Polaris: An Interactive and Scalable Data Infrastructure for Polar Science

Yuchuan Huang, Ana Elena Uribe, Youssef Hussein, Kareem Eldahshoury, Grant Ogren, Mohamed F. Mokbel
University of Minnesota, USA
{huan1531,uribe055,husse408,eldah001,ogren091,mokbel}@umn.edu

ABSTRACT

Though polar scientists entertain having huge amounts of publicly available datasets, they face the challenge that working with such data is a cumbersome process that requires downloading tons of unnecessary data and writing various scripts on top of it. This hinders their ability to perform any kind of interactive analysis. This paper presents Polaris; a novel open-source system infrastructure for Polar science that is highly Interactive and Scalable. Polaris is designed based on three observations that distinguish the query workload of polar scientists, namely, all queries are spatio-temporal, not all data are equal, and the large majority of queries are aggregates. Polaris is equipped with a hierarchical spatio-temporal index structure that stores precomputed aggregates for data of interest. Experimental results with a real Polaris prototype and real scientific data show that it achieves highly interactive and scalable data access, enabling interactive analysis of polar science data.

PVLDB Reference Format:
Yuchuan Huang, Ana Elena Uribe, Youssef Hussein, Kareem Eldahshoury, Grant Ogren, Mohamed F. Mokbel. Polaris: An Interactive and Scalable Data Infrastructure for Polar Science. PVLDB, 18(11): 4644 - 4652, 2025.
doi:10.14778/3749646.3749719

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/iharp3/Polaris-VLDB2025/.

1 INTRODUCTION

Polar scientists face the challenge that even though there are huge amounts of climate and environment datasets archived online (e.g., ERA5 [18], CARRA [7], MERRA2 [31], ICESat-2 [24], CESM [9]), they are not easy to access and work on. For most of such datasets, scientists have to wait hours or days to first download the data, then write their own scripts for their analysis needs. This is definitely far from being an interactive user experience , which hinders the polar scientists' ability to perform their analysis. For example, ERA5 [18] is one of the most widely used datasets by polar scientists for simulation, prediction, and modeling [15, 23, 30, 45, 46]. It has 262 climate variables, including temperature, sea level, ice sheet, etc. Each variable is recorded at 0.25×0.25 latitude longitude degree spatial resolution for the whole world and for every hour from 1940 to present. So, one variable for the whole world consumes an annual storage of 17GB, which makes the overall ERA5 dataset size for 262 variables and 84 years approximately 374TB.

Query	Running time	Result size	Data download	Polaris
Daily	12m46s	34 MB	20.12 GB	1.2s
Monthly	12m55s	1.2 MB	20.12 GB	0.3s

Table 1: Query Performance Metrics

Due to its size, it is hard for polar scientists to host and manage such data, not to mention have efficient data access and interactive analysis. Currently, polar scientists employ one of the following three options to work with ERA5 data: (1) Call public APIs to download the parts of ERA5 data of interest to local storage and run local scripts. This is pretty inefficient as the downloading itself can take hours or more depending on the requested data size and network conditions, and then considerable efforts are needed to write and execute the analysis scripts. (2) Pre-download and store major parts of the data on a High Performance Computing (HPC) environment and run computations on HPC directly. Though this option will be best in terms of interactive analysis, it is the least used approach, as HPC environments are not available to the large majority of polar scientists worldwide. (3) An emerging trend from the geoscience community [35] is to transform the data into Analysis-Ready Cloud-Optimized (ARCO) format stored in a cloud storage [1, 43], e.g., ARCO-ERA5 is an ARCO version of ERA5 data hosted on Google Cloud [8]. Compared with the public APIs, ARCO integrates better with modern data ecosystems, yet it is still far from being interactive. Table 1 gives performance measures of running two queries on ARCO-ERA5 [8] for daily and monthly temperatures of Alaska in 2020. Though the result size for the daily query is 34MB and the monthly query is 1.2MB, both queries end up downloading the same amount of data of 20GB. The main reason is that ARCO-ERA5 must scan all the raw data before aggregation, making it less interactive on aggregate queries that are highly used by polar scientists. Since it takes close to 13 minutes to execute both queries, this is not suitable for any interactive analysis.

This paper presents Polaris; a novel open-source system infrastructure for Polar science that is highly Interactive and Scalable. Polaris came out as part of the iHARP project (institute for HArnessing data and model Revolution in the Polar regions) [25], which is a large collaboration effort between computer and polar scientists. Polaris is tailored to the query workload and access patterns of polar scientists, and hence it provides a highly interactive and scalable performance for the large majority of the queries it receives. For example, Polaris answers the daily and monthly queries of the ERA5 data in Table 1 in 1.2 and 0.3 seconds, respectively.

This work is supported by NSF grants OAC-2118285 and IIS-2203553.

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749719

Polaris is built with three main concepts in mind: (1) *All polar scientists queries are spatio-temporal.* Queries about any of the 262 variables of ERA5 request specific spatial and temporal ranges. If no ranges are specified, the whole temporal domain (1940 to present) and the whole world will be considered. (2) *Data Inequality.* Not all data are of equal importance. The desired resolution of a variable depends on the region of the world; snowfall may be needed in polar areas at an hourly temporal resolution and 0.25 degree spatial resolution for the last 10 years, then at daily and 0.5 degrees resolutions for earlier dates. Non polar areas could be kept at a much lower 1-degree monthly resolution. (3) *Most queries are aggregates.* Queries are mostly asking about an aggregate value of a specific variable, spatio-temporal range, and spatial and temporal resolutions, e.g., get the maximum snowfall (aggregate) of the days (temporal resolution) in the last year (temporal range) in 1-degree area (spatial resolution) in Alaska (spatial range).

To accommodate the *data inequality* concept, Polaris is equipped with a GUI interface that a polar scientist can easily use to declare their interest for each variable along with the required spatial and temporal resolutions. Polaris is supposed to be deployed by each research group with their administrators who would decide on their best interests. Once defined, Polaris accommodates the *aggregate queries* concept by triggering an offline process to download all the raw data of interest, which is available at the highest resolution. Then, it precomputes all aggregates per the requested lower resolutions, while throwing away unneeded higher resolution raw data. To accommodate the *spatio-temporal* queries concept, Polaris is equipped with a multi-layer spatio-temporal index structure built offline during the aggregation process, and queried online for interactive query analysis. The main objective of Polaris to answer the large majority of its incoming queries from its own local storage. When receiving a query that cannot be fully answered locally, Polaris partially answers it from local storage, then uses its index to decide on the minimum API calls needed to complete the query answer. Finally, Polaris is equipped with a *query monitoring* module that tracks incoming queries and whether they are answered from local storage. If the ratio of queries that need API calls is increasing, Polaris uses its query workload to update its data of interest and trigger the offline process to download and organize data accordingly.

2 POLARIS QUERY SIGNATURE

This section presents the query signature of all queries received by Polaris along with its 5 most common query types, which drives Polaris' design of index structure and query processing.

2.1 Query Signature

Polaris is designed to support the following query signature, which represents the family of queries posed by polar scientists to widely used public repositories like ERA5 [18].

```
SELECT <Spatial Resolution>, <Temporal Resolution>,
       <[ <Min/Max/Avg> ] variable>,
  FROM Data
 WHERE <Spatial Predicate> AND <Temporal Predicate>
[ GROUP BY <Spatial Resolution (0.25/0.5/1-degree)>,
           <Temporal Resolution (Hour/Day/Month/Year)>,
  [ HAVING <group predicate> ] ]
```

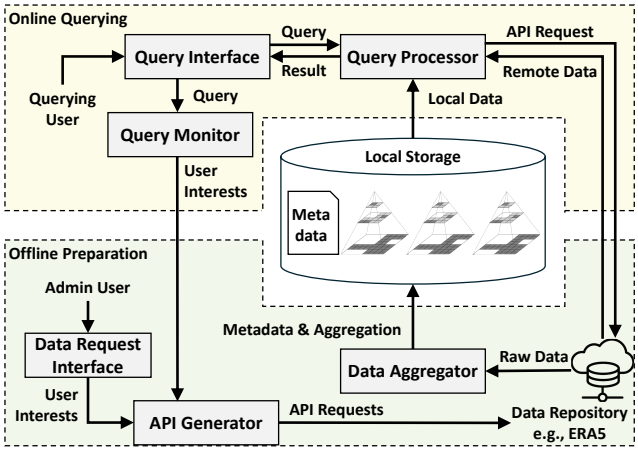


Figure 1: Polaris **System Architecture**

The signature assumes spatial and temporal predicates in all queries. Default predicates cover the whole spatial and temporal range. In ERA5 data, that is data from 1940 to present for the whole world. The *Data* in the signature refers to the whole 370TB ERA5 repository [18]. The query result can be grouped and aggregated by spatial resolution defined in coordinate degrees and/or temporal resolution defined in time units, and filtered on a group predicate.

2.2 Most Common Query Types

Below are five of the most common basic query types posed to Polaris. Other queries can be composed from these basic ones.

Get Variable Query. This query requests the aggregate value of a variable for each spatial area and time (at designated resolutions) within certain spatial and temporal ranges. For example, in the query *"Get the daily average temperature in 2020 of Alaska at 1-degree"*, the spatial predicate is Alaska, temporal predicate is 2020, spatial resolution is 1-degree, and temporal resolution is day.

```
SELECT Day, 1-degree, Avg(temperature)
  FROM ERA5
 WHERE SpatialRange = Alaska AND TemporalRange = 2020
GROUP BY Day, 1-degree
```

Heatmap/Timeseries Query. These two queries build on the previous one by aggregating over the temporal or spatial range. A heatmap query returns a list *<area, value>* for the average value for each area. A timeseries query returns a series of values over the whole area over time. For example: *"Get the average temperature of Alaska for each 1-degree/day during 2021/01/01 to 2023/01/31"*

```
SELECT 1-degree/Day, Avg(temperature)
  FROM ERA5
 WHERE SpatialRange = Alaska
   AND TemporalRange = 2021/01/01~2023/01/31
GROUP BY 1-degree/Day
```

Find Area/Time Query. These queries find the areas/time periods where a certain value predicate is satisfied at the given resolutions. For example, *"Find the 0.25-degree areas/days in Antarctica that had an average temperature greater than 270 Kelvin in 2023"*.

```
SELECT 0.25-degree/Day
  FROM ERA5
 WHERE SpatialRange = Antarctica AND TemporalRange = 2023
GROUP BY 0.25-degree/Daily
  HAVING Avg(temperature) > 270
```

3 POLARIS SYSTEM ARCHITECTURE

Figure 1 depicts the system architecture of Polaris, which is comprised of three main components, described below:

Offline Data Preparation Layer. This layer stores the most important data of ERA5 such that the large majority of queries received by Polaris are completely supported from local storage, providing an interactive analysis experience. The input to this layer comes from an admin user through a GUI data request interface that specifies the data of interest, which is then downloaded and aggregated to the requested resolutions. Details are in Section 4.

Local Indexed Storage. The input to the local storage is the aggregated data from the offline data preparation layer. Local storage is equipped with a hierarchical spatio-temporal index structure designed specifically to support Polaris query signature and workload. Local storage is accessed by the online query layer for interactive data analysis. Details are in Section 5.

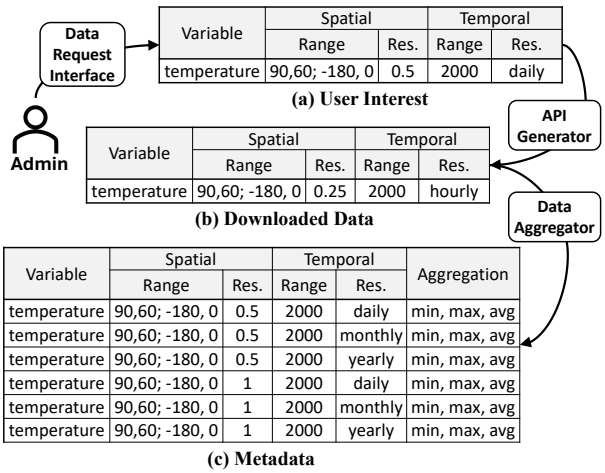
Online Query Layer. This layer gets user queries through a GUI map interface. It is then responsible for providing efficient query processing through the indexed local storage. It is also equipped with a query monitoring module that monitors the query workload and prompts the offline data preparation layer to dynamically adjust the local storage per workload changes. Details are in Section 6.

This architecture can be easily deployed to a distributed environment using existing techniques. For example, Polaris could use HDFS [22] to stitch the distributed disks for local storage and use Dask [13] to stitch the distributed memory for query processing. When storing the local storage index in the unified HDFS file space, the query process logic can remain unchanged through the array data abstraction provided by Dask.

4 POLARIS OFFLINE DATA PREPARATION

This layer handles two of the three concepts driving the design of Polaris, *data inequality* and *most queries are aggregates*. To realize data inequality, this layer adds only the most important data to the local storage. Data importance is either explicitly specified by an admin user or inferred from the system query workload. To efficiently support aggregate queries, this layer pre-computes various aggregates at the requested spatial and temporal resolutions. As in Figure 1, this layer has the following three components:

User Interest Table. This table has five attributes <*variable*, *spatial range*, *spatial resolution*, *temporal range*, *temporal resolution*>, which capture the data stored in the local storage. Figure 2a gives an example of one row of such table, where the user is interested in the temperature variable for the year 2000 in the rectangular geographical area bounded by latitudes 60 to 90 and longitudes 0 to -180, at a 0.5 degree spatial and daily temporal resolution. The table can be filled in two ways: (1) Through a GUI map interface where admin users specify their areas of interest, temporal range of interest, variables of interest, and the preferred spatial and temporal resolutions. If multiple users' interests overlap in the desired spatio-temporal ranges/resolutions, Polaris consolidates the interests to avoid storage redundancy. In particular, Polaris ignores the coarser resolution since it will be covered by the finer resolution as a result of the resolution hierarchy (Section 5). For example, if an additional row is added to Figure 2a with the same spatial range and resolution for temperature, but a yearly temporal resolution for



Variable	Spatial		Temporal	
	Range	Res.	Range	Res.
temperature	90,60; -180, 0	0.5	2000	daily

(a) User Interest

Variable	Spatial		Temporal	
	Range	Res.	Range	Res.
temperature	90,60; -180, 0	0.25	2000	hourly

(b) Downloaded Data

Variable	Spatial		Temporal		Aggregation
	Range	Res.	Range	Res.	
temperature	90,60; -180, 0	0.5	2000	daily	min, max, avg
temperature	90,60; -180, 0	0.5	2000	monthly	min, max, avg
temperature	90,60; -180, 0	0.5	2000	yearly	min, max, avg
temperature	90,60; -180, 0	1	2000	daily	min, max, avg
temperature	90,60; -180, 0	1	2000	monthly	min, max, avg
temperature	90,60; -180, 0	1	2000	yearly	min, max, avg

(c) Metadata

Figure 2: Offline Data Preparation

2000 to 2010, Polaris will re-write this temporal range to 2001-2010. The yearly resolution for 2000 is produced when pre-computing the temporal hierarchy for 2000. (2) Through the query monitoring module (Section 6.2) that automatically populates this table based on the observed query workload. In either case, any significant change in the table triggers the following API generator module.

API Generator. This module is triggered by any significant change in the user interest table. It digests all the table entries to find the minimum set of APIs needed to issue to a remote repository such as ERA5 cloud services [18] to download the requested data. Additionally, it adjusts the requested resolutions to those available for that repository. Figure 2b depicts the API request sent to ERA5. The spatial and temporal resolutions are changed from the user interest table to the closest available resolutions in ERA5.

Data Aggregator. This module takes the raw data downloaded from the API generator and performs three operations: (1) It aggregates the raw data to the requested resolution and deletes unneeded raw data. The example in Figure 2 only requests spatial and temporal resolutions of 0.5 and daily, so after aggregation we delete the raw data. (2) It precomputes the aggregations at the requested resolutions along with coarser ones. For Figure 2, the data is requested for 0.5 daily resolution, but we also compute and store the aggregate values (min, avg, max) for all areas of 1-degree spatial resolution with monthly and yearly temporal resolutions. (3) It stores a metadata table for all locally stored aggregated data, which is used by the index structure and query processor to locate such data. Figure 2(c) gives the metadata for this particular example, where the six rows refer to the set of all precomputed daily, monthly, and yearly aggregates for each of the 0.5 and 1 degrees spatial resolutions.

With these three components, the offline data preparation layer achieves the following: (1) Significant storage saving. In Figure 2, the size of the downloaded hourly 0.25 raw data is 1.4GB, while the size of the aggregated data is only 15MB for each of the first three rows of Figure 2c and 3.6MB for the last three rows. So, total local storage is 56MB, which is only 4% of the raw data size of 1.4GB. This means 96% storage saving by only storing the data needed by polar scientists in this area. (2) With storage saving, we are able to store more of the user's defined or inferred data of interest. This allows the large majority of queries to be answered completely from local storage, enabling interactive query responses. (3) As

most of the received queries are aggregates, the answer is already pre-computed, which enables an interactive query response.

5 POLARIS INDEX STRUCTURE

Given the data we have from the offline data preparation module, we can just dump this data into a big data system, e.g., TileDB [37], Apache Sedona [41], or SpatialHadoop [16], and use that system for all queries. However, these systems lack the spatio-temporal index structures that are immensely useful for the spatio-temporal queries posed by polar scientists. The systems that most closely support such query workload are Sedona [41] and TileDB [37]. However, Sedona has only spatial index which only indexes the whole spatial bounding box of the entire array but cannot support the spatial predicates (i.e., slicing selection) and resolutions inside the array. Not to mention its lack of temporal index and support for temporal predicates and resolution. Meanwhile, even though TileDB [37] efficiently supports queries over three-dimensional slices, these slices are not spatial- or temporal-aware, so it also does not natively support aggregation for spatial and/or temporal resolutions. This is why Polaris has to design its own index structure guided by the query signature defined in Section 2.1, which calls for a native index structure with the following three properties: (a) *spatio-temporal*, as both the spatial and temporal predicates are mandatory in all queries, (b) *hierarchical*, as queries impose a natural hierarchy across spatial and temporal resolutions (e.g., 0.25 to 1 degree and hour to year for ERA5) (c) *precomputed aggregates in index entries*, as most queries request aggregate values.

5.1 Basic Spatial Indexing Unit

The basic indexing unit in Polaris is an incomplete pyramid structure [3]. It indexes a variable of interest and stores its aggregate values over the supported spatial resolutions in hierarchical levels: The lowest pyramid level divides the geographical space of the whole world into non-overlapping equal-size cells of the highest resolution. Higher levels store coarser resolutions such that each cell is an aggregation of four cells of its lower level. Not all levels have the same temporal interval and temporal resolution, and not all cells are maintained, hence the term *incomplete* pyramid. Maintained cells store the *minimum*, *average*, and *maximum* values of the indexed variable. All ancestor cells of a maintained cell are maintained with another level of aggregation. Thus, the basic incomplete pyramid index structure supports various spatial resolutions through its multi-level hierarchy, supports spatial predicates as it partitions the world into spatial areas, and stores aggregates over spatial areas in its maintained cells.

5.2 Spatio-temporal Index Structure

Figure 3 depicts the full spatio-temporal index structure of Polaris for each indexed variable. It is composed of multiple instances of the basic incomplete pyramid unit to support temporal aggregation and resolutions. Yearly, monthly, and daily pyramids are aggregates over the lower level pyramids. Like the spatial levels of a basic pyramid unit, not all pyramids are maintained at all temporal levels. For example, consider the ERA5 data of interest presented in Figure 2 that requests the temperature variable in a certain spatial and temporal range at 0.5 and daily resolutions. For this variable, we

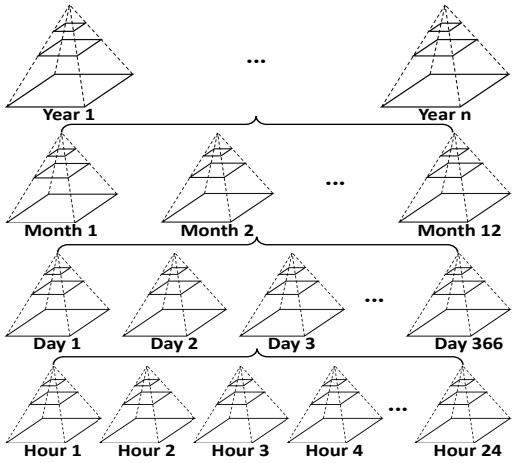


Figure 3: Pyramid Hierarchy

are only interested in maintaining the yearly, monthly, and daily pyramids, and no hourly pyramids will be maintained anywhere. Also, each pyramid unit will only have its two top levels storing 0.5 and 1-degree data. Finally, all pyramids will only be maintained for the year 2000, and for the specified area.

Practically speaking, the table in Figure 2a could have tens or hundreds of areas of interest. All areas of the same variable would share the same hierarchy of pyramids of Figure 3. As both pyramid and hierarchical structure are incomplete by design, the same variable may have hourly pyramids for certain areas, and daily/monthly/yearly for others. For example, if there is another interest for the temperature value over a different area in the world at 0.25 and hourly resolution for March 2020, we would add this new area at the lowest pyramid unit level of 0.25, and update the monthly pyramid for March 2020 to include this area of interest.

It is important to emphasize that this hierarchical index structure already includes the aggregates needed to answer queries without the raw data. So, even though it may seem like an overhead to maintain that many pyramids in the index, it actually saves a huge amount of storage and provides highly efficient query support. As mentioned in Section 4, the storage needed to support the data of interest of Figure 2 is 56MB, which is only 4% of the 1.4GB raw data. Thus, using only 4% of the raw data storage, and organizing data within our index structure, we can answer the same queries we would normally need the raw data for.

5.3 Index Update

As the ERA5 dataset is updated on a daily basis, in case the admin user indicates interest in the most recent data, Polaris downloads the new data, computes pre-aggregation and updates its hierarchial index every day. Fortunately, the overhead for updating the pyramid hierarchy index is pretty manageable. On the hourly and daily level, only 24 hourly pyramids and one daily pyramid are added. No existing pyramids are changed. For the monthly and yearly level, only the current month/year pyramid on each level is affected. The minimum and maximum aggregate values are updated by comparing the new data with the current aggregates. The new average aggregate is also updated by taking the weighted average of the current value, whose weight is the number of current days in the month/year, and the new day average whose weight is 1.

6 POLARIS ONLINE QUERY LAYER

In addition to a map GUI interface, Polaris online query layer is composed of two main modules, a query processor that exploits the index structure for highly efficient query execution, and a query monitoring to ensure that incoming queries are supported locally.

6.1 Query Processor

Polaris query processor aims to exploit its hierarchical index structure to efficiently answer incoming queries from local storage. For those queries that cannot be fully locally answered, it aims to partially answer these queries, and call an API to augment the partial answer. The promise of Polaris is that a large ratio of incoming queries, default 95%, will be answered from local storage, mainly due to the careful design of the data preparation layer and the query monitoring module. Below, we discuss the query processor operation for the five query types mentioned in Section 2, when the query answer is completely in the local storage.

Get Variable Query. Consider the query: *"Get the daily average temperature in 2020 in Alaska at 1-degree"*. Without Polaris, answering such query would be computationally exhaustive as all the data for Alaska in 2020 will be downloaded at hourly level with 0.25-degree resolution as this is the only available resolution for ERA5 [18]. Then, a scan with aggregation over this data is needed. With Polaris, we just go right away to the 366 daily level pyramids of 2020, where we just visit the very top level of each pyramid that represents the 1-degree resolution and return the areas that are within Alaska. This shows that Polaris precomputations and index structure significantly help in supporting such queries.

Heatmap Query. Consider the query: *"Get a 1-degree heatmap of average temperature in Alaska during the days from 2021/01/01 to 2023/01/31"*. Without Polaris, producing such heatmap would need to download 0.82GB of data that correspond to the hourly 0.25 data over Alaska for the requested time range, and then perform aggregation on top of it. It takes hours to download such data. With Polaris, we just need to use three pyramid structures, the two yearly pyramids of 2021 and 2022, and a monthly pyramid of January 2023. The top level of each pyramid for Alaska area represents the 1-degree heatmap for Alaska for its corresponding temporal resolution. Then, the query answer is the weighted average of these three top level areas. This makes the query response in real time, showing the impact of Polaris for polar scientist analysis.

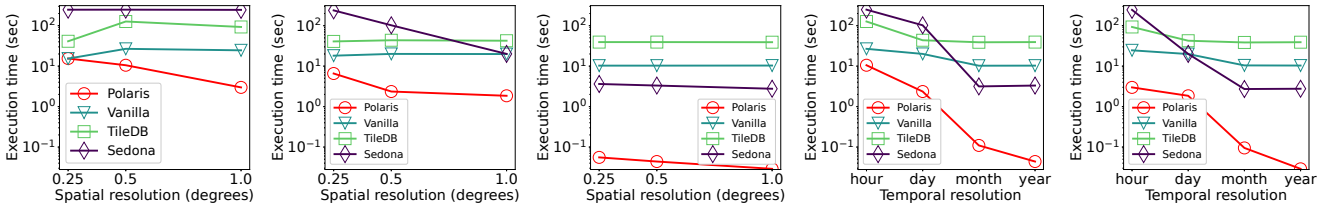
Timeseries Query. Consider the query: *"Get the daily average temperature in 2022 in Greenland"*. Without Polaris, we would need to download all the hourly 0.25-degree data for Greenland for 2022, which is ~0.4GB. Then, scan over the downloaded data to aggregate for the daily resolution and over the whole Greenland area. With Polaris, we will only access the 365 daily pyramids of 2022. For each of these pyramids, we may only need to check the top pyramid level to cover Greenland area with the 1-degree cells. If such cells cover an area that is significantly more than Greenland, we use the 0.5 or 0.25-degree cells of lower levels to give a more accurate average for the whole area. If we use cells from different levels, we take a weighted average. This shows the impact of Polaris pre-computations and index to provide interactive performance.

Find Area Query. Consider the query: *"Find the 0.25-degree areas in Alaska that had an average temperature more than 300 Kelvin in 2023"*. Without Polaris, we would need to download all the 0.25-degree hourly data of 2023 in Alaska, and scan them to find those areas with temperature over 300. With Polaris, we take advantage of the pyramid hierarchy for early filtering. We start with the yearly pyramid of 2023, and focus on its top level of 1-degree resolution for the area covering Alaska. If none of these areas have temperature more than 300, then there is no need to proceed further. Only for those 1-degree areas that satisfy the predicate, we will check on their 12 monthly pyramids. Similarly, we will only go for the daily pyramids if a parent monthly pyramid shows that there is a temperature more than 300. In the daily pyramid, we start by the top level, where if any of the Alaska cells does not satisfy the query predicate, then, it is another early filter, and we do not need to proceed further for such cells. For those cells that satisfy the query predicate, we go to the medium level for another early filter, and then only for the 0.25 cells that have parents satisfy the predicate. We finally output the result from the lowest pyramid level with minimal overhead due to the early filtering from Polaris index.

Find Time Query. Consider the query: *"Find days from 2021/01/01 to 2023/01/31 where the average temperature in Antarctica is greater than 300 Kelvin"*. Without Polaris, we will need to download all the raw hourly 0.25-degree data for Antarctica for two years and one month. Then, scan such data to get the answer. With Polaris, similar to Find Area query, we use the spatial and temporal aggregation for filtering. We start by the yearly pyramid of 2021. If none of the cells that cover Antarctica in the top level satisfy the query predicate, then we know that no day in this year will satisfy the query. If any of the cells satisfy the predicate, we visit the 12 monthly pyramids for 2021 and check for the same cells that correspond to the satisfying cell in the yearly pyramid. We do similar check and visit the daily pyramids only for those monthly pyramids satisfying the predicate. We will also do the same for the yearly pyramid of 2022 and the monthly pyramid of January 2023.

6.2 Query Monitoring

The main promise of Polaris is that the large majority of queries will be answered from local storage. This assumes that the offline data preparation layer has successfully captured the user interests and has pre-downloaded and pre-aggregated the data needed for incoming queries. Yet, the system usage and query workloads may change over time, which would reduce the ratio of queries answered from local storage. To avoid such performance drop, Polaris actively monitors the query workload and updates local storage if new data of interest is detected. Polaris employs a query monitoring module that tracks: (1) user-issued queries in terms of their variable and spatial/temporal range and resolution, kept in a query table Q that has the same schema of the table in Figure 2a, and (2) the ratio of queries α that were not completely supported from local storage. Once α exceeds a user-specified value, default 5%, the query monitor analyzes table Q and infers the actual user interest. Since the spatial and temporal query ranges may overlap, Polaris determines the ranges that cover most queries and adds those as user interests. The inferred interests are sent to the Data Preparation layer of Section 4 to trigger a local storage data update. Since it is treated as an additional user interest input, the same condensing logic that avoids storage redundancy applies here.



(a) Hour (b) Day (c) Year (d) 0.5 degrees (e) 1.0 degrees

Figure 4: Impact of Spatial and Temporal Resolutions

Resolution	Get Variable	Heatmap	Find Time
3-hour, 0.5x0.625^o	3.205	3.232	5.093
day, 1x1.25^o	0.088	2.140	0.078
year, 2x2.5^o	0.024	0.655	0.030

Table 2: Polaris Execution time (sec) for MERRA-2 data

Metric	Polaris	Vanilla	TileDB	Sedona
Agg. time	3h 51m	0	>20h	est. 67h
Storage	292GB	170GB	648GB	est. 348GB

Table 3: Offline Preparation Time and Storage

7 EXPERIMENTAL RESULTS

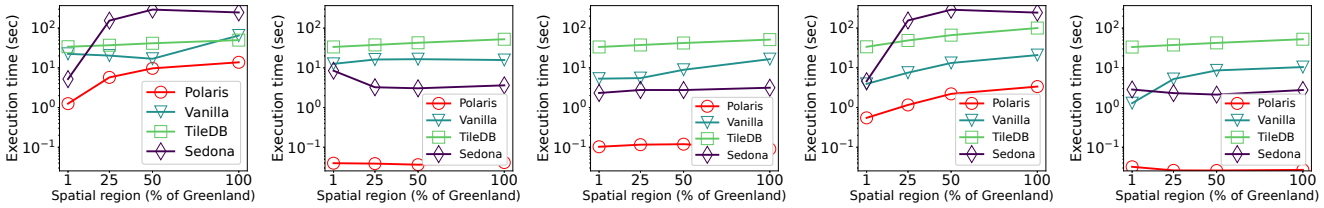
This section evaluates the performance of Polaris, based on its first released version [39]. As mentioned in Section 1, the direct competitor of Polaris is what is currently in use by polar scientists, which is having the data in a cloud storage [1, 43], e.g., ARCO-ERA5 [8]. Yet, as reported in Table 1 in Section 1, this solution is far from being interactive. Hence, it is unreasonable to compare it against Polaris, which provides sub second performance. Instead, we focus our experiments on the main internal components of Polaris: its incomplete spatio-temporal pyramid index structure supported by the aggregation module in the offline layer and the query processor module in the online layer.

We compare Polaris with three alternatives: (1) A **vanilla** approach of Polaris, where we pre-download all data of interest but do not perform any aggregation or indexing. (2) **TileDB** [37]. We store and query the downloaded data of interest in TileDB, a state-of-the-art storage engine for multi-dimensional arrays, using its up-to-date open-source distribution [44]. (3) **Apache Sedona** [41]. Although Sedona can load array data, it does not support spatial/temporal predicates and aggregation, which is always needed by the query signature of Section 2. In essence, it has no ability to select a subset of an array within a spatio-temporal range, or to aggregate the array into heatmaps or timeseries. Thus, to include Sedona in our experiments, we had to first convert downloaded array data to tabular data. For example, we store temperature data in table schema (*time*, *latitude*, *longitude*, *temperature*). Then, we perform the experiments using the SQL queries that Sedona supports. All experiments are performed on a Linux server with 20 CPU@2.2GHz, 96GB memory and 800GB SSD.

We benchmark Polaris and competitors on the ERA5 dataset [18] that has 0.25×0.25-degree spatial resolution and 1-hour temporal resolution. We use ten years of temperature data for the whole world, and ingest it into each system. An additional dataset, MERRA2 [31], which has coarser resolutions (0.625×0.5-degree and 3-hour) is used to confirm Polaris performance is generalizable to other climate datasets. Table 2 shows that Polaris is able to support such data for three main queries (Get variable, Heatmap, Find time) and three different resolutions with an interactive response, where all the queries were for all data over the last five years. Due to space limitation, in the rest of this section, we will only focus on ERA5 data, yet the trends in all experiments should be very similar when applied to MERRA2 data.

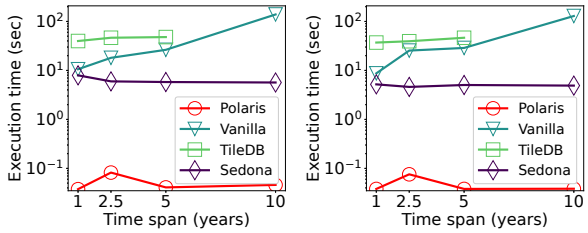
Offline Preparation Time. Table 3 compares the offline preparation time of Polaris, vanilla, TileDB, and Sedona. Our intention is to run these experiments for one variable for the whole world over the last 10 years, however, only Polaris and vanilla were able to do so. In particular, the vanilla approach does not encounter any offline preparation time as it just works with raw data as is. Also, the storage is the minimum, which is the 170GB of raw data. Meanwhile, Polaris completes the data aggregation described in Section 4 in less than 4 hours, and adds 122GB of metadata, on top of the raw data. Even though we are reporting total storage as 292GB, this would be the case only if a user explicitly specifies their interest in raw data, which is unlikely. For TileDB, the data overhead for 10 years exceeded our storage limit of 800GB, so, we had to run it for 7 years only, which took more than 20 hours with a storage overhead of 648GB. For Sedona, it takes excessive time to convert the downloaded 170GB raw data into the tabular form, and we could not do it for the whole world. So, we did it only for the area of Greenland over the last 10 years, and extrapolated the result to estimate what it would take for the whole world. While the storage overhead is reasonable, the preparation time is prohibitive. Overall, this experiment shows that Polaris has the least overhead time and storage compared to TileDB and Sedona.

Impact of Spatial and Temporal Resolutions. Figure 4 studies the impact of varying the spatial and temporal resolutions on the performance of Polaris, TileDB, and Sedona for the Get Variable Query: "*Get the average temperature over Greenland over the last five years in T temporal resolution and S spatial resolution*". Figures 4(a)-4(c) fix the temporal resolution T to hour, day, year, while varying the spatial resolution from 0.25 to 1. Except for the very first point of 0.25 hourly resolution, Polaris consistently performs one to three orders of magnitude better than vanilla, TileDB, and Sedona. This is mainly because Polaris has less computations to perform on its aggregate values while other systems must scan and aggregate the raw data. Sedona also improves with lower resolutions but still maintains a larger execution time compared to Polaris. Figures 4(d)-4(e) fix S to 0.5 and 1-degree, while varying T from hour to year. It also show a performance gain improvement for Polaris as the temporal resolution gets lower.



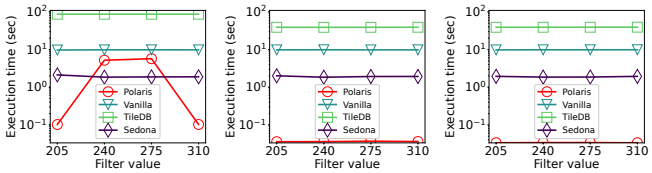
(a) 0.25 degrees, hourly (b) 0.25 degrees, yearly (c) 0.5 degrees, monthly (d) 1.0 degrees, hourly (e) 1.0 degrees, yearly

Figure 5: Impact of Result Size



(a) 0.25 degrees (b) 1.0 degrees

Figure 6: Heatmap Query Performance



(a) Hour (b) Month (c) Year

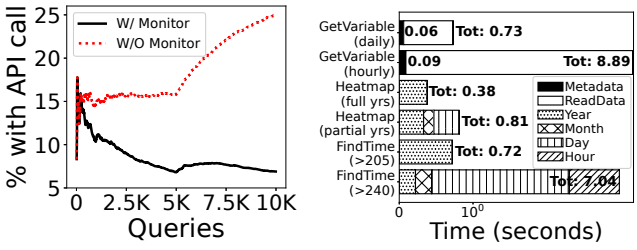
Figure 7: Find Time Query Performance

Impact of Result Size. Figure 5 studies the impact of the query result size on the performance of Polaris, TileDB, and Sedona for the Get Variable Query: "*Get the average temperature over area A in Greenland over the last five years for spatial and temporal resolutions S and T*", when varying area A from 1% to 100% of the total area of Greenland. In all aggregation cases, Polaris achieves one to three orders of magnitude better performance than vanilla, TileDB and Sedona. Again, we see the performance of Polaris and Sedona improve at lower spatial and temporal resolutions. For example, the performance gap between Polaris and alternatives in Figure 5(d) is about an order of magnitude when S is 1-degree and T is hourly, but three orders of magnitude in Figure 5(e) when T is yearly. Polaris achieves sub second execution time except when T is hourly and it must read 8,760 values per year requested (one per hour).

Heatmap Queries. Figure 6 studies the performance Polaris, TileDB, and Sedona when plotting a heatmap at various spatial resolutions over a certain area, while varying the time span. This is done through the heatmap query: *"Build an S-degree heatmap of average temperature in Greenland during the last T yeas"*, where T varies from 1 to 10 years, while S is 0.25 in Figure 6(a) and 1 in Figure 6(b). Polaris consistently gives up to four orders of magnitude better than the vanilla approach, three orders of magnitude better than TileDB and two orders of magnitude better than Sedona. We were not able to run TileDB for more than 7 years of data, due to its oversized storage. The main reason behind such performance is that heatmaps are mainly plotted based on the aggregated values that are already precomputed in Polaris, and hence it can be done in milliseconds for interactive analysis. The little bump in Polaris performance when plotting the heatmap for 2.5 years is due to exploiting the monthly pyramids, while in all other cases, we only need to exploit the yearly pyramids.

Filter Queries. Figure 7 studies the impact of varying the filter value on the performance of Polaris, TileDB, and Sedona for the Find Time query: *"Find the hours/days/months/years in the last 5 years in Greenland when the average temperature is greater than K Kelvin"*. We vary the value of K from 205 to 310, when the query asks about hours, days, and years that has average temperature above K across Greenland. TileDB, Sedona, and vanilla are not impacted by the filter value, as they need to scan the hourly raw data in all cases. Polaris performance is more than four orders of magnitude better than other approaches when returning the *months* and *years* that encounter higher temperature above K. This is mainly due to the precomputed aggregation. For example, in the yearly case (Figure 7(c)), Polaris only needs to check five values that correspond to five years, and will report only that year that has higher temperature. For querying over the hour (Figure 7(a)), Polaris would exploit its pyramid structure for early pruning or early result reporting. To find those hours when temperature is more than 205, Polaris can return all hours of a certain day/month/year without checking any hour, if the minimum of that day/month/year is above 205. This is why Polaris has two orders of magnitude better performance in the values of 205 and 310. At the middle temperatures, where there is not much filtering, Sedona outperforms Polaris by 3 seconds since Polaris cannot filter out as many hours and Sedona has better parallelism in scanning its tabular data.

Impact of Query Monitoring. Figure 8(a) studies the impact of Polaris query monitoring module using a synthetic workload of 10K queries. The x-axis represents the total amount of queries received by Polaris so far, while the y-axis represents the percentage of queries that required API calls. The workload is designed such that 85% of the first 5K queries can be completely answered from local data, then for the next 5K queries the percentage is 65%. So, the first half of the workload imitates the case where the initial user interest input mostly captured the requested queries, while the second half imitates the case where there is an interest shift. The red dotted line in Figure 8(a) indicates the percentage of API calls when we disable Polaris monitoring and updating strategy. For the first 5K queries, this percentage converges to around 15%, then continuously increases once the interest shifts. With Polaris monitoring module that is triggered with 5% threshold, the number of API-involving queries is suppressed to around 7.5% in the first half. More importantly, it is constrained at the same level even after the workload changes. This demonstrates the resilience to dynamic workloads of the Polaris monitoring and updating strategy.



(a) Simulation on Workload (b) Query Time Breakdown

Figure 8: Workload Simulation and Query Breakdown

Query Time Breakdown. Figure 8(b) shows the query time breakdown for three representative queries, Get Variable, Heatmap, Find Time. The top two bars in the Figure are for Get Variable query with daily and hourly temperature for Greenland over five years. Both take only few milliseconds to check the metadata. Yet, the hourly query takes more overall time as it needs to access more pyramids structures, one for each hour. Overall, the experiment shows that the overhead of metadata access is minimal for this query. The middle two bars of the Figure are for heatmap queries on Greenland for five years at 1-degree. Yet, the five years of the top bar are full ones from 2015/01/01 to 2019/12/31, while the five years of the other bar are from 2015/03/10 to 2020/03/09. The query time breakdown is depicted by the time spent in each temporal hierarchy level. The first query only needs the pyramid index structures at the yearly level, while the second query needs to access pyramids at yearly, monthly, and daily levels. The last two bars in the Figure are for Find Time queries asking about the hours in the last five years with an average temperature greater than 205 and 240 over Greenland. The first query only needs the yearly pyramids as it happened that all the five years have a minimum temperature greater than 205, hence, there is no need to dig further, and we will just report all the hours of this year. However, for the second query, the answer cannot be determined solely by the yearly level, so Polaris checks the monthly, daily, and hourly levels, which takes more time.

Partial Query Execution. Polaris aims to have 95% of its queries executed completely from local storage. To justify the need for this, we run two sets of queries, where the first set is completely answered from local storage, while the second set needs to issue an API call as 10% of the requested data is not in the local storage. We observed a rapid increase in query execution time from sub-second to over 4000 seconds with only a small API call. Regardless of the query request size, the ERA5 data repository adds a huge overhead for API requests. This shows that it is impossible to achieve an interactive user experience with remote data access, which justifies our design choices in Polaris.

Multi-Variables Query. Polaris supports multi-variable queries by issuing multiple queries and combining the result. We test this by running queries asking for the daily average for one to five variables for five years on Greenland, which are executed in 0.66, 1.38, 1.95, 2.66, and 3.31 seconds, respectively. As expected, the response time grows relatively linearly with respect to the number of variables. Since the overhead for one variable query is ~0.66, the overhead of combining the results is negligible. This shows that the design of Polaris supports multi-variable queries.

8 RELATED WORK

Array Database systems. Motivated by the fact that massive data are generated in various scientific domains, e.g., earth, space, and life sciences, huge efforts were dedicated to manage and analyze such data [5, 50]. This results in full-fledged systems, including Rasdaman [4], SciDB [6], TileDB [37], and ChronosDB [49]. TileDB, as a representative of such efforts, has its own specialized storage format and can support spatial and temporal predicates. However, it does not have native support for the resolution-based aggregation as it deals with the spatial and temporal dimensions as plain discrete values with no awareness of the space and time context. Meanwhile, the rise of Apache Hadoop [22] and Spark [42] inspired the efforts of SpatialHadoop [16], SHAHED [17], Apache Sedona [41, 48], and GeoTrellis [19] to manage big spatial raster data, though none is addressing the temporal dimension. Among those, Sedona is most active, yet it focuses on in-memory computing, which is not suitable for polar scientists data and query workload.

Spatio-temporal Indexing. While there is already a rich literature for spatio-temporal index structures [29, 32, 33], none of them can be directly applied to support polar science data. The main reason is that most of these index structures are designed for vector-like moving objects such as trajectories [11, 14, 20, 21, 27, 36, 40] and multimedia [10, 14, 28]. Although there are some studies on indexing raster satellite images [17, 26, 38], the temporal dimension is studied as an attribute associated with each image, not a dimension of the raster. So, temporal predicates in polar science queries cannot be directly supported. Though the ST-Hadoop [2] system is equipped with spatio-temporal index structures, it is mainly for offline analysis with no support to interactive data analysis.

Polar and Geoscience community. Polar and geo-scientists have made their efforts to store and share large raster data. Most of these efforts, e.g., Pangeo [35], CryoCloud [12] and OpenDataCube [34], focus on the integration with modern cloud-based infrastructure and democratizing data access to broader audience. For example, Pangeo [35] hosts raster data in an Analysis-Ready Cloud-Optimized (ARCO) format [1, 43], which can be accessed directly via open source libraries like Dask [13] and Xarray [47]. However, this does not provide any kind of interactive data analysis as data always needs to be transmitted from cloud to some disaggregated compute resources to run the analysis.

9 CONCLUSION

This paper presented Polaris; a novel system built to support polar science data and query workload in an interactive and scalable way. Polaris architecture is designed with an offline data preparation layer to harvest user interests on data, an online query processing layer to answer the queries efficiently, and a local storage that indexes the data spatio-temporally. Polaris is equipped with a novel spatio-temporal index structure, made of a hierarchy of incomplete pyramids that support efficient data access within a spatio-temporal range at any spatial and temporal resolution. Experimental results on real datasets show that Polaris achieves interactive response time and is scalable to support polar science workload. Although Polaris is designed for polar science workloads, it has potential application in general climate, environment, oceanology, atmosphere science as they share similar spatio-temporal data and query signature as that of polar scientists.

REFERENCES

[1] Ryan Abernathey, Tom Augspurger, Anderson Banihirwe, Charles C. Blackmon-Luca, Timothy J. Crone, Chelle L. Gentemann, Joseph Hamman, Naomi Henderson, Chiara Lepore, Theo A. McCaie, Niall H. Robinson, and Richard P. Signell. 2021. Cloud-Native Repositories for Big Scientific Data. *Comput. Sci. Eng.* 23, 2 (2021), 26–35.

[2] Louai Alarabi and Mohamed F. Mokbel. 2017. A Demonstration of ST-Hadoop: A MapReduce Framework for Big Spatio-temporal Data. *PVLDB* 10, 12 (2017), 1961–1964.

[3] Walid G. Aref and Hanan Samet. 1990. Efficient Processing of Window Queries in The Pyramid Data Structure. In *PODS* (Nashville, TN, USA).

[4] Peter Baumann, Andreas Dehmel, Paula Furtado, Roland Ritsch, and Norbert Widmann. 1998. The Multidimensional Database System RasDaMan. In *SIGMOD*.

[5] Peter Baumann, Dimitar Misev, Vlad Merticariu, and Bang Pham Huu. 2021. Array databases: concepts, standards, implementations. *J. Big Data* 8, 1 (2021), 28.

[6] Paul G. Brown. 2010. Overview of sciDB: large scale array storage, processing and analysis. In *SIGMOD* (Indianapolis, Indiana, USA).

[7] CARRA [n.d.]. Arctic regional reanalysis on single levels from 1991 to present. https://cds.climate.copernicus.eu/datasets/reanalysis-carra-single-levels.

[8] Robert W. Carver and Alex Merose. [n.d.]. ARCO-ERA5: An Analysis-Ready Cloud-Optimized Reanalysis Dataset. https://github.com/google-research/arco-era5.

[9] CESM [n.d.]. NCAR Community Earth System Model. https://www.cesm.ucar.edu/.

[10] Lisi Chen, Gao Cong, and Xin Cao. 2013. An efficient query indexing mechanism for filtering geo-textual data. In *SIGMOD* (New York, NY, USA).

[11] Su Chen, Beng Chin Ooi, Kian-Lee Tan, and Mario A. Nascimento. 2008. ST^{2}B-tree: a self-tunable spatio-temporal b$^+$-tree index for moving objects. In *SIGMOD* (Vancouver, BC, Canada).

[12] CryoCloud [n.d.]. CryoCloud - Accelerating discovery and enhancing collaboration for NASA Cryosphere communities. https://cryointhecloud.com/.

[13] Dask [n.d.]. Dask. https://www.dask.org/.

[14] Harish Doraiswamy, Huy T. Vo, Cláudio T. Silva, and Juliana Freire. 2016. A GPU-based index to support interactive spatio-temporal queries over historical data. In *ICDE* (Helsinki, Finland).

[15] Job CM Dullaart, Sanne Muis, Nadia Bloemendaal, and Jeroen CJH Aerts. 2020. Advancing global storm surge modelling using the new ERA5 climate reanalysis. *Climate Dynamics* 54 (2020), 1007–1021.

[16] Ahmed Eldawy and Mohamed F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In *ICDE* (Seoul, South Korea).

[17] Ahmed Eldawy, Mohamed F. Mokbel, Saif Al-Harthi, Abdulhadi Alzaidy, Kareem Tarek, and Sohaib Ghani. 2015. SHAHED: A MapReduce-based system for querying and visualizing spatio-temporal satellite data. In *ICDE* (Seoul, South Korea).

[18] ERA5 [n.d.]. ERA5 hourly data on single levels from 1940 to present. https://cds.climate.copernicus.eu/datasets/reanalysis-era5-single-levels.

[19] GeoTrellis [n.d.]. GeoTrellis. A geographic data processing engine for high performance applications. https://geotrellis.io/.

[20] Yang Guo, Zhiqi Wang, Jin Xue, and Zili Shao. 2024. A Spatio-Temporal Series Data Model with Efficient Indexing and Layout for Cloud-Based Trajectory Data Management. In *ICDE* (Utrecht, The Netherlands).

[21] Marios Hadjieleftheriou, George Kollios, Vassilis J. Tsotras, and Dimitrios Gunopulos. 2006. Indexing spatiotemporal archives. *VLDB J.* 15, 2 (2006), 143–164.

[22] Hadoop [n.d.]. Apache Hadoop. https://hadoop.apache.org/.

[23] Hans Hersbach, Bill Bell, Paul Berrisford, Shoji Hirahara, András Horányi, Joaquín Muñoz-Sabater, Julien Nicolas, Carole Peubey, Raluca Radu, Dinand Schepers, et al. 2020. The ERA5 global reanalysis. *Quarterly journal of the royal meteorological society* 146, 730 (2020), 1999–2049.

[24] ICESat-2 [n.d.]. Ice, Cloud, and land Elevation Satellite-2 (ICESat-2). https://icesat-2.gsfc.nasa.gov/.

[25] iharp [n.d.]. iHARP: NSF HDR Institute for Harnessing Data and Model Revolution in the Polar Regions. https://iharp.umbc.edu/.

[26] Lubos Krcál and Shen-Shyang Ho. 2015. A SciDB-based Framework for Efficient Satellite Data Storage and Query based on Dynamic Atmospheric Event Trajectory. In *SIGSPATIAL* (Bellevue, WA, USA).

[27] Ruiyuan Li, Huajun He, Rubin Wang, Yuchuan Huang, Junwen Liu, Sijie Ruan, Tianfu He, Jie Bao, and Yu Zheng. 2020. JUST: JD Urban Spatio-Temporal Data Engine. In *ICDE* (Dallas, TX, USA).

[28] Amr Magdy and Mohamed F. Mokbel. 2017. Demonstration of Kite: A Scalable System for Microblogs Data Management. In *ICDE* (San Diego, CA, USA).

[29] Ahmed R. Mahmood, Sri Punni, and Walid G. Aref. 2019. Spatio-temporal access methods: a survey (2010 - 2017). *GeoInformatica* 23, 1 (2019), 1–36.

[30] Pedro Mateus, João Catalão Fernandes, Virgílio B. Mendes, and Giovanni Nico. 2020. An ERA5-Based Hourly Global Pressure and Temperature (HGPT) Model. *Remote. Sens.* 12, 7 (2020), 1098.

[31] MERRA-2 [n.d.]. Modern-Era Retrospective analysis for Research and Applications, Version 2. https://gmao.gsfc.nasa.gov/reanalysis/MERRA-2/.

[32] Mohamed F. Mokbel, Thanaa M. Ghanem, and Walid G. Aref. 2003. Spatio-Temporal Access Methods. *IEEE Data Engineering Bulletin* 26, 2 (2003), 40–49.

[33] Long-Van Nguyen-Dinh, Walid G. Aref, and Mohamed F. Mokbel. 2010. Spatio-Temporal Access Methods: Part 2 (2003 - 2010). *IEEE Data Engineering Bulletin* 33, 2 (2010), 46–55.

[34] OpenDataCube [n.d.]. OpenDataCube - Open Source Earth Observation at Scale. https://www.opendatacube.org/.

[35] Pangeo [n.d.]. Pangeo: A community for open, reproducible, scalable geoscience. https://www.pangeo.io/.

[36] Dimitris Papadias, Yufei Tao, Panos Kalnis, and Jun Zhang. 2002. Indexing Spatio-Temporal Data Warehouses. In *ICDE* (San Jose, CA, USA).

[37] Stavros Papadopoulos, Kushal Datta, Samuel Madden, and Timothy G. Mattson. 2016. The TileDB Array Data Storage Manager. *PVLDB* 10, 4 (2016), 349–360.

[38] Gary Planthaber, Michael Stonebraker, and James Frew. 2012. EarthDB: scalable analysis of MODIS data using SciDB. In *SIGSPATIAL* (Redondo Beach, CA, USA).

[39] POLARIS [n.d.]. POLARIS first System release. https://iharpv.cs.umn.edu/.

[40] Keven Richly, Rainer Schlosser, and Martin Boissier. 2021. Joint Index, Sorting, and Compression Optimization for Memory-Efficient Spatio-Temporal Data Management. In *ICDE* (Chania, Greece).

[41] Sedona [n.d.]. Apache Sedone. https://sedona.apache.org/.

[42] Spark [n.d.]. Apache Spark - Unified Engine for large-scale data analytics. https://spark.apache.org/.

[43] Charles Stern, Ryan Abernathey, Joseph Hamman, Rachel Wegener, Chiara Lepore, Sean Harkins, and Alexander Merose. 2022. Pangeo forge: crowdsourcing analysis-ready, cloud optimized data production. *Frontiers in Climate* 3 (2022), 782909.

[44] TileDB [n.d.]. TileDB - The Universal Storage Engine. https://docs.tiledb.com/main.

[45] Claudia Vitolo, Francesca Di Giuseppe, Christopher Barnard, Ruth Coughlan, Jesus San-Miguel-Ayanz, Giorgio Libertá, and Blazej Krzeminski. 2020. ERA5-based global meteorological wildfire danger maps. *Scientific data* 7, 1 (2020), 216.

[46] You-Ren Wang, Dag O Hessen, Bjørn H Samset, and Frode Stordal. 2022. Evaluating global and regional land warming trends in the past decades with both MODIS and ERA5-Land land surface temperature data. *Remote Sensing of Environment* 280 (2022), 113181.

[47] Xarray [n.d.]. Xarray. https://xarray.dev/.

[48] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. 2016. A demonstration of GeoSpark: A cluster computing framework for processing big spatial data. In *ICDE* (Helsinki, Finland).

[49] Ramon Antonio Rodriges Zalipynis. 2018. ChronosDB: Distributed, File Based, Geospatial Array DBMS. *PVLDB* 11, 10 (2018), 1247–1261.

[50] Ramon Antonio Rodriges Zalipynis. 2021. Array DBMS: Past, Present, and (Near) Future. *PVLDB* 14, 12 (2021), 3186–3189.