TECHNISCHE UNIVERSITÄT ILMENAU
Faculty of Computer Science and Automation
Institute for Practical Computer Science
Department of Databases and Information Systems

Research Project

Definition of a Benchmark for Raster Data Processing

Submitted By

Proteeti Prova Rawshan
Matrikel Nr. 63752
Winter Semester 2023/24
Research in Computer and Systems Engineering

Supervised By:

Dr. Marcus Paradies

Ilmenau, March 1, 2024

Contents

1 Introduction 1

2 Background and Related Work 2

3 Project Overview 4
- 3.1 Problem Definition and Requirements . . . 4
- 3.2 ERA5 Climate Reanalysis Data . . . 4
- 3.3 Query Functions and Their Role in Benchmarking . . . 4

4 Methodology 5
- 4.1 Requirements . . . 5
- 4.2 Dataset . . . 5
- 4.3 Workload Generator . . . 5
- 4.4 Query formulation . . . 6
- 4.5 Benchmark execution and collected measurements . . . 7

5 Implementation: Benchmark Design 8
- 5.1 Benchmark Goals and Metrics . . . 8
- 5.2 Query Definition and Parameterization . . . 8
- 5.3 Workload Generation and Execution . . . 9

6 Evaluation 11
- 6.1 Benchmark Setup and Performance Metrics . . . 11
- 6.2 Result and Discussion . . . 11

7 Future Work and Enhancements 13

8 Conclusion 14

Appendix

A API Request

B ERA5 Climate Reanalysis Data Exploration

C Query Functions Implementation

D Workload Generation & Execution Scripts

1. Introduction

Benchmarking [1] plays a critical role in assessing the performance of various raster data processing systems that are widely used in applications such as environmental modeling and geospatial analysis. These systems, which include open-source tools such as xarray [12], have diverse functional scopes and performance characteristics, making it difficult to match the right tool with a particular task. The lack of a standardized benchmark for raster data [4] processing bars unbiased comparisons of these tools, thereby preventing users from making informed choices. A comprehensive benchmark that covers a range of representative queries, from basic data access to complex analytics, is urgently needed to address this gap. This benchmark will offer an objective means for comparing different systems, and enable users to select the most suitable tool for their needs [2, 13, 15].

2. Background and Related Work

According to numerous researchers, benchmarking is crucial in evaluating the performance of different data processing systems. Raster data processing systems are no exception, since they are used in a wide variety of applications, including, but not limited to, environmental modeling, and geospatial analysis. Currently, however, existing works focus on different data types as well as do not provide a comprehensive set of queries and detailed workload generation and execution mechanisms.

Dayarathna et al.[3] conduct an extensive survey of the literature of graph data management and processing benchmarks and identify 20 different benchmarks that have been proposed in last 15 years. It identifies several open issues in this area. A major problem is that benchmarks do not exist that exercise the system with a very high workload. In another study by [8], benchmarks spatially enabled RDF stores for geospatial data integration within Spatial Data Infrastructures (SDIs), identifying advancements in GeoSPARQL compliance and query performance. It highlights despite these advancements, if the challenge to ensure query correctness across different databases for interoperability is not resolved, there will remain a gap in achieving seamless integration and utilization of geospatial data across diverse data platforms. Meanwhile, Haynes et al. [6] introduces a benchmark used to compare how well big data platforms process raster geospatial data and to identify which systems might be best suited for different raster analysis tasks. It provides a means of identifying which platforms are particularly apt for certain prominence raster analysis tasks. It does not tackle the challenge of identifying which platforms are suitable for the large-scale processing of raster data. It does not deal with comparing these platforms in terms of their performance within ”real-world” application scenarios.

Existing literature on database systems and benchmarks provides an exhaustive treatment of the area. However, none of these works addresses the unique needs of raster data processing in the context of climate data analysis. While the spatial database benchmark by Mauroux et al. [2] is enlightening about the nuances of spatial data handling, it does not specifically address the peculiarities of climate data and the challenges that raster data brings. Efficient storage and access mechanisms are pivotal in the context of array databases. The array database study by Merticariu et al. [9] emphasizes this need and highlights the gap in performance evaluation for complex data types, such as raster data. This is an important step in the array and raster database area. Optimization of distributed systems is also a topic of much research, as is the challenge of achieving high availability and consistency [15]. However, these databases typically do not address scientific data processing requirements. Lastly, [14], offers an alternative to current benchmarks that are fine tuned towards general-purpose big data systems but do not translate well to the computational and storage needs of scientific data processing.

The proposed benchmarking effort tackles the most significant gaps from prior studies, particularly in addressing handling of high workloads and efficient processing of large-scale raster data operations in real-world settings. By constructing a benchmark to evaluate how big data platforms perform during extensive raster analysis tasks, it gives a design for data management

and processing methodologies for geospatial information. Also, it tackles specific challenges of scalability and real-world applications which might be one of the critical void in current literature in geospatial data integration and analysis.

3. Project Overview

3.1. Problem Definition and Requirements

The project capitalizes on the preliminary development of a benchmark to undertake the performance assessment of data processing systems handling ERA5 climate reanalysis data. The justification for the need for a benchmark is the voluminous nature and complexity of climate data, which is too large and complex for tools to efficiently analyze, so analysis will be fast enough to support policy within the climate research time frame.

3.2. ERA5 Climate Reanalysis Data

The European Centre for Medium-Range Weather Forecasts provides ERA5 climate reanalysis data [7], offering an updated record of comprehensive global atmospheric reanalysis from 1979. Nevertheless, its use in this research is bound by precision, resolution, and earlier-wide usage when it comes to climatic studies.

The ERA5 dataset [11] possesses complete atmospheric, land, and oceanic climate variables and, therefore, a well-articulated representation of climate patterns with time. With its high spatial and temporal resolutions, climate studies based on ERA5 will include diachronic weather forecasting to long-term climate change analysis.

3.3. Query Functions and Their Role in Benchmarking

Query functions try to mimic the most basic simulation query structures common for data analysis on climate information and also other aspects. They offer simple benchmarks whereby performance evaluated by the systems should ensure effective service in executing these simple operations on big climate datasets. These include time and space aggregations, transformation of data, and derived variable calculations that are typical patterns in which climate data are analyzed.

4. Methodology

4.1. Requirements

The objective of this work is to critically evaluate the effectiveness and accuracy of different data processing systems in managing and analyzing large climate datasets. The benchmark has been crafted to stress test the systems' ability to survive and perform under heavy computational workloads common in climate research data analysis tasks, with the ultimate goal of identifying systems that can achieve both high throughput and low latency in processing and analyzing climate-related data.

4.2. Dataset

The ERA5-Land climate dataset was used as the foundation for this benchmarking study, owing to its extensive spatiotemporal coverage of climate variables. Specifically, this dataset includes land surface temperatures across Europe, spanning from January 2000 through the present day, detailing the climatic variations across the years. A request was formulated with which to retrieve this dataset, being sure to provide the Climate Data Store (CDS) API [10, 5] with all necessary parameters that they may return exactly the data necessary for this study's objectives. Successful acquisition of this dataset then allowed climate variables to be probed in detail, in which the xarray library was employed to facilitate the manipulation and analysis of the multi-dimensional data array.

The dataset was chosen for its comprehensive coverage of climate data across a vast temporal and spatial scale, mirroring the complexities encountered in real-world climate data analysis. This preliminary examination allowed to develop SQL queries, and also to establish a robust methodology for benchmarking, both with respect to the data's structure and granularity.

4.3. Workload Generator

The Workload Generator is conceived of as a tool to create a varied and representative set of analytical tasks meant to mimic the extensive and complex data analysis operations typical in climate research. This aspect of the process is essential to evaluate how well the data processing systems are able to manage diverse computational demands. The main objective of the Workload Generator is to produce tasks that simulate the key aspects of climate data analysis research, so that they provide a sound basis for evaluating the quality of the data processing systems' performance as the computational regime varies.

- Configuration-Driven Approach: The generator is implemented as a configuration-driven tool, which allows a set of tasks to be defined in a flexible manner, using a variety of established parameters so that the workload is able to include many different scenarios of data retrieval and transformation.
- Representation of Tasks: The tasks that are produced are chosen to reflect different aspects of climate data analysis, such as the retrieval of data, the performance of data aggregation, and carrying out various kinds of data transformation. Together, they are devised to put a full range of fascinating challenges in front of the systems, whose performance they are meant to probe.

4.4. Query formulation

A further exploration of the ERA5-Land dataset by formulating and executing a series of simple queries aimed at extracting specific climate data insights (with visualizations) for better understanding. This involved applying various data retrieval techniques to pinpoint trends, anomalies, and patterns in the temperature data across different time periods and geographical locations within Europe. Then, it was taken up to a higher level with more complicated stuff, such as figuring out long-term climate patterns and oddities. This step was pivotal in establishing a framework that tests both the efficiency and scalability of data management systems in handling raster data.

The queries detailed in Table 4.1 offer the means to tame the climate dataset in order to manipulate and explore it. Each query is intended to respond to an actual data operation commonly done in climate research.

Function	**Query**	**Purpose**
Average Temperature Over Time	TemporalRangeAggregation	Reflects the system's ability to handle time-series data.
Geographical Temperature Analysis	SpatialSubsetExtraction	Tests the system's spatial data handling efficiency
Adjusted Climate Data	TemporalDataModification	Evaluates the system's capacity for data manipulation
Derived Climate Metrics	AddDerivedVariable	Tests the system's computational functionality
Seasonal Weather Patterns	SeasonalAggregation	Assesses the system's ability to aggregate data.

Table 4.1.: Descriptive summary of the query functions implemented

4.5. Benchmark execution and collected measurements

In this phase, the aim is to convert the conceptual framework to an empirical method through which real-world climate data analysis scenarios can be executed and managed by each system.

The benchmark execution process itself is intended to measure how well each system runs through the workload. That is to say that the focus of the benchmark is on measuring execution time, CPU usage and memory consumption. These performance metrics are chosen because of their direct relevance in measuring the critical aspects of system performance such as computational speed, resource efficiency and the ability to scale under increased loads. By capturing and then analyzing these metrics, the study seeks to provide objective insight into the operational strengths and limitations of each system in the context of large-scale climate data set analysis.

5. Implementation: Benchmark Design

5.1. Benchmark Goals and Metrics

The benchmark was designed to assess the performance and efficiency of processing climate data. It evaluates the processing of ERA5-Land temperature data across Europe from 2000 to 2022.

The primary metrics for evaluation are execution time, CPU usage, and memory consumption, which will allow us to identify the most efficient data handling techniques for large-scale climate datasets. Such an assessment is critical to enhance climate data analysis workflows, which are essential to climate research and decision-making.

5.2. Query Definition and Parameterization

Query formulation is a critical step in translating scientific questions into code that is actually run. It involves defining a set of queries that collectively cover a wide spectrum of typical data analysis tasks found in climate datasets. In the ***query_fucntions.py***, these functions were used:

1. **temporal_range_aggregation**
    - Purpose: This function aggregates (mean/sum) the 't2m' (2-meter temperature) variable over a given temporal range.
    - Query Type: Analyzes how temperature changes over a given period, which is useful for understanding climatic trends/ variations within that timeframe.
2. **spatial_subset_extraction**
    - Purpose: This function extracts the subset of 't2m' variable data within the given latitudinal and longitudinal ranges.
    - Query Type: Essential to narrow down on a specific geographical area and hence observe climatic patterns or conditions localized to that area.
3. **temporal_data_modification**
    - Purpose: This function modifies the 't2m' variable data over the specified month and year for all or a selection of latitude and/or longitude. It can increase/decrease the data by a specified multiple as well. It also checks for empty selections to ensure that the modifications are not applied where there is no data.

- Query Type: Could be used to see the impact of any condition/intervention on the temperature data, which will assist in understanding potential climate change scenarios/ conditions where the actual data values could be missing or any other anomalies in the dataset.

4. **add_derived_variable**
   - Purpose: This function adds a derived variable to the dataset. The new variable can be the existing variable 't2m' with any of the following operations performed with the given factor: multiply, add, subtract, divide.
   - Query Type: Useful in generating new insights/ variables from the existing data and is critical for more complex climate analysis or model inputs.
5. **seasonal_aggregation**
   - Purpose: This function provides a helper for the user to aggregate the 't2m' data over the specified season (winter, spring, summer, autumn) for the given year. The helper internally uses a predefined month range for each season (e.g. winter is Dec, Jan, Feb).
   - Query Type: Essential for looking at the conditions/ variations over a year across all the seasons and thus, the indication of each season's impact on the temperature.

5.3. Workload Generation and Execution

Benchmark Configuration: Defined in the ***benchmark_config.json***, is a configuration file that specifies parameters for benchmarks, including which functions to run, their parameters, and perhaps in what order. Here, the dataset was specified (GRIB file). Also, in an array of objects, the queries were defined, where each object has a type corresponding to a specific operation and a weight,which indicates the relative importance or frequency of each query type in the benchmark.

For example: `"type": "TemporalRangeAggregation", "weight": 30` - meaning this query will aggregate data over a specified time range, where it's 30% of the total queries performed.

Workload Generation: Workload generation is a key aspect of benchmarking, where the goal is to create a diverse and representative set of queries to evaluate the performance of data processing systems. This is done through the ***workload_generation.py*** script, where a sequence of tasks are programmatically defined to mimic realistic data analysis scenarios on climate datasets such as the ERA5-Land temperature data, and the workload is generated in (***workload.csv***).

The script follows a configuration-driven approach where the types of queries, configuration parameters, and under which conditions they are run are specified in (***benchmark_config.json***). Over-all, the workload consists of a balanced mix of data retrieval, aggregation, data transformation tasks which push on different aspect of the system's performance.

Workload Execution: The workload execution, outlined in ***workload_execution.py***, is responsible for running the generated workload against the data processing system. The script reads in the list of tasks created by the workload generator and executes each query sequentially. At the

conclusion of the query, key performance metrics (e.g., execution time, CPU, memory) are captured. These metrics are essential input for understanding the aforementioned system efficiency and scalability under differing computational loads.

To ensure accurate and reliable benchmark results, the execution mechanism is robust to errors and includes mechanisms to handle errors, logging and the ability to retry failed tasks. This functionality is particularly important for handling the complexity and potential unpredictability of processing large-scale climate datasets.

The exact order and relationship between these components would be like this: benchmark_config would be read by workload_generation to create a series of tasks, which are then executed by workload_execution using the functions defined in query_functions.py.

6. Evaluation

6.1. Benchmark Setup and Performance Metrics

The benchmark was designed to evaluate the performance and efficiency of various query functions, each aimed at testing different aspects of data handling and analysis capabilities.

- **Execution Time:** This metric denotes the total time for a query to complete its execution - from initiation to completion. This is important because understanding spiky or long latencies times for data processing can be important, since a rapid response helps build data analytics around issues and opportunities that are current, topical, and timely.
- **CPU Usage:** This is a measure of how intensively a query is exercising the computational elements of query execution. It expresses the percentage of the cycles the CPU is busy during the life cycle of the query.

  But getting this exactly quantified is actually hard, as a modern operating system is likely to have thousands of tasks switching in and out, and in and out, and this will also make the CPU look as if it's executing other processes during the time in question, changing readings. So, CPU usage only gives a rough approximate idea of the load in terms of computation, but does not often wholly reflect the resource consumption of the query itself.
- **Memory Usage:** This is a measure of the extra memory (beyond the database memory) that is going to be utilized as this query executes. Memory usage is a critical measure in capacity planning, and a rich source of data for understanding the scalability of a new and growing data processing operation. Memory utilization that is excessive will lead to thrashing and will degrade all the performance of the system, not just the performance of the query execution.

6.2. Result and Discussion

Examination of 't2m' climate data operations reveals significant variability in computational demands. It is observed in 6.1 that, execution times varied from approximately 1.46E+15 to 2.21E+16 secs. Changes in CPU usage ranged up to 130%. Memory usage fluctuations varied from a decrease of 390625 MB to an increase of 3125 MB. Operations are associated with a 'query_index' that categorizes different types. The operations types range from SpatialSubsetExtraction and AddDerivedVariable to various aggregations. Examining the data it appears that operations such as temporal modification and aggregation are particularly computationally demanding. The wide variability in resource usage clearly illustrates the complexity of performing analysis on climate data, with some types of operations consuming more resources (either due

to granularity of the data or the introduction of new variables).

	result	execution_time_sec	cpu_usage_diff_percent	memory_usage_diff_mb	query_index	query_type
1	<xarray.DataArray 't2m' (time: 264, latitude: 0, longitude: 375)> [0 values with dtype=float32] Coordinates: number int32 ... * time (time) datetime64[ns] 2000-01-01 2000-02-01 ... 2021-12-01	2,211E+16	126	3125	0	SpatialSubsetExtraction
2	<xarray.DataArray 't2m' (time: 264, latitude: 0, longitude: 0)> [0 values with dtype=float32] Coordinates: number int32 ... * time (time) datetime64[ns] 2000-01-01 2000-02-01 ... 2021-12-01	5,575E+15	0	4296875	1	SpatialSubsetExtraction
3	<xarray.Dataset> Dimensions: (time: 264, latitude: 501, longitude: 701) Coordinates: number int32 ... * time (time) datetime64[ns] 2000-01-01 2000-02-01 ... 2021-12-01	1,66438E+16	47	42888671875	2	AddDerivedVariable
4	<xarray.DataArray 't2m' (latitude: 501, longitude: 701)> array([[259.49878, 259.58472, 259.66675, ..., nan, nan, nan], [259.83667, 259.8933 , 259.948 , ..., nan, nan, nan],	1,78335E+16	0	1171484375	3	SeasonalAggregation
5	<xarray.DataArray 't2m' (latitude: 501, longitude: 701)> array([[257.97607, 258.0874 , 258.19482, ..., nan, nan, nan], [258.28662, 258.3706 , 258.4546 , ..., nan, nan, nan],	1,45776E+15	130	-133203125	4	SeasonalAggregation
6	<xarray.DataArray 't2m' (latitude: 501, longitude: 701)> array([[243.10631, 243.17467, 243.23717, ..., nan, nan, nan], [243.48131, 243.53502, 243.5897 , ..., nan, nan, nan],	1,35145E+16	62	-3125	5	TemporalRangeAggregation
7	<xarray.Dataset> Dimensions: (time: 264, latitude: 501, longitude: 701) Coordinates: number int32 ... * time (time) datetime64[ns] 2000-01-01 2000-02-01 ... 2021-12-01	1,45725E+16	-63	-390625	6	TemporalDataModification

Figure 6.1.: A snippet of the benchmarking results

Lastly, the computational workflow clearly illustrates a methodical process from refinement of an initial version of the data, to a detailed extraction, and ultimately to an aggregation. A systematic processing approach is necessary to be able to efficiently transform climate data for informative environmental research.

7. Future Work and Enhancements

The exploration of ERA5 climate data with direct query execution and real-time visualization is a stepping stone to climate analytics. The possible improvements & future work might aim to build upon it in the following ways:

Tool Evaluation: Comparing the existing query execution and data analysis techniques with the likes of Dask or PySpark or similar tools for benchmarking and identifying improvements with respect to execution time, scalability and resource usage.

Benchmarking Enhancements: Developing a comprehensive benchmark setup with a variety of query types, larger datasets, and different computational environments to identify strengths and optimization opportunities.

Dataset Expansion: The scope of the work can be broadened to consider other variables from ERA5 as well as other climate datasets to enable the analysis of broader climate phenomena. To support these, new query functions will be needed that can extend the data volume processed and adapt the implemented data model to accommodate multi-dimensional datasets.

Tool Comparison: Systematic comparison with more climate data analysis tools to understand the salient features of the implemented tool in terms of speed, accuracy and scalability, for example.

User Interface Development: Development of a user-friendly interface that facilitates interactive queries, real-time visualization and easy data interpretation from complex queries and large datasets and hence, bridges the gap between the current tool and the climate science community.

8. Conclusion

In this report, a benchmarking framework was introduced for the analysis of ERA5 climate reanalysis data, highlighting the development, and particularly, the insights gathered from the exploration phase as well. The work has not only advanced the efficiency of climate data analysis, but has also laid the groundwork for furthering contributions to climate science research. The successes seen while working with these complex climate data should pave the way for a myriad of future work, ranging from comparisons against other tools to broadening the tool's applicability and user accessibility. It might also produce some make meaningful contributions to the understanding and addressing the challenges of the changing climate, with a continued focus on innovative thinking and broadening the analytical reach.

A. API Request

In order to obtain the climate data, an API request was formulated and executed using the cdsapi library to the Climate Data Store (CDS), targeting the monthly averaged reanalysis data for the 2m_temperature variable for a specific geographical bounding box over a particular time range (Europe in this case, 2000-2021).

```
import cdsapi

c = cdsapi.Client(key=cdsapi_key)

variables_list = ['2m_temperature']
for variable in variables_list:
    c.retrieve(
        'reanalysis-era5-land-monthly-means',
        {
            'format': 'grib',
            'product_type': 'monthly_averaged_reanalysis',
            'variable': variable,
            'year': [str(year) for year in range(2000, 2022)],
            'month': ['01', '02', '03', '04', '05', '06', '07', '08', '09',
    '10', '11', '12'],
            'time': '00:00',
            'area': [25, -25, 75, 45],  # Europe bounding box
        },
        'ERA5-Land_2m_temperature_2000-2022_europe_2.grib'
    )
```

The 'retrieve' method of the CDS API client accepts the dataset name, the request parameters, and the destination file path. Each parameter in the request serves a specific purpose:

`format` The format of the downloaded file.

`product_type` The type of reanalysis product to download.

`variable` The specific climate variable of interest.

`year` and `month` The temporal range for the data retrieval.

`time` The time of day for which the data is required.

`area` The geographical bounding box specifying the spatial extent of the data.

B. ERA5 Climate Reanalysis Data Exploration

In this study, the ERA5-Land dataset was interacted with using and explored with the xarray library. Xarray is an incredibly powerful Python package for using/manipulating multi-dimensional arrays especially when working with climate and meteorological data. The ERA5-Land dataset is made up of so many climate variables with high spatial and temporal resolution. The dataset comprises a lot of climate variables including '2m_temperature' which is the temperature at 2 meters above the surface.

Key aspects of the ERA5-Land dataset include:

- Spatial resolution: Contains detailed latitude and longitude data. Consequently, allows for detailed geographical based analysis.
- Temporal resolution: Monthly averages of all variables from 2000, up until recently (up to the 22nd of February 2022).
- Variables: This study primarily focuses on the '2m_temperature' variable.

The code below shows how the dataset was explored using xarray:

```
import xarray as xr

# Open the GRIB file using xarray with cfgrib engine for GRIB files
grib_file_path = "path_to_grib_file.grib"
ds_2 = xr.open_dataset(grib_file_path, engine='cfgrib')
print(ds_2)
```

There is also 'Attributes' section showing trustworthy metadata on the source of these data, the standard to which the data adhere, and the dataset's transformation history, making it a reliable, standards-compliant dataset that is suitable for studying climate and climate change.

```
<xarray.Dataset>
Dimensions:     (time: 264, latitude: 501, longitude: 701)
Coordinates:
  * time        (time) datetime64[ns] 2000-01-01 2000-02-01 ... 2021-12-01
  * latitude    (latitude) float64 75.0 74.9 74.8 74.7 ... 25.3 25.2 25.1
    25.0
  * longitude   (longitude) float64 -25.0 -24.9 -24.8 -24.7 ... 44.8 44.9
    45.0
Data variables:
    t2m         (time, latitude, longitude) float32 ...
Attributes:
    GRIB_edition:            1
    GRIB_centre:             ecmf
    GRIB_centreDescription:  European Centre for Medium-Range Weather
    Forecasts
```

```
    Conventions:              CF-1.7
    institution:              European Centre for Medium-Range Weather
    Forecasts
    history:                  2024-02-07T22:56 GRIB to CDM+CF via cfgrib
    -0.9.1...
```

As observed above, the dataset has 264 time points, covering 22 years including 501 latitudinal and 701 longitudinal points across Europe. The implementation focuses on variables like 2m temperature (t2m), which were vital for the analysis, giving detailed time and location information crucial for the comparisons.

C. Query Functions Implementation

This section describes the implementation of core query functions within the project which are designed to analyze ERA5 climate reanalysis data. Each of these functions are tailored to address a specific analysis pattern which manipulates the queried data based on the xarray objects:

```
    # Aggregate 't2m' over a specified temporal range.
def temporal_range_aggregation(ds, start_date, end_date, variable="t2m",
    operation="mean"):
    # Selects the data slice based on the time range and performs the
    specified aggregation operation
    # Returns: xarray.DataArray with the aggregation result

# Extract a spatial subset for 't2m'
def spatial_subset_extraction(ds, lat_range, lon_range, variable="t2m"):
    # Selects a subset of the data based on the specified latitude and
    longitude range
    # Returns: xarray.DataArray containing the spatial subset

# Modify 't2m' data for a specific month and year
def temporal_data_modification(ds, year, month, modification="increase",
    value=1.0, variable="t2m"):
    # Applies a specified modification to 't2m' data for the given month
    and year, if data is available
    # Returns: xarray.Dataset with the modified 't2m' data

# Add a derived variable based on 't2m'
def add_derived_variable(ds, new_variable_name, base_variable='t2m',
    operation='multiply', factor=1.0):
    # Creates a new variable in the dataset based on the specified
    operation and factor
    # Returns: xarray.Dataset with the new derived variable added

# Aggregate 't2m' over a specified season
def seasonal_aggregation(ds, year, season, variable="t2m", operation="mean"
    ):
    # Performs an aggregation of 't2m' data over the specified season for
    the given year
    # Returns: xarray.DataArray with the seasonal aggregation result
```

Each of these functions are critical for building a climate pattern analysis that is flexible and allows a user to build functionality that does several operations and allows to build up more and more complex climate anlayses.

D. Workload Generation & Execution Scripts

The workload generation script (workload_generation.py) creates a set of queries dynamically, with different types and parameters. It helps to simulate many different types of data processing tasks. Some of the key components of this script are:

- Parameter Generation: Functions to randomly generate different types of parameters for different types of queries to make sure that we test a variety of cases.
- Workload File Creation: The script creates a CSV file that contains the generated queries which can be run by the workload execution script.

```
# Example of parameter generation for a Temporal Range Aggregation query
def generate_parameters(query_type):
    if query_type == "TemporalRangeAggregation":
        # Generates start and end dates, variable, and operation

# Main function to generate the workload
def generate_workload(config_path="benchmark_config.json", workload_file="
    workload.csv"):
    # Loads configuration, generates queries based on weights, and writes
    them to a CSV file
```

The workload execution script (workload_execution.py) reads the generated queries and executes them against the dataset, capturing and logging performance metrics such as execution time, CPU usage, and memory usage. It demonstrates integration with the xarray library for data manipulation and psutil for monitoring system resources.

- Query Execution: Implements logic to execute various types of queries, including temporal and spatial data manipulations.
- System Usage Monitoring: Captures system metrics before and after query execution to measure impact.
- Results Aggregation: Collects and stores the execution results for analysis in csv.

```
# Function to execute a single query and measure system usage
def execute_query(ds, query):
    cpu_before, mem_before = get_system_usage()
    # Executes the query based on its type (e.g., TemporalRangeAggregation)

# Main function to run the benchmark
def run_benchmark(workload_file, config):
    # Opens the dataset, iterates through the workload, and executes
    queries
```

Bibliography

[1] Dina Bitton, David J DeWitt, and Carolyn Turbyfill. Benchmarking database systems-a systematic approach. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1983.

[2] Philippe Cudre-Mauroux, Hideaki Kimura, Kian-Tat Lim, Jennie Rogers, Samuel Madden, Michael Stonebraker, Stanley B Zdonik, and Paul G Brown. Ss-db: A standard science dbms benchmark. *Under submission*, 114, 2010.

[3] Miyuru Dayarathna and Toyotaro Suzumura. Benchmarking graph data management and processing systems: A survey. *arXiv preprint arXiv:2005.12873*, 2020.

[4] QGIS 2.18 Documentation. Raster data), 2024. Accessed on 2024-02-27.

[5] ECMWF. https://cds.climate.copernicus.eu/#!/home, 2024. Accessed on 2024-02-27.

[6] David Haynes, Philip Mitchell, and Eric Shook. Developing the raster big data benchmark: A comparison of raster analysis on big data platforms. *ISPRS International Journal of Geo-Information*, 9(11):690, 2020.

[7] Hans Hersbach, Bill Bell, Paul Berrisford, Shoji Hirahara, András Horányi, Joaquín Muñoz-Sabater, Julien Nicolas, Carole Peubey, Raluca Radu, Dinand Schepers, et al. The era5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730):1999–2049, 2020.

[8] Weiming Huang, Syed Amir Raza, Oleg Mirzov, and Lars Harrie. Assessment and benchmarking of spatially enabled rdf stores for the next generation of spatial data infrastructure. *ISPRS International Journal of Geo-Information*, 8(7):310, 2019.

[9] George Merticariu, Dimitar Misev, and Peter Baumann. Towards a general array database benchmark: Measuring storage access. In *Big Data Benchmarking: 6th International Workshop, WBDB 2015, Toronto, ON, Canada, June 16-17, 2015 and 7th International Workshop, WBDB 2015, New Delhi, India, December 14-15, 2015, Revised Selected Papers 6*, pages 40–67. Springer, 2016.

[10] Baudouin Raoult, Cedric Bergeron, A López Alós, Jean-Noël Thépaut, and Dick Dee. Climate service develops user-friendly data store. *ECMWF newsletter*, 151:22–27, 2017.

[11] Copernicus Climate Change Service. Ecmwf reanalysis v5 (era5), 2023. Accessed on 2023-10-30.

[12] Joe Hamman Stephan Hoyer. Xarray n-d labeled arrays and datasets in python, 2024. Accessed on 2024-02-28.

[13] Gábor Szárnyas, Jack Waudby, Benjamin A Steer, Dávid Szakállas, Altan Birler, Mingxi Wu, Yuchen Zhang, and Peter Boncz. The ldbc social network benchmark: Business intelligence workload. *Proceedings of the VLDB Endowment*, 16(4):877–890, 2022.

[14] Xinhui Tian, Shaopeng Dai, Zhihui Du, Wanling Gao, Rui Ren, Yaodong Cheng, Zhifei Zhang, Zhen Jia, Peijian Wang, and Jianfeng Zhan. Bigdatabench-s: an open-source scientific big data benchmark suite. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1068–1077. IEEE, 2017.

[15] Ramon Antonio Rodriges Zalipynis. Array dbms: past, present, and (near) future. *Proceedings of the VLDB Endowment*, 14(12):3186–3189, 2021.