$ using an external software provided that the retiling was applied to the input dataset with $\rho_i \geqslant 1$ for $\forall i$. Therefore, the convolution can be efficiently applied to each system-level array (subarray) in parallel without data exchange between cluster nodes because subarrays overlap.

We also introduce another, new convolution operation (operator) in section 3.3.1.

Array (Tensor) Mosaicking

Please refer to sections 2.5, 3.5 and 4.4 for the definition of array (tensor) mosaicking and its respective techniques.

2.3 Tunable Queries, Indexing and Data Structure

2.3.1 Introducing a New R&D Direction: Tunable Queries

Dozens of parameterized math functions are applied daily to tensors to tackle vital practical tasks, including urban planning, agriculture monitoring, forestry control, and rapid response for disaster relief [4, 78], fig. 2.10.

$$\text{SAVI} = \frac{\text{NIR} - \text{R}}{\text{NIR} + \text{R} + \boxed{L}} \times (1 + \boxed{L}) \quad \text{ARVI}^{\ddagger} = \frac{\text{NIR} - (\text{R} - \boxed{\gamma}(\text{B} - \text{R}))}{\text{NIR} + (\text{R} - \boxed{\gamma}(\text{B} - \text{R}))} \quad \text{PVI} = \frac{cos(\boxed{@}) \times \text{NIR}}{-sin(\boxed{@}) \times \text{R}}$$

$$\text{GARI}^{\ddagger} = \frac{\text{NIR} - [\text{G} - \boxed{\gamma}(\text{B} - \text{R})]}{\text{NIR} + [\text{G} - \boxed{\gamma}(\text{B} - \text{R})]} \quad \text{TSAVI} = \frac{\boxed{@}(\text{NIR} - \boxed{@}\text{R} - \text{B})}{\text{R} + \boxed{@}\text{NIR} - \boxed{@}\boxed{b}} \quad \text{TVI} = \sqrt{\frac{\text{NIR} - \text{R}}{\text{NIR} + \text{R}} + 0.5\boxed{L}}$$

$$\text{AVI} = tan^{-1}\left[\frac{\boxed{\lambda_3} - \boxed{\lambda_2}}{\boxed{\lambda_2}} \frac{1}{(\text{NIR} - \text{R})}\right] + tan^{-1}\left[\frac{\boxed{\lambda_2} - \boxed{\lambda_1}}{\boxed{\lambda_2}} \frac{1}{(\text{G} - \text{R})}\right] \quad \text{WDRVI} = \frac{\boxed{@}\text{NIR} - \text{R}}{\boxed{@}\text{NIR} + \text{R}}$$

Figure 2.10: Examples of Typical Math Functions with **Tunable Parameters** [53]

As a typical example, consider Soil-Adjusted Vegetation Index (SAVI, fig. 2.10) which aims to minimize soil brightness influence. NIR and R are 2-d arrays with intensities of reflected solar radiation in the near-infrared and visible red spectra respectively. L is a soil fudge factor varying from 0 to 1 depending on the soil [78]. The user may tune L many times to find appropriate SAVI values for a given area of interest.

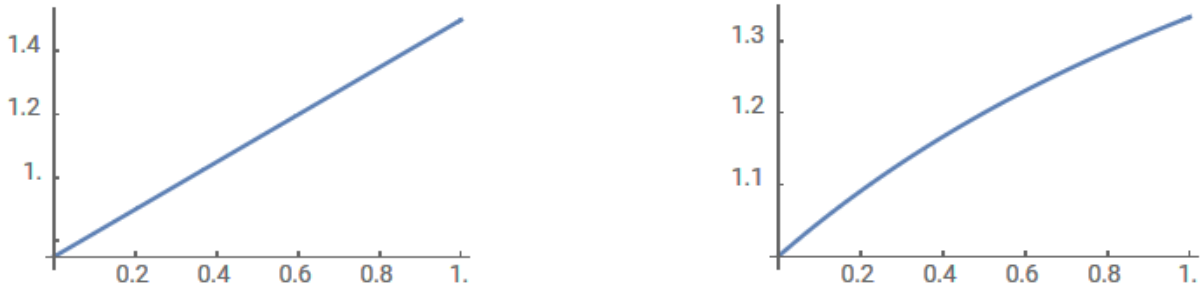
BitFun tackles an important class of queries not explicitly considered before in the context of Array (Tensor) DBMSs: tunable queries. BitFun explicitly focuses on the fact of tunability [53]. At the time of the BitFun invention, modern Array (Tensor) DBMSs were not equipped with mechanisms of indexing tensor cells: SciDB [15], TileDB [46], RasDaMan [48], PostGIS [47], and Oracle Spatial [43]. Array indexing approaches did not explicitly consider tunable scenarios [11, 76].

Before BitFun, state-of-the-art Array (Tensor) DBMSs approaches treated two subsequent queries of computing SAVI with slightly different L values (e.g. 0.7 and 0.8) as two distinct queries. This triggered computing SAVI for the new L value from scratch despite the fact that usually only a small fraction of the resulting tensor (array) will differ significantly from the previous result. This led to wasting I/O and compute resources [53].

[‡]Input reflectances can be corrected for the molecular effects

2.3.2 Novel Tunable Function Indexing Techniques

We developed novel indexing techniques for 3 types of tunable queries: (1) computing $f(\tau)$ values, (2) classification of $f(\tau)$, (3) inequality evaluation $f(\tau) < const$, where τ is a tunable parameter and $f(\tau)$ is a differentiable, possibly non-linear function [53]. Let us briefly describe key ideas of (1) by taking SAVI as an example.



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

Figure 2.11: SAVI values for the whole range of $L \in [0, 1]$

Let us fix NIR and R values. Now SAVI becomes a function of only 1 variable. If we look even closer, we will notice that SAVI is almost a line in the vast majority of cases, that is, for most combinations of NIR and R values, fig. 2.11. In general, SAVI is not an affine function. There are NIR and R values for which SAVI exhibits slightly non-linear behavior, fig. 2.11b. However, even in such cases, our techniques can perform efficient indexing with good precision.

We can approximate SAVI by a significantly simpler expression, a linear fit: $aL + b$, where $a, b \in \mathbb{R}$. To construct it, we need two pairs of values of L and SAVI to compute a and b. As indexing is interleaved with query answering in a DBMS (computing $f(\tau)$), we can already have one of the pairs for each array cell. We can derive the second pair by selecting some L in its value range. However, not any L can be used to compute a and b, as the potential approximation error can exceed the user-provided tolerance.

The solution to $[\text{SAVI}(L) - (aL + b)]'dL = 0$ yields the maximum error. We obtain several roots after solving and simplifying the solution: $L_{1,2} = \pm(\sqrt{a(\text{NIR} - \text{R})(\text{NIR} + \text{R} - 1)} \mp a(\text{NIR}+\text{R}))/a$. Afterwards, we can use $L_{1,2}$ to compute the potential error and make further indexing decisions [53].

Now we have a set of pairs a, b for each cell. Next we assign a unique $\texttt{ID} = (\bar{a}, \bar{b})$ to $\forall \overline{f} = aL + b$ such that $\bar{a}, \bar{b} = a\times, b \times 10^{|\lg(\Upsilon)|}$, where Υ is the user-specified precision. Next, our novel data structure indexes {(ID, Freq)}, where Freq is the frequency of ID occurrence, section 2.3.3.

2.3.3 A New and Fast Hierarchical Data Structure

Our novel hierarchical data structure can index the objects {(ID, Freq)} defined in section 2.3.2. The data structure is fast to create and read [53] and makes it possible to answer tunable queries by up to 8× faster compared to previous approaches, section 4.2.

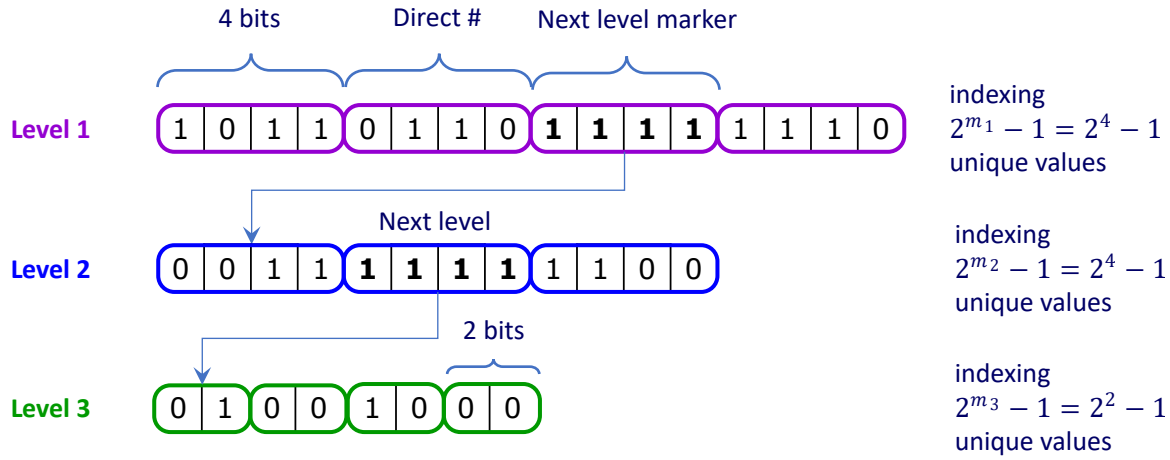


Figure 2.12: Bitmap Index Example [53]

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 (tensor) to be indexed (the array/tensor notation is defined in section 2.1.2), $n \leqslant |A|$ and $\text{Freq}(e_j)$ is the count of e_j in A, where $j \in [1, n]$. Level i (L_i) is a 1-d array where each cell is a fixed-length code of m_i bits, where $m_i \leqslant \lceil \log_2 |A| \rceil$. The index is hierarchical with at most K levels, where $K \in \mathbb{N}$, fig. 2.12.

For example, the structure in fig. 2.12 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 0011 from level 2 to continue. These bits index possibly less frequent object e_{16} ($2^{m_1} - 2 + 1 = 15$).

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, Input/Output Operations per Second (e.g., in the Cloud).

The motivation behind the index is that real-world arrays do not typically contain random data. For example, a typical Landsat 8 satellite scene with $\approx 64 \times 10^6$ cells and diverse land cover, can be indexed only by ≈ 650 unique linear fits of SAVI, 50% of which may approximate 99% of cells and show an exponential usage frequency distribution.

2.4 New R&D Direction: Physical World Simulations Entirely Inside Array (Tensor) DBMSs

2.4.1 Rationale, Shortcomings & Benefits

Now we present a novel Research and Development direction in the area of Array (Tensor) DBMS that opens a wide range of promising research opportunities [56, 61]. Array (Tensor) DBMSs manage large multidimensional array (tensor) storage, processing, and even visualization and machine learning in some cases. Physical world simulations have been traditionally implemented on diverse types of grids and meshes that can be modeled as N-d arrays (tensors). As multidimensional arrays are at the core of Array (Tensor) DBMSs, it is logical to apply these systems to simulations in order to benefit from Array (Tensor) DBMS capabilities.

Various grids, meshes, swaths, and many other data types are stored in computer systems during simulations or data exchange. Many of such data types, even exotic ones, can be represented in well-known array data models and conventions (detailed examples are in [6, 39]), mappable to our new data model, section 2.1.

From one point of view, simulation scenarios open a wide range of new R&D opportunities and expand the area of Array (Tensor) DBMS applications. From a modeler perspective, an Array (Tensor) DBMS can provide numerous benefits and DB-style simulation data management. We leveraged our data model, section 2.1, and started integrating simulation into Array (Tensor) DBMSs from Cellular Automata (CA). We pioneered the incorporation of simulation capabilities into an Array (Tensor) DBMS using Traffic Cellular Automata (TCA) at SIGMOD 2021 [56].

Currently, the application of an Array (Tensor) DBMS (SimDB, section 3.3) to CA simulations has some shortcomings. To date, SimDB introduces scheduling overhead to CA simulations and users need to write 2 types of UDFs (User Defined Functions): in Java and a new, but relatively simple language. We believe these shortcomings are very minor and future R&D efforts will alleviate or even eliminate them. Hence, numerous benefits outweigh a few 'cons' against using SimDB.

We showed that SimDB Array (Tensor) DBMS can be used for end-to-end CA simulations, from initialization to computing the resulting statistics. It turns out that SimDB is an excellent choice for this workload. This immediately showcases numerous benefits of using an Array (Tensor) DBMS, and SimDB in particular, for CA simulations:

- **Data Ingestion and Fusion**. CA may use diverse datasets as inputs, so SimDB can readily prepare (e.g., ingest, resample, and slice) data for simulations.
- **Automatic Parallelization**. SimDB runs the simulation in parallel, managing all necessary data exchanges between cluster nodes in the Cloud.
- **Debugging UDFs**. It is easy to debug step-by-step the UDFs utilized by SimDB, prepared for a CA model.
- **Interactive visualization** is essential for data understanding, so SimDB provides CA lattice imagery via the open, popular protocol, called OGC WMTS [75].
- **Data management**. A user can archive, query, and compare simulation data with the help of SimDB.
- **Interoperability**. SimDB storage layer is built on top of raw files in standard formats. Simulation arrays are full-fledged, georeferenced GeoTIFF files readily accessible to other software.
- **End-to-End Simulations**. SimDB serves all phases of the simulations within the single system, even computing statistics (the goal of simulations).

2.4.2 New Traffic Cellular Automaton (TCA)

To enable simulations in Array (Tensor) DBMSs, we started with the area of road traffic simulations, as they are quite important in practice and used to plan road changes, optimize traffic lights, analyze throughput, and integrate objects, to name a few [3]. In addition, CA road traffic models can be very complex CA models which are challenging to simulate.

We based on the literature (a good survey of CA road traffic models is in [32]) to investigate which entities (e.g., vehicles, traffic lights, road intersections) can be represented by a cellular automaton. We designed a new Traffic Cellular Automaton (TCA) that is sufficiently complex in order to challenge Array (Tensor) DBMS principles, section 2.4.3. The solutions to the challenges will increase the efficiency of simulations entirely inside Array (Tensor) DBMSs, help them become more robust systems in general, and provide researchers and practitioners in the simulation domain benefits specific to Array (Tensor) DBMSs, section 2.4.1.

Our CA model has 4 traditional components: the physical environment, cells' states and neighborhoods, and local transition rules [56], fig. 2.13.

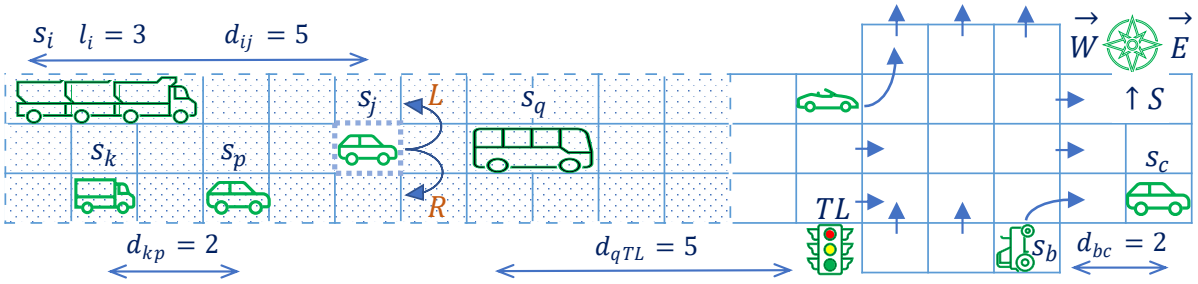


Figure 2.13: Traffic Cellular Automaton [56]

In turn, the physical environment of our model represents the entire road network as a 2-d lattice (a 2-d array). The distinctive features of the physical environment of our model are listed below and rely on fig. 2.13.

- A lattice has diverse cell types, e.g., vehicles & impassable parts.
- We support multiple traffic lights located throughout the lattice (TL).
- We account for vehicles of different lengths (denoted by l_*).
- Vehicles can have different moving directions, speeds (denoted by s_*).
- The model considers road intersections (RI) regulated by traffic lights.
- Vehicles can change their moving directions at RI (see s_b).
- Each road can consist of multiple lanes; vehicles are allowed to change lanes, e.g. in order to overtake a slower-moving vehicle (see s_j).
- It is possible to locate vehicles on any cell of a road.

Cells' states and neighborhoods have the following peculiarities.

- Cells change their states at discrete time steps (iterations).
- All cells in a certain window constitute the local neighborhood of a cell, including the cell itself, 5 cells back and forward (see s_j).
- Local transition rules are applied to all cells simultaneously (drivers make decisions independently of each other, in parallel, but obeying some generally accepted rules).

A large and diverse set of local transition rules are responsible for the correct evolution of the TCA in time and can be categorized as follows.

- Move Forward Rules account for vehicle lengths and include acceleration, braking, randomization, and movement rules that avoid vehicle collisions and account for individual driver behavior.
- Lane Change Rules allow for left and right lane changes. They check for constraints for inter-vehicle distance (e.g., d_{ij}) and vehicle speeds.
- Traffic Light Rules help to avoid collisions at RIs. Traffic lights can be red, yellow, and green. Vehicles can queue in front of traffic lights.
- Road Crossing Rules regulate vehicle behavior at RIs. Vehicles check for diverse constraints & can turn left/right (see d_{bc}).

2.4.3 Challenges & New Enabling Components

It might seem that Array (Tensor) DBMSs can readily run CA simulations, as they have a common data model. However, such simulations pose sophisticated design challenges to Array (Tensor) DBMSs. We solved multiple design challenges to enable CA simulations possible inside Array (Tensor) DBMSs related to iterative workloads, CA transition rules, parallel execution, and other challenges which we briefly summarize below.

Design Challenge 1. *How to support arbitrary cellular automata local transition rules in Array (Tensor) DBMSs?*

A CA rule looks like a well-known convolution operator [64]. However, a CA rule is much more complex: it is a procedure that (1) is applied for each cell of several input arrays, (2) checks for constraints, (3) may update several cells within the neighborhood.

To support arbitrary CA rules, we introduced a new convolution operator for Array (Tensor) DBMSs, section 3.3.1.

Note that we cannot simply extend our Array (Tensor) DBMS model to store tuples in a cell as it will lose compatibility with industrial software, popular, standardized file formats, and the ability to work in situ. Some formats support records, but they are complex and rarely used in practice.

Design Challenge 2. *How to efficiently (natively) support iterations directly inside an Array (Tensor) DBMS?*

Iterations are inherent to physical world simulations. Python/C++ UDFs (User Defined Functions) are supported by some Array (Tensor) DBMSs, so a user may code iterations in these languages. However, such UDFs are black-boxes that Array (Tensor) DBMSs cannot optimize [36]. Existing query languages are unable to express iterations. Running iterations query by query requires a query output to be completely materialized before the next query. In addition, holistic optimizations for several iterations ahead are unavailable as the overall picture is unclear.

To solve this challenge, we introduced the first native Array (Tensor) DBMS language for UDFs, section 3.3.2.

Design Challenge 3. *How to correctly and efficiently execute a native UDF for an Array (Tensor) DBMS?*

Although the UDFs written using our new language look relatively small, they are challenging to execute. We introduced numerous new Array (Tensor) DBMS components, including proactive simulation plans, versioning, and locking to solve the challenge, section 3.3.3.

2.5 New Scalable Data Science Techniques

Machine Learning (ML) and Data Science (DS) are just paving their ways to Array (Tensor) DBMSs [52, 72]. Although the integration of ML and DS techniques into an Array (Tensor) DBMS is facing multiple challenges, it also has numerous benefits. One of such major advantages is the absence of time-consuming data movements between Array (Tensor) DBMSs and ML/DS systems in order to run ML/DS algorithms on data under the management of an Array (Tensor) DBMS. An example of an important practical application is the fast creation of a high-quality seamless mosaic.

2.5.1 Array (Tensor) Mosaicking Challenges

The MOSAIC operator, found in Array (Tensor) DBMSs, ingests a massive collection of overlapping tensors into a single large tensor, called mosaic. The operator can apply sophisticated ML/DS techniques to generate a high quality seamless mosaic: the visible contrasts between the values of cells taken from input overlapping tensors is reduced as much as possible.

For example, CHRONOSDB has the MOSAIC command [54], ORACLE SPATIAL provides the `SDO_GEOR_AGGR.mosaicSubset` procedure [42], and RASDAMAN is equipped with the MOSAIC recipe [49].

High-quality mosaicking techniques rely on advanced approaches. For example, IR-MAD (Iteratively Re-weighted MAD), where MAD stands for Multivariate Alteration Detection, utilizes Canonical Correlation Analysis (CCA) [29]. CCA is a popular approach for determining correlations in multidimensional datasets [23]. CCA is widely used in Data Science for dimensionality reduction and discovering latent variables.

However, the performance bottleneck becomes a major challenge when applying such advanced techniques to increasingly growing tensor volumes. Recent surveys have identified speed as a key challenge and improvement aspect of next generation mosaicking techniques [29, 31].

As an example, consider the following use-case. A single tensor, a satellite scene, often does not fully cover the area of interest. In this case, the mosaic operator, equipped with Data Science techniques, is used to fuse a large collection of input tensors into a single seamless mosaic, a large multidimensional array where the visibility of stitches between individual input arrays is reduced as much as possible. To produce high-quality mosaics, algorithms resort to the advanced techniques mentioned above. Figures 4.15 and 4.16 illustrate the input scenes and the resulting mosaic.

2.5.2 Scaling MAD & IR-MAD

FastMosaic enhances IR-MAD (using a new, scalable way to CCA), as it can run with no or little manual interaction: an important property for Big Data applications. IR-MAD can also produce high-quality tensor mosaics, even for complex value distributions of overlapping input cells [29].

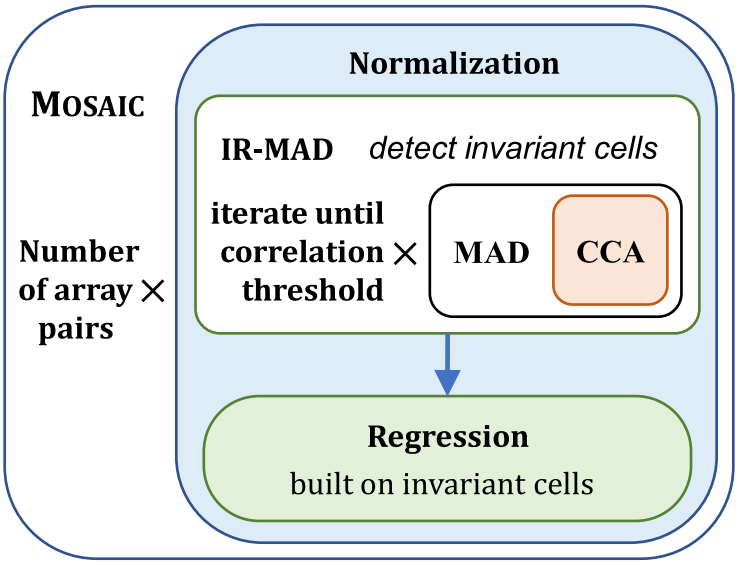


Figure 2.14: FastMosaic Structure

Figure 2.14 presents the FastMosaic structure. As input, FastMosaic takes a set of overlapping arrays (tensors) and performs pair-wise fusion of overlapping arrays (tensors). We treat M cell pairs of two overlapping arrays (tensors) as a pair of random variables X and Y of dimension K (X and Y represent the subject and the reference array/tensor respectively): $X_k = \{X_{k,1}, X_{k,2}, \ldots, X_{k,M}\}$, where $X_{k,j}$ is the value of cell $j \in [1, M]$ and spectral band $k \in [1, K]$. We define Y in the same way [59].

For each cell pair, MAD estimates the probability of no change P_j using our new, scalable way to perform CCA (section 2.5.3) on weighted $X_k w$ and $Y_k w$, where $w = \{w_1, w_2, \ldots, w_M\}$. IR-MAD iterates while w changes significantly. Cells with $P_j > \Theta \in [0.95, 0.99]$ are called invariant. The relative normalization builds an orthogonal regression on invariant cells $Y_k = \beta_k X_k + \epsilon_k$, where $\beta_k, \epsilon_k \in \mathbb{R}$, to get K pairs of transformation coefficients β_k, ϵ_k to apply to all cells of the subject array (tensor). Finally, the transformed array (tensor) is fused with the reference one; this larger array (tensor) replaces the pair in the input set. These steps are repeated until only one array (tensor) appears in the set: the resulting mosaic.

Section 4.4 demonstrates step-by-step construction of a mosaic on real-world geospatial arrays (tensors).

2.5.3 Scaling Canonical Correlation Analysis (CCA)

Recall that CCA maximizes the correlation $corr(a^T X, b^T Y)$ by seeking coefficients a and b. Random variables $U = a^T X, V = b^T Y$ are called a pair of canonical variables. We designed linear-time formulae to compute U, V together with IR-MAD transformation coefficients β_k, ϵ_k (section 2.5.2) in the same pass over the input data in $O(M \times K^2 + K^3)$, fig. 2.15.

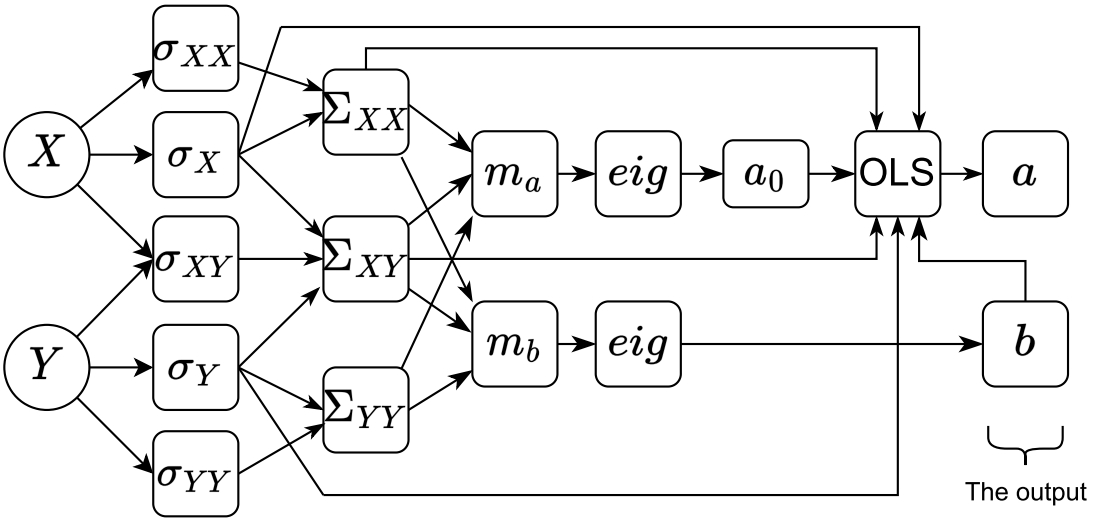


Figure 2.15: CCA Compute Graph: Proposed Approach [59]

The idea is to collect certain statistics (takes over 95% of total runtime on real-world data) to compute U, V and β_k, ϵ_k (5% of total runtime), all in the same pass. This equips Array (Tensor) DBMSs with a scalable and high-quality MOSAIC operator. First, we compute $\sigma_{X,k} = \sum_j^M X_{k,j} w_j$, the weighted sum of cells for band k: $\sigma_X = \{\sigma_{X_1}, \sigma_{X_2}, \ldots, \sigma_{X_K}\}$. Then we compute σ_{XY}, a matrix of weighted sums of products X and Y: $\sigma_{XY} = X_k^T (Y_k \odot w)$, $\sigma_{XX} = X_k^T (X_k \odot w)$, $\sigma_{YY} = Y_k^T (Y_k \odot w)$, where $\odot$ is a cell-wise product. In addition, we compute Σ_{XY}, Σ_{XX}, and Σ_{YY}, matrices of weighted covariances (formulae for Σ_{XX} and Σ_{YY} are similar):

$$\Sigma_{XY} = \frac{\sigma_{XY}}{\sum w - 1} - \frac{\sigma_X \sigma_Y^T}{\sum w (\sum w - 1)}$$

We use these statistics to compute matrices m_a, m_b, vectors a_0, b as eigenvectors of matrices m_a, m_b, derive a and $M = \{M_1, M_2, \ldots, M_K\}$, where $M_k = ((U_k - V_k) - \overline{M_k})/std(M_k)$ to estimate $P(no\ change) = \chi^2_{cdf}\left(\sum M_k^2\right)$ and get β_k, ϵ_k in the same pass. Complete formulae are in [59].

Chapter 3

Software: Architectural & Implementation Aspects

In the context of systems research, it is insufficient to propose theoretical results on their own: we must also have appropriate architectures and implementations, presented in this chapter, to ensure and reinforce the algorithmic impact and compete with popular software systems. The novel aspects provide DBMS-style tensor management and efficiently organize our tensor algorithms and techniques to yield significant accelerations.

The results of this chapter are new architectural and implementation aspects of ChronosDB (section 3.1), BitFun (section 3.2), SimDB (section 3.3), WebArrayDB (section 3.4.2), ArrayGIS (section 3.4.3), and FastMosaic (section 3.5). Chapter 4 presents additional aspects.

ChronosDB outperforms SciDB **by up to 75× on average**. It is always faster and can outperform SciDB **by up to 1024×**. SciDB is developed by the Paradigm4 company under the supervision of M. Stonebraker, an ACM Turing Award Recipient ("Nobel Prize of Computing").

BitFun is equipped with novel tensor indexing strategies to continuously re-index tensors during queries with similar mathematical functions. It can be **up to 8× faster** than computing the results from scratch.

SimDB is the first Array (Tensor) DBMS that can run end-to-end simulations entirely inside itself: from data preparation to simulation to computing statistics, with runtime **competitive to hand-written code**.

WebArrayDB is the first Array (Tensor) DBMS that runs completely inside a Web browser. ArrayGIS is a new Web GIS based on WebArrayDB. In tandem, they can be **over 2× faster** than querying only Sentinel-Hub, a popular Cloud service for providing Sentinel data.

FastMosaic is a new Array (Tensor) DBMS operator that can run an **order of magnitude faster** for array (tensor) mosaicking than the popular software library.

3.1 ChronosDB: An Innovative Array (Tensor) DBMS

3.1.1 ChronosDB Architecture & Components

Figure 3.1 presents the ChronosDB architecture with some of its key components and peculiarities.

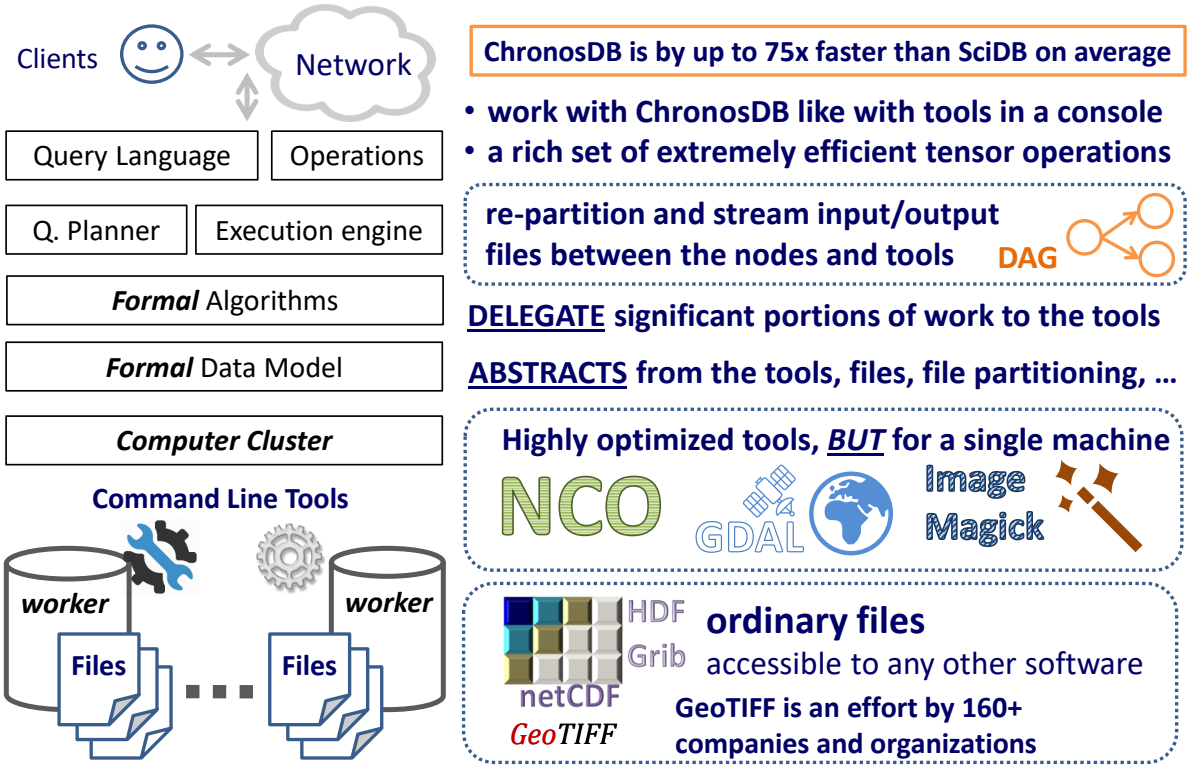


Figure 3.1: ChronosDB Architecture: A Bird's Eye View

The formal data model abstracts from the files and treats related files as a single large tensor, section 2.1. Files can be distributed among cluster nodes and processed in parallel. This enables ChronosDB to work on a computer cluster of commodity hardware, possibly in the Cloud, and base its storage layer on established, standardized file formats [54].

Moreover, unlike other DBMSs, ChronosDB works in situ (directly with diverse file formats like NetCDF and GeoTIFF) and omits the time-consuming import phase into an internal DBMS format. This approach provides powerful storage capabilities, including efficient support of diverse data types, missing values, and numerous compression techniques out-of-the-box. In addition, ChronosDB arrays are readily accessible to any other software as a set of ordinary files. For example, to Geographic Information Systems (GISs), R, Python, and many other software that can read such files.

The model also facilitates designing formal tensor management, processing, and visualization algorithms, section 2.2. They are designed to work in situ and delegate significant portions of work the elaborate and optimized command line tools, section 4.1.1. Novel architectural and implementation aspects make it possible for the algorithms implemented in ChronosDB to run with exceptional efficiency [54].

In particular, ChronosDB features a sophisticated query planner and execution engine. ChronosDB re-partitions and streams input/output files between the nodes and tools to scale out the processing. For this purpose, execution plans are built, section 3.1.3. ChronosDB exploits multi-core CPUs and can benefit from multi-disk cluster nodes.

Finally, from the user perspective, it is easy to work with ChronosDB even for a novice, as it provides a query language resembling the syntax of well-known command line tools. We showcase this in section 3.1.2.

3.1.2 Novel Tensor Management Approaches

Let us form a dataset to serve as a running example: 24937×38673 arrays R and NIR, the mosaic of a set of 4×8 Landsat 8 scenes, bands 4 (visible red, R) and 5 (near-infrared, NIR), paths 191–198, rows 24–27, 01–15 July 2015, GeoTIFF. Landsat is the longest continuous space-based record of Earth's land running from 1972 onwards [27]. Amazon and Google provide Landsat scenes via commercial clouds due to Landsat's popularity [18].

Unlike any existing Array (Tensor) DBMSs, ChronosDB maintains a hierarchical Dataset Namespace to make it easier to navigate in a large number of datasets. For example, the fully qualified name of a dataset can look like `Landsat8.Level_1.SurfaceReflectance.Band4` (dots separate the names of dataset collections, the dataset name is the last).

System-level tensors (subarrays, files of diverse formats) can be placed on cluster nodes by manual copying or via a special ChronosDB command. In contrast to parallel or distributed file systems, ChronosDB always keeps a file entirely on a node. It is possible to replicate a subarray over several nodes for fault tolerance and load balancing.

Workers are designed to discover their datasets upon startup. They receive the dataset hierarchy from the coordinator node and scan their local storage systems to report the dataset properties back. They obtain this information by parsing subarray file names or reading file metadata.

ChronosDB performs Dataset Ingestion before querying a given dataset (a set of files in a certain format). Whereas existing Array (Tensor) DBMSs

convert files (or require a third-party tool to convert the files) into an internal DBMS format, ChronosDB works with files in situ, in their original formats. In this way, ChronosDB eliminates a time-consuming and error-prone phase of format conversion that may run longer than a typical query over the given data. ChronosDB makes some assumptions about the given files, e.g., the same set of metadata keys.

Typically, data providers disseminate their products as a set of files that already satisfy the criteria for ChronosDB regular datasets. However, even if ChronosDB classifies the files as its raw dataset, it is possible to quickly "cook" them into a regular dataset by specific ChronosDB commands. The ingestion of a raw dataset is still orders of magnitude faster compared to the state-of-the-art import procedures [54].

Array (Tensor) DBMSs can also print an array (tensor) schema, a notion which is also found in other, e.g. relational, DBMSs. Array (tensor) schema notations greatly differ between Array (Tensor) DBMSs. The ChronosDB schema for Band4 from our running dataset is presented below and is from [54].

```
gdalinfo Landsat8.Level_1.SurfaceReflectance.Band4
Driver: GTiff/GeoTIFF
Size is 38673, 24937
Coordinate System is:
PROJCS["WGS 84 / UTM zone 32N", ...skipped... AUTHORITY["EPSG","32632"]]
Origin = (-53110.000000000000000,5878570.000000000000000)
Pixel Size = (30.000000000000000,-30.000000000000000)
Metadata:
  AREA_OR_POINT=Area
Corner Coordinates:
Upper Left  (  -53110.000, 5878570.000) (  0d47'37.30"E, 52d46'20.14"N)
Lower Left  (  -53110.000, 5130460.000) (  1d50'34.16"E, 46d 6'11.15"N)
Upper Right ( 1107080.000, 5878570.000) ( 17d59'48.38"E, 52d42'52.21"N)
Lower Right ( 1107080.000, 5130460.000) ( 16d50'57.11"E, 46d 3'26.67"N)
Center      (  526985.000, 5504515.000) (  9d22'26.95"E, 49d41'33.18"N)
Subarray=2048x2048 Block=2048x1 Type=UInt16, ColorInterp=Gray
  NoData Value=0
```

The meaning of the fields in output above is the same as for the well-known `gdalinfo` tool that handles only one file at a time. As `gdalinfo` is quite popular, most users avoid learning a new array schema notation and inspect ChronosDB tensors in a way they are accustomed to, unlike with other Array (Tensor) DBMSs.

Metadata Management in ChronosDB takes place via the commands that mirror the functionality of the respective command line tools. For example, ChronosDB provides the capabilities of `ncatted` (attribute editor, NCO) to append, create, delete, modify, and overwrite internal file metadata [41]. However, unlike the `ncatted` tool, the `ncatted` command applies to a potentially large number of files simultaneously.

3.1.3 New & Efficient Query Execution Techniques

We illustrate our new efficient query execution techniques for Array (Tensor) DBMSs using the following complex analytic pipeline that is detailed in section 3.4 of [54]. As input, we take R and NIR arrays (tensors) with different subarray shapes to avoid their collocation on cluster nodes and trigger the distributed retiling that involves network exchange (section 2.2.1):

1. Interpolate 2× Band4 ↦ Warp4
2. Interpolate 2× Band5 ↦ Warp5
3. Perform 2-way array (tensor) join of Warp4 and Warp5 ↦ SAVI
4. Downsample 64× SAVI ↦ SAVIoutlook

First, the 2× interpolation is performed (section 2.2.3) which outputs are joined (section 2.2.2) to compute SAVI (section 2.3). Finally, a small 1209×780 "quick outlook" array (tensor) (64× less than the join output) is derived to visually estimate the result.

Let us refer to the above scenario as the SAVI pipeline.

ChronosDB builds and works according to execution plans. They are used to determine the order of producing subarrays and their exchange between workers. It is possible to build such plans due to the formal data model and formal algorithms that have well-defined inputs/outputs.

Formally, an execution plan is a directed acyclic graph $G = (V, E)$, where $E = \{((\mathbb{D}_i, key_i^j), (\mathbb{D}', key_m)) : \mathbb{D}_i\langle\langle key_i^j \rangle\rangle$ is required to compute $\mathbb{D}'\langle\langle key_m \rangle\rangle\}$. For an array operation, $\mathbb{D}_i$ is an input dataset and $\mathbb{D}'$ is the output dataset. The formal dataset definition is in section 2.1. Execution plans are static (immutable at runtime) which makes it easier to reason about the data flow. An execution plan example is in section 3.4 of [54].

ChronosDB commands cannot be nested, but output datasets can be inputs to another commands. This clear and easy-to-use approach is not a limitation or performance penalty. ChronosDB does not materialize the whole output of a command before launching the next command.

In the pipeline, Warp4, -5, and SAVI can be Intermediate Datasets. They can be reused in the pipeline, but cannot be registered as permanent datasets. Intermediate Datasets are subject for diverse optimizations [54].

Intermediate (or even resulting) system-level tensors may grow too large or become too small. This may cause load imbalance or increase runtime. ChronosDB can rewrite execution plans to avoid this. If we find operations that yield subarrays whose sizes deviate 2× of a given threshold, we insert the retiling to merge/split small/large subarrays. To maintain load balance, a retiling can also be inserted into the execution plan.

3.2 BitFun: Fast Answers to Tunable Queries

BitFun is a ChronosDB [54, 55] component that provides novel bitmap indexing techniques for queries with tunable mathematical functions and novel, space-efficient hierarchical bitmap index structure to support tunable indexing. BitFun features a specialized architecture, a novel indexing workflow, a specifically designed user interface (section 3.2.2), as well as a number of important applications (section 4.2).

3.2.1 BitFun Architecture

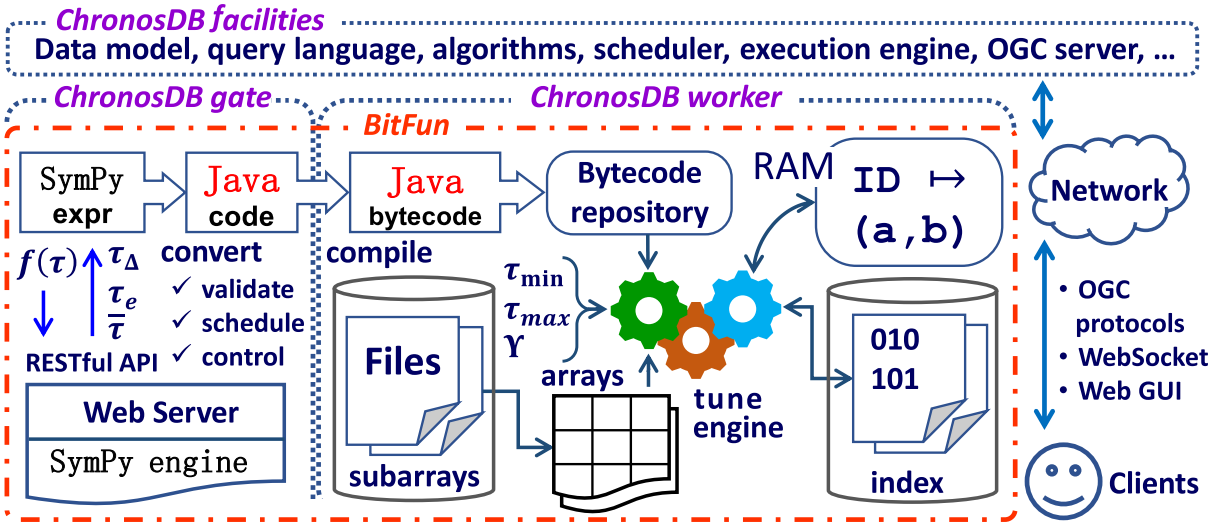


Figure 3.2: BitFun Architecture [53]

As BitFun is a ChronosDB component, its primary language is Java, but it also uses Python for its Web server. Java lacks symbolic computing libraries. Hence, BitFun utilizes SymPy in order to find derivatives, solve equations, simplify expressions, and perform other required steps according to the BitFun approach, section 2.3.2.

Jython and similar tools are limited in functionality compared to SymPy. Therefore, as part of BitFun, we developed a Web server in Python in order to submit formulas, related parameters and retrieve the SymPy output via our self-designed RESTful API.

BitFun evaluates symbolic expressions hundreds of millions of times. Hence, BitFun translates the SymPy output into Java code and runs a compiler to produce Java bytecode. The code serves as part of diverse indexing procedures and is soon compiled into machine code. Consequently, the native code executes much faster than evaluating expressions in symbolic form. BitFun architecture is depicted in fig. 3.2.

3.2.2 Interactive User Interface

The Web interface is designed to (a) showcase the BitFun performance in query answering that relies on innovative indexing approaches, (b) make it possible for users to investigate the novel index structure and explore indexing workflow insights, (c) provide an opportunity to extend user knowledge of Array (Tensor) DBMSs and real-world tunable query applications.

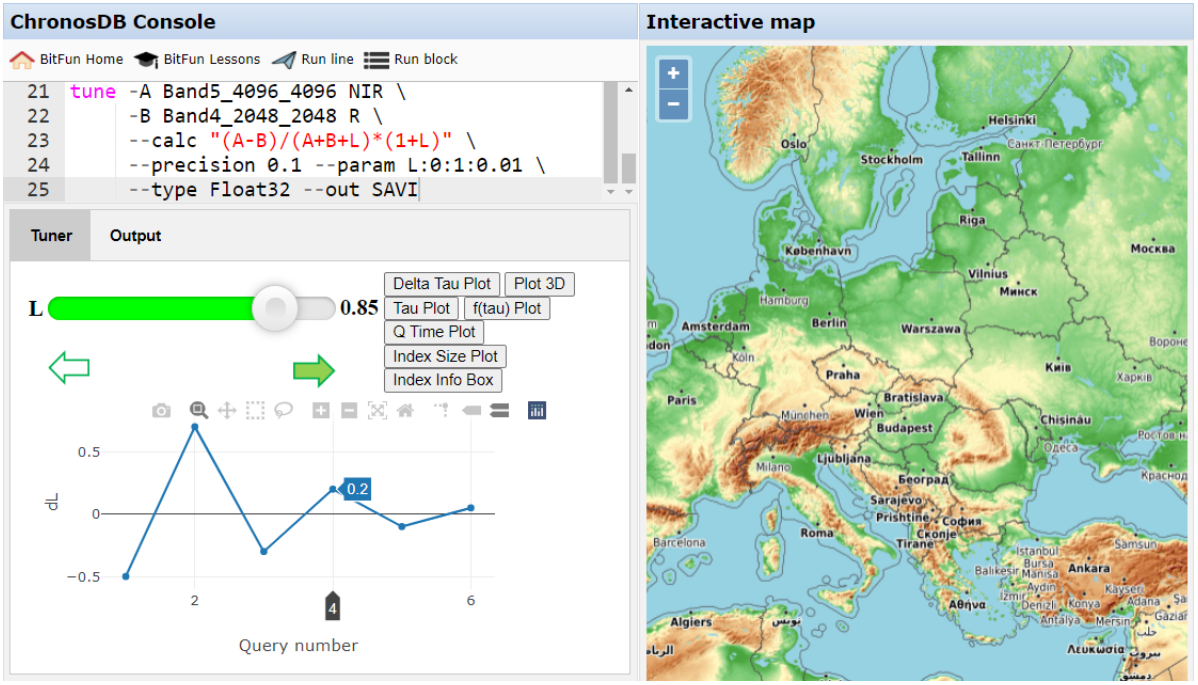


Figure 3.3: BitFun Web GUI with Some of its Components

The Web interface consists of (1) the syntax-highlighted editor for creating and submitting queries; (2) 2-d and 3-d charts showing the properties of the index, the indexing process, and the specifics of the query answering; visual components (3) to help with expression tuning and (4) to provide lesson guidance; (5) interactive map featuring query results and input data.

The GUI also provides engaging lessons centered around real-world applications. To succeed, the user should find the correct mathematical function parameter value by experimentally tuning it. Each new parameter value triggers the rendering of a new interactive map based on the result of re-applying the function to an input array (tensor).

BitFun can be up to 8× faster than computing the results from scratch. The appropriate map view is produced as a result of using the correct parameter value. The goal is to illustrate fast computations due to novel indexing techniques. During a lesson, the user gets interactive hints.

More information on the lessons and interface components, i.e., tune slider, tune hints, plots, index info box, interactive map, etc. is in [53].

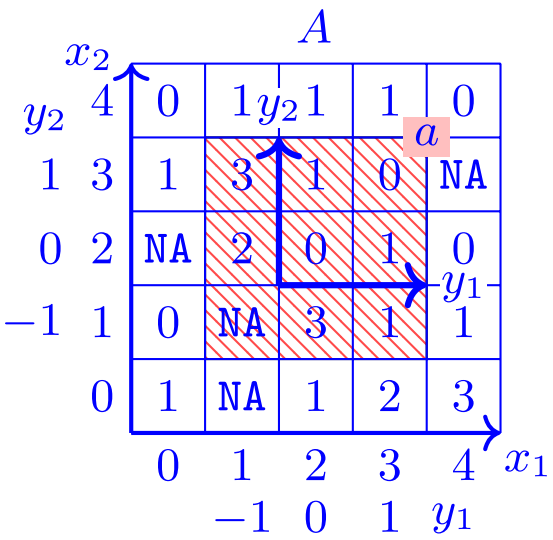
3.3 SimDB: Physical World Simulations Completely Inside Array (Tensor) DBMS

SimDB solves key design challenges (section 2.4.3) to enable end-to-end Cellular Automata simulations entirely inside an Array (Tensor) DBMS for the first time via new Array (Tensor) DBMS components [56, 61].

3.3.1 Novel Array (Tensor) DBMS Convolution Operator

To support arbitrary Cellular Automata (CA) simulations, we introduced a new convolution operator for Array (Tensor) DBMSs, fig. 3.4.

Let us formally define the convolution operator Ξ, leveraging our novel Array (Tensor) DBMS data model, section 2.1. The operator $\Xi : K, A_1, A_2, \ldots, A_n \mapsto B_1, B_2, \ldots, B_m$ takes n input and yields m output 2-d arrays, all shaped $l_1 \times l_2$. A kernel $K\langle k_1, k_2\rangle$ is the mapping $K : a_1^x, a_2^x, \ldots, a_n^x \mapsto b_1^x, b_2^x, \ldots, b_m^x$ of n input (windows of A_j, fig. 3.4) and m output 2-d arrays, all shaped $2k_1 + 1 \times 2k_2 + 1$, indexed by $(y_1, y_2) : y_i \in [-k_i, +k_i] \subset \mathbb{Z}$, $a_j^x[y_1, y_2] = A_j[x_1 + y_1, x_2 + y_2]$, $j \in [1, n]$, $a_j^x[y_1, y_2] =$ NA if $x_i + y_i \notin D_i$, $k_i \leqslant |l_i|$ div 2, $B_q[u_1, u_2] \in \{b_q^x[y_1, y_2] : x_i + y_i = u_i\}$ (choose the latest produced value), where $q \in [1, m]$, $x = (x_1, x_2)$, $x_i, u_i \in D_i$ [56].




```
interface ConvolveWindow<T> {
   enum Degrees
     {_0, _90, _180, _270}

   int getArrayX();      // x1
   int getArrayY();      // x2
   int xWindowSize();    // k1
   int yWindowSize();    // k2
   int xSubarraySize(); // l1
   int ySubarraySize(); // l2
   T rotate(Degrees degrees);
   Random random();
   void move(int x, int y);
}
```

Figure 3.4: Illustration of the Novel Convolution Operator and Its Interface [56, 61]

Users provide the logic as UDFs in a high-level language, currently in Java. SimDB iterates over arrays, forms read/write windows, equipped with helper functions, fig. 3.4. Unlike a traditional convolution, our operator feeds a convolution UDF several input windows and allows the UDF to modify an arbitrary number of cells within multiple output windows. This enables the operator to produce several output arrays [56, 61].

3.3.2 Native UDF Language for Array (Tensor) DBMSs

To efficiently and natively support simulations directly and entirely inside an Array (Tensor) DBMS, addressing Challenge № 2, we introduced the first native Array (Tensor) DBMS language for UDFs [56, 61], fig. 3.5.

```
cp tca.speed $speed ↲ [newline] cp tca.length $length
val {window} "-xWindowSize 11 -yWindowSize 11"
foreach {step} -from 0 -to 100 ↲ begin
  calc tca.lane:in $speed:in $length:in tca.temperature:in \
       --overwrite $speed_move:out $length_move:out    \
       -ot Int16 {window} -classfile "TCA.java"        \
       -method_name moveForwardPhase
  ...
  calc tca.lane:in $speed_lights:in $length_lights:in \
       --overwrite $speed:out $length:out              \
       -ot Int16 {window} -classfile "TCA.java"        \
       -method_name lightsPhase
  append $speed speedh ↲ append $length lenh ↲ end
```

Figure 3.5: Part of the TCA simulation native Array (Tensor) DBMS UDF [61]

SimDB UDFs are easy to code: they consist of commands with a syntax similar to command line tools. This syntax is familiar to most users. A part of the UDF for TCA simulations is in fig. 3.5. The traffic simulation consists of the following steps repeated in a loop multiple times:

1. Each vehicle is advanced several steps forward
2. Each vehicle decides whether to turn left and possibly turns left
3. Each vehicle decides whether to turn right and possibly turns right
4. Traffic lights change their color
5. The timer advances

To keep the initial arrays, `tca.speed` and `tca.length` are copied to intermediate `$speed` and `$length` arrays managed with optimizations.

The `calc` command runs a convolution operator supplied as a Java UDF, section 3.3.1. SimDB compiles the Java UDF to bytecode for faster execution. `calc` accepts/produces an arbitrary number of input/output arrays. The `-ot` parameter specifies $\mathbb{T}$, section 2.1. Quantiles `:in` and `:out` distinguish between the in/out arrays as their number is not fixed.

An iteration ends by appending new 2-d arrays `$speed` and `$length` to 3-d arrays `speedh` and `lenh` along the virtual *time* axis.

Virtual axes were also introduced as part of making simulations possible and efficient entirely inside an Array (Tensor) DBMS [56].

3.3.3 New Scheduling, Versioning & Locking Mechanisms

Although the UDF looks small in fig. 3.5, it is challenging to execute. Hence, we immediately face Challenge № 3. We had to teach SimDB, a ChronosDB Array (Tensor) DBMS extension, to schedule execution in a new way by building Proactive Simulation Plans (PSPs) [56, 61].

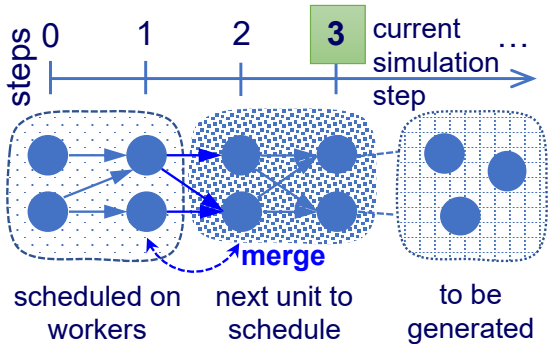


Figure 3.6: Proactive Simulation Plan [61]

For example, during loop unrolling, the same array name appears 100 times: e.g., the last `calc` command deletes current `$speed` array and creates a new array with the same name. We must be able to keep and address all arrays (deleted and new) and write/read all of them simultaneously when we build and execute a simulation plan.

SimDB uses strict formal definitions of array (tensor) operations, section 2.2 and compiler techniques to build and execute PSPs for several iteration steps ahead. This lets SimDB avoid redundant materializations and reduce scheduling overheads; e.g., "`scheduled on workers`" tasks can be executed without communicating with the coordinator, fig. 3.6.

Figure 3.7: Exclusive Locking States: circles are states, arrows are allowed transitions

To physically maintain several arrays (tensors) with the same name during runtime, we introduced array (tensor) versioning and locking mechanisms that work in tandem. SimDB refers to an array (tensor) by its name and version, operating simultaneously on both deleted and new arrays (tensors): (n, v), where n is a fully qualified name (section 3.1.2) and $v \in \mathbb{Z}$ is version. Datasets of different versions but the same name are stored separately. In this way, we build PSPs that contain deleted arrays (tensors) upon which depend other arrays (tensors) [56, 61].

Unlike ChronosDB, SimDB maintains two new additional dataset types: staging and permanent. Given a name n it is possible to acquire an exclusive or non-exclusive lock on n. The latter allows UDFs to share a dataset for read-only purposes. The former allows UDFs to control the dataset state (fig. 3.7) and prevent other UDFs from modifying the dataset. Locked datasets become staging: their metadata is in a special repository, not visible to users. A staging dataset can become permanent, see below.

The lifecycle of a SimDB dataset is as follows. A UDF command must ask the Dataset Pool to acquire an exclusive lock for name n and the latest version if it is going to perform a `stage`: discover (section 3.1.2), read, remove, or create a dataset named n. To overwrite, perform remove & create stages. Note that `remove` is a new stage compared to ChronosDB, causing the most complexity. Read is also supported by non-exclusive locks, so we omit its description. Any stage on a dataset is not immediate, e.g. it is impossible to immediately delete a large distributed dataset, so intermediate states exist, fig. 3.7. A `commit` operation finishes any stage.

If an exclusive lock on n is successful, a new staging dataset is created in state ① if no permanent dataset named n exists or in state ② otherwise. A UDF command can perform the `create` stage: ① initiate create $\mapsto$ ⑤ specify the metadata, fill the dataset with new subarrays, commit $\mapsto$ ③ successfully register the metadata & subarrays $\mapsto$ ②. Similarly for `remove`: ② initiate remove $\mapsto$ ⑦ commit $\mapsto$ ⑧ successfully apply the changes in the Dataset Pool $\mapsto$ ⑨. The metadata and subarrays of a deleted dataset still physically exist and can be referred to by the UDF commands that have previously accessed them. If a dataset named n is deleted, a UDF command can create a new dataset named n with a new version v', but possibly with completely different metadata: via states ⑩, ⑪, and ⑫. Note that v is the same in all states. State ⑫ indicates that (n, v) cannot go any stage further: it is deleted and possibly a newer version exists.

Once a lock is released (e.g., when a UDF completes) and a permanent dataset (n, v) does not exist, SimDB registers a staging dataset in state ② as permanent. Otherwise, SimDB overwrites (n, v) if a newer dataset exists (n, v') or deletes (n, v) if its staging state is ⑨. SimDB garbage collector physically deletes subarrays of other staging datasets.

When users code UDFs, they are not required to know anything about dataset versions, locking, or scheduling: all mechanisms in this section are transparent to users. They enable efficient simulations, but required deep modifications to our Array (Tensor) DBMS [56, 61].

3.4 The First Array (Tensor) DBMS Entirely in a Web Browser

3.4.1 Time to Operate on Tensors in Web Browsers

Big tensor data with its rapid growth stimulates the development of client-server applications, especially Web-based. Therefore, Web browsers are becoming increasingly popular platforms for client applications that provide capabilities of working with tensors: Web GISs (Geographic Information Systems). However, Web GISs are far from being mature because they largely underutilize the power of modern Web browsers.

Contemporary Web GISs perform all array (tensor) processing on the server side. They use diverse protocols, open and/or proprietary, to submit queries (array/tensor processing commands) to the Cloud (server side) and retrieve results in small portions (e.g., 2-d image tiles to display in a Web browser on the client side). This increases response times, up to several seconds, significantly degrading the user experience [63].

Companies claim that only 0.1s of reduction in latency can influence the user journey and ultimately increases conversion rates [37]. ArrayGIS and WebArrayDB, our novel Web GIS and Web-based Array (Tensor) DBMS, prove that Web GISs can submit loads of tensor-related queries to an Array (Tensor) DBMS that runs entirely in a Web browser in order to significantly reduce query response times.

ArrayGIS relies on WebArrayDB. They can be over 2× faster compared to querying only Sentinel-Hub [68], a Cloud service for disseminating and processing very popular Sentinel data, section 4.2.3.

3.4.2 WebArrayDB Organization

WebArrayDB is the first Array (Tensor) DBMS in pure JavaScript that runs entirely in a Web browser [63], fig. 3.8.

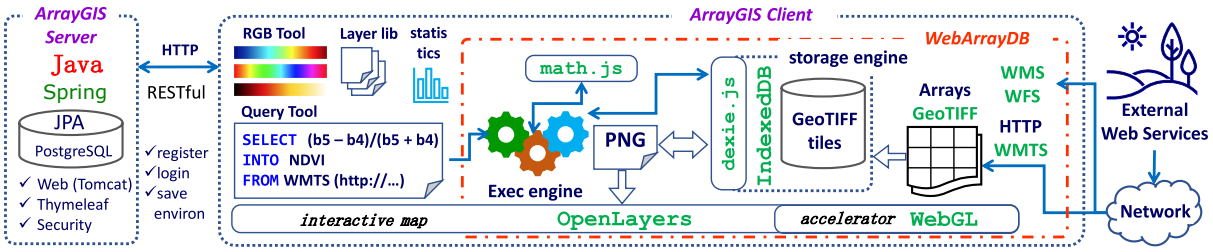


Figure 3.8: WebArrayDB and ArrayGIS Architectures [63]

Data formats. WebArrayDB utilizes both types of formats: those carrying raw data and imagery, e.g. GeoTIFF and PNG. The former formats are quite complex and were traditionally used by desktop software, but recent advancements in Web development have made it possible to operate with GeoTIFF and similar formats in a Web browser.

Data Ingestion. As WebArrayDB runs in a Web browser, it expects other web services (not local files) to be its primary data sources. WebArrayDB can ingest GeoTIFF tiles via the popular OGC WMTS (Web Map Tile Service) [75]. WebArrayDB performs (1) on-the-fly tile-by-tile conversion of GeoTIFF files to the structure supported by OpenLayers (for visualization), and (2) saves raw tiles into the WebArrayDB storage engine for future use.

Storage Engine. WebArrayDB stores raw data tiles as BLOBs directly via a Web browser API: tiles, N-d arrays (tensors), along with certain metadata, e.g., URL key and extent. WebArrayDB keeps on the client side only a limited array data volume, controlled by a parameter. When the space exhausts, the least frequently used tensor tile is deleted.

Query Parsing. WebArrayDB exploits an SQL-like query syntax:

```
SELECT (band8 - band4)/(band8 + band4)
INTO NDVI
FROM WMTS (https://services.sentinel-hub.com/ogc/wmts/
           <personal_api_key>?REQUEST=GetCapabilities)
```

The query computes NDVI, a popular vegetation index [78]. Unlike a conventional SQL, the query takes tensors as input: Sentinel bands 8 and 4 (2-d arrays). The `FROM` clause instructs WebArrayDB to retrieve the inputs from a remote service via the WMTS protocol. As a result, the query also generates tensors. In this case, a 2-d array called NDVI. It will be saved by the storage engine inside the Web browser.

Query Execution plans consist of the following phases: (1) load, (2) join, (3) compute, and (4) render. ArrayGIS and WebArrayDB work in tandem: the former requests the latter to emit only those resulting tensor tiles that will be immediately visible to the user. When the user pans or zooms the map, ArrayGIS and WebArrayDB quickly generate new resulting tiles on-the-fly. Array joins and GPU can be used.

Array Joins. When a query involves multiple arrays (tensors), a K-way array (tensor) join may be required [52]. WebArrayDB supports extracting input tensors (1) from a WMTS response or (2) different layers. An output tensor is tiled using the smallest input tile. Instead of full retiling [54], WebArrayDB emits tiles incrementally [63].

3.4.3 ArrayGIS: WebGIS Components

ArrayGIS is an innovative Web GIS (Geographic Information System) with an interactive Web GUI [63], fig. 3.9. WebArrayDB is the engine of ArrayGIS. A distinctive ArrayGIS feature is that it can operate on raw tensor data directly in a Web browser, as we showcase in section 4.2.3.

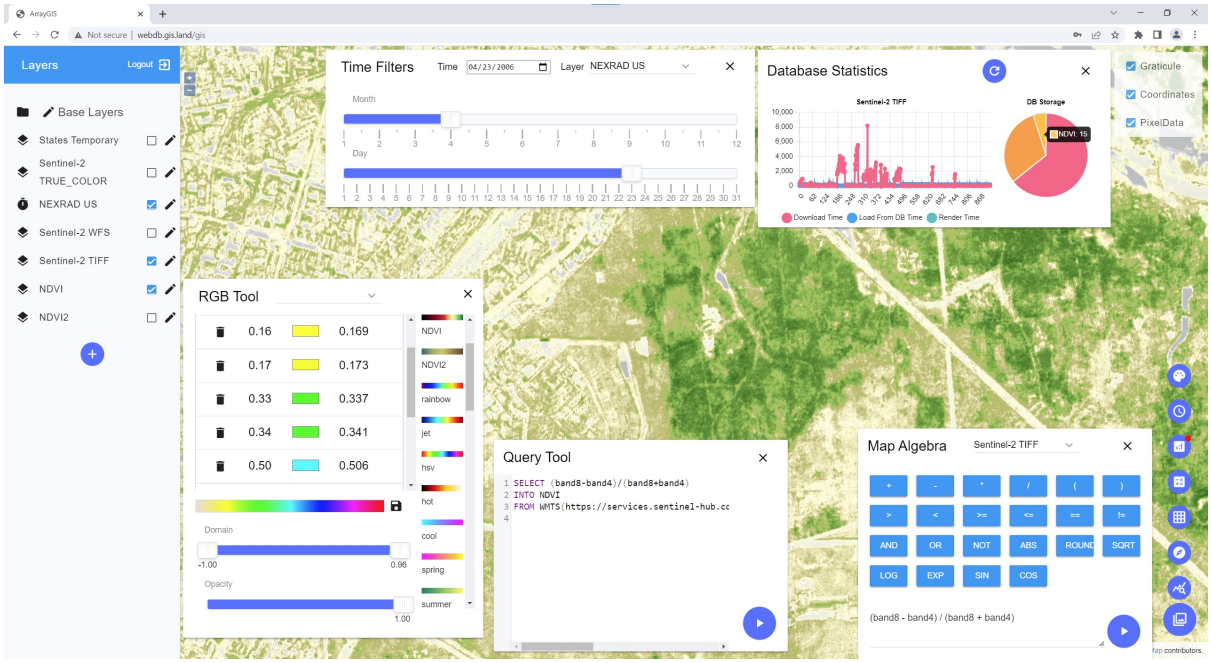


Figure 3.9: ArrayGIS GUI [63]

Layer Library allows users to manage 2-d and 3-d layers via WMS, WFS, and WMTS protocols. GeoTIFF or ordinary images are supported.

Value under cursor tool displays source raw cell values on mouse click events. It is lightning-fast as it happens without client-server communication since ArrayGIS can work with raw source data.

RGB Tool provides predefined color palettes that users can modify or create a new one (e.g., add/remove/re-arrange colors, change opacity). A layer in new colors is re-rendered in a split second, as ArrayGIS can set/tune color palettes without client-server communication.

Map Algebra is a popular analysis language [70] and one of the most frequent Array (Tensor) DBMS workloads [52]. ArrayGIS and WebArrayDB accept SQL queries with Map Algebra expressions and run fast computations directly in a Web browser.

The WebArrayDB and ArrayGIS performance can be experienced via GUI responsiveness and interactive statistical charts in the GUI.

ArrayGIS and a video about it are freely available via its homepage[1].

[1] https://wikience.github.io/webdb2022

3.5 FastMosaic: A Novel, Scalable Mosaic Operator

3.5.1 End-To-End Mosaicking Workflow

FastMosaic realizes techniques from section 2.5, fig. 3.10. In the text we use M and K instead of N and k as in [59]. Two mosaicking modes are available, designed for different purposes. Each mode generates a mosaic.

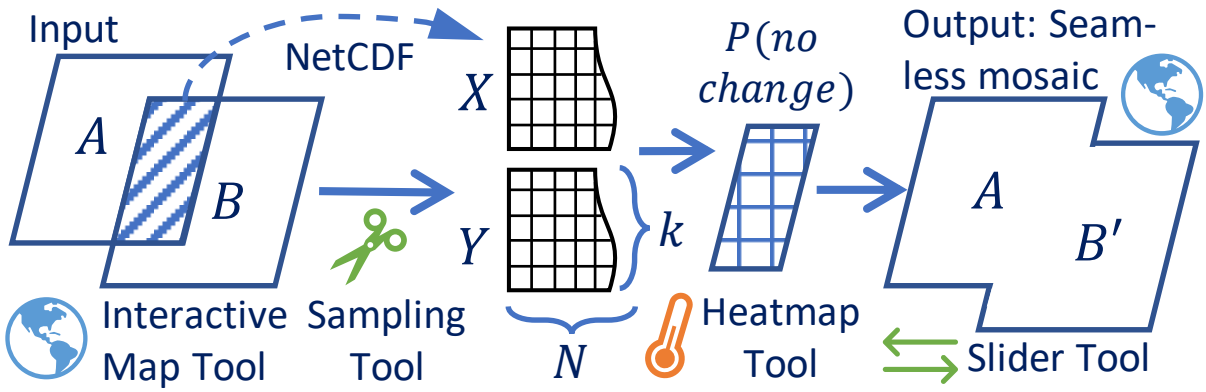


Figure 3.10: FastMosaic Workflow Overview (2 input arrays) [59]

The fist mode, a manual plan, runs in batch on all input arrays using a previously created mosaic execution plan. The user interacts with FastMosaic to provide a plan and get a large output mosaic as a result built on all input arrays according to the plan. This mode invites experimenting with the impact caused by the order of adding tensors to the overall, final mosaic. This is because transformation coefficients are computed for tensor pairs. Hence, they superimpose during the mosaicking which leads to a non-linear transformation of input tensors.

The second mode, a step-by-step user guidance, takes only two tensors as input: the mosaic execution plan that instructs FastMosaic to fuse two tensors. This mode enables users to perform in-depth investigation of FastMosaic internals at each step of the mosaicking process.

In the detailed, step-by-step mode (fig. 3.10), the user can explore input tensors on an interactive map. The user must initiate a series of steps to construct a mosaic of two overlapping tensors. First, the user runs the Sampling Tool to extract cells from the overlapping area of two tensors: X and Y ($M \times K$ arrays, where M is the number of overlapping cells and K is the dimensionality of A and B, without spatial dimensions). Next, the user launches CCA and other algorithms to generate the map of no change probabilities which can be interactively explored. Finally, the Mosaic Tool can apply transformation coefficients and build the final seamless mosaic.

3.5.2 Rich and Interactive GUI

The FastMosaic GUI (Graphical User Interface) has several windows: the Main Window, Guidelines Window, Console Window, Coefficients Tool Window, and Correlations Plot Tool Window. In addition, the user sees the folder for storing the data during the work of FastMosaic, fig. 3.11.

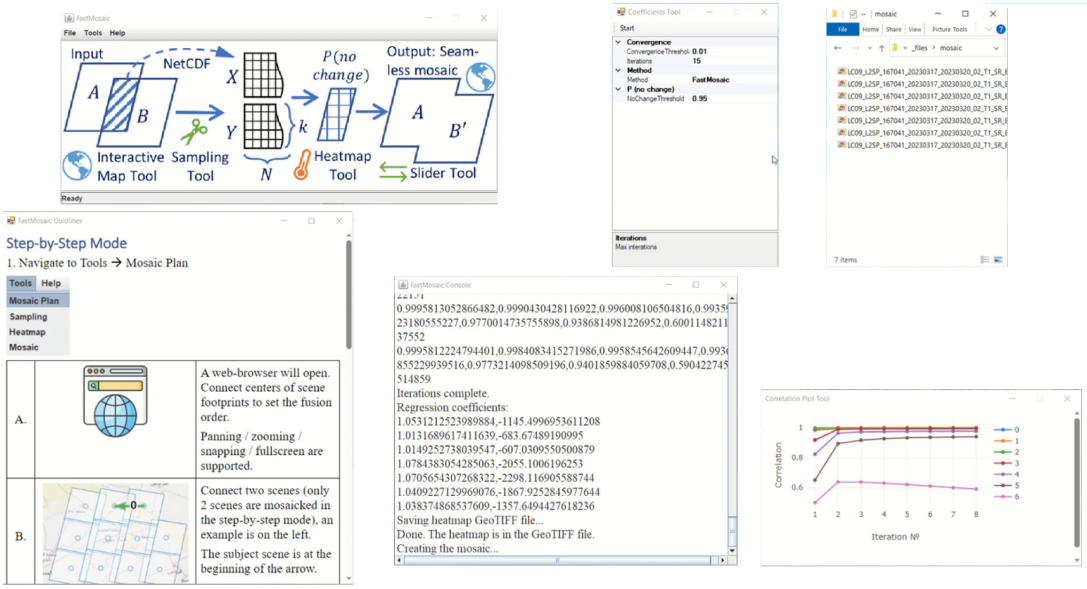


Figure 3.11: A Screenshot of FastMosaic GUI

The Guidelines Window contains detailed instructions with illustrations for the user on how to perform step-by-step construction of a seamless mosaic (create mosaic execution plan, run data sampling, and other FastMosaic algorithms), as well as guidelines on how to interactively investigate inputs and outputs.

The Console Window is read-only and prints important information during the work of FastMosaic algorithms. For example, the Console Window displays mosaic execution plans and canonical correlation coefficients together with final transformation coefficients as human-readable text during the CCA, MAD, and IR-MAD execution.

The Coefficients Tool Window makes it possible to set parameters for executing CCA, MAD, and IR-MAD.

The Correlations Plot Tool Window contains a highly interactive plot that is updated at each iteration step during the FastMosaic execution. The window displays correlations of pairs of canonical variables. With this tool, the user can investigate the convergence of the algorithm.

In this section we presented a high-level, bird's eye view of the FastMosaic interface. The respective details of its functioning are in section 4.4 that dives into the step-by-step mosaicking process.

Chapter 4

Applications: Real-World Data & Use-Cases Revisited

How are the contributions presented earlier applicable to important practical use-cases? We utilize real-world data and demonstrate the efficiency of new theoretical techniques and software architectural & implementation aspects. This is an evaluation on real-world data and problems and an additional way to showcase the practical significance of our contributions.

We refer to this study as "revisiting" real-world use-cases, as the problems we address can be solved in other ways. For example, practitioners can use batch-style scripts or MPI programs for a supercomputer. However, such approaches suffer from data management problems [9, 54]. Therefore, Array (Tensor) DBMSs, with inherent qualitative and quantitative DBMS-style advantages, are attractive alternatives in one cases or enabling systems for the others due to the need of managing large tensor volumes, e.g. in national initiatives [9].

Array (Tensor) DBMSs propose new ways to tackle the problems, increasingly more robust and efficient compared to existing frameworks [54]. In many cases, an Array (Tensor) DBMS can become a new enabling tool which is faster, easier, and more scalable compared to existing solutions.

Even if Array (Tensor) DBMSs will not completely obsolete some particular systems, they can definitely serve as excellent complements to the world of approaches to big multidimensional array management, processing, visualization, and other tensor-related tasks. Hence, Array (Tensor) DBMSs are altering already existing pipelines developed for real-world use-cases, as well as enable new opportunities, especially taking into account the rapid and continuous increase of tensor volumes [9].

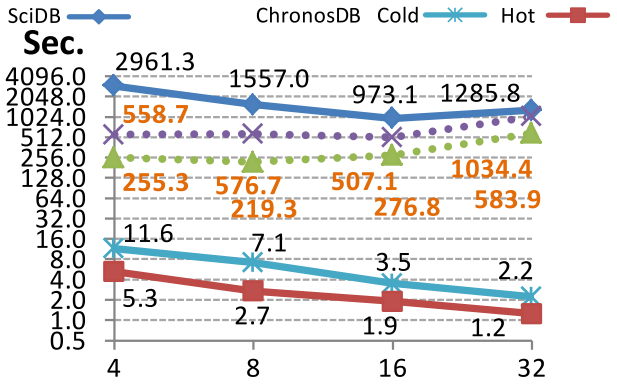
A contemporary role of Array (Tensor) DBMSs is to provide their qualitative and quantitative benefits (page №11) to pipelines that involve large multidimensional arrays (tensors).

4.1 Earth and Climate Data: Manage, Process, and Visualize

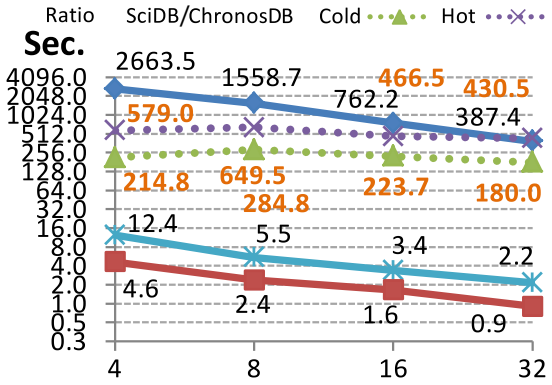
4.1.1 High-Performance Tensor Management & Processing

The new data model (section 2.1), novel array (tensor) algorithms (section 2.2), and management approaches (section 3.1.2) are implemented in ChronosDB and outperform SciDB by up to 75× on average. They are always faster and can outperform SciDB by up to 1024×. At the time of comparison, SciDB was the only freely available distributed Array (Tensor) DBMS [54]. SciDB is developed by Paradigm4 and M. Stonebraker, an ACM Turing Award Recipient ("Nobel Prize of Computing").

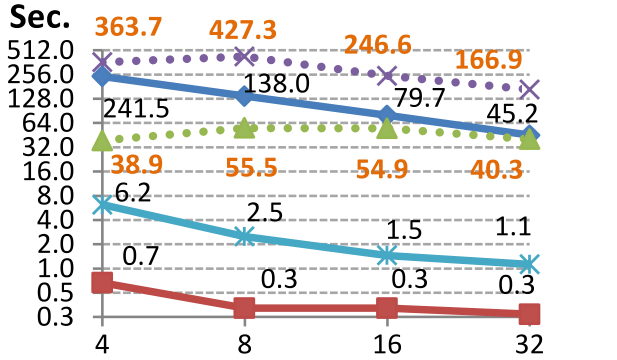
We deployed computer clusters that consists of 4, 8, 16, and 32 virtual machines in the Cloud. ChronosDB and SciDB were deployed on their own computer clusters in the Cloud. Please, refer to [54] for detailed characteristics of the hardware and software used for performance evaluation. To get maximum performance for SciDB, we thoroughly tuned it [54].



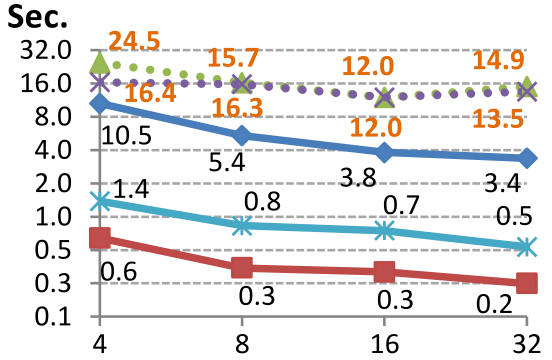
(a) $1 \times 94 \times 192 \mapsto 730 \times 2 \times 2$



(b) $100 \times 20 \times 16 \mapsto 730 \times 2 \times 2$



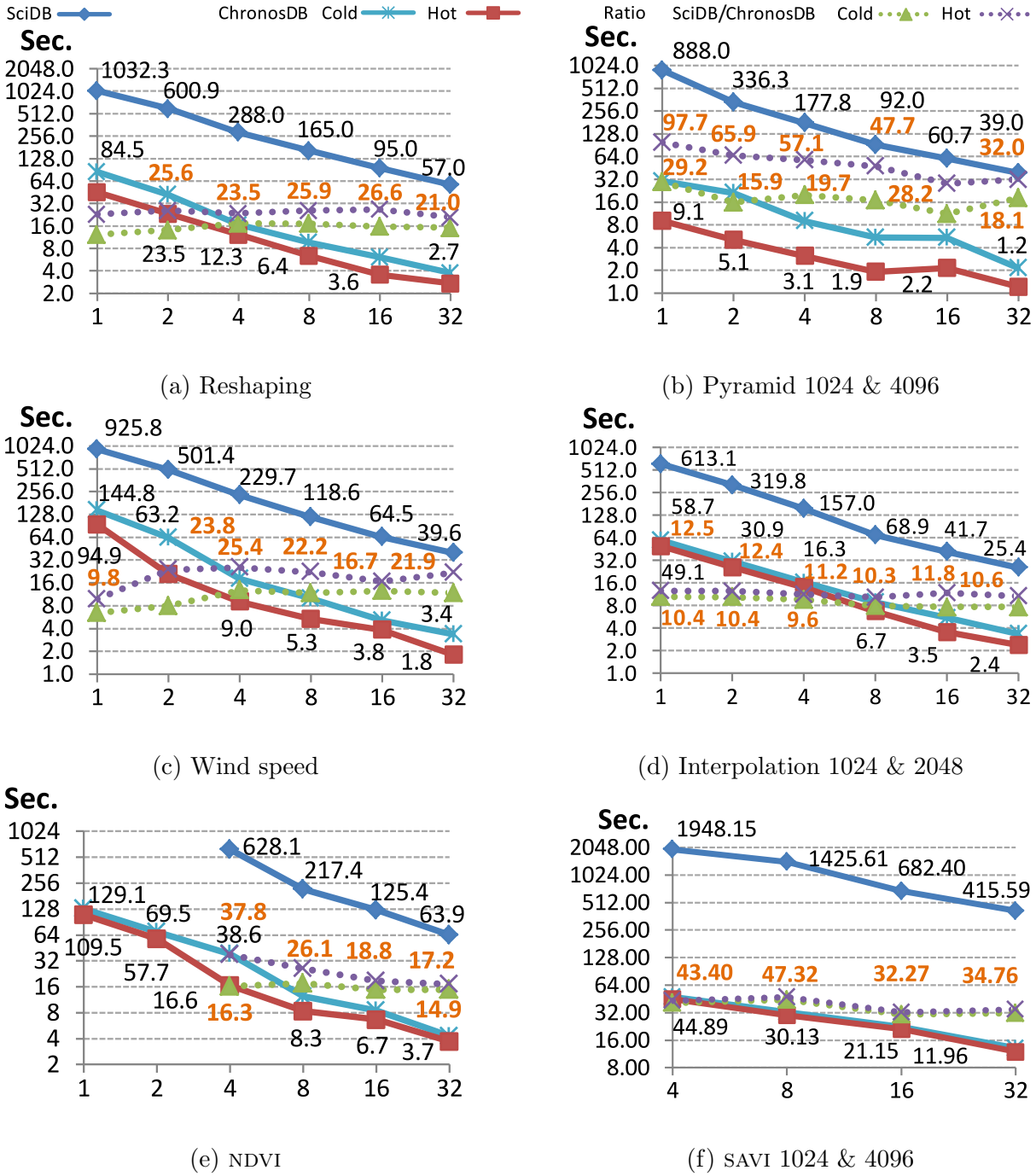
(c) $[, 0:20, 0:20]$, $1 \times 94 \times 192$



(d) $[, 0:20, 0:20]$, $100 \times 20 \times 16$

Figure 4.1: (a, b) chunking, (c, d) hyperslabbing $[0:46751, 0:20, 0:20]$, The horizontal axes plot the number of cluster nodes.

We experimented with the Landsat dataset from section 3.1.2. The raw dataset had to be retiled into a regular dataset in ≈**30 seconds** on the 8-node cluster. For SciDB, for each scene band, we had to merge several GeoTIFF files into a single large mosaic. SciDB imported such a mosaic in ≈**2 hours** on a powerful server to avoid burning Cloud time. On the contrary, ChronosDB operates with GeoTIFF files in situ, without import into an internal DBMS format.



(a) Reshaping

(b) Pyramid 1024 & 4096

(c) Wind speed

(d) Interpolation 1024 & 2048

(e) NDVI

(f) SAVI 1024 & 4096

Figure 4.2: a & b mean that $a \times a$ SciDB chunks and $b \times b$ ChronosDB subarrays were used. The horizontal axes plot the number of cluster nodes.

We also formed a dataset of eastward (u-wind) and northward (v-wind) wind speeds at 10 meters above surface between 1979–2010 (32 years) from NCEP/DOE AMIP-II Reanalysis (R2) [40]. These are Gaussian grids in the NetCDF3 format.

ChronosDB operates on NetCDF files directly, in situ. ChronosDB can readily query the data in NetCDF, but we had to develop a dedicated software to import the data into SciDB. The import took over **45 hours** on a powerful server to avoid wasting Cloud time.

We evaluated cold and hot query runs: a query is executed for the first and the second time respectively. ChronosDB benefits from native OS caching and is much faster during hot runs. This is especially useful for continuous experiments with the same data, section 2.3.1. There is no significant runtime difference between cold and hot SciDB runs.

The resulting performance for chunking and hyperslabbing are presented in fig. 4.1. Figure 4.2a reports the performance of reshaping $(time, lat, lon) \mapsto (lon, lat, time)$. The ratio is up to 26×. We benchmarked the creation of 3 levels of the multiresolution pyramid (fig. 4.2b) and the 2× interpolation (fig. 4.2d). ChronosDB outperforms SciDB by up to 97× and 12× respectively.

Wind speed (ws) at each grid cell and time point is calculated as $ws = \sqrt{\text{u-wind}^2 + \text{v-wind}^2}$. The ratio is up to 25× (fig. 4.2c).

The calculation of NDVI demonstrates the distributed K-way array (tensor) join, section 2.2.2. SciDB fails to compute NDVI on 1- and 2-node clusters with a `not enough memory error`. ChronosDB is exceptionally superior to SciDB (up to 37×).

We also benchmarked computing SAVI (not only SAVI itself, but the complete complex execution plan: the SAVI pipeline, section 3.1.3). In the result, ChronosDB is from 32× to 47× faster than SciDB, fig. 4.2f. This proves ChronosDB to be efficient for complex analytic pipelines.

The reader can find more details on the performance evaluation in [54].

4.1.2 GUI & DWMTS for Array (Tensor) DBMS

Utilizing ChronosDB, it is possible to interactively visualize large volumes of array (tensor) data. Web GUI and a specialized, proprietary DWMTS (Distributed WMTS) implementation provided by ChronosDB out-of-the-box facilitate the aforementioned goal [55].

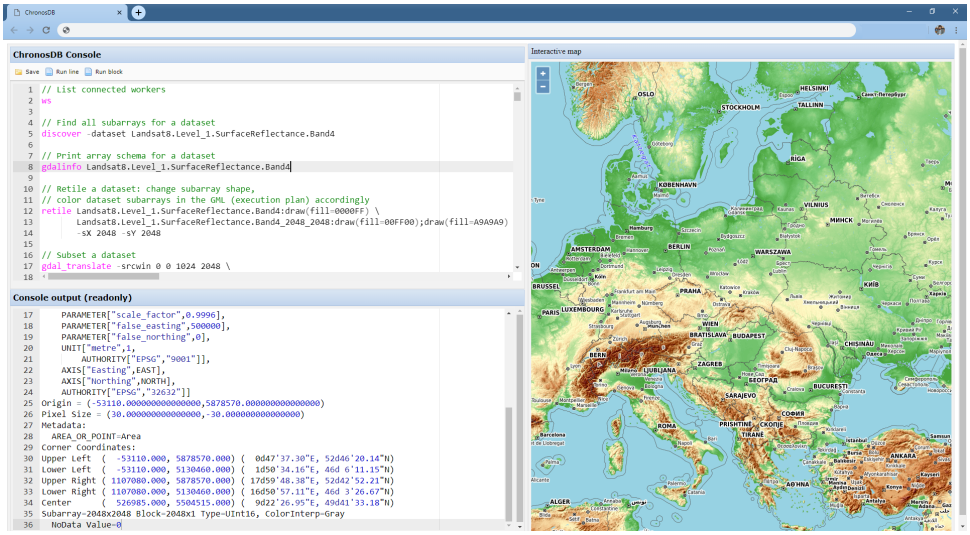


Figure 4.3: ChronosDB Web GUI [55]

The ChronosDB Web GUI has three main parts: (1) ChronosDB Console, (2) Console Output, and (3) Interactive Map, fig. 4.3. It is possible to edit several scripts in the ChronosDB Console (syntax highlight is supported), submit a single line or a code fragment for execution. The Web GUI establishes a session with ChronosDB via a custom network protocol. A GUI $\mapsto$ ChronosDB message carries a script, ChronosDB $\mapsto$ GUI messages carry the script output, attached to the Console Output.

Visualization is essential for data understanding. The Interactive Map displays ChronosDB datasets. Users can switch base layers, add/remove arrays from the map, and adjust their color schemes. ChronosDB visualizes arrays by rendering subarrays and delivering imagery via WMTS (Web Map Tile Service), a popular OGC protocol for serving georeferenced map tiles over the HTTP [75]. Any interactive web map or desktop software that supports WMTS can visualize ChronosDB datasets.

ChronosDB analyses a WMTS tile request and provides the rendered image directly from the node on which the subarray resides. Most popular WMTS servers work on a single machine. ChronosDB enables large array (tensor) visualization while reducing data movement between the nodes. ChronosDB facilitates users to feel input data, as well as visually evaluate derivative data resulting from script execution [55].

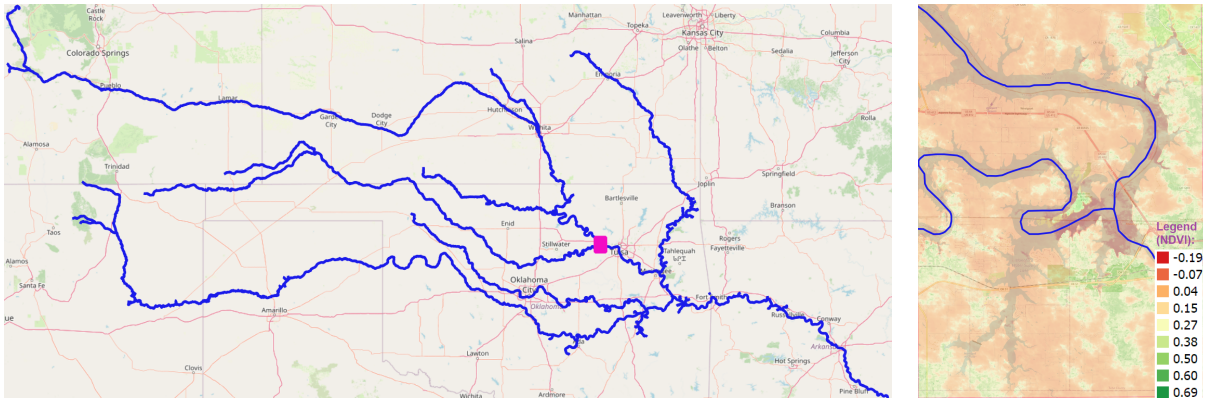
4.2 Fast Interactive Data Science: Quick Tensor Recomputing (Updates)

Data scientists can experience increased response times from software that enables working with big arrays (tensors) interactively. As a user typically spends their time waiting for such responses in front of a computer, each successive data processing delay, even within 1–2 seconds, increases human fatigue and thereby reduces work quality and data understanding.

Here we describe several important practical applications that benefit from BitFun, WebArrayDB, and ArrayGIS [53, 63], heavily boosting tensor updates (up to 8×) in response to manual or automatic inputs.

4.2.1 Water Management & Flood Mapping

This application illustrates fast evaluation of $f(\tau) < const$ due to novel indexing techniques, section 2.3.2. To map a flood, we create a water mask: a 2-d array with two cell values: 1 (water) and 0 (no water).



(a) The Arkansas River with its most noticeable tributaries (b) Zoomed NDVI box

Figure 4.4: The area for the BitFun Water Lesson (River Flood Mapping)

We selected the Arkansas river basin as an example. At 1,469 miles (2,364 km), the Arkansas is the 6th and 45th longest river in the U.S. and the world, respectively. Its drainage basin covers 161,000 sq miles (417,000 sq km) and has a total fall of 11,400 feet (3,500 m) [5]. Figure 4.4b (May 28, 2019; Landsat 8) zooms the pink box in fig. 4.4a.

In the spring of 2019, the Southern and Central U.S. experienced severe flooding. The problem was most acute in late May along the Arkansas River. Every county in Oklahoma was in a state of emergency, and evacuations were ordered or recommended in several communities in Arkansas [38].

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, and rapid-response for disaster relief [4].

NDVI values close to zero or negative represent zones with the presence of water, section 2.3.1. The task is to quickly create an accurate water mask by tuning τ in $f(\tau) = \text{NDVI} - \tau < 0$. As a reference, the analyst sees the RGB map, resulting mask, and the set of points of two colors (ground truth) located in flooded and non-flooded areas [53].

The challenge is to tune τ such that for most ground truth points that indicate flood and no-flood, water mask values are 1 and 0, respectively. The selected area serves as a good example, as the Arkansas River Basin occupies a relatively large area. BitFun quickly re-creates the water mask for this area each time the user tunes τ [53][1].

4.2.2 Food Security & Crop Yield Prediction

Now we illustrate the fast computing of $f(\tau)$ due to novel indexing techniques, section 2.3.2. Values of $f(\tau)$ are required for a crop yield model.

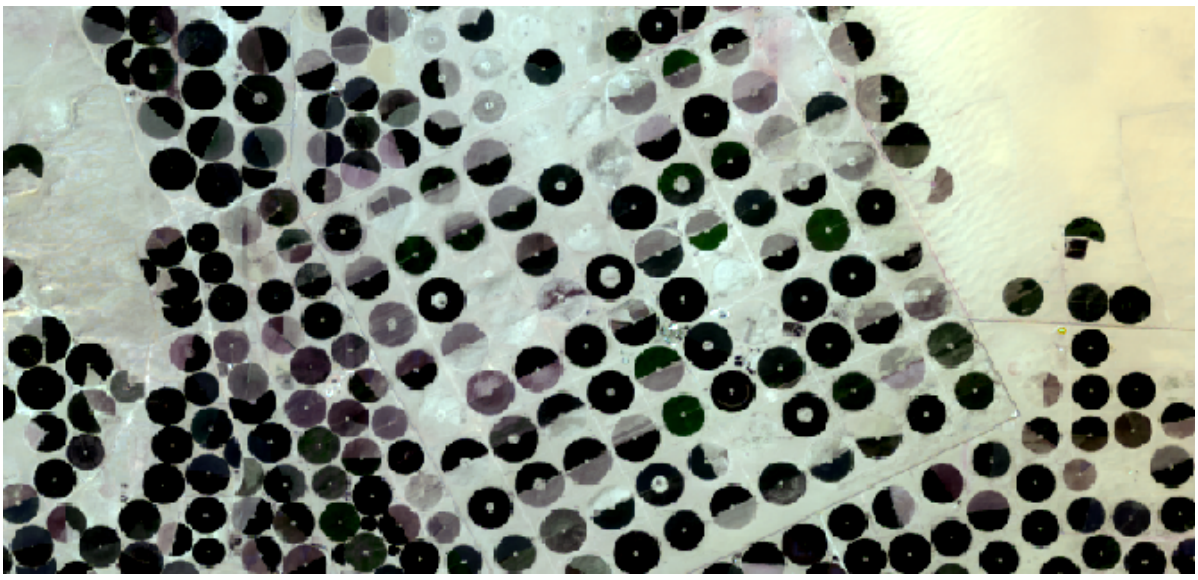


Figure 4.5: Saudi Arabia: Center Pivot Irrigation Systems (Wadi Al-Dawasir area, to the south of Riyadh, the capital city, 11/Feb/2016, RGB, Landsat 8 Collection 1 Level 2)[1]

Vegetation indexes are heavily used in ML/AI & Food Security: classification [16], segmentation [79], drought threats [74], cropland health [21], precision agriculture [35], crop yield prediction [2], to name a few.

SAVI is used for arid regions (such as Saudi Arabia's agricultural fields) with sparse vegetation and exposed soil surfaces [78]. SAVI performs much

[1] BitFun Homepage: http://bitfun.gis.land

better than other vegetation indexes, as it aims to minimize soil brightness influence by introducing its tunable parameter L, a soil fudge factor varying from 0 to 1 depending on the soil, section 2.3.1.

Center pivot irrigation is popular in the arid and hyper-arid regions of the Earth. In Saudi Arabia, fig. 4.5, the main crops grown in winter are wheat, potato, tomato and melon. Fodder crops are grown throughout the year. Circles are cultivated in the desert with temperatures up to 43°C [2].

Center (or Central) Pivot Irrigation is a method for irrigating crops using sprinklers rotating around a central pivot [69]. This type of irrigation uses less labor (lower costs) compared to other irrigation types and can reduce water runoff, soil erosion, and compaction.

A BitFun lesson [53] takes [2] as an example: the authors build empirical crop yield models for pivots of the Saudi Agricultural Development Company (INMA) in the area, fig. 4.5. It is possible to feed different SAVI values to the models depending on the L value. The challenge is that the value of L is not known beforehand and is tuned experimentally. BitFun provides novel indexing techniques to avoid computing SAVI from scratch.

BitFun quickly re-computes SAVI for a large area each time the user tunes L to check whether new, adjusted SAVI values for this new L are more appropriate for the models. Moreover, as the crop yield is estimated up to some digits after the floating point, the BitFun ability to specify precision is very helpful here and significantly accelerates computations[2].

4.2.3 Accelerated Web-Based Processing & Visualization

Web GISs (Geographic Information Systems) are vivid examples of how tensor processing and visualization are becoming commodities via Web Browsers. Whereas other Web GISs use Web Browsers just as thin clients, displaying only server-rendered read-only images, WebArrayDB and ArrayGIS operate on raw data entirely inside a Web Browser, section 3.4.

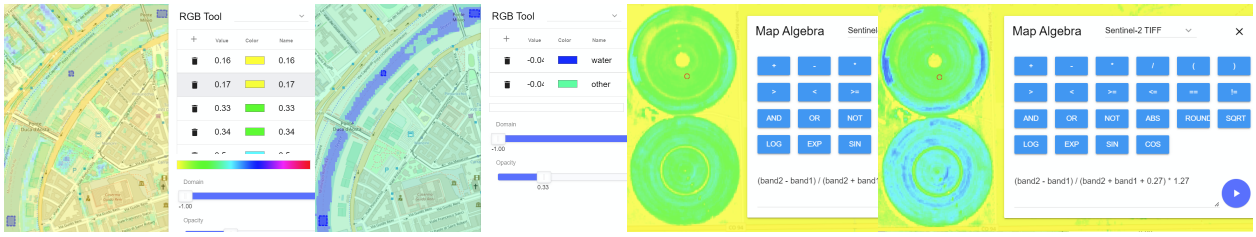


Figure 4.6: WebArrayDB and ArrayGIS Lessons: Initial States and Solutions [63]

[2]BitFun Homepage: http://bitfun.gis.land

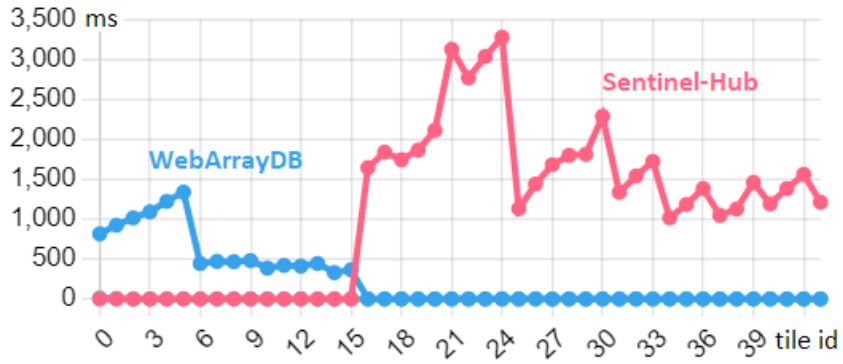


Figure 4.7: Latency: WebArrayDB vs. Sentinel-Hub

We illustrate accelerated Web-based computations in two lessons [63]: mapping water bodies and assessing agricultural crops, fig. 4.6. Although the scenarios are somewhat reminiscent of those described in sections 4.2.1 and 4.2.2, the underlying acceleration techniques are different [63].

The Water Lesson, in particular, showcases near instantaneous updates of array color palettes by keeping raw array data in the Web Browser. WebArrayDB, the foundation of ArrayGIS, enables access to source tensor data without client-server interactions, unlike other Web GISs. We also use the popular NDVI index as a water indicator, section 4.2.1, but for Rome (Italy) and Sentinel-2 data. Users see the ground truth and submit queries **SELECT** NDVI $< \beta$, where β is a tunable parameter, fig. 4.6.

ArrayGIS is an innovative Web GIS, a front-end for WebArrayDB. ArrayGIS features interactive Web GUI and is freely accessible at `https://wikience.github.io/webdb2022` Hence, it is easy to reproduce the lessons and confirm the advantages of ArrayGIS and WebArrayDB.

Users can inspect the WebArrayDB & ArrayGIS performance by exploring interactive charts in the Database Statistics infobox [63]. Figure 4.7 is a screenshot from the Database Statistics infobox.

ArrayGIS relies on WebArrayDB. They can be over 2× faster compared to querying only Sentinel-Hub [63], a popular Cloud service for disseminating and processing Sentinel data, fig. 4.7[3].

Detailed guidelines on the lessons in the ArrayGIS GUI are in [63]. Video presentations of BitFun, WebArrayDB and ArrayGIS can also be found on their home pages [53, 63].

[3]Note that WebArrayDB retrieves data (typically tiles 256×256 or 512×512) over WMTS directly to a Web browser; there is no intermediary. Client machine: Intel Core i5 1.6 GHz, 8 GB RAM, 256 GB SSD, Chrome 100 (64-bit), Windows 10, Internet up to 100 MBit/s (WiFi).

4.3 Road Traffic Simulations: An Array (Tensor) DBMS End-To-End Approach

This section guides through a step-by-step, end-to-end Cellular Automata simulations by SimDB (section 3.3) in its interactive GUI [61], fig. 4.8. We also showcase the benefits of using SimDB for the simulations to answer the question why SimDB is an excellent choice for this workload.

We rely on the article [61] and the web page of [56] with very detailed examples[4]. In addition, we also encourage to visit the homepage of [61] where the link to the respective video is also located[5].

Users can create a road network via a constructor. The road network is converted to input arrays. Native Array (Tensor) DBMS UDFs are used to initialize the simulation and generate proactive simulation plans. Integrating native and Java UDFs is also supported, section 3.3. SimDB can animate arrays on an interactive map and is interoperable. The goal of a simulation is to derive statistics from arrays with simulation history (e.g., `speedh` and `lenh`, section 4.3.1) for decision support. SimDB facilitates simulations with its powerful capabilities.

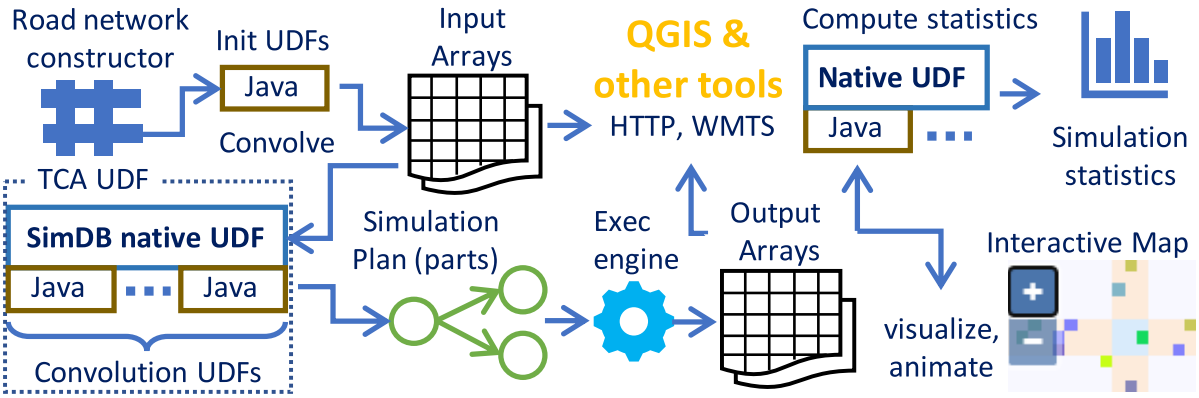


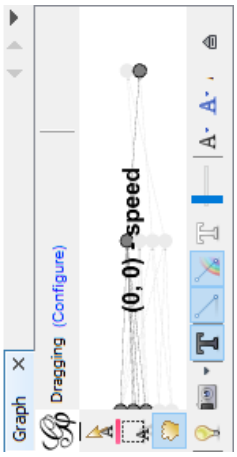
Figure 4.8: SimDB End-to-End Simulation Overview [61]

4.3.1 Simulation Initialization & Plan Investigation

The physical environment and cells' states can be modeled as 2-d arrays. New Array (Tensor) DBMS components are partially driven by our new flexible convolution operator & native UDF language enabling us to apply local transition rules & code the simulation logic respectively, section 3.3.

As input, we create and initialize at least three 2-d arrays shaped $lat \times lon$ (more arrays can serve as inputs, e.g. weather conditions, elevation):

[4] http://sigmod2021.gis.gg (also contains the code for our TCA implementation)
[5] https://wikience.github.io/simdb2022



(a)

```
public void setSpeed(ConvolveWindows w) {
   double val = w.input(0).get(0, 0);
   Double output = null;
   if (val == 1 || val == 0) {
      output = (double) w.input(0).random().nextInt(4);
   } else {
      if (val == 4) {   // traffic lights
         int rnd = w.output(0).random().nextInt(2);
         output = (double) (200 + rnd + 1);
      }
   }
   w.output(0).set(output, 0, 0);
}
```

(b)

Figure 4.9: (a): Part of a Proactive Plan, (b) UDF for Assigning a Speed for a Vehicle [61]

- `tca.lane`: road grid, cells: -1: impassable, 0/1: west-east/south-north moving direction, 2: road intersections, 3: traffic lights
- `tca.speed`: initial vehicles' speeds further updated by the TCA rules (0 – indicates that a vehicle is not moving, positive values are speeds)
- `tca.length`: vehicles' lengths (a vehicle of any length is modeled by a cell having its rear bumper)

Note that we do not store vehicle positions explicitly as they are implicitly coded by cell coordinates.

Typically, the goal of a simulation is to derive statistics. Hence, we incrementally build history 3-d arrays ($time \times lat \times lon$): `speedh` and `lenh` (vehicles' speeds and lengths for each time step) by appending `tca.speed` and `tca.length` along the virtual *time* axis, section 3.3.2.

The initialization has two sequential phases: (1) decide whether a cell is occupied by a vehicle, (2) assign a length and a speed to each vehicle.

The UDF for setting a vehicle's speed is in fig. 4.9b. We assign a random speed to a cell after we assure whether the convolution window that is centered at this cell permits placing a vehicle. Along the way, we also assign the number of ticks to traffic lights if we encounter them.

SimDB also makes it possible to interactively investigate Proactive Simulation Plans (PSPs, section 3.3.3) to get insights of its scheduling and execution capabilities. Plans are exposed in the Graph Modeling Language [20]. SimDB lays out tasks on the plane to avoid clutter, assigns colors, and annotates them with extensive statistics: task assignment, network I/O, dataset info, etc. It is possible to zoom/pan PSPs, create statistics, filter, highlight tasks and their dependencies in Gephi [20], fig. 4.9a.

4.3.2 Interactive Visualization & Animation

Visualization is essential for data understanding. SimDB provides arrays via the open, popular, standard WMTS protocol [75] and displays them in its interactive GUI, fig. 4.10. It is possible to add/remove arrays (e.g. `tca.lane`) to/from the map, pan, zoom & adjust their color palettes (map layers in fig. 4.10 are transparent, so colors slightly merge).

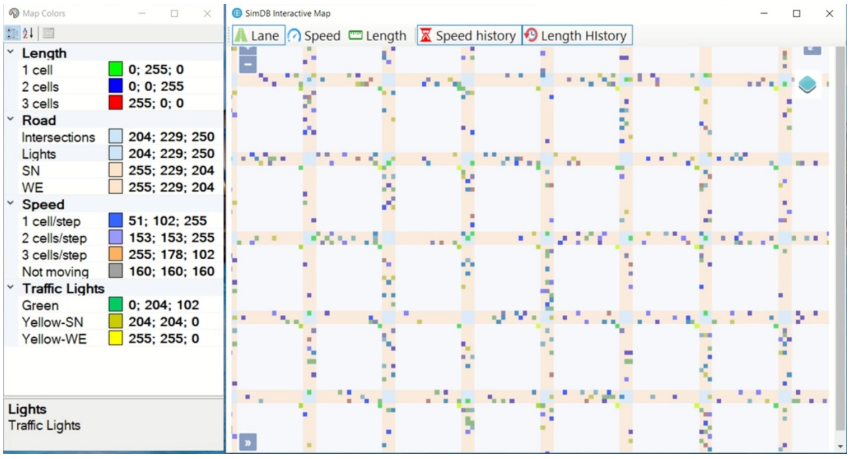


Figure 4.10: SimDB GUI: Interactive Visualization and Animation [61]

SimDB animates history arrays, so users can watch modeling at work: how vehicles move along the roads, queue at traffic lights, turn at road intersections, change speed, and overtake each other [61], fig. 4.10.

4.3.3 Experiencing Interoperability

SimDB storage layer relies on raw files in standard formats, e.g. GeoTIFF. Hence, tensors are readily accessible to other software as well-known binary files or rendered images. For example, to view a tensor on an interactive Quantum GIS map, add a SimDB dataset as a WMTS layer, fig. 4.11.

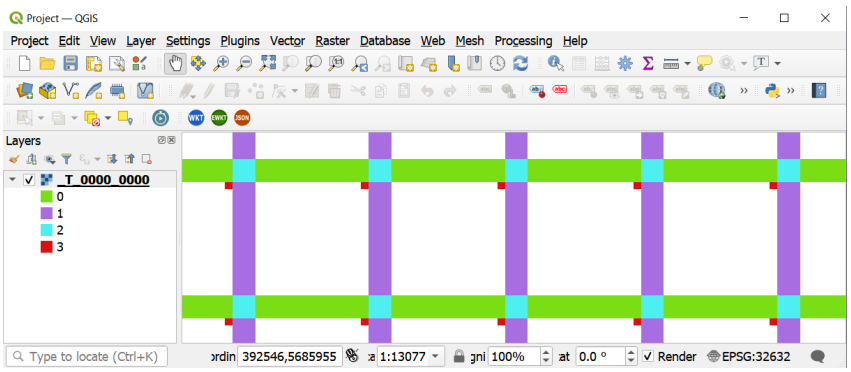


Figure 4.11: A SimDB Array in Quantum GIS [61]

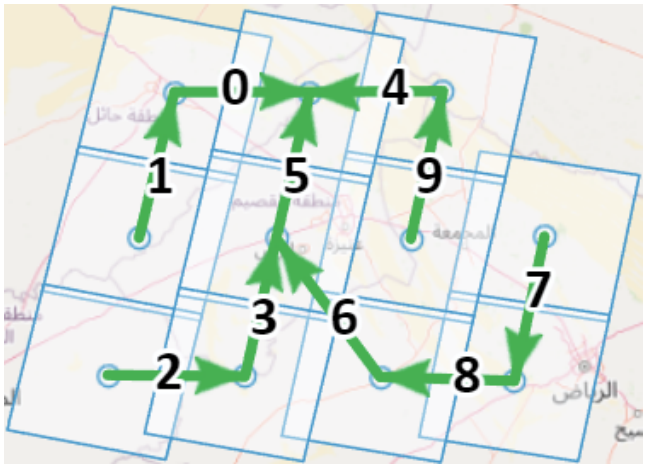
More information is in [61] and homepages of [56, 61].

4.4 Fast, Seamless Tensor Mosaicking: Step-By-Step

FastMosaic can guide users step-by-step through the end-to-end workflow (section 3.5.1) of constructing a seamless mosaic on input tensors via an interactive and rich GUI (section 3.5.2). Here we briefly describe the main steps of the mosaicking workflow. We encourage the reader to check the FastMosaic homepage[6] for the video showcasing all phases.

4.4.1 Creating a Mosaic Plan

First, we must define the mosaic plan using the Mosaic Plan Tool in the interface. The user interactively builds a tree, the mosaic execution plan, by drawing arrows to connect array pairs and set the fusion order. At step № i, the tensor at the start of arrow № i joins the mosaic built so far, fig. 4.12a.



(a) Mosaic Execution Plan

```
{"type":"FeatureCollection",
 "features":[{
  "type":"Feature",
  "geometry": {
     "type":"LineString",
     "coordinates":
        [[510598.029, 3043050.697],
         [357855.877, 3042011.635]]
     },
     "properties": {"id": 0},
     "id":0
  }]
}
```

(b) A Tiny Mosaic Plan in GeoJSON

Figure 4.12: Illustrations of Mosaic Execution Plans

Figure 4.12a shows an execution plan for the batch mode, which fuses tensors in the given order. Tensors, Landsat 8 satellite scenes, are represented as footprints. The subject tensor is at the beginning of an arrow, so FastMosaic calculates transformation coefficients for subject tensors.

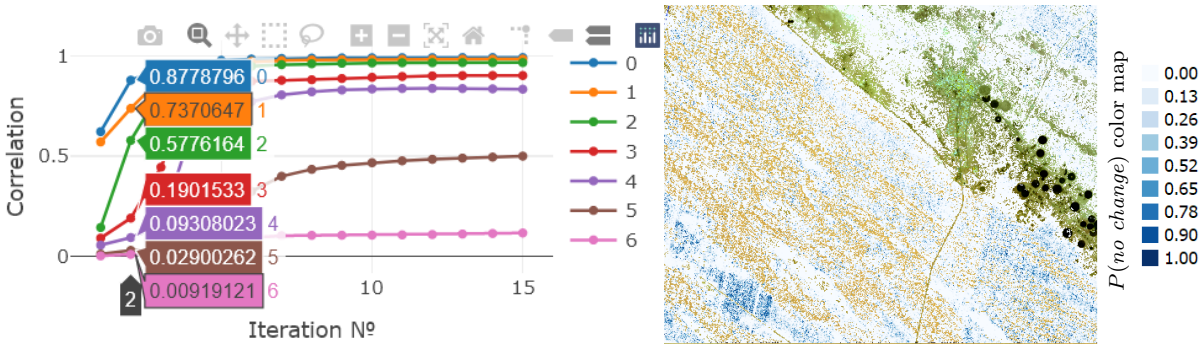
Mosaic execution plans are stored in the GeoJSON format. Figure 4.12b presents an execution plan that will fuse two tensors in the step-by-step mode (arrow № 4 in fig. 4.12a).

Once the plan is defined, FastMosaic is ready to sample the data for estimating no-change probabilities and proceed to the next steps.

[6] https://wikience.github.io/fastmosaic2023

4.4.2 Sampling, Execution & Heatmaps

Once the reference and subject arrays (tensors) are defined, we are ready to create X and Y arrays (fig. 3.10) with the Sampling Tool in the GUI for the input to canonical correlation analysis. The tool creates a NetCDF file with the sampled cell values from overlapping cells of the inputs.



(a) Correlations of Canonical Variables

(b) P(no change) Heatmap

Figure 4.13: Interactive Plot and Heatmap

Next, we can use X and Y to execute CCA, MAD, IR-MAD, generate the map of no change probabilities and the regression coefficients. Recall that X and Y are $M \times K$ arrays, where M is the number of overlapping cells and K is the dimensionality of input arrays (tensors), excluding spatial dimensions (the number of bands, in the case of Landsat 8 scenes).

During the execution, the user can visually track the convergence of the algorithm with the interactive plot updated at each iteration step, fig. 4.13a. In addition, the coefficients also appear in the GUI console. The line № i plots the correlation of the pair № i of canonical variables.

The options for the algorithms are described in the article [59] and shown in the video, section 4.4: (1) CCA implementation: the proposed or Python scikit-learn, and the stopping condition: (2) correlation threshold (the significance of the change in correlations of canonical variables), or (3) the maximum number of CCA iterations.

The user can see that our CCA runs considerably faster compared to the Python scikit-learn both for real-world and synthetic data: it can be about 30× faster even for two overlapping Landsat 8 scenes ($M \approx 6 \times 10^6$, $K = 7$), fig. 4.15, and can be over 30× faster for samples from normal distribution, fig. 4.14.

We generated random input variables using the normal distribution. For example, let X is a random variable sampled from the standard normal distribution, where the mean is 0 and the variance is 1: $X \sim N(0, 1)$. We can create the other variable as $Y = X + Z$, where $Z \sim N(0, 0.5)$

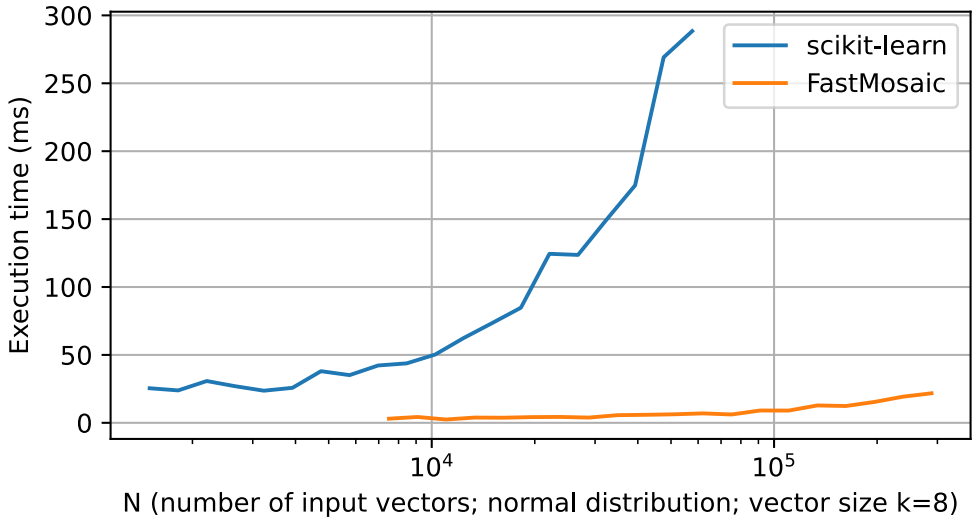


Figure 4.14: CCA: FastMosaic VS. Python's "scikit-learn" [59]

(the parameters of the distribution can vary without much impact on the runtime). We also varied M, the number of input K-sized vectors (sample size) and set $K = 8$ (recall that we use M and K instead of N and k as in [59], because N denotes the number of tensor dimensions in this Dissertation), fig. 4.14[7]. For $M = 10^6$, the runtime of Python's CCA and FastMosaic is 5056.7 ms and 127.8 ms, respectively; the ratio is 39.57.

The default tolerance parameter of Python's CCA implementation make it run significantly slower, so in our experiments we set 0.01 for tolerance (the same as for FastMosaic). The subsequent increase of the tolerance is hardly acceptable. The same parameters were used to compare the performance of Python's CCA and FastMosaic on Landsat 8 satellite scenes. Let U_P, V_P and U_F, V_F be pairs of canonical variables generated by Python's CCA and FastMosaic respectively. We found that $|corr(U_P, U_P) - corr(U_F, U_F)| < 10^{-u}$ for $u = 2$ and above, where $u \in \mathbb{Z}$ both for synthetic and real-world data. This confirms that both methods are of comparable quality regarding their outputs (canonical variables).

The correlations of the variables can be tracked using the interactive chart, fig. 4.13a. Correlation values and coefficients generated step-by-step can also be found in the FastMosaic video, see Article № 8.

As we have computed the P(no change) array, we can generate the respective heatmap to thoroughly inspect cells in the overlapping area that are likely invariant, fig. 4.13b. FastMosaic saves the heatmap in GeoTIFF, so it is possible to interactively explore it in a GIS (Geographic Information System) software, for example, the popular Quantum GIS.

[7]Intel Core i7 2.60 GHz, 64 GB RAM, 512 GB NVMe, Win 10, sklearn (scikit-learn) version 1.4.0.

4.4.3 Transformation (Normalization)

Finally, we can build the mosaic. As we have 2 input arrays (tensors) for the step-by-step mode, we need only K pairs of coefficients $\beta_k, \epsilon_k \in \mathbb{R}$ to transform the subject array (tensor) $B\langle lat, lon, K\rangle$ (on the right, fig. 4.15) $\beta_z B[x, y, z] + \epsilon_z$ ($z, k \in [1, K]$, section 2.5.2) to generate a seamless mosaic.

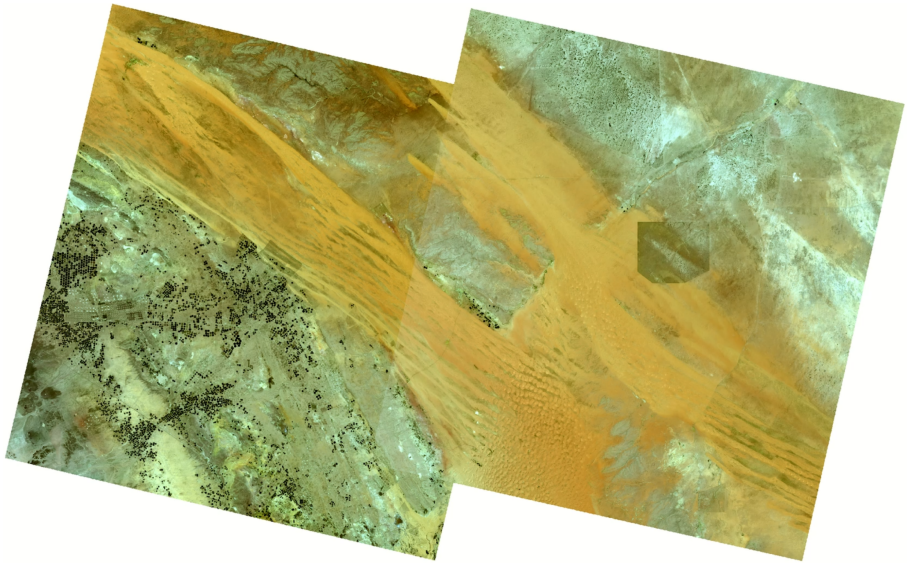


Figure 4.15: Input Satellite Scenes (paths 167 and 168, row 41): RGB

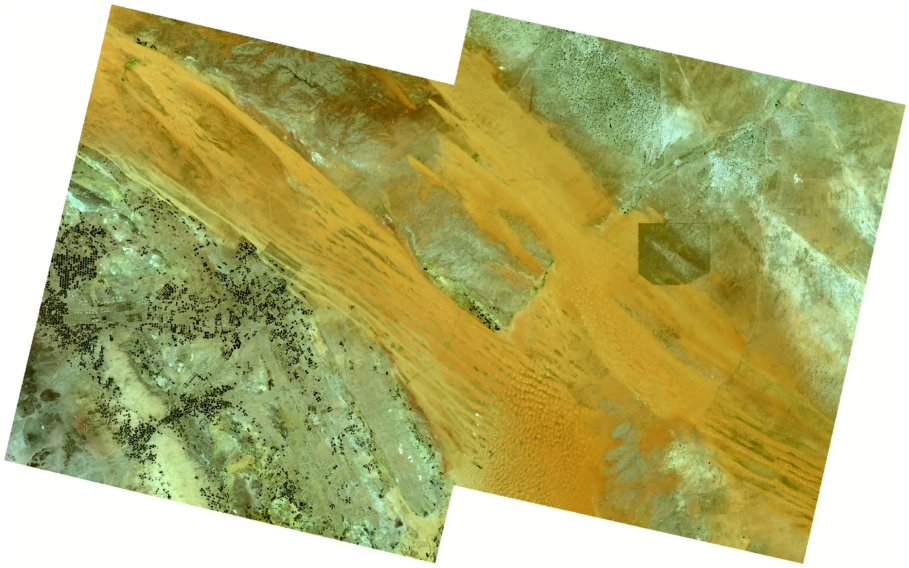


Figure 4.16: Seamless Array (Tensor) Mosaic for the Input Data in fig. 4.15

It is apparent that the visual transition between the scenes is abrupt and looks like a stitch, fig. 4.15. It is challenging to build a high-quality mosaic: a simple contrast tuning is not sufficient to fix this. Clearly, FastMosaic features scalable CCA, MAD, IR-MAD to quickly build seamless mosaics that look like a single continuous image in natural colors, fig. 4.16.

Chapter 5

Conclusion

The conclusion outlines the outcomes of the completed R&D, recommendations, and prospects for further development of the topic.

In the area of Array (Tensor) DBMSs, we have established novel theoretical foundations, introduced novel architectural and implementation aspects, and demonstrated the significance of our contributions on real-world data and important practical applications. The results included in this Dissertation are presented at premier international conferences in computer science: VLDB and SIGMOD.

A detailed list of the results is in section 1.3. The main results submitted for defense are also in section 1.3.

As we noted, the area of Array (Tensor) DBMS is young by right, so the work in this area has just commenced. Array (Tensor) DBMSs could account for other data types like spatial polygons, relational tables, and graphs, or work within polystore systems. More attention is required to utilize novel hardware, e.g. NVM and GPU. One of the most perspective R&D directions is the exploration of novel Array (Tensor) DBMS applications, like simulations. Applications pose unique challenges to Array (Tensor) DBMSs helping them to become more robust systems in general.

Data Science and Machine Learning are just paving their way to Array (Tensor) DBMSs, whose one of the major advantages is native tensor support. It is attractive to run DS/ML directly inside an Array (Tensor) DBMS to avoid costly data exchanges with DS/ML systems.

Right now, there are the best conditions to begin making contributions to the R&D area of Array (Tensor) DBMSs.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, et al. TensorFlow: a system for large-scale machine learning. In *OSDI*, pages 265–283, 2016.

[2] K. A. Al-Gaadi, A. A. Hassaballa, E. Tola, et al. Prediction of potato crop yield using precision agriculture techniques. *Plos One*, 11(9):e0162219, 2016.

[3] AnyLogic. anylogic.com/road-traffic, 2024.

[4] ArcGIS book. learn.arcgis.com/en/arcgis-imagery-book, 2024.

[5] Arkansas River. newworldencyclopedia.org/entry/Arkansas_River, 2024.

[6] V. Balaji, A. Adcroft, and Z. Liang. Gridspec: a standard for the description of grids used in Earth system models. *arXiv preprint arXiv:1911.08638*, 2019.

[7] L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *SIGMOD*, pages 1363–1375, 2016.

[8] P. Baumann and S. Holsten. A comparative analysis of array models for databases. *Int. J. Database Theory Appl.*, 5(1):89–120, 2012.

[9] P. Baumann, D. Misev, V. Merticariu, and B. P. Huu. Array databases: concepts, standards, implementations. *Journal of Big Data*, 8(1):1–61, 2021.

[10] P. Baumann, D. Misev, V. Merticariu, B. P. Huu, and B. Bell. DataCubes: a technology survey. In *IGARSS*, pages 430–433. IEEE, 2018.

[11] S. Blanas, K. Wu, S. Byna, B. Dong, and A. Shoshani. Parallel data analysis directly on scientific file formats. In *SIGMOD*, pages 385–396, 2014.

[12] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. Saltz. Titan: a high-performance remote-sensing database. In *ICDE*, pages 375–384, 1997.

[13] D. Choi, H. Yoon, and Y. D. Chung. Resky: efficient subarray skyline computation in array databases. *Distributed and Parallel Databases*, 40(2-3):261–298, 2022.

[14] D. Choi, H. Yoon, and Y. D. Chung. Subarray skyline query processing in array databases. In *SSDBM*, pages 37–48, 2021.

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

[16] V. S. da Silva, G. Salami, M. I. O. da Silva, E. A. Silva, J. J. Monteiro Junior, and E. Alba. Methodological evaluation of vegetation indexes in land use and land cover (LULC) classification. *Geology, Ecology, and Landscapes*, 4(2):159–169, 2020.

[17] D. J. DeWitt et al. Client-server Paradise. In *VLDB*, pages 558–569, 1994.

[18] Earth on AWS. https://aws.amazon.com/earth/, 2024.

[19] ECWMF report. https://www.ecmwf.int/en/computing/our-facilities/data-handling-system, 2022.

[20] GML. https://gephi.org/users/supported-graph-formats/, 2024.

[21] A. T. Hammad and G. Falchetta. Probabilistic forecasting of remotely sensed cropland vegetation health and its relevance for food security. *Science of the Total Environment*, 838:156157, 2022.

[22] O. Horlova, A. Kaitoua, and S. Ceri. Array-based data management for genomics. In *ICDE*, pages 109–120, 2020.

[23] H. Hotelling. Relations between two sets of variates. *Biometrika*, 28(3/4):321–377, 1936.

[24] C. E. Kilsedar and M. A. Brovelli. Multidimensional visualization and processing of big open urban geospatial data on the web. *ISPRS International Journal of Geo-Information*, 9(7):434, 2020.

[25] B. Kim, K. Koo, U. Enkhbat, S. Kim, J. Kim, and B. Moon. M2bench: a database benchmark for multi-model analytic workloads. *PVLDB*, 16(4):747–759, 2022.

[26] S. Ladra, J. R. Paramá, and F. Silva-Coira. Scalable and queryable compressed storage structure for raster data. *Information Systems*, 72:179–204, 2017.

[27] Landsat missions. https://landsat.usgs.gov/, 2024.

[28] É. Leclercq et al. Polystore and tensor data model for logical data independence and impedance mismatch in big data analytics. In *LNCS*, pages 51–90. 2019.

[29] X. Li, R. Feng, X. Guan, et al. Remote sensing image mosaicking: achievements and challenges. *IEEE Geoscience and Remote Sensing Magazine*, 7(4):8–22, 2019.

[30] L. Libkin, R. Machlin, and L. Wong. A query language for multidimensional arrays: design, implementation, and optimization techniques. In *ACM SIGMOD Record*, volume 25 of number 2, pages 228–239, 1996.

[31] B. A. Lungisani, C. K. Lebekwe, A. M. Zungeru, and A. Yahya. The current state on usage of image mosaic algorithms. *Scientific African*:e01419, 2022.

[32] S. Maerivoet and B. De Moor. Cellular automata models of road traffic. *Physics reports*, 419(1):1–64, 2005.

[33] A. P. Marathe and K. Salem. Query processing techniques for arrays. *VLDBJ*, 11(1):68–91, 2002.

[34] Maxar AWS re:Invent, 80 TB/day. https://youtu.be/mkKkSRIxU8M, 2017.

[35] V. Mazzia, L. Comba, et al. UAV and machine learning based refinement of a satellite-driven vegetation index for precision agriculture. *Sensors*, 20(9):2530, 2020.

[36] P. Mehta, S. Dorkenwald, D. Zhao, et al. Comparative evaluation of big-data systems on scientific image analytics workloads. *PVLDB*, 10(11):1226–1237, 2017.

[37] Milliseconds make millions. https://www2.deloitte.com/content/dam/Deloitte/ie/Documents/Consulting/Milliseconds_Make_Millions_report.pdf, 2020.

[38] NASA EO. earthobservatory.nasa.gov/images/145108/floods-in-the-arkansas-river-watershed, 2019.

[39] S. Nativi, J. Caron, B. Domenico, and L. Bigagli. Unidata's common data model mapping to the ISO 19123 data model. *Earth Sci. Inform.*, 1:59–78, 2008.

[40] NCEP-DOE AMIP-II Reanalysis. http://www.esrl.noaa.gov/psd/data/gridded/data.ncep.reanalysis2.html, 2024.

[41] NCO. http://nco.sourceforge.net/, 2024.

[42] Oracle database release. https://docs.oracle.com/en/database/oracle/oracle-database/21/geors/image-processing-virtual-mosaic.html, 21c.

[43] Oracle SG. oracle.com/database/technologies/spatialandgraph.html, 2024.

[44] C. Ordonez, Y. Zhang, and S. L. Johnsson. Scalable machine learning computing a data summarization matrix with a parallel array DBMS. *Distributed and Parallel Databases*, 37(3):329–350, 2019.

[45] I. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[46] S. Papadopoulos, K. Datta, S. Madden, and T. Mattson. The TileDB array data storage manager. *PVLDB*, 10(4):349–360, 2016.

[47] PostGIS. http://postgis.net/, 2024.

[48] RasDaMan home. http://rasdaman.org/, 2024.

[49] RasDaMan mosaic. https://doc.rasdaman.org/05_geo-services-guide.html#data-import-recipe-mosaic-map, 2024.

[50] J. A. Richards. *Remote Sensing Digital Image Analysis: An Introduction*. Springer-Verlag Berlin Heidelberg, 5th edition, 2013.

[51] R. A. Rodriges Zalipynis. Array DBMS in environmental science: satellite sea surface height data in the cloud. In *IDAACS*, pages 1062–1065. IEEE, 2017.

[52] R. A. Rodriges Zalipynis. Array DBMS: past, present, and (near) future. *PVLDB*, 14(12):3186–3189, 2021.

[53] R. A. Rodriges Zalipynis. BitFun: fast answers to queries with tunable functions in geospatial array DBMS. *PVLDB*, 13(12):2909–2912, 2020.

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

[55] R. A. Rodriges Zalipynis. ChronosDB in action: manage, process, and visualize big geospatial arrays in the Cloud. In *SIGMOD*, pages 1985–1988, 2019.

[56] R. A. Rodriges Zalipynis. Convergence of array DBMS and cellular automata: a road traffic simulation case. In *SIGMOD*, pages 2399–2403, 2021.

[57] R. A. Rodriges Zalipynis. Distributed in situ processing of big raster data in the Cloud. In volume 10742 of *LNCS*, pages 337–351. Springer, 2017.

[58] R. A. Rodriges Zalipynis. Evaluating array DBMS compression techniques for big environmental datasets. In *IDAACS*, volume 2, pages 859–863, 2019.

[59] R. A. Rodriges Zalipynis. FastMosaic in action: a new mosaic operator for Array DBMSs. *PVLDB*, 16(12):3938–3941, 2023.

[60] R. A. Rodriges Zalipynis. Generic distributed in situ aggregation for earth remote sensing imagery. In volume 11179 of *LNCS*, pages 331–342. Springer, 2018.

[61] R. A. Rodriges Zalipynis. SimDB in action: road trafic simulations completely inside Array DBMS. *PVLDB*, 15(12):3742–3745, 2022.

[62] R. A. Rodriges Zalipynis. Towards machine learning in distributed array DBMS: networking considerations. In volume 12629 of *LNCS*, pages 284–304. Springer, 2021.

[63] R. A. Rodriges Zalipynis and N. Terlych. WebArrayDB: A geospatial array DBMS in your web browser. *PVLDB*, 15(12):3622–3625, 2022.

[64] R. A. Rodriges Zalipynis et al. Array DBMS and satellite imagery: towards big raster data in the Cloud. In volume 10716 of *LNCS*, pages 267–279. Springer, 2018.

[65] R. A. Rodriges Zalipynis et al. Retrospective satellite data in the cloud: an array DBMS approach. In volume 793 of *CCIS*, pages 351–362. Springer, 2017.

[66] F. Rusu. Multidimensional array data management. *Foundations and Trends in Databases*, 12(2-3):69–220, 2023.

[67] Sentinel data access annual report. https://sentinels.copernicus.eu/web/sentinel/-/copernicus-sentinel-data-access-annual-report-2021, 2021.

[68] Sentinel Hub. https://www.sentinel-hub.com/, 2024.

[69] W. E. Splinter. Center-pivot irrigation. *Scientific American*, 234(6):90–99, 1976.

[70] D. C. Tomlin. *Geographic Information Systems and Cartographic Modeling*. Prentice-Hall, 1990.

[71] A. van Ballegooij. RAM: a multidimensional array DBMS. In *EDBT*, volume 3268, pages 154–165, 2004.

[72] S. Villarroya and P. Baumann. A survey on machine learning in array databases. *Applied Intelligence*, 53(9):9799–9822, 2023.

[73] S. Villarroya and P. Baumann. On the integration of machine learning and array databases. In *ICDE*, pages 1786–1789, 2020.

[74] W. Wen et al. A review of remote sensing challenges for food security with respect to salinity and drought threats. *Remote Sensing*, 13(1):6, 2020.

[75] WMTS. https://www.opengeospatial.org/standards/wmts, 2024.

[76] H. Xing and G. Agrawal. Accelerating array joining with integrated value-index. In *SSDBM*, pages 145–156, 2020.

[77] H. Xing and G. Agrawal. COMPASS: compact array storage with value index. In *SSDBM*, pages 1–12, 2018.

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

[79] M.-D. Yang, H.-H. Tseng, Y.-C. Hsu, and H. P. Tsai. Semantic segmentation using deep learning with vegetation indices for rice lodging identification in multi-date UAV visible images. *Remote Sensing*, 12(4):633, 2020.

[80] W. Zhao, F. Rusu, B. Dong, and K. Wu. Similarity join over array data. In *SIGMOD*, pages 2007–2022, 2016.

[81] W. Zhao, F. Rusu, B. Dong, K. Wu, A. Y. Ho, and P. Nugent. Distributed caching for processing raw arrays. In *SSDBM*, pages 1–12, 2018.

[82] W. Zhao, F. Rusu, B. Dong, K. Wu, and P. Nugent. Incremental view maintenance over array data. In *SIGMOD*, pages 139–154, 2017.