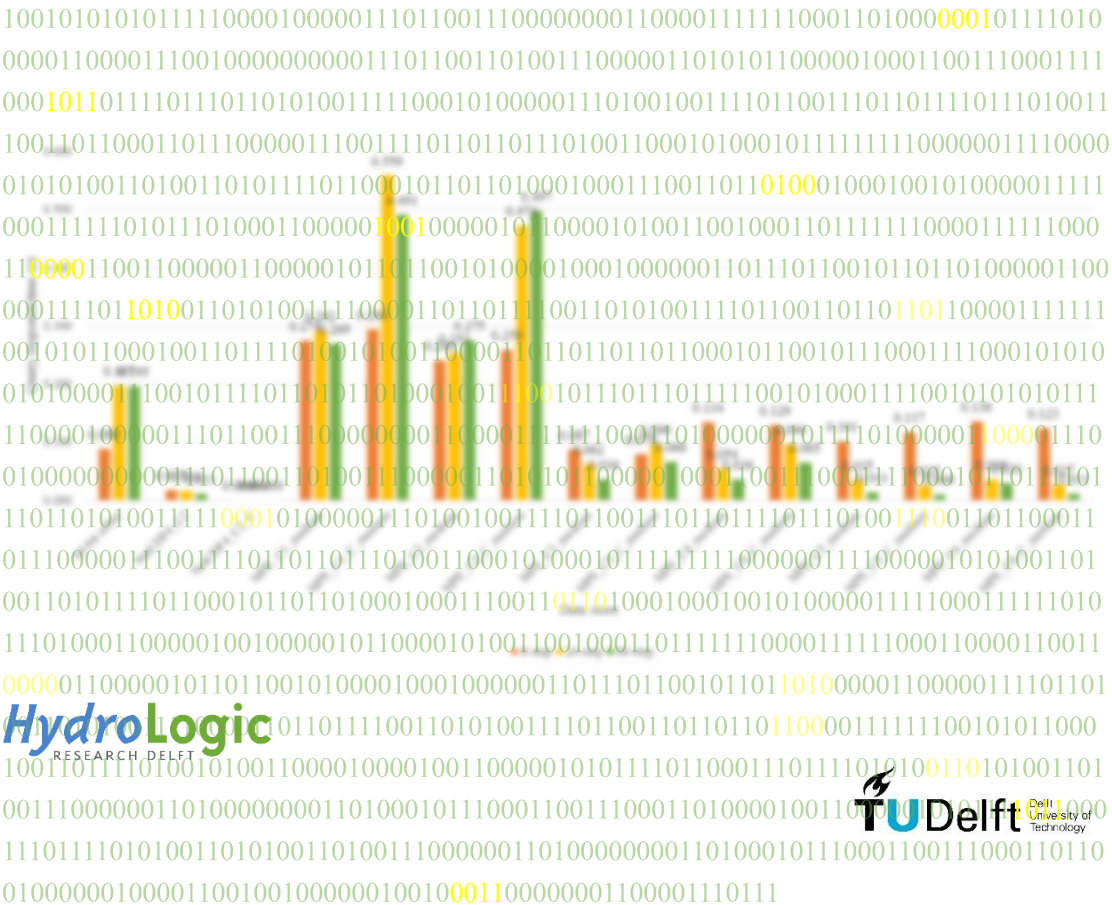
Comparing NetCDF and a multidimensional array database on managing and querying large hydrologic datasets: a case study of SciDB

Haicheng Liu



29 October 2014

Title: Comparing NetCDF and a multidimensional array database on managing and querying large hydrologic datasets: a case study of SciDB

Name: Haicheng Liu
Student number: 4252438
Master program: Geomatics

Graduation professor:
Prof. Dr. Ir. Peter van Oosterom
GIS technology, Faculty of Architecture,
TU Delft

Supervisor:
Drs. Theo Tijssen
GIS technology, Faculty of Architecture,
TU Delft

Advisor:
Ir. Tom Commandeur
Hydrologic Research B.V.

Co-reader:
Dr. Roderik Lindenbergh
Geoscience and Remote Sensing, Faculty of Civil Engineering and Geosciences,
TU Delft

Date: 29 October 2014

Abstract

Like many ICT related domains, hydrology enters the era of big data and managing large volume of data is a potential issue facing hydrologists. However at present, hydrologic data research is mostly concerned with data collection, interpretation, modelling and visualization. Management and query of large datasets do not draw much interest. The motivation of this research originates from a specific data management problem reflected by Hydrologic Research B.V. and that is, time series extraction costs intolerable time when the large multidimensional dataset is stored in NetCDF classic or 64-bit offset format. The essence of this issue lies in the contiguous storage structure adopted by NetCDF. So in this research, NetCDF-4 format and a multidimensional array database applying chunked storage structure are benchmarked to learn whether and how chunked storage structure can benefit queries executed by hydrologists.

To achieve a convincing and representative benchmark result, expert consultancy was conducted to collect queries and sample datasets frequently handled by water experts. From the raw consultancy records, 5 classes of query were summarized and specific queries for benchmarking were devised. After this, 9 criteria were established to assess which multidimensional array database is most suitable for benchmarking and finally SciDB was chosen. To establish a fair benchmark test environment, HydroNET-4 system was utilized and adapters for NetCDF files and SciDB were developed to manage and query data. For final benchmark tests, influence of data compression on query, and scalability of diverse data solutions, i.e. 64-bit offset, NetCDF-4 and SciDB solutions were investigated. In addition, chunk size and dimensions order effect of SciDB arrays on query performance were also explored.

It turns out that NetCDF-4 solution with a recommended chunk size has the best overall management and query performance among all solutions. SciDB arrays utilizing small chunk sizes present favorable performance. However with current implementation of SciDB, large amount of small chunks cause huge overload of main memory which constraints SciDB's scalability. For SciDB, DEFLATE compression can either have negative or no effect on query performance. In time series extraction test, compression effect is found to be correlated with chunk size and the negative impact of compression on query decreases as chunk size reduces. It is deduced that with hypercubic and modest chunk sizes, the internal data structure of chunks in SciDB has no significant influence on query performance. The research demonstrates that for large data management and query, a file based solution can be a better choice than a database utilizing smart caching and indexing strategies. But due to the limited scope of the research, for instance no parallel query processing tested, more work need to be conducted in the future.

Acknowledgments

First of all, I give my great gratefulness to my parents who supported my study in TU Delft which is the very starting point yet crucial one for producing the thesis. I wish them all the best with their health and work.

This thesis was conducted in Hydrologic Research B.V., Delft. It was actually a collaborative work between the company and TU Delft. I would like to thank Dr. Ir. Slavco Velickov, the director of the company to provide me the opportunity to perform the thesis research and also the second chance to continue the research when my graduation was delayed.

I owe my great gratitude to the most crucial collaborator, also my daily advisor, Ir. Tom Commandeur. We proposed the topic and implemented the research together. From him, I improved practical programming skills. Besides, he also taught me essential skills to work in a company, for which I had zero background. I cannot guarantee I have mastered those skills but I can feel that I gained a lot.

This work wouldn't have happened without Prof. Dr. Ir. Peter van Oosterom and Drs. Theo Tijssen's help. Peter provided advices constantly either through long emails or skype meeting in despite of his sabbatical leave. More importantly, he offered kindly guidance when I lost myself in the research. My writing skill progresses under his instructions. Theo also imparted useful experience and knowledge related to the research, for which I appreciate. I thank Dr. Roderik Lindenbergh and Drs. Dirk Dubbeling for their comments on the research and draft thesis.

My sincere thankfulness also goes to Dr. Ir. Susan Steele-Dunne, Prof. Dr. Nick van de Giessen and Dr.ing. Sisi Zlatanova for their offers and suggestions when I started to define the research topic. Sisi also helped a lot for managing other affairs in my graduation process and so did Dr. Ir. Stefan van der Spek. I thank them a lot.

There are still people helping me during the research and I give my best wishes to them all.

Haicheng Liu
20/10/2014
Delft University of Technology

Table of content

List of Figures

List of Tables

List of Appendix Figures

Glossary

AMSR-E Advanced Microwave Scanning Radiometer - Earth Observing System
BUFR Binary Universal Form for the Representation of meteorological data
CDL Common Data Language
DBMS Database Management System
GEFS Global Ensemble Forecast System
GPM Global Precipitation Measurement
GRIB Gridded Binary
HDF Hierarchical Data Format
MPE Multi-Sensor Precipitation Estimate
NetCDF Network Common Data Form
TRMM Tropical Rainfall Measuring Mission

1 Introduction

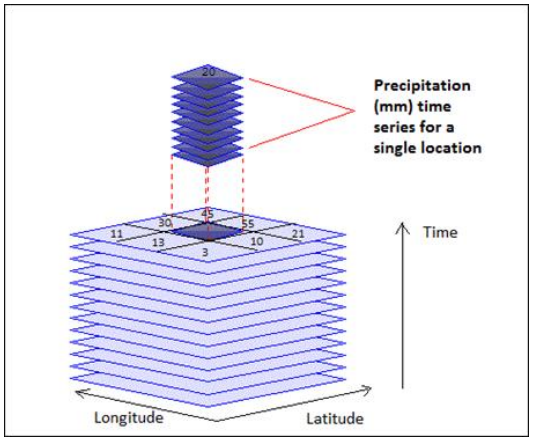
1.1 Problem statement

Original data collecting techniques with improved accuracy become increasingly prevalent currently, radar systems for instance. Meanwhile, new sensor platforms and sources such as citizen-supplied observations are arising all the time. All kinds of models never stop running to produce essential results for decision making. These all make hydrology more and more data intensive. Take rainfall measurement as an example. Doppler radars installed on the ground are widely used for rainfall detection at present. Besides, the same precipitation information can be retrieved from several satellite data products such as TRMM (Tropical Rainfall Measuring Mission), AMSR-E (Advanced Microwave Scanning Radiometer - Earth Observing System) and also the GPM (Global Precipitation Measurement) mission dedicated for collecting observations of rainfall and snow. Not only the integration of these diverse datasets stored in different formats is problematic, but also due to the large amount of data, it indeed brings inconvenience for professionals to extract effective information for certain applications efficiently.

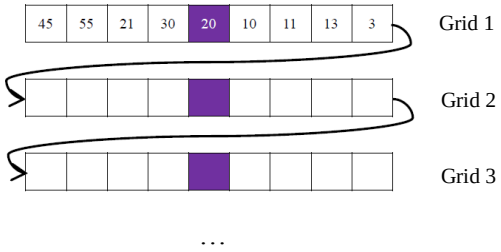
Several data models originally designed for meteorological purposes like HDF5 (Hierarchical Data Format), NetCDF (Network Common Data Form), GRIB (Gridded Binary) and BUFR (Binary Universal Form for the Representation of meteorological data) are recently introduced to the hydrologic domain to ease the exchange and storage of data. Among them, NetCDF (Unidata, 2014) is notable for its simple data model, ease of use, portability, and strong user support infrastructure (Rew, Hartnett, & Caron, 2006). NetCDF also provides schema language to define metadata explicitly and it is implemented as the NetCDF format which is then widely applied to record meteorologic, oceanographic as well as hydrologic observation or simulation data. The data are normally multidimensional. Each dimension is one property to confine the data of interest which is the core information for a certain multidimensional dataset. For hydrology, the core data, also called variable can be precipitation, temperature, etc. They are usually organized into spatial grids which are mostly represented by X and Y under a certain reference system. Each cell in the spatial grid has a value of the hydrologic variable at a particular time step. Since processes and variables change dynamically, all cell values at discrete time steps are then separated into different time layers to be stored, which means a temporal dimension is also involved. Moreover, for some forecasted datasets, they even have an additional ensemble dimension which represents different initial conditions by adding perturbations for the forecast simulation. A model run dimension recording the time to run the forecast model in reality is yet another common dimension in forecast datasets.

However, NetCDF performs inefficiently in retrieving information from large datasets for certain queries according to practical experience. This is radically caused by the way it stores the data, which is known as contiguous storage structure. Basically, for a grid full of cell values in a certain spatial area, NetCDF stores the variable in rows consecutively, i.e. a row is attached to the end of its previous row to form a one-dimensional array (Figure 1.1b). And in this way, if one wants to query the variable value stored in some cell, calculating the position of the cell value in the one-

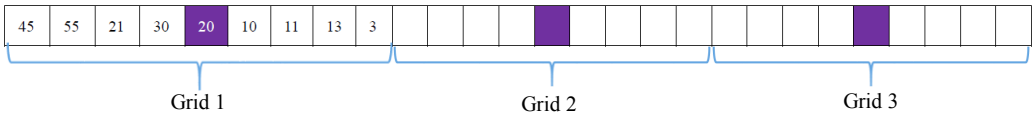
dimensional array is needed. If multiple temporal layers of spatial arrays are stored in the NetCDF, the extraction of a time series (Figure 1.1a), i.e. a series of cell values at all the time steps for a certain location becomes very expensive. This is because the specific cell value of each layer should be extracted from the whole array (Figure 1.1c) and finally these single values are combined to form the time series. It is possible to store time series of specific locations as a one-dimensional variable in NetCDF but then retrieving the grid at a single time step becomes the problem. Thus, queries targeting at a dimension which is not the primary dimension used to store the data are indeed the bottleneck for NetCDF. Although chunked storage structure is introduced to the later version of NetCDF, i.e. NetCDF-4, contiguous storage structure is the implementation for most cases (This is derived from discussion with geo-data experts as well as checking NetCDF data offered by popular providers on the Internet).



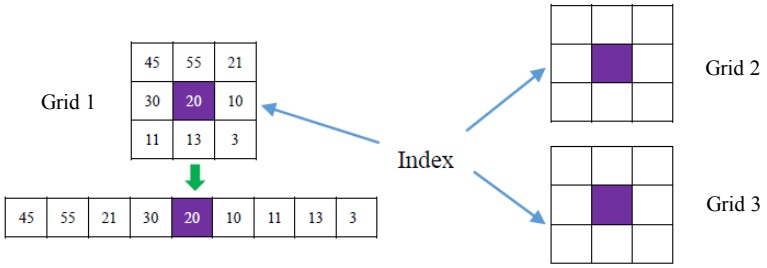
a. 3 dimensional precipitation data and a time series



b. Storing each grid as one-dimensional array and connecting. Purple cells are in the time series.



c. Storage of first 3 grids of precipitation dataset with NetCDF classic format



d. Storage of first 3 grids of precipitation dataset with chunked storage structure. Each chunk is stored as a one-dimensional array on the disk

Figure 1.1. Storing a 3 dimensional precipitation dataset with contiguous storage structure and chunked storage structure

This then brings us to the multidimensional array database solution. It can specify metadata (Pedersen & Jensen, 2001) and supports storage of multidimensional arrays. It employs the chunked

storage structure (Figure 1.1d) which divides a whole dataset into separate chunks (Baumann, Dehmel, Furtado, Ritsch, & Widmann, 1998; Brown, 2010). Based on this storage structure, multidimensional array databases then apply specific indexing approaches, which is proved to be of high query efficiency (Colliat, 1996). In addition, set-oriented language provided by multidimensional array databases like RasQL (Baumann, 1999), AQL (Brown, 2010) can be directly used for executing queries, which is a lack for NetCDF. Last but not least, the multidimensional array databases owns all features of a data base management system (DBMS) such as support of abundant data types, transaction process. Also they are mostly optimized for scalability. Based on these characteristics, multidimensional array databases seem to be an alternative to NetCDF for storing and retrieval of large hydrologic datasets.

Hence, this research is aimed at investigating whether the multidimensional array database can have a better performance in processing queries toward large multidimensional hydrologic datasets than the NetCDF file based solution.

1.2 Research questions

Main research question:
Can a multidimensional array database process frequently implemented queries faster than NetCDF solutions for large hydrological datasets?
We further divide the main question into 4 sub questions which then come through the whole research process:

1. What datasets and queries should be used to fully assess the performance between a multidimensional array database and NetCDF solutions?
2. Which specific multidimensional array database should be applied for tests?
3. For the multidimensional array database, is the performance in handling queries on different dimensions at the same level using one data storage schema?
4. Does data compression in the multidimensional array database have an impact on the query performance?

1.3 Methodology

To address the main research question, benchmark tests are performed. Several steps should be done to construct the benchmark test.

1. Establishing benchmarks. Several hydrologic experts are interviewed to collect typical queries they executed and corresponding datasets. The collected queries are then classified in order to design queries suitable and available for this research. While for datasets, similar data products as what experts mentioned are provided by Hydrologic Research and used for the benchmarking.
2. Selecting multidimensional array database. The multidimensional array database is designed for efficiently manage and query exploding multidimensional datasets in many domains like business and remote sensing. However there are many implementations of multidimensional array database. In this research, first from literature study and former experience with data

storage systems, nine criteria are established for the assessment of multidimensional array databases. Then focus is given to two most promising solutions, SciDB and Rasdaman. Each criterion will be graded and the final grade determines which option will be implemented for benchmarking. Evidence is concluded from literature study, source code checking and discussion with product developing team. No practical tests are involved for the assessment.

3. Building benchmark test environment. On the one hand, queries for benchmarking should be executed on NetCDF, which means an additional API for management and query need to be developed because NetCDF does not provide direct native tools. On the other hand, the selected multidimensional array database should be set up and configured. NetCDF system and multidimensional array database are supposed to be constructed with same hardware and operating system to guarantee a fair comparison. After setting up the benchmarking environment, initial tests can be performed with NetCDF and database.
4. Implementing full benchmarks and analyzing results. Several data management options are of interest to this research.
    a. NetCDF files in classic format
    b. NetCDF files in NetCDF-4 format with uncompressed chunks
    c. NetCDF files in NetCDF-4 format with compressed chunks
    d. Multidimensional array database with uncompressed chunks
    e. Multidimensional array database with compressed chunks

    During the research, datasets with varying sizes are stored with all these options and same queries are executed for all the options to get query response time for later analyzing. Scalability with data size is also assessed for each option. The effect of chunk size and data compression on query performance is also investigated. The final judgment of the best solution for managing and querying large hydrologic datasets is based on the overall performance of data storing, loading and querying.

1.4 Thesis outline

In Chapter 1, the motivation for this research topic, specific research questions and brief description of methodology are introduced. Then Chapter 2 provides the background knowledge which includes NetCDF, multidimensional array database and previous work related to this research. After it, from Chapter 3 to Chapter 6, four phases constituting the main research are described respectively. More specifically, Chapter 3 describes how final queries and datasets for benchmarking are determined based on the expert consultancy. Chapter 4 discusses the process for selecting one multidimensional array database which is applied for final benchmarking. This is then followed by specific description of benchmark test environment including both hardware and software aspects in chapter 5. In Chapter 6, the main output of this research, i.e. the benchmark test results and analysis are elaborated. The whole thesis ends up with conclusions and future work in Chapter 7.

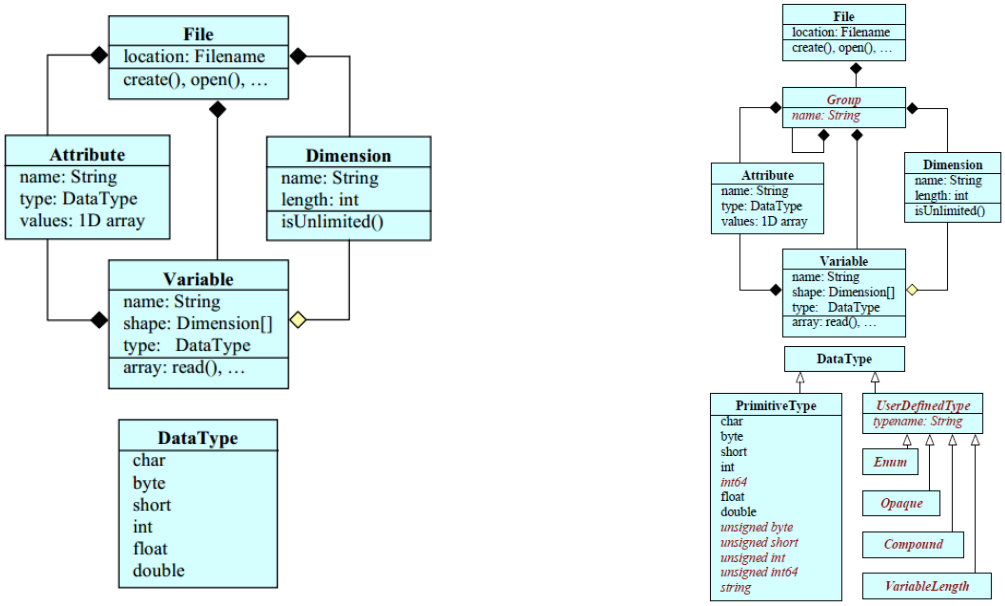
2 Background

2.1 NetCDF

Developed by the Unidata Program of the University Corporation for Atmospheric Research (UCAR), NetCDF is typically referred to as a self-describing, machine-independent binary data format that supports the creation, access, and sharing of array-oriented scientific data. It also represents a data model that organizes a collection of associated abstractions into a high-level view of how to access data (Rew et al., 2006). While NetCDF can refer to a software implementation with associated application program interfaces (APIs) released by Unidata as well. Here only data model and format which constitute the abstract and implementation layers of the NetCDF storage system are introduced. APIs to manage and query NetCDF files applied in this research are developed by Hydrologic Research, which is described in Chapter 5.

2.1.1 Data model

There are only two NetCDF data models, one is the classic model and the other is the enhanced model, also known as NetCDF-4 data model. UML representations of both models are shown in Figure 2.1 below,



a. b.

Figure 2.1. a. Classic data model; b. Enhanced model with red words showing differences from classic data model (Rew et al., 2006)

In the classic model (Figure 2.1a), there are three key elements, namely, dimension, variable and attribute. A dimension may represent a real physical dimension such as longitude, latitude or time. It can also be used to index other quantities, for example, model run number (Unidata, 2011). In

definition, a dimension includes a name and a length. The length can be arbitrary positive integer or UNLIMITED. However, only one dimension has the main grouping effect and this is often the case for temporal dimension which allows grids in new time steps added in (Rew et al., 2006). As to variable, it refers to data of interest like precipitation, temperature, etc. A variable has a name, a data type, associated dimensions and attributes. Each variable represents an array of values of the same type. There are totally six data types available for the classic model (Figure 2.1a). The shape of a variable is described by its list of dimensions specified when the variable is created. A special variable in NetCDF is the coordinate variable which has the same name as an existing dimension and stores the dimension values. The attributes of a variable may be added, deleted or changed after the variable is created. For the last element, attribute, it defines the metadata. Each attribute has an associated variable, a name, a data type, a length and a value. Most attributes provide information about variables, for instance, unit, missing value, valid range, etc. While global attributes that describe the whole file like origin and history can also exist. The attribute is more flexible than variable or dimension since it can be deleted or modified after creation, which is not possible for the others.

The enhanced model incorporates more features than the classic model (Figure 2.1b). Group is the added element and each acts as an entire NetCDF dataset in the classic model with multiple variables, dimensions and attributes. It is similar to dictionaries in operating systems, i.e. hierarchically organized and arbitrary depth enabled, which indicates its capability to hold larger dataset than classic model. Group establishes namespace for variables, groups and types and in the same group, their names must be unique. A notable feature is that dimensions can be shared between variables in different groups which are defined under the same parent group. It is possible to define multiple unlimited dimensions with enhanced model. Another strength of the enhanced data model is the extension of data types. Not only has the number of primary data types increased to 12, but users can also define specific data type, for example, an opaque blob of bytes.

2.1.2 Data format

NetCDF format includes four variants, namely, NetCDF classic, 64-bit offset, NetCDF-4 and NetCDF-4 classic model format. They are implementations of NetCDF data models (Figure 2.2).

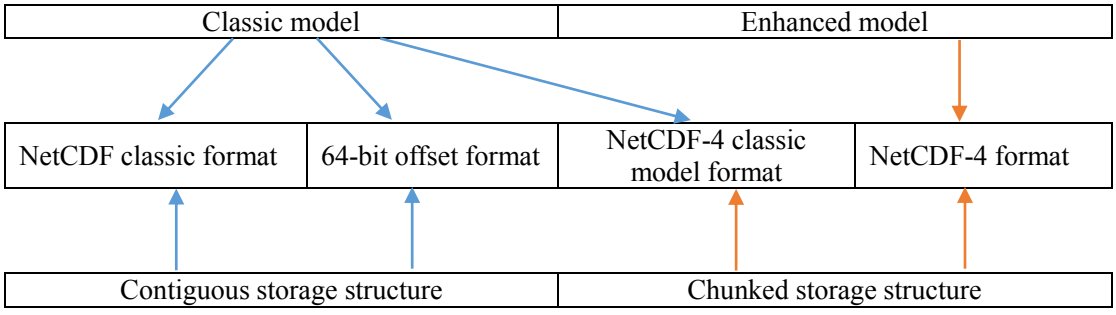


Figure 2.2. Relationships between NetCDF formats, data models and storage structures

Both NetCDF classic and 64-bit format are based on NetCDF classic data model. For storage, they both consist of two parts, a header which is always the first part in the file and then followed by a

data part which contains variable (including coordinate variable) values occupying the main body of the NetCDF file. Header contains information of dimensions, variables and attributes. In addition, the offset to the beginning of each variable and dimension length information is also recorded, which can then be used for locating specific variable value. However, a problem with the header is that it reserves little extra space and any change requiring it to grow would cause copying and moving the whole following data part. Regarding the data part, it consists of fixed-size data part containing data for variables that do not have an unlimited dimension, and the record data part which is reserved for variables holding an unlimited dimension. The whole data part is stored contiguously on the disk (Figure 1.1c). A sample NetCDF classic file recording precipitation time series of 3 time steps is provided in Figure 2.3. In the binary storage, i.e. Figure 2.3b, the first line displays the bytes in hex while a line below presents the interpretation of the bytes in terms of NetCDF components and values. Each line records 16 bytes. The main difference between NetCDF classic and 64-bit offset format lies in the OFFSET entity which indicates the offset in bytes of the beginning of variables in the header of NetCDF. For classic format, this is a 32 bit integer while 64-bit offset format uses a 64 bit integer for this to allow a larger file size, i.e. larger offset for specific data value. More details can be acquired from Unidata (2011).

```
netcdf precipitation {
    dimensions:
            time = 3;
    variables:
            float p(time);
    data:
            p = 3.1, 1.2, 0.6 ;
}
```

a. File content in CDL ("Common Data Language", the original ASCII form of binary netCDF data)

```
4344    4601    0000    0000    0000    000a    0000    0001
['C' 'D' 'F' 1] [ 0 records ]  [NC_DIMENSION]  [1 dimension]

0000    0004    7469    6d65    0000    0003    0000    0000
[  4 char name = "time"    ]   [ size = 3 ]    [ 0 global

0000    0000    0000    000b    0000    0001    0000    0001
attributes ]   [NC_VARIABLE] [ 1 variable  ]  [ 1 char name

7000    0000    0000    0001    0000    0000    0000    0000
 =  "p"    ] [1 dimension  ]   [ with ID 0 ]  [ 0 attributes

0000    0000    0000    0005    0000    000c    0000    0050
(ABSENT)   ] [type NC_FLOAT] [size 12 bytes]    [offset: 80]

4046    6666    3f99    999a    3f19    999a
[   3.1    ]    [    1.2   ]    [    0.6   ]
```

b. Binary storage and interpretation

Figure 2.3. A sample NetCDF file in classic format and its storage.

NetCDF-4 format offers new features such as group, compound types and multiple unlimited dimensions since it is built on enhanced model of NetCDF. Besides, as has been pointed out, contiguous storage structure cannot perform efficiently if the access pattern to variable values does not follow a favored order. To tackle this issue, HDF5 is introduced as the storage layer for NetCDF-4 format, which is known as the chunked storage structure. Basically, a dataset is partitioned into fixed-size pieces that are transferred independently of each other to and from the disk. The fixed-size pieces are referred to as chunks. With this storage sructure, it is possible to compress variable values. Moreover, chunked storage structure does not need a contiguous header region and it can then support dynamic schema change which allows new variables, dimensions and attributes added dynamically without copying any data. However, chunking is a complex process and concerns several parameters such as chunk size and chunk cache size, which significantly influences query performance (Lee, Yang, & Aydt, 2008). If not tuned properly, it can results in performance penalty compared to the use of contiguous storage structure.

Previous libraries which can operate NetCDF classic format or 64-bit offset format cannot work on

NetCDF-4 format which applies more advanced features like group. On the other hand, users may not need to adopt those advanced features. While HDF5 storage structure can indeed bring benefits in some cases. As a result, NetCDF-4 classic model format is created. It is the favorable solution if no advanced features are needed while chunking or compression is wanted.

2.2 Multidimensional array database

In this thesis, multidimensional array database refers to a database of which the abstract model for data management and query is multidimensional array consisting of dimensions and attributes. The dimension shares the same concept defined in NetCDF data models. It can represents commonly referred physical dimensions, i.e. spatial and temporal dimensions or any specific indexes. Attribute is known as variable in NetCDF, i.e. representing the information of interest. For hydrologic purposes, common attributes are precipitation, rainfall rate, evapotranspiration and so on. The multidimensional array includes one attribute value case, i.e. only a single value is stored without dimension information.

To avoid the ambiguous understanding of terms appearing in NetCDF and multidimensional array database, the mapping of concepts for the two storage systems is listed in table 2.1.

NetCDF	**Multidimensional array database**
Dimension	Dimension
Variable	Attribute
Attribute	Array metadata
Chunk	Chunk
File	Array

Table 2.1. Equivalent terminologies in NetCDF and multidimensional array database

Analogous concept of multidimensional array database has occurred in previous publications. Array database which provides database services to manage scientific and business data is the most correlative terminology. However, its definition is vague and only some common features of array databases are concluded. For example, the target to manipulate is multidimensional array. It has the capability to manage “Big Data”. Chunks are used to store the array (Baumann and Stamerjohanns, 2014). Typical examples of array database are Rasdaman (Baumann et al., 1998), MonetDB (Zhang, Kersten, & Manegold, 2013), SciDB (Brown, 2010) and Oracle Georaster (Oracle, 2014). Another commonly mentioned concept multidimensional database should be considered as the superset of the multidimensional array database. Its abstract data model, as indicated by Pedersen & Jensen (2001) is the data hyper cube (Figure 2.4) which is indeed the multidimensional array. Multidimensional database is designed for the efficient and convenient storage and retrieval of large volumes of data that is intimately related and stored, viewed and analyzed from different dimensions (Kenan Technologies, 1996). Its concept is originated from OLAP (OnLine Analytical Processing) systems for business use (Pedersen & Jensen, 2001), such as Essbase from Orcacle and Netezza from IBM. Both solutions for array database and multidimensional database will be concerned when multidimensional array database is referred to.

Traditional relational databases employ table as the abstract data model for data management.

Although columns in the table can be regarded as different dimensions or attributes, the role or importance of each column is equal. There is no distinction between a dimension and an attribute. While in the multidimensional array database, dimensions are used to organize attribute data and the focus is put on attributes. Queries to a multidimensional array database will never target at and only select dimension values.

For data storage, most multidimensional array databases apply chunked storage structure. Basically, a large multidimensional dataset can be chunked into small pieces and then stored on the disk.

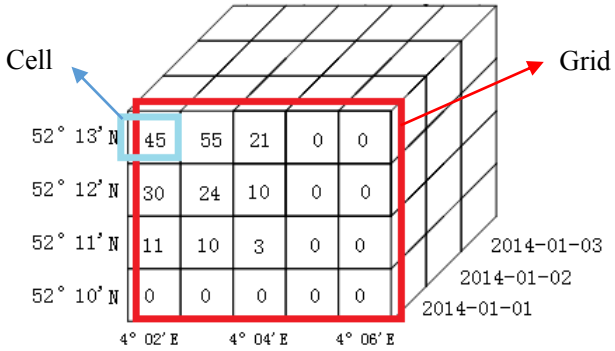


Figure 2.4. Precipitation data for the Netherlands which is organized in a three dimensional array or a data cube. The values shown in cells are precipitation in millimeter.

Figure 2.4 presents an array storing daily precipitation data in a certain region in the Netherlands. The horizontal axis is longitude while the vertical axis represents latitude. Also a temporal dimension is involved as shown at the right side in the figure. The spatial layer, also referred to as a grid of precipitation for each day, for example, the front face in the cube can be saved as a chunk and therefore for the whole dataset, there are totally three chunks stored. Index such as B-tree can be built on these chunks. And when retrieving a specific value in the dataset (Colliat, 1996), first through index, the chunk which the value is in can be returned. After it, within the chunk, the position of the value is calculated using relative offsets, the same way as NetCDF contiguous structure. Finally, the value can be extracted.

2.3 Previous work

Specific storage systems aside, previous studies have published results of relevant benchmark tests. These include for example, comparison between relational database MySQL and multidimensional array database SciDB on dealing with big datasets (Cudre-Mauroux et al., 2012). Researchers cooperate with astronomical domain experts to propose scientific data processing benchmark. The benchmark includes nine specific designed queries. The final results show that SciDB performs one or two orders of magnitude faster on most queries than MySQL. But queries they propose are based on astronomical case, which is not applicable to hydrologic domain. Nevertheless, their approach on how to establish specific benchmarks provide crucial indication to this research.

Lee et al. (2008) present contrast between NetCDF-3 with contiguous storage and NetCDF-4 with

both contiguous and chunked storage on handling datasets of diverse dimensions. In their research, it is pointed out that the chunk size has crucial influence on data write rate and with the size of each chunk increasing, the write rate fluctuates. As a result, chunk size is taken into account in this study.

Su and Agrawal (2012) develop a light weight management tool built on NetCDF classic format files. They then evaluate the performance of the tool with OPeNDAP (Cornillon, Gallagher, & Sgouros, 2003), a data access layer for files in different formats. Conclusion is that for all tasks tested, the tool they built has a better sequential performance than OPeNDAP. However, their address is more on the data management and access layer, i.e. the tool they developed, rather than data storage. Databases are not discussed.

More relevant research is about the comparison between the relational database and NetCDF classic format on executing four aggregate (sum, average, minimum, and maximum) and three index queries (first, middle, and last element) (Cohen, Hurley, Schulz, Barth, & Benton, 2006). And the result demonstrates the superiority of NetCDF over relational database for mentioned queries on large datasets. But for all the publications reviewed until now, no research has been done to investigate the query efficiency of NetCDF compared to multidimensional array databases for hydrologic applications.

In the end, it should be mentioned that some parts of this research get inspiration from the Project Massive point clouds for eSciences conducted in parallel. These include surveying costumers to design benchmarks (Suijker, Alkemade, Kodde, & Nonhebel, 2014) as well as scale the data size to explore the scalability of diverse data solutions (van Oosterom et al., 2014).

3 Queries and datasets

Determining appropriate benchmarks for testing is a crucial factor which influences the evaluation of different storage systems. For comparing performance of databases, storage size, memory usage, data loading time and query response time are benchmarks frequently considered. Recent research on multidimensional array database performance, for example, Cudre-Mauroux, et al. (2012), Su and Agrawal (2012) select typical queries representing different classes and then measure query response time to show the performance. This research applies similar approach. First, several hydrologic experts are consulted for queries they frequently execute as well as datasets on which queries are based. After interviews, queries are classified, which paves the way for designing queries for final benchmark test.

3.1 Expert consultancy

Totally 6 hydrologic experts are consulted. Five of them are interviewed in person while the left one responses through questionnaire. More specifically, the interviewees are imparted what the main questions are before the official structured interview. For the one answering with questionnaire, crucial explanation is provided in the questionnaire to help understand. Then each interview is conducted, which lasts for around half an hour. Essential text notes are made during the meeting. After the interview, records are modified, organized and sent back to interviewees for verification. The reply in questionnaire is kept without any further correction.

In this research, the concept of query comes from databases and is related to SQL, i.e. the query language. More specifically, processes on data that can be executed with SQL are regarded as queries. However queries should not be too complex as well. Tasks such as numerical computation provided by professional software may also be expressed by SQL but these are excluded from this research. During the consultancy, not only datasets and queries frequently handled by hydrologists are discussed (Table 3.1), also their experience with advanced ICT techniques such as parallel structure, compression to manage the data as well as their opinions about big data in hydrology are covered. Interestingly, those experts who come from ICT related companies all think big data will be or is currently the bottleneck in hydrology domain. While responses from hydrologic researchers express that big data has slight influence on their daily work which are only concerned with limited geographical area. The questionnaire as well as records of consultancy are presented in the Appendix A and B.

Expert number	Expert	Queries	Datasets
I	Dr. Ir. Steele-Dunne (Delft university of technology)	1. Detection of observations, i.e. selecting non-null variable values. 2. Quality control, selecting value from variable grid according to quality grid	Remote sensing data products, such as TRMM, GRACE and SMOS; Two spatial dimensions
II	Dr. Ir. Siek		Observation records collected

	(Hydrologic Research B.V.)		from rain gauges; One temporal dimension
III	Prof. Wang (Hohai university)	1. Accumulating daily variable value to yearly based value for whole dataset 2. Calculating average extreme variable value per year for whole dataset 3. Quality control. No-value area should be below 10% for the whole spatial grid according to empirical experience	1. Observation records collected from spot gauges; One temporal dimension 2. Satellite imageries; Two spatial dimensions 3. Data products of remote sensing, like MODIS NDVI and NPP data; Two spatial dimensions
IV	Ir. Commandeur (Hydrologic Research B.V.)	1. Sub-selection according to spatial or temporal dimension from the multidimensional datasets 2. Statistical operation for the dataset, sum 3. Statistical operation for the dataset, average 4. Locating the maximum value of the dataset 5. Calculating the percentile curve for the forecast dataset 6. Subtracting one grid of one dataset from the other grid of another dataset 7. Detecting observations from data captured by orbiting satellites	1. Time series recorded by gauges like precipitation, discharge and temperature; One temporal dimension 2. Precipitation data derived from Doppler radar and also calibrated radar precipitation data using data recorded by gauges; Two spatial dimensions and one temporal dimension 3. Processed satellite data like precipitation and soil moisture. Two spatial dimensions and one temporal dimension 4. Forecast datasets like GEFS calculated from models; Five dimensions, longitude, latitude, time, ensemble and model run time 5. Orbiting satellite observation datasets in swaths; Two spatial dimensions
V	Ing. Van der Wielen (Hydrologic B.V.)	1. Selection of time series data recorded by gauges 2. Sum of time series data recorded by gauges 3. Extracting time series from grids 4. Subtraction of accumulated data, e.g. precipitation 5. Combining data of two grids with different cell size, intersection and multiplication are needed 6. Assigning color to polygons according to values got by intersecting polygons with grids 7. Pyramid calculation from	1. Time series data from gauges; One temporal dimension 2. Radar data Two spatial dimensions and one temporal dimension 3. Results computed from hydrologic models; Four dimensions, longitude, latitude, time and model run date 4. Forecast data from models; Five dimensions, longitude, latitude, time, perturbation and model run time

		fine to coarse resolution and query on pyramid.	
VI	Ir. Villa Real (IBM)	Selecting data from one variable grid according to a NODATA grid	1. Topographic and land cover data; Two spatial dimensions 2. Precipitation raster calculated from forecast model; Two spatial dimensions

Table 3.1. Queries and datasets collected by consultancy

Table 3.1 lists conceptual queries and datasets collected by interview. The first three experts are doing water research. Databases or NetCDF processing software are tools for their work, which might be the reason why they contribute less queries. The second interviewee explained that his work was mainly about modeling and he normally programs own scripts to process data and run the model. Script does not belong to query according to his opinion. While the latter three experts are specialized in ICT techniques and hydrology is the area for application. They have much experience of what kind of queries users often request. Since customers of these ICT experts are mainly hydrologic professionals or water related decision makers, queries offered by them can still be regarded as typical hydrologic queries. As to the dataset, dimensions in the description refers to dimensions that a specific dataset contains in a single file. These files may be combined to get larger region or longer temporal coverage.

Queries listed in Table 3.1 are trivial. To make the overall query types more clearly, classification is then performed (Table 3.2).

Class number	Class	Raw query
A	Selection based on dimension value	IV1, V1, V3
B	Selection based on variable value	I1, IV7
C	Selecting one array according to information in another array, i.e. masking	I2, III3, VI
D	Mathematical calculation	III1, III2, IV2, IV3, IV4, IV5, IV6, V2, V4, V7
E	Spatial operation	V5, V6

Table 3.2. Classification of queries collected from consultancy

In table 3.2, the column raw query references queries listed in table 3.1. On the whole, five query classes are defined. The classification approach draws on experience from previous studies. Specifically speaking, Su and Agrawal (2012) proposed three query types, i.e. query based on dimensions, coordinate variables and variable values to test the NetCDF query tool they developed. Class A and B above covers those three query types. In addition, Cohen, et al. (2006) implement four aggregate (sum, average, minimum, and maximum) and three index queries (first, middle, and last value). The aggregate queries are included in class D which additionally contains algebraic calculations such as subtraction of two arrays and statistical analysis like computing percentile. A problem with previous researches is that they only focus on queries on single arrays and do not consider queries using information more than one arrays. Through consultancy, masking queries such as data quality check (Figure 3.1) for which the selection of variable is based on the information recorded in the reliability or quality array are also collected. This type of query is also defined as a separate class C. Class E represents spatial operations like intersection of a polygon of a city with

precipitation raster. Although hydrologists work on calculation based on rasters, sometimes intersection of raster and vector is performed for visualization. In addition, even for two rasters, spatial operation is needed at times. For example, given a rainfall raster calculated from forecast model and a Doppler radar rainfall product, if the deviation is required but two rasters have different cell sizes, then resampling one of them may be the solution. This is indeed what query V5 indicates. Spatial operation is apparently another unique query type.

145	201	107	567
23	23	80	29
-4	-10	32	16
-35	-107	-67	3

a.

2	2	1	0
2	2	1	1
3	3	3	1
3	3	3	3

b.

Figure 3.1. Sample fragment of MODIS NDVI product. Axis are longitude and latitude. Both arrays share the same location and cell size. a. Array storing value related to NDVI. Real NDVI value should be multiplication of the cell value and a factor 0.0001. b. Pixel reliability array where 0 refers to good data, 1 represents marginal data, 2 indicates snow/ice cover and 3 implies cloud cover. If for a certain cell, its reliability value equals to 3, then the corresponding NDVI value is contaminated and thus not going to be used.

Table 3.2 clearly shows that query of class D, i.e. statistical calculation is most frequently run by hydrologists. While class B and E are the least crucial types. In practice, class B and A are often combined as one query type. Specifically, in most cases, users first select the area of interest or a certain period of time, i.e. selection according to dimension value and then focus on variable values. Only executing query B, i.e. selecting certain range of values from a full grid has no actual meaning because a full grid most of the time does not represent a whole country or a city. The grid is often a part of a dataset with larger scale. As a result, the output can be nowhere. So query purely based on variable value will not be included in the final benchmark test. As to spatial operation, for one thing, it is not very common in the daily work of hydrologic experts; for another, realizing it with no spatial extension enabled NetCDF or database would bring excessive developing work, which deviates from the core of this research. So class E is excluded from implementation as well. Class C is also removed since dataset which is capable to perform the query is in fairly small size due to coarse spatial and temporal resolution, which is not the interest of the research.

3.2 Query design

This section describes how the queries for benchmark test are determined. Raw queries from experts are not applicable for testing directly. This is because first, these queries are at a rather conceptual level, no specific parameters attach to these queries. For instance, temporal dimension a query concerns can range from 2001 to 2010 or from 01/01/2014 to 31/01/2014. While these specific parameter values definitely influence query performance. Second, datasets at which raw queries

target are in large numbers, which brings impediment for management. Moreover, some of these datasets are inaccessible due to their licenses. Given above reasons, query design is required to modify and specify the conceptual queries. Query design should be based on query classification as well as research questions, which will be interpreted in the following.

There are several notes when designing queries:

1. First, the target of queries, i.e. specific dataset should be selected. Dataset size needs to be sufficiently large and has the potential to exceed TB level. Dimensions of the dataset should focus on spatial and temporal dimensions, which keep in line with commonality of datasets described by experts. Multiple dimension is also a requirement. One dimensional dataset is the strength of NetCDF's contiguous storage structure and it does not cause the problem stated in section 1.1, so it is not the interest of this research.
2. Each query should be a realistic request sent by users. That is to say, parameters in a query should avoid being randomly fabricated. For example, it makes sense to select precipitation time series for last 24 hours while it is odd to request information for the past 25 hours. Spatial selection should also be country or city based and escape from being half sea and half land situation. Besides, the scope of all queries is confined in class A and D as is explained in previous Section 3.1.
3. Queries should address sub-research question 3, i.e. query performance on different dimensions. As indicated in section 1.1, if variable values are stored in grids at different time steps, then extracting time series for a single location is problematic for contiguous storage structure. So for one dataset, queries on different dimensions for example, spatial and temporal should be both included to show the performance variation.

3.2.1 Datasets for benchmarking

Considering all the requirements for datasets, i.e. data size, dimension and accessibility, two datasets (Table 3.3) are selected for testing. Dataset **MPE** (Multi-Sensor Precipitation Estimate) stores the rainfall rate data processed from raw satellite data. Besides, the Availability information indicating whether a grid at a certain time step is missing and the Quality marking if the satellite data have been corrected according to ground measurements are also recorded. Each file contains information for one hour with four time steps. Hydrologic Research can provide records for more than two years, which results in the total amount of data larger than 4.18 TB (2 x 365 x 24 x 250 MB). Dataset **GEFS** (Global Ensemble Forecast System) is calculated from a global forecast model and it stores 8 meteorological variables as shown in the table. It also includes the auxiliary information, Datastatus which is the combination of Availability and Quality. GEFS contains more dimensions. The forecast dimension refers to the time steps simulated while model run represents the time to run the forecast model in reality. Increments for both dimensions are 6 hours, i.e. every 6 hours the model will be run and forecast temporal interval is also 6 hours. Ensemble represents initial condition and 20 ensembles simulate 20 different initial conditions as the input for the forecast model. This is done to decrease the uncertainty of the forecast as later the percentile and ensemble mean are derived. GEFS latitudes range from -90 to 0 and then to 90 (181 integers). For both datasets, the spatial reference system is WGS 84.

Dataset	Information stored	Dimension count	Dimension Span (single file)	Temporal resolution	Spatial resolution and coverage	Single file size	Data format
MPE rainfall rate data, satellite data product	Rainfall rate; Availability; Quality	3	x, y, time (4000,4000,4)	15 minutes	0.03 degree (3.3 km), 1/3 world	250 MB	64-bit offset
GEFS weather forecast data	Temperature 2m above ground; Maximum temperature 2m above ground; Minimum temperature 2m above ground; Relative humidity 2m above ground ; Total precipitation; Total Cloud Cover; U-Component of Wind 10m above ground; V-Component of Wind 10m above ground; Data status	5	Longitude, latitude, forecast, ensemble, model run (360,181,40,20,1)	6 hours	1 degree (111 km), Global	1.55 GB	64-bit offset

Table 3.3. Datasets for benchmarking

3.2.2 Queries for benchmarking

Based on the above two datasets, designed queries are listed below.

Dataset MPE:

1. *Selection on spatial dimension for Delft and northern part of the Netherlands (Class A):*
   a. Delft: Select rainfall rate in the bounding box (4.31, 51.97) (4.39, 52.03) at 16:30 01-09-13
   b. Netherlands: Select rainfall rate in the bounding box (4.72, 51.93) (6.5, 53.36) at 16:30 01-09-13

   Expected output: a. a two dimensional array with size (4, 3), 12 cells. b. a two dimensional array with size (60, 48), 2,880 cells.

2. *Extraction of time series for Delft (Class A):*
   Select rainfall rate from 0:00 01-09-13 to 0:00 01-10-13 at (4.37, 52.02)

   Expected output: a one dimensional array with length 30 x 4 x 24 = 2,880

3. *Pyramid query at the Netherlands scale(Class D):*
   After building the rainfall rate pyramid (x4), i.e. average every 16 cells for the whole grid at 0:00 07-09-13, then retrieve the Netherlands grid with bounding box (3.36, 50.76) (7.21, 53.51) from the pyramid.

   Expected output: a two dimensional array with size 33 x 23 = 759

4. *Historical average value for the Netherlands (Class D):*
   Select average rainfall rate from 0:00 01-09-13 to 0:00 01-10-13 in the bounding box (3.36, 50.76) (7.21, 53.51)

   Expected output: a two dimensional array with size (129, 92), 11,868 cells while the calculation goes through 129 x 92 x 30 x 24 x 4 = 34,179,840 cells

5. *Historical maximum value for the Netherlands (Class D):*
   Select maximum rainfall rate in the bounding box (3.36, 50.76) (7.21, 53.51) from 0:00 01-09-13 to 0:00 01-10-13

   Expected output: a two dimensional array with size (129, 92), 11,868 cells while the calculation goes through 129 x 92 x 30 x 24 x 4 = 34,179,840 cells

Dataset GEFS:

1. *Forecast time series for Delft in 10 days (Class A):*

Select precipitation for all ensembles at (4, 52) for 40 forecast steps, model run time is 6:00 15-05-2014

Expected output: a one dimensional array of precipitation forecast and the length is 20 x 40 = 800.

2. *80th percentile for Delft in 10 days (Class D, percentile indicates the value below which a given percentage of observations in a group of observations fall. For instance, suppose there are 10 natural numbers ranging from 1 to 10, the 80th percentile is 8 since 8 values are below it including itself):*
   Select the 80th percentile at (4, 52) from all ensembles for 40 forecast steps, model run time is 6:00 15-05-2014

   Expected output: a one dimensional array with length 40 while the calculation goes through 40 x 20 = 800 cells.

3. *Ensemble mean for the Netherlands in 10 days (Class D):*
   Select mean of 20 ensembles of precipitation for 40 forecast steps in the bounding box (3, 51) (7, 54), model run time is 6:00 15-05-2014

   Expected output: 40 two dimensional arrays of which the size is (5, 4), 20 cells, i.e. totally 800 values are returned, while the calculation goes through 40 x 20 x 5 x 4 = 16,000 cells.

Totally 8 queries are designed. Output size is also taken into account when designing queries since it influences the query performance as well. As is presented, the output size in cells span several levels from 12 to 11,868. Another point should be noted is that the shape of the output refers to the array size at the abstract level. So selecting a sub-grid, i.e. selection based on two spatial dimensions from the whole grid results in a two dimensional array intuitively. While in practice, result is a series of numbers in the row-major order (Figure 3.2). Query 1 and 2 of MPE dataset can be used for evaluating whether querying on different dimensions can achieve same performance. This is reasonable because the output size does not deviate much from each other. For GEFS dataset, there is no queries for spatiotemporal aggregation since its resolution is fairly large, i.e. 1 degree and 6 hours. While aggregation like average on all ensembles are more general query types for ensemble forecast datasets. It should be mentioned that although queries designed above consider several crucial factors, in the real benchmarking (Chapter 6), more aspects are taken into account, for example whether query results retrieved are all zero values or diverse non-zero values.

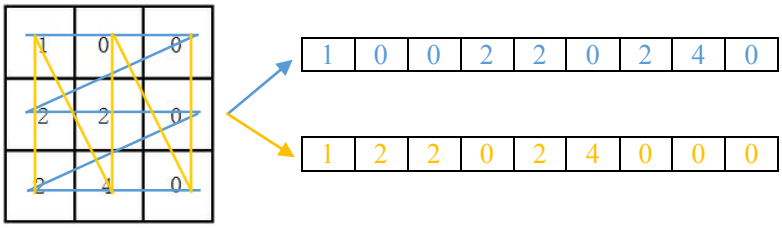


Figure 3.2. Data organization with row-major order (blue) and column-major order (yellow)

4 Selection of multidimensional array database

Due to large numbers of various specific multidimensional array databases and the complexity to implement full benchmarks, for example, importing data, cache management and employing different storage schema, it is decided that only one database will be selected for benchmarking. Literature study is the main methodology to select this databasee. In the following content, first several solutions including both open-source and commercial multidimensional array databases are presented. Then two typical multidimensional array databases, i.e. Rasdaman and SciDB which are covered intensively by previous researches are compared in more detail with nine criteria. In the end, based on their overall grades for all criteria, the final multidimensional array database applied in this research for benchmarking is determined.

4.1 Current multidimensional array databases

Rasdaman

Rasdaman strives for domain-independent support for arrays of arbitrary size, dimension, and base type through a general-purpose declarative query language RasQL, paired with internal execution, storage, and transfer optimization (Baumann, 1998). Rasdaman implements chunked storage structure and makes use of BLOB (binary large object) type within MySQL, Oracle, IBM DB2, IBM Informix, PostgreSQL and file systems to store chunks. So it is always combined with other DBMSs.

SciDB

SciDB is an open-source DBMS intended primarily for use in application domains that involve very large (petabyte) scale array data. Scientific applications include astronomy, remote sensing, climate modeling and bio-science information management. Commercial applications like risk management systems in the financial services sector as well as web log analysis software can also be constructed on SciDB. It is a bottom-up designed array database from the physical storage layer to logical access layer. It utilizes chunked storage structure using native binary format.

MonetDB

MonetDB is applied for data mining, business intelligence, OLAP, scientific databases, XML Query, text and multimedia retrieval required by high-performance applications (Idreos et al., 2012). Arrays are stored using column-store structure and each dimension as well as the core data is stored as one BAT (Binary Association Table) in the database. All BATs are related together through one column OID stored inside each BAT. SciQL, a SQL-based declarative query language is later introduced to MonetDB for scientific data management (Zhang, Kersten, & Manegold, 2013).

Essbase

Essbase is originally a multidimensional DBMS that serves for analytic applications. It is now held

by Oracle as a multidimensional OLAP server product, providing a rich environment for effectively developing custom analytic and enterprise performance management applications. It separates dense and sparse dimensions to organize data more efficiently. More specifically, in Essbase, dimensions which usually have data in every cell are dense dimensions and are represented as blocks, i.e. chunks on disk. Other dimensions are set to sparse dimensions and for those combinations of sparse dimensions where data exists, pointers to corresponding dense blocks are stored in an array (Figure 4.1). Through the array index and relative offset in the data
block, a specific value can be retrieved (Colliat, 1996; Oracle, 2008).

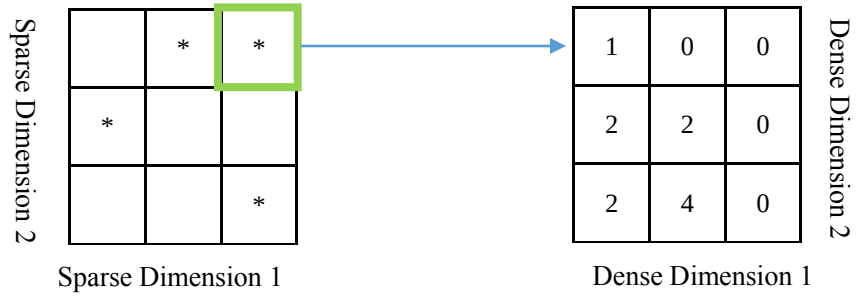


Figure 4.1. Storage of a fourth dimensional dataset in Essbase. The left array is constructed by sparse dimensions and the cell with star, i.e. pointer implies there is data in the cell. The right array is one data block confined by dense dimensions.

InterSystems Caché

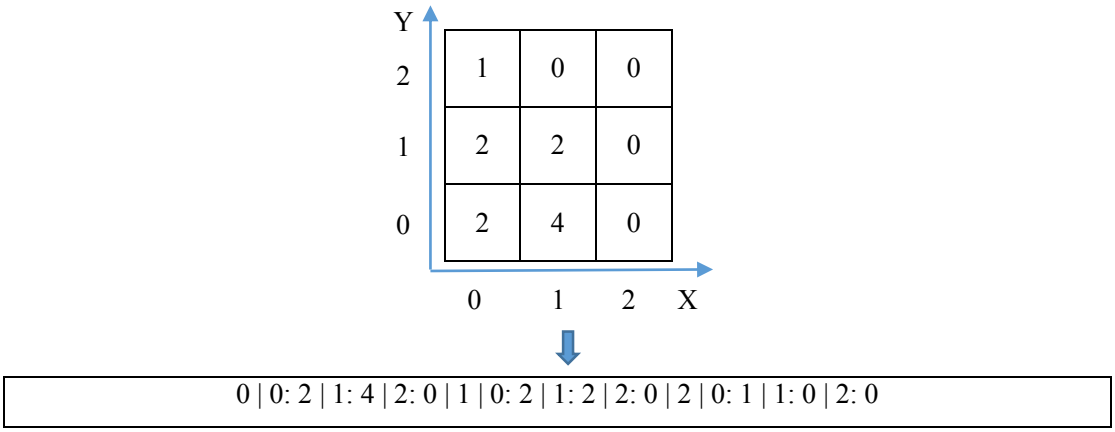


Figure 4.2. Dimension values and attributes of a two dimensional array are stored sequentially in Caché's data block.

Intersystems Caché is a multidimensional DBMS based on which users are able to process and analyze complex data, developing web and mobile applications. Caché provides native scripting language to manipulate arrays directly, which is the multidimensional mode. Multidimensional arrays in Caché are named "globals" which are stored on disk within a series of data blocks (InterSystems, 2014). Dimension values and attributes are compressed by default and then stored together in data blocks sequentially (Figure 4.2). A special case is for arrays with large object as the attribute value, for example, a very long string. In this case, dimension values and attributes are stored in separate blocks. And what is then stored with dimension values are pointers to corresponding attribute blocks.

Oracle spatial

Oracle spatial provides users the ability to store, index, query, analyze and deliver raster images and other gridded data together with associated metadata within the Oracle Relational Database Management System (Xie, 2008). Basically, a raster data, for example, an image has an entry stored in Oracle as a GeoRaster object. For each GeoRaster object, there is a corresponding raster data table which records information of small blocks composing the image. While the actual BLOB with image data for each block is stored separately from the raster data table (Oracle, 2014; Xie, 2008). Figure 4.3 presents the overall physical raster storage structure adopted by Oracle spatial.

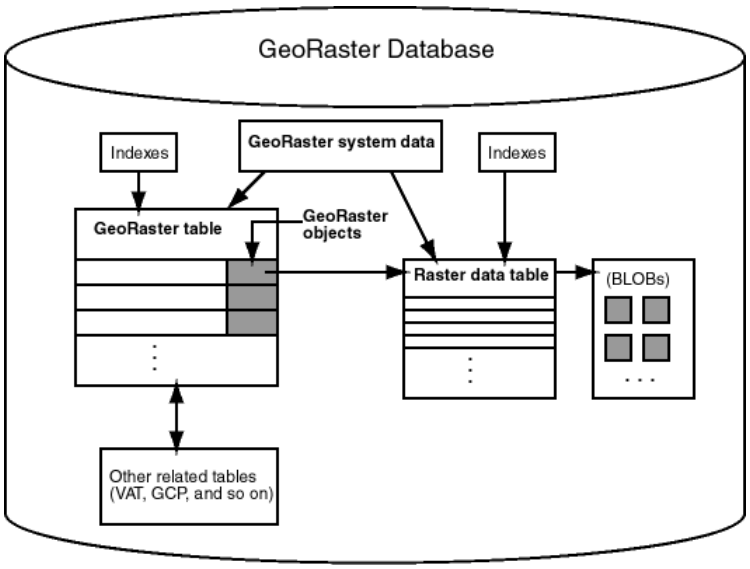


Figure 4.3. Storage of raster data in Oracle spatial (Oracle, 2014).

However, Oracle spatial can maximally supports storage of raster of three dimensions, i.e. x, y and band. For higher dimensional datasets, they cannot be incorporated as a GeoRaster object. Strictly speaking, the abstract data model of Oracle Spatial for GeoRasters is based on tables while the array, i.e. raster is treated as a data type. But since many specific functionalities are developed for managing multidimensional rasters (Oracle, 2014), so in a way, users can still handle arrays directly. Consequently, Oracle spatial is included as one multidimensional array database.

Universal File Interface (UFI)

UFI (BCS, 2014) is IBM Informix database extension to index and query files stored outside the database. The aim of this solution is to avoid the traditional difficult and slow data loading process. Regarding its working principle, basically, it maps selected elements like columns in CSV files and variables in NetCDF files into UFI Virtual Table (VT) columns. That is to say, one VT column corresponds to one column or variable of a file. Then though querying the table with SQL, data stored in the files can be retrieved (BCS, 2012). The UFI server communicates with various files through different adapters. Specific adapters can be developed by users with UFI Adapter SDK released by the development team. Figure 4.4 shows the query workflow of UFI.

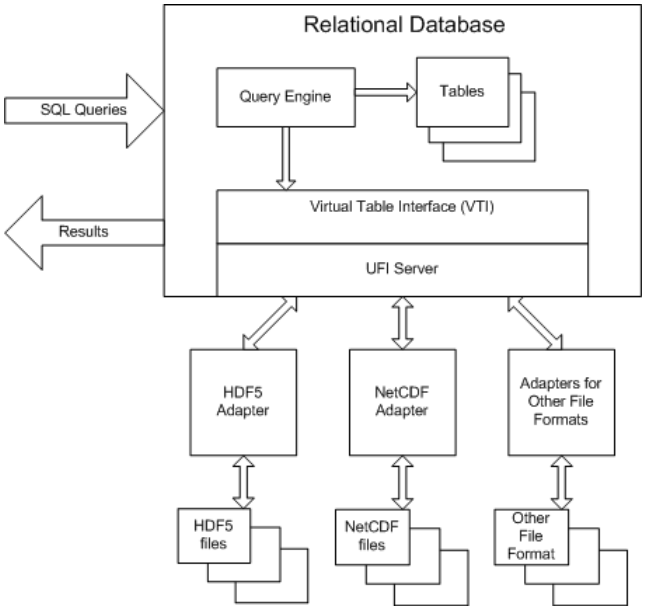


Figure 4.4. Workflow of query execution with UFI

4.2 Comparison between Rasdaman and SciDB

Due to large numbers of variants of multidimensional array databases, it is insensible to evaluate them all in detail to select the one for final benchmarking. Among these solutions, Rasdaman and SciDB provide sufficient documentation for study and research. Besides, their source code is accessible online, which is crucial for exploring more details of data structures later. Consequently, for determining final multidimensional array database used for test, emphasis will be put on these two options.

In the following parts, essential criteria for the comparison are listed. And features for both databases with respect to each criterion are described. How those descriptions come into being are also provided. It should be noted that no practical test is involved for the comparison conducted in this phase.

1 License

Open source product is preferable because on the one hand, more details of storage and query can be acquired from checking the source code and on the other hand, additional budget is unavailable for buying licenses for commercial ones. The licenses applied by Rasdaman and SciDB are listed in table 4.1 below,

Rasdaman	SciDB
Two versions: open source, i.e. Rasdaman community and commercial version, Rasdaman enterprise For Rasdaman community, its client library, applications and OGC frontends are based on LGPL-3 (http://www.rasdaman.org/wiki/License)	It includes both community and enterprise edition. SciDB community applies Affero GPLv3 license (Feature chart，http://www.paradigm4.com/licenses/; SciDB, 2014)

While the server-side executables (such as rasserver and rasmgr) are based on GPL-3 license For Rasdaman enterprise, it extends rasdaman community with improved features, performance and maintenance. (Feature chart, http://www.rasdaman.org/wiki/Features) and is held by Rasdaman GmbH	Enterprise edition provides more functionalities and support, e.g. additional math and machine learning functions, database replication, etc. (Feature chart). It is held by paradigm4, Inc.

Table 4.1. Licenses of Rasdaman and SciDB

Both databases have free versions, i.e. Rasdaman community and SciDB community. As is indicated, they are preferable over commercial counterparties, so features of free versions are the stress for the comparison.

2 Implementation of multidimensional array storage

As has been indicated in Chapter 2, chunked storage structure are employed by most multidimensional array databases. But the details for real implementation such as spatial index can influence the query performance as well, which should be considered. Table 4.2 lists essential items for both databases towards this criterion.

Rasdaman	SciDB
1. Chunked storage structure. Rasdaman divides the original dataset into chunks which have same dimensions as the original dataset. (Baumann, Furtado, Ritsch, & Widmann, 1997) 2. Chunks can be regular, i.e. share the same size and shape or arbitrary with different shapes and sizes. (http://rasdaman.org/wiki/Tiling) 3. A multidimensional index (e.g. R+) is applied to identify chunks. (Baumann, Furtado, Ritsch, & Widmann, 1997; Baumann, Dehmel, Furtado, Ritsch, & Widmann, 1998; http://en.wikipedia.org/wiki/Rasdaman)	1. Chunked storage structure with chunks of fixed logical size but variable physical size (Stonebraker, Brown, Poliakov, & Raman, 2011). 2. Based on the information of how the array is divided into chunks and how the chunks are distributed into different instances, queries can be implemented (SciDB forum, http://www.scidb.org/forum/viewtopic.php?f=13&t=1259&p=2546&hilit=index#p2546). 3. Original dataset can be divided into basic chunks of any dimensions (Stonebraker et al., 2011; Email communication with Prof. Stonebraker) 4. Overlap of chunks are applied to facilitate querying process. (Brown, 2010)

Table 4.2. Implementation of data storage structure of Rasdaman and SciDB

The fixed logical size but variable physical size of chunks in SciDB means the size of a chunk is fixed by lengths of dimensions while since there may be no value or null value in the chunk, the physical size of such a chunk is variable. This design is considered more efficient for array operation than the approach with variable logic size with fixed physical size. This is because for joining arrays, a frequently implemented operation, fixed logical chunk size can guarantee chunks of both arrays being processed in pairs while variable logical chunk size implies that the chunk of one array may join to several chunks in the second array, which is costly (Stonebraker, Brown, Poliakov, & Raman, 2011).

3 Compression support

Compression support. The availability of compression is one of the interests of this research because for hydrologic datasets, usually a large amount of zero values or no values can exist and it is deduced that compression should thus have a notable influence on query efficiency. Table 4.3 concludes the compression support for both databases.

Rasdaman	SciDB
1. Compression is not available for Rasdaman community. But through format conversion, like JPEG, compression can be incorporated in a way. Besides, null value is not supported. (Feature chart; Source code: compression and conversion module; Email communication with Prof. Baumann) 2. Format conversion as well as additional compression techniques like Zlib, RLE and wavelet are available for Rasdaman enterprise. The commercial version also supports null value. (Feature chart)	1. Delta encoding for version control. One base array is stored while all other updated array versions are stored as deltas chain layers. (Stonebraker et al., 2011; Seering, Cudre-Mauroux, Madden, & Stonebraker, 2012) 2. Support for various types of chunk compression (e.g., Run Length encoding, Null suppression, Huffman encoding) (Stonebraker et al., 2011; SciDB forum, http://www.scidb.org/forum/viewtopic.php?f=6&t=181) 3. SciDB supports null value and its default value can be changed (SciDB, 2014).

Table 4.3. Compression support of Rasdaman and SciDB

Format conversion in Rasdaman refers to that some data formats such as JPEG implement a certain lossy compression, so conversion can result in compression. However, the lossless compression technique like zlib is not served by Rasdaman community, which is a critical drawback of Rasdaman for this research. Delta encoding is a technique applied by SciDB to achieve equivalent effect as compression. Basically, for an array of which values are constantly changing, for example, precipitation array in different time steps, only the latest version of the array is stored while all its previous versions are stored as a chain of "deltas" referenced from the latest array. This "no overwrite" storage approach is implemented by SciDB automatically.

4 Parallelization

Parallel processing capability is needed because it is currently an effective technique dealing with big data to improve query performance. The parallel architectures of both databases are summarized in Table 4.4.

Rasdaman	SciDB
Both shared memory, i.e. multi-core and shared nothing parallel architecture are supported (Hahn, Reiner, Höfling, & Baumann, 2002)	Both shared nothing and shared memory parallel architecture. Queries can be run on several instances on a single server with multiple threads or distributed to a cluster and executed on several servers in parallel (Cudre-Mauroux et al., 2012).

Table 4.4. Parallel architectures of Rasdaman and SciDB

Shared memory architecture refers to the database making use of multiple processors which access the same memory. It adopts a method of inter-process communication. That is, one process creates a space in memory which is accessible by other processes, which significantly improve the speed of communication. However, all communicating processes should run on the same machine and is not

suitable for systems which do not have cache coherent architecture. Shared nothing architecture represents a group of computers which are treated as nodes and communicate with each other through network. And query optimizer runs portions of the query on data stored locally. In essence, it adopts a "send the query to the data" model which is applicable for petabyte scalability (Stonebraker et al., 2011).

5 .Net C# API

.Net C # interface is a requirement because the HydroNet-4 system developed by Hydrologic which later will be used for benchmarking is constructed on .Net platform. To keep consistency, it is preferable that the database can be accessed through .Net C# API. Table 4.5 presents the comparison on this criterion.

Rasdaman	SciDB
No direct C# API for Rasdaman. But Rasdaman has a GDAL driver which can connect to rasdaman by defining a query template. And GDAL indeed provides a .Net C# API.	1. No, it only provides APIs of C++, Java/JDBC, R and Python. 2. C# wrapper can be created from C++ code. 3. There is tutorial for Java (SciDB User Guide), R (https://github.com/Paradigm4/SciDBR/wiki/_pages) and Python (SciDB-py, Online tutorial), but no material about C++. It is implied in the SciDB forum that from source code, many C++ examples can be used for learning.

Table 4.5 .Net API for Rasdaman and SciDB

Although SciDB can wrap C++ interfaces into C# classes to realize the C# API, but the documentation of C++ API is not provided. Even though its source code can be used for leaning, the source code is not as readable as Rasdaman which provides more comments.

6 Query language

Query language of the database should have a simple yet powerful interface which provides plenty functionalities for communicating with the database. The query language for each database is provided in Table 4.6 below.

Rasdaman	SciDB
RasQL, constructed on standard SQL. (Rasdaman, 2013)	AQL and AFL. Both are query languages. AQL is modeled after SQL, so the grammar is like SQL. While AFL provides a more powerful functional language interface. (SciDB, 2014)

Table 4.6. Query language of Rasdaman and SciDB

Below some sample commands with RasQL, AQL and AFL for manipulating arrays are provided,

Query: *create a two dimensional rainfall array which is a spatial grid of 100 x100 cells with default cell value 0.*

RasQL:

1. *typedef marray <float, [0:99, 0:99]> rainfallmap;*

2. *typedef set <rainfallmap> RainfallSet;*
3. *create collection rainfall RainfallSet;*
4. *insert into rainfall values marray x in [0:99, 0:99] values 0f;*

AQL: *create array rainfall <rainfall: float default float(0.0)> [x=0:99, 100, 0, y=0:99, 100, 0];*
AFL: *create array rainfall <rainfall: float default float(0.0)> [x=0:99, 100, 0, y=0:99, 100, 0];*

Query: *select a sub array of which both x and y dimensions range from 0 to 9, i.e. 10 x 10 array from the original rainfall array created above.*
RasQL: *select rainfall[0:9, 0:9] from rainfall;*
AQL: *select * from rainfall where x < 10 and y <10;*
AFL: *between(rainfall, 0, 0, 9, 9);*

The grammar of both RasQL and AQL is analogous to SQL. Apart from AQL, SciDB supports the functional query language AFL. With AFL, general information of the array can be retrieved conveniently, for example, with operator "analyze" (SciDB, 2014). Also commonly used mathematical functions like "stdev" which calculates the standard deviation of the input array are implemented as commands. RasQL also supports corresponding functional operators. For instance, through "SELECT dbinfo() FROM…", some general information of the array can also be acquired (Rasdaman, 2013).

7 Spatial calculating capability

The database is expected to be able to perform spatial operations such as intersection, distance calculation and projection conversion since some of the hydrologists work is spatial relevant (Table 3.2). Information of this aspect is shown in Table 4.7.

Rasdaman	SciDB
No specific spatial operator is provided (Rasdaman, 2013)	No specific spatial operator is provided. (SciDB, 2014)

Table 4.7. Spatial calculating capability of Rasdaman and SciDB

Some operators of AFL like "regrid" can achieve the effect of resample, a common spatial operation. But actually it is more a statistical calculation than spatial function. With RasQL, it is able to develop a script to perform the same task. On the whole, both databases lack essential support for spatial calculation.

8 NetCDF importer

A direct NetCDF importer for a database is also desired because datasets originally in different formats can be converted to NetCDF formats inside HydroNET-4 system. Then it would be convenient to import the NetCDF files directly into database with the specific importer.

Rasdaman	SciDB
Rasdaman has a module supporting NetCDF import. But sometimes it cannot work well and needs prior process using extra tools like ncdump, ncks, etc. (Google mail list, https://groups.google.com/forum/#!topic/rasdaman-users/1UcadWb5aHw)	No function for loading NetCDF directly, but it is possible to program a specific SciDB plugin of the loader. SciDB can support CSV import natively and HDF5 data through the HDF5 loader plugin, i.e. SciDB-HDF5 (https://github.com/wangd/SciDB-HDF5) The approach can thus be converting NetCDF to CSV or HDF5 and then importing data into

	SciDB.

Table 4.8. Information of NetCDF importer for Rasdaman and SciDB

Table 4.8 indicates Rasdaman has implemented specific NetCDF importer despite errors occur sometimes. While through format conversion, there also exists a way to load NetCDF files into SciDB. Besides, HydroNET-4 can provide such format conversion service.

9 Maintenance

The database should keep consistent maintenance to fix bugs, improve and add new features if they are required by large numbers of users. Besides, the consultancy support for using the database should also be sufficient, which can benefit this research. Table 4.9 lists relevant information of maintenance for two databases.

Rasdaman	SciDB
1. Community version is governed by its Project Steering Committee (PSC) consisting of 3 members. (http://rasdaman.org/wiki/Governance) while Rasdaman enterprise is maintained by Rasdaman GmbH. 2. Source code is maintained through Git where developers can contribute code for Rasdaman (https://github.com/PublicaMundi/rasdaman). It is then determined by PSC whether the code will be adopted (http://rasdaman.org/wiki/Governance). 3. Google groups for Rasdaman users (https://groups.google.com/forum/#!forum/rasdaman-users) and developers for Rasdaman community (https://groups.google.com/forum/#!forum/rasdaman-dev) 4. Obsolete website for both community (http://rasdaman.org/) and enterprise (http://rasdaman.com).	1. SciDB open source project is developed by MIT, Brown University, university of Washington (http://scidb.cs.washington.edu/#), Portland State University and University of Wisconsin-Madison. (http://database.cs.brown.edu/projects/scidb/) 2. A forum publishes code and essential documentation with respect to each community version of SciDB. Developers also communicate through this forum and reflect to SciDB project team. (http://www.scidb.org/forum/) 3. Advisory Board consists of engineers and scientists from several specialized research institutes and universities in the US. (http://www.scidb.org/about/science-advisors.php) 4. Enterprise version is held by Paradigm4, Inc. which also contributes to SciDB community.

Table 4.9. Maintenance of Rasdaman and SciDB

In Table 4.9, it clearly shows that SciDB has more professional support, which can be utilized for this research as well.

Given the above information, the performance of two databases with respect to all the criteria can be summarized and compared (Table 4.10).

	C1	C2	C3	C4	C5	C6	C7	C8	C9	Final
Rasdaman community	1	1	0	1	0.5	1	0	1	0.5	6
SciDB community	1	1	1	1	0	1	0	0.5	1	6.5

Table 4.10. Relative grade for the open-source versions of two databases

In Table 4.10, C1 to C9 correspond to criterion from one to nine. The maximum grade is 1 while the

minimum is 0. There exists the 0.5 referring to that the database has partial support for the corresponding criterion. The whole grade shows that SciDB has slight superiority. It should be noted that the grade is based on the assumption that all criteria are of equal weight. As a matter of fact, the C2 indicator, i.e. compression support is a critical factor since the research is concerned with it. And SciDB community has compression functionality which is on the other hand, a lack for Rasdaman community. So, the final multidimensional array database for benchmarking is determined to be SciDB.

5 Testing environment setup

To benchmark two storage systems, i.e. NetCDF and SciDB, interfaces to query and manage data stored in them should be available to use. Although there exist official tools to manipulate data for both systems, to achieve fair benchmark purpose, customized data query and management layers are developed for this research. The benchmark software environment is then built on a server.

In the following, first the overall benchmark architecture along with its working principle is presented. Then the hardware and software parts of the architecture are described. Regarding software, only the implementation of NetCDF connector and SciDB connector which are developed for the research is discussed.

5.1 Overall architecture

HydroNET-3 system developed by HydroLogic provides functions to access data stored in NetCDF classic and 64-bit offset format. Benchmark test is thus based on this system to avoid excessive code development. However, HydroNET is built on Widows .Net platform while SciDB currently is only available on Linux. In order to guarantee a fair comparison, a connector from HydroNET in Windows to SciDB on Linux needs to be developed. So the query performance of both NetCDF and SciDB can be assessed in the same HydroNET system. Besides, the company integrates the interface to query and manage NetCDF-4 files in the next version of HydroNET, i.e. HydroNET-4. Therefore, the whole benchmark environment (Figure 5.1) is elaborated in a beta version of HydroNET-4.

A data retrieving query starts at a web client which sends an http post request with a JSON object describing essential parameters for executing the query. Parameters include the span of each dimensions of the dataset for selection, for example, time steps and spatial region. Besides, a unique ID for each dataset available, so called data source ID is also a crucial parameter involved. The data access API receives the request and then parses it. API communicates with the catalog database, a MySQL database and finds metadata of the corresponding dataset through ID. The metadata indicates where the dataset is stored and according to this information as well as the query, API retrieves the data from the specific data storage system through connectors. For example, if the concerned dataset is stored as an array in SciDB, the catalog database will have a record for such a data source. Through matching data source ID, the API can then send the query parameters to SciDB connector which initiates an AFL command and transfers it to SciDB to extract the result. After it, the result is returned to HydroNET-4 and stored as a HydroNET in-memory object. For NetCDF files, API can determine which file contains the data needed for the query according to data source ID and then retrieves the data through NetCDF connector. The result is also stored in a HydroNET object. All connectors can communicate with Processor containing functions to execute more complex computation such as aggregation and percentile calculation on data retrieved. For benchmarking, the Processor is utilized for realizing complex queries for NetCDF while SciDB process all calculation itself and communicates with API directly. It is though that the calculating capability of a database is also important to understand. The total time spent for executing a query

can be measured in HydroNET-4, which forms the benchmark indicator.

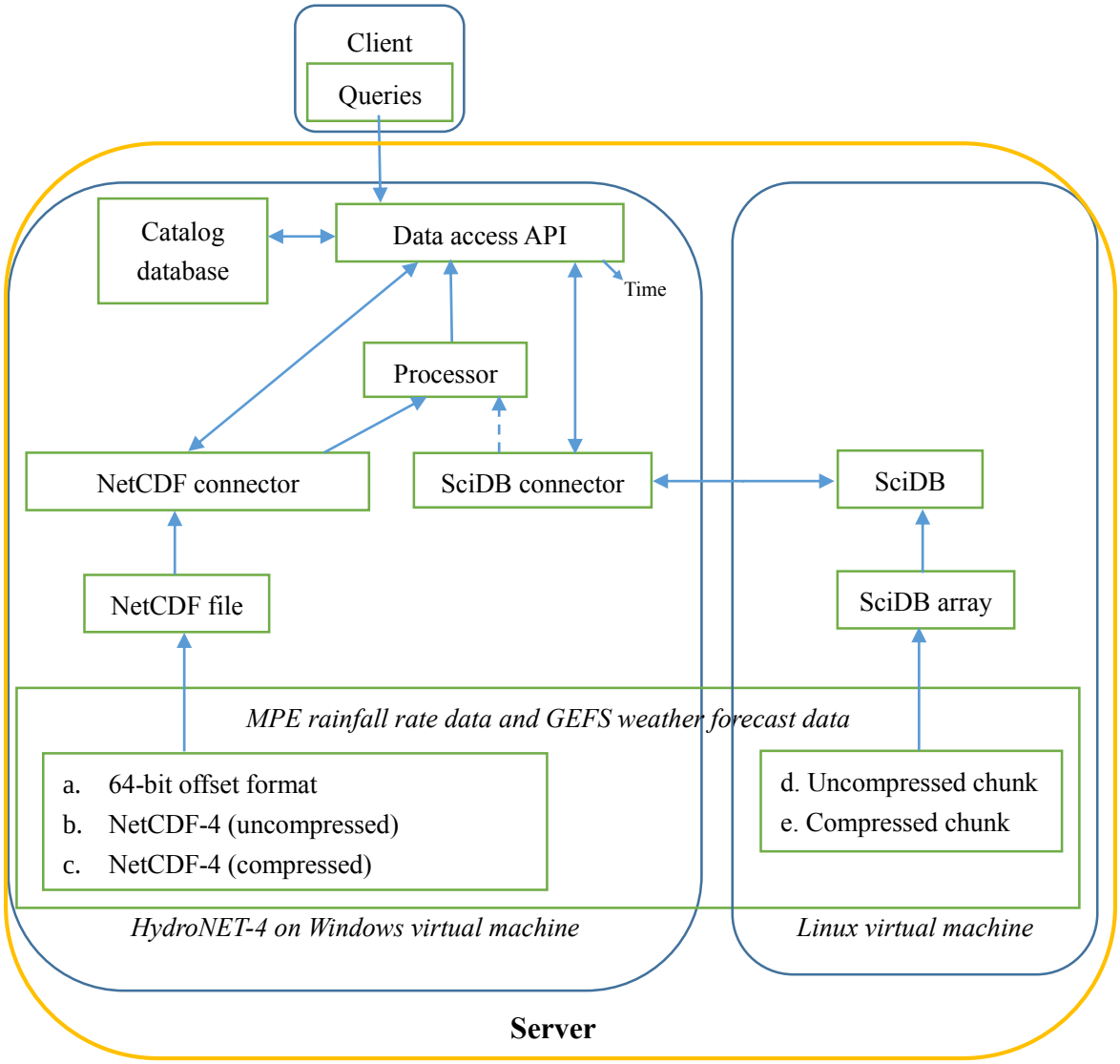


Figure 5.1. Benchmarking architecture

In practice, VMware Workstation, ESXi version 5.5 which holds two virtual machines is installed on a server. On one virtual machine, Windows Server 2012 64-bit is installed to set up HydroNET-4, and Ubuntu 12.04 64-bit version is established on the other virtual machine to run SciDB. Total memory capacity of the server is 8 GB while each virtual machine takes a half, i.e. 4 GB. Each virtual machine is equipped with 2 virtual CPUs sharing the same physical CPU with 2 cores. Besides, each virtual machine owns 1 TB storage space allocated from the same 3 TB hard disk. Another alternative architecture for benchmarking is to employ two physical machines installed with Windows and Ubuntu separately and let the SciDB connector communicate through network. But considering the possible delay in the network, which influence query measurements, the approach is not implemented.

5.2 Hardware

Figure 5.2. Server used for benchmarking

Server
Dell Inc. OptiPlex 745

CPU
Intel(R) Core(TM) 2 CPU 6600 @ 2.40GHz
Processor Sockets: 1
Processor Cores per Socket: 2
Logical Processors: 2

RAM
4x 2GB DDR2
Crucial DDR2 - 667 MHz / PC2-5300 - CL5 - 1.8 V

HDD
Western Digital
WD Red WD30EFRX - 3 TB - intern - 3.5" - 5400 rpm - SATA-600 - 64 MB cache (note: the drive is SATA-600 but the motherboard has SATA-300)

5.3 NetCDF connector

Unidata has released programming libraries with different language interfaces such as C, Fortran, Java, Python and MATLAB to access NetCDF files. Starting from the standard NetCDF library version 4.0, it is possible to read and write files in NetCDF-4 format. In order to optimize specific data flow for NetCDF classic files, earlier versions of HydroNET adopt HydroNetCDF library developed themselves. In HydroNET-4, the NetCDF connector still uses functions provided by HydroNetCDF to access NetCDF classic files. However, HydroNetCDF only supports classic and

64-bit offset format. To manage data stored in NetCDF-4 format, the connector applies SDS (Scientific DataSet) 1.3 library released by Microsoft Research after comparing available options. SDS's NetCDF access module is constructed on the standard library NetCDF-4.1.3.

For the NetCDF connector, it implements the reader and writer class for the three formats. The data structures designed and used by HydroNET objects can only be several types such like *Grid* and *TimeSeries* (Appendix C) since the data access API only recognizes them. Basically, when a query (Figure 5.3) reaches the reader, through processing the query with functions from NetCDF connector, a HydroNET object is retrieved as the result. Functions are developed with SDS and C# standard library, and are mostly constructed by iterations and assignment statements. After it, the object is stored in a JSON ASCII file to accomplish the query process. The NetCDF writer writes HydroNET objects read from other formats such as GRIB2 into NetCDF files. For NetCDF-4 format, the writer also incorporates chunk size and compression parameters to implement chunked storage approach.

```
{
    "DataSourceId": 70,
    "DataFormatCode": "json",
    "ReadSettings": {
        "DataStructure": "Grid",
        "StartDate": "20130901000000",
        "EndDate": "20130901004500",
        "Extent": {
            "XLL": -50.54,
            "YLL": -8.3,
            "XUR": -50.24,
            "YUR": -8,
            "Projection": { "EPSG": 4326}
        },
        "TimeZoneId": "UTC",
        "VariableCodes": [ "IRRATE" ],
        "Interval": 15,
        "IntervalType": "Minutes",
        "Settings": null
    },
    "Processors": [
        {
            "Code": "Statistics",
            "Settings": { "CalculationType": "Average" }
        }
    ],
    "ExportSettings": {
        "Formatting": "Indented",
        "Projection": { "ProjectionId": 3 }
    }
}
```

Figure 5.3. A sample API JSON request to calculate the average MPE rainfall rate in one hour in a specific area

Considering queries listed for benchmarking, apparently query class A, i.e. sub-selection (Table 3.2) can be directly processed by the reader. While for other queries, they are handled in the Processor (Figure 5.1). The final results are also written into JSON files. Principle of functions for querying is the same as those in SciDB connector, which is explained in the following section.

5.4 SciDB connector

SciDB, like many databases, provides a native query and management tool in Linux, iquery, with which it is possible to use AQL and AFL to manipulate data stored. SciDB also has interfaces for C++, Python and R. However, none of the above approaches is implementable on .Net platform. Through discussing with developers of SciDB

(http://www.scidb.org/forum/viewtopic.php?f=13&t=1337), there are two options to realize the SciDB connector:

1. Protocol Buffers: RPC (Remote Procedure Call) (https://developers.google.com/protocol-buffers/docs/proto#services). It is a tool from Google for serializing structured data for use in communications protocols, data storage and so on. Basically, it defines data structures for transferring in a .proto file and through protocol buffer compiler, classes to access data stored with the defined data structure in many languages including C# can be generated automatically.
2. Shim (https://github.com/Paradigm4/shim) "Shim is a super-basic SciDB client that exposes limited SciDB functionality through a simple HTTP API." Through parsing HTTP request encapsulating queries expressed in AQL and AFL, Shim then communicates with SciDB to access arrays. This approach is adopted for SciDB-py (https://github.com/Paradigm4/SciDB-py), i.e. python implementation of SciDB API. Continuous versions of Shim have been published online by SciDB team, so its stability can be guaranteed. Besides executing SciDB queries, Shim provides several other useful functionalities (Table 5.1).

Function name	Aim
execute_query	Execute a query expressed in AFL in SciDB
cancel_query	Cancel a query process in SciDB
upload	Upload a file or a binary string to the server using POST method
loadcsv	Load an uploaded CSV file into SciDB
read_bytes	Read a query result as byte array
read_lines	Read a query result as ASCII lines
stop_scidb	Stop a SciDB database
start_scidb	Start a SciDB database
measurement	Post telemetry data such as free memory of SciDB passively
getlog	Retrieve the running log of SciDB

Table 5.1. Functions provided by Shim

In theory, Protocol Buffer is more efficient because it can be directly constructed on web socket. But the implementation of Protocol Buffers in C# still needs intensive programming work. Transplanting SciDB-py to C# platform is considered, but counterparts of many libraries used by Python have to been found for C#, which is yet another time consuming work. Shim approach adds additional layers on sockets to realize the generic HTTP service. However the overhead introduced into HTTP request layer is acceptable, especially the HTTP service is built on localhost (Figure 5.1). Besides, Shim offers essential functions which facilitate the realization of SciDB connector. So Shim approach is adopted. Analogous to NetCDF connector, two classes, i.e. writer and reader are implemented in the SciDB connector.

5.4.1 Writer

The writer is to load HydroNET objects into SciDB. In this research, since MPE and GEFS datasets are initially stored in NetCDF files on Windows, the data is first read into memory object using NetCDF connector and then used as the input to start the loading process on Linux SciDB server. Totally four procedures are involved in the writer,

1. Creating arrays needed for storage of the data in SciDB.
2. Uploading data stored in byte array to SciDB server.

3. Loading uploaded data into one-dimensional load array in SciDB.
4. Redimensioning one-dimensional load array to final multidimensional array.

Creating arrays

The NetCDF header is not critical for benchmarking, so it is not loaded into a SciDB array. While the data part of NetCDF should be completely imported into SciDB arrays. Arrays for storing MPE and GEFS dataset should be set up before populating data later (Table 5.2 and 5.3).

Array name	Schema
MPE_X	<X:double> [X_id=0:3999, 4000,0]
MPE_Y	<Y:double> [Y_id=0:3999, 4000,0]
MPE_Time	<Time:int64> [T_idx=0:*,1,0]
MPE_Availability	<Availability:int8> [T_idx=0:*,1,0]
MPE_Quality	<Status:int8> [T_idx=0:*,1,0]
MPEinSciDB	<IRRATE:float> [Time=0:*,1,0,Y_idx=0:3999,4000,0,X_idx=0:3999,4000,0]

Table 5.2. Arrays created for the storage of MPE dataset

Array name	Schema
GEFS_X	<X:double> [X_idx=0:*,360,0]
GEFS_Y	<Y:double> [Y_idx=0:*,181,0]
GEFS_Modelrun	<Modelrun: int64> [M_idx=0:*,1,0]
GEFS_Ensemble	<Ensemble:string> [E_idx=0:19,20,0]
GEFS_Forecast	<Time:int64> [F_idx=0:39,40,0]
GEFS_Datastatus	<Datastatus:int64> [E_idx=0:19,20,0,F_idx=0:39,40,0]
GEFSinSciDB	<TMP:float,TMAX:float,TMIN:float,RH:float,APCP:float,TCDC:float,UGRD:float,VGRD:float>[M_idx=0:*,1,0,E_idx=0:19,1,0,F_idx=0:39,1,0,Y_idx=0:180,181,0,X_idx=0:359,360,0]

Table 5.3. Arrays created for the storage of GEFS dataset

GEFSinSciDB array is taken as an example to interpret the schema. Items listed in angle brackets are attribute names followed by their data types. "TMP" is the short name for temperature 2m above ground (Table 3.3). And dimensions are listed in square brackets and there are 5 dimensions for GEFSinSciDB. The first number of a dimension is the lower boundary of the dimension value while second number is the upper boundary value. "*" means the dimension is unlimited. The third number is the chunk size of the specific dimension while the last number is the overlap parameter (a unique feature provided by SciDB to facilitate querying). In this research, overlap feature is not investigated, so in all cases, its value is set to 0. For MPEinSciDB and GEFSinSciDB listed above, no compression is used. Later when testing compressed version of MPEinSciDB array, the schema then becomes,

<IRRATE:float COMPRESSION 'zlib'>
[Time=0:*,1,0,Y_idx=0:3999,4000,0,X_idx=0:3999,4000,0]

It should also be noted that schemes for MPEinSciDB and GEFSinSciDB listed here are not the final implementation for benchmarking. Specific data storage is presented in section 6.1.

In MPE NetCDF files, although X, Y and Time are dimensions, they are stored as coordinate variable, i.e. values of these dimensions are explicitly stored. On the other hand, dimensions in SciDB can

only allow int64 as the data type. As a result, specific dimension values should be stored in separate arrays, which is the case for GEFS storage as well. This is not a suitable design for multidimensional array database which is supposed to provide more advanced support for managing dimensions. Diverse arrays storing dimension values are related to main arrays through indexes such as X_idx and T_idx. This implementation indeed pushes SciDB into the framework of relational databases. Tracing all the released versions of SciDB, before version 13.12, the feature of non-integer dimension is still supported (http://www.scidb.org/forum/viewtopic.php?f=18&t=1172). Majorly due to efficiency problems, the feature is removed from SciDB. It can be imagined that when many non-integer dimensions are involved, management of relationships is going to be an impediment. Other attributes like Availability and Status in MPE are not stored in MPEinSciDB array along with IRRATE attribute as they are only based on the temporal dimension.

After creating essential arrays, MPEinSciDB_X, MPEinSciDB_Y, GEFSinSciDB_X, GEFSinSciDB_Y, GEFSinSciDB_Ensemble, GEFSinSciDB_Forecast and GEFSinSciDB_Datastatus are populated. This is because values for these arrays can be calculated in SciDB according to the prior knowledge of the datasets and then filled in. This on the other hand, can decrease data that need to be transferred to the SciDB server, which speeds up the whole data writing process. Other arrays are populated later.

Uploading byte array

The second step of the loading is to upload data from Windows virtual machine to Linux virtual machine. The upload module is realized through Shim's upload function using POST method. First the SciDB writer transforms the input HydroNET objects, i.e. grids read from NetCDF files into CSV memory stream. For each grid of MPE, three attributes are exported into CSV stream and they are X, Y and IRRATE (i.e. rainfall rate). When the grid of size 4000 x 4000 is exported into a CSV file, the file size becomes 808 MB, which can cause huge overhead if transferred through network. However, a unique character for MPE dataset is that approximately 98% of its values equal to 0, i.e. no precipitation. Then a substitute solution is to only export non-zero values (Figure 5.4a) and use them to overwrite the MPE array created with default value 0 at corresponding cells. By exporting non-zero values of a grid, the file size is decreased to 15 MB, 54 times smaller than previous version. For uploading, the CSV stream is written into byte array instead of files. This is because writing stream to local files has additional I/O cost, i.e. exchange with disk. In the iteration, only one grid of MPE is uploaded every time.

For GEFS, also three attributes, X, Y and value for a specific meteorological variable is exported into the CSV stream (Figure 5.4b) and then uploaded in the byte array. Since information such as temperature and humidity in GEFS is normally unequal to 0, and each grid only contains 65160 (360 x 181) cells, all the data are exported and sent.

At the server side, the byte array is received and stored as CSV file in Linux /tmp folder, which is automatically done by Shim's upload function.

```
X_idx,Y_idx,IRRATE
771,81,1.4409
772,81,1.4409
773,81,2.8784
774,81,2.8784
775,81,2.8784
776,81,3.4565
777,81,3.4565
793,81,1.4409
794,81,1.4409
795,81,1.1596
796,81,1.1596
797,81,0.7221
798,81,0.7221
```

a. MPE data with only non-zero values

```
X_idx,Y_idx,TMP
0,0,-12.25
1,0,-12.25
2,0,-12.25
3,0,-12.25
4,0,-12.25
5,0,-12.25
6,0,-12.25
7,0,-12.25
8,0,-12.25
9,0,-12.25
10,0,-12.25
11,0,-12.25
```

b. GEFS temperature data with all values

Figure 5.4. Fragments of MPE and GEFS upload file shown in ASCII

Loading CSV file into load array

Shim provides loadcsv function which calls the csv2scidb tool in SciDB's installation to import the CSV file received into one dimensional load array in SciDB. And the schema of load array (Table 5.4) is an essential parameter of loadcsv.

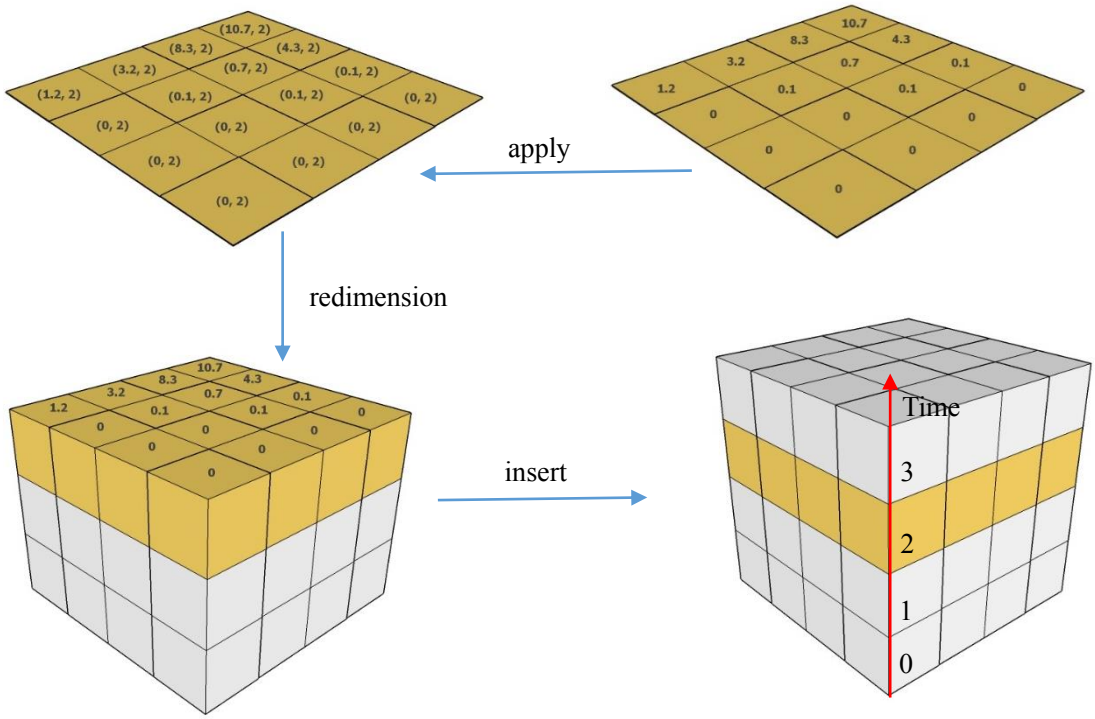
Name of load array	Schema
MPE_Flat	<X:double, Y:double, IRRATE:float>[row=0:*,1000000,0]
GEFS_Flat	<X:double, Y:double, Variable: float>[row=0:*,10000,0]

Table 5.4. Schema of one dimensional load arrays of MPE and GEFS

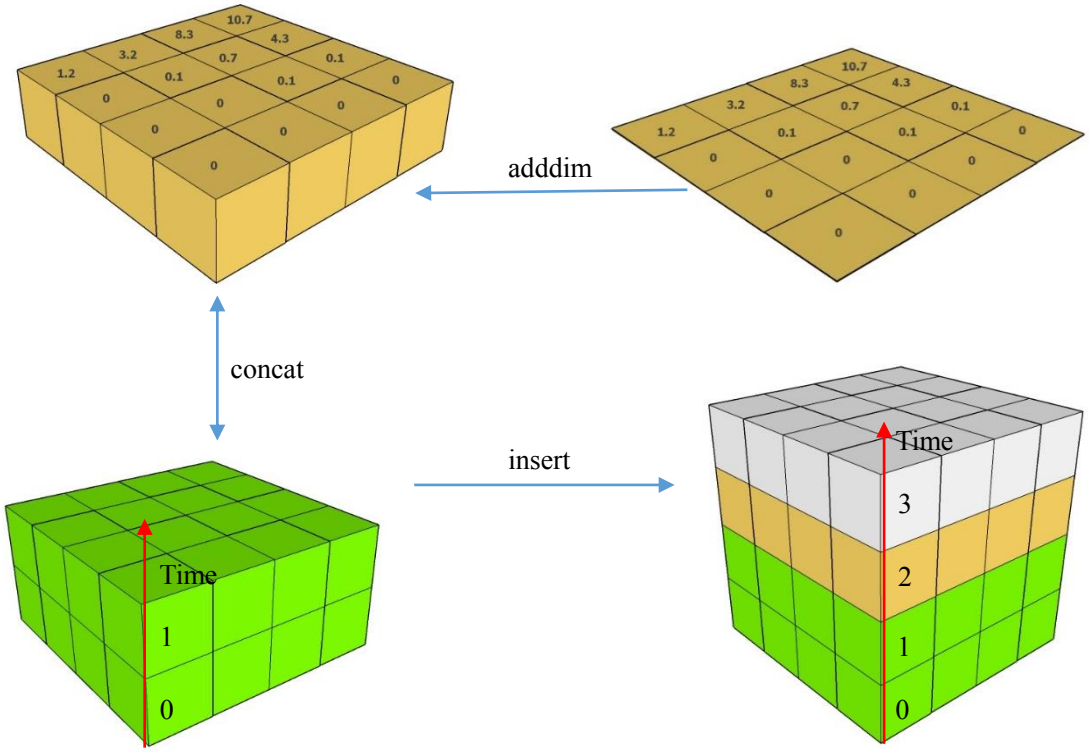
After loadcsv, the uploaded gird of MPE or GEFS is stored in the corresponding load array in SciDB.

Redimensioning load array

When inserting a gird into MPE, first a two-dimensional template array with size 4000 x 4000 is created and initiated with default value 0. Then the MPE_Flat with non-zero values in the grid is inserted into this template array. After it, the updated template array, i.e. a 2D MPE slice array is inserted into final MPE array. The normal way to perform an inversion is utilizing "redimension" operator which can transform attributes to dimensions (Figure 5.5a). However through testing, "redimension" is an expensive operation and the improvement found in the research is applying "adddim" and "concat" operators (Figure 5.5b). The latter approach is around four times faster than the previous one according to tests. But "concat" can only function when chunk size of temporal dimension equals to one. Otherwise, "redimension" is the only possibility to import grids into the three dimensional MPE array.



a. *"Redimension" approach. The grid to insert is at the third time step. First add the time step value ("apply" operator) to the 2D MPE slice array. Second, redimension the multi-attribute 2D array to a partial 3D array which then only contain values from the 2D array. Last, insert the partial 3D array to the final MPE array.*



b. "Concat" approach. First with "adddim" operator, the 2D MPE slice array is extruded to a 3D array. Second, build an empty 3D array with only 2 time steps. Third, "concat" the populated

3D array above the empty array. Last, insert the concated array to MPE, and the empty cells inserted will no overlay the original values stored.

Figure 5.5. Two approaches to insert the 2D MPE slice array containing a gird into MPE.

Since all the data including zeros are uploaded and imported into GEFS_Flat, the array can be redimensioned directly into GEFS. Since one GEFS grid includes 65160 (i.e. 181 x 360) values, yet the total amount of GEFS to load is 1.6 GB, i.e. one model run, the overload of "redimension" is not significant. The issue for importing GEFS data is that its 8 attributes are stored in each grid instead of only one attribute. This is because when reading grids from GEFS NetCDF files, every time only one variable is exported. Then the grid of one variable is uploaded and loaded into GEFS_Flat. And after exporting all grids of one variable, grids of second variable starts to be read. With SciDB 14.3, it is impossible to insert data of one variable into GEFS when there are other variables already stored. The solution adopted is to first import grids of all variables into individual 5D GEFS arrays with only one attribute each. For example, the individual array of variable TMP, i.e. temperature 2m above ground has the following schema,
<TMP:float>[M_idx=0:*,1,0,E_idx=0:19,1,0,F_idx=0:39,1,0,Y_idx=0:180,181,0,X_idx=0:359,360,0]
After all individual arrays are populated, "join" is adopted to combine all these arrays and insert the combined one into GEFS. All individual arrays are deleted in the end.

5.4.2 Reader

The reader is used to send queries to SciDB to execute and then returns the results. Although dimension values for X, Y and Time are explicitly stored, they are not referenced when executing queries. The transformation of the real longitude, latitude and time to corresponding index values of the MPE and GEFS arrays is hardcoded in the C# classes. This makes query processing procedure consistent with NetCDF library which also calculates dimension index values rather than extracting them.

Function	Aim
Spatiotemporal selection of MPE	Select subset either in spatial or temporal dimension from MPE dataset.
Spatiotemporal selection of GEFS	For GEFS, the temporal dimension refers to the forecast index, i.e. time step in the forecast series. This function selects spatiotemporal subset of GEFS for specified ensembles in one model run.
Pyramid calculation of MPE	Build pyramid, i.e. coarse grid from original MPE data at a certain pyramid level on the fly.
Spatiotemporal aggregation of MPE	Do a certain type of aggregation on the temporal dimension on a sub array from the whole MPE array.
Percentile calculation of GEFS	Calculate percentiles from all ensembles of one variable for a sub grid in a certain period for the GEFS forecast data.
Ensemble aggregation of GEFS	Perform a certain kind of aggregation on all ensembles belonging to one variable in a sub grid of the GEFS forecast dataset at multiple time steps. If the aggregation type is mean, then the result is the ensemble mean for the sub grid. If ensemble spread is need, then the standard deviation is going to be computed.

Table 5.5. List of all functionalities in the reader

Table 5.5 presents all functions included in the reader. All functionalities make use of certain operators of SciDB encapsulated in HTTP requests. Through Shim's execute_query and read_bytes functions on the SciDB server side, queries are executed and results in binary array are then returned. Below, the working principle of each of these functions are described.

1. Spatiotemporal selection of MPE

HTTP request:

```
http://URL/execute_query?id=#&query=project(between(MPE,
T_idxlower, Y_idxlower, X_idxlower, T_idxupper, Y_idxupper,
X_idxupper), IRRATE)&save=(float)
```

The "between" operator in SciDB takes the lower and upper boundary of each dimension of an array as parameters and return the corresponding sub array. "Project" is for selecting specific attributes and this is done for generic use. For other three dimensional datasets with multiple variables, "project" may be needed. The "save" command in the end is to save query result into a temporary binary file on the disk, which is retrieved by Shim later.

In addition to "between", "subarray" and "filter" operator in SciDB can also be choices for the same job. But they have different performance. And according to initial tests, in most cases, "between" and "subarray" are much faster than "filter". This is caused by underlying algorithms, "between" is an operator based on chunks. That is to say, "between" first judge whether the chunks located in the between box and then goes into cells in the chunk. Another operator "subarray" implements the same procedure as "between" while it provides another ability to build a map of all chunks that are present, which is preferred when the input array is very sparse and the subarray box contains millions of possible chunks. "Filter" on the other hand is a more generic operator, it simply checks every cell to judge if it can satisfy the requirement. However, when selection based on attribute is involved, then filter is the choice, and this capability is unavailable for "between" and "subarray".

2. Spatiotemporal selection of GEFS

HTTP request:

```
http://URL/execute_query?id=#&query=project(between(GEFS, M_idx,
E_idxlower, F_idxlower, Y_idxlower, X_idxlower, M_idx,
E_idxupper, F_idxupper, Y_idxupper, X_idxupper), attributeName)&
save=(float)
```

For GEFS, it has additional model run and ensemble dimension. The model run is set to one value, i.e. the latest model run while there are 20 different ensembles. In a request, it is possible to indicate ensembles in which range need to be returned. Besides, GEFS contains several attributes such as temperature and precipitation. There is an iteration for attributes and users can define interest attributes in the request as well. In regard to benchmark test later, APCP, i.e. total precipitation is selected, so do other GEFS queries.

3. Pyramid query of MPE

HTTP request:

```
http://URL/execute_query?id=#&query=regrid(project(between(MPE,
T_idxlower, Y_idxlower, X_idxlower, T_idxupper, Y_idxupper,
X_idxupper), IRRATE), 1, pyramidFactor, pyramidFactor,
aggregationType(IRRATE))&save=(float)
```

The input of the pyramid function is a 3D sub array while the output is also a 3D array sharing the same dimension span as the input. Since the aggregation is done on spatial dimension, so the output has coarser resolution on spatial dimensions than the input. When executing pyramid query on MPE, it is unnecessary to pre-build pyramid which can be generated on the fly. The "regrid" operator in SciDB functions for aggregation like average on top of a multidimensional block with defined sub block size (Figure 5.6).

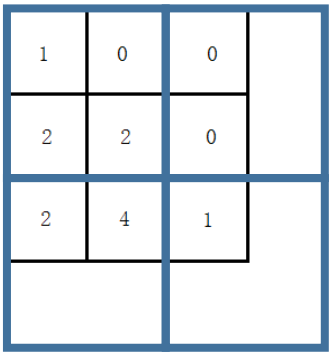


Figure 5.6. Regrid a 3 x 3 block with 2 x 2 as sub block size. If the aggregation type is average, then for pyramid built, the upper left value is 1.25, upper right value is 0, lower left is 3 and lower right value is 1. The target of "regrid" can be an array with more than 2 dimensions and more aggregation types such as maximum and minimum are available (SciDB, 2014).

The sub block size is composed by pyramid factors, so if 4 is assigned as the pyramid factor in each dimension for instance, then for a two dimensional grid, the resultant grid has 1/16 the number of cells in original grid. After "regrid" operation, the coarse blocks calculated are returned as the result. In the HTTP request, pyramid factor on temporal dimension is set to 1 while for spatial dimensions, factors are dynamic. The function is specifically for MPE array. There is no need to build pyramid for GEFS since its spatial span 360 x 181 is small, which is also the reason why pyramid is not listed in query list for GEFS.

4. Spatiotemporal aggregation of MPE

HTTP request:

```
http://URL/execute_query?id=#&query=aggregate(project(between(MPE
, T_idxlower,
 Y_idxlower, X_idxlower, T_idxupper, Y_idxupper, X_idxupper),
IRRATE),
 aggregationType(IRRATE), Y_idx, X_idx)&save=(float)
```

The input for MPE aggregation is a 3D sub array and the output is a 2D array sharing the same spatial extent as the input. So basically aggregation is performed on time series at all locations in the spatial extent. With "between" and "project" operator, a sub array can be selected with assigned span of X, Y and Time dimension. Then "aggregate" operator is used to perform average, maximum or minimum calculation on such a sub array on the temporal dimension. The "aggregate" operator is a compact version of "regrid" operator because "regrid" can support aggregation with specified length on each dimension while "aggregate" can only aggregate the whole span of certain dimensions. However, two operators share the same functional core by checking source code.

5. Percentile calculation of GEFS

HTTP request:

```
R = Percentage / 100 x SeriesLength

http://URL/execute_query?id=#&query=(select avg(attributeName) as
attributeName_lower from (select B.attributeName,
B.attributeName_rank as lower, A.attributeName_rank*2 -
B.attributeName_rank as upper from avg_rank(between(GEFS, M_idx,
0, F_idxlower, Y_idxlower, X_idxlower, M_idx, ensembleCount-1,
F_idxupper, Y_idxupper, X_idxupper), attributeName, F_idx) as A,
rank(between(GEFS, M_idx, 0, F_idxlower, Y_idxlower, X_idxlower,
M_idx, ensembleCount-1, F_idxupper, Y_idxupper, X_idxupper),
attributeName, F_idx) as B) as C where lower<=R and upper>=R
group by F_idx) &save=(float)
```

A specific spatiotemporal cell in GEFS dataset has 20 values corresponding to diverse ensembles. The percentile query is to assign a specific percentile value calculated from 20 ensembles to a cell. The input of the query is a 4D sub array (1 modelrun) and the output is a 3D sub array with unchanged spatiotemporal range as input.There are many ways to calculate percentile from a series of numbers. The method adopted by SciDB connector is from the second definition of percentile used by Lane (2013) which is implemented for percentile calculation in HydroNET-4 as well. Using the 75th percentile (i.e. the percentage is 75) as an example, the 75th percentile is the smallest value that is greater than or equal to 75% of all values in the series.

The central part of the calculation in the HTTP request lies in the ranking of values. "rank" operator in SciDB calculates the lower bound ranking of a specific value while "avg_rank" calculates average rank as the average of the upper bound and lower bound rankings. Suppose a series contains 1, 7, 7, 7 and 8 and the ranking is listed in Table 5.6.

Value	1	7	7	8	7
Ranking by "rank"	1	2	2	5	2
Ranking by "avg_rank"	1	3	3	5	3
Connector lower bound ranking	1	2	2	5	2
Connector upper bound ranking	1	4	4	5	4

Table 5.6. Rankings calculated by "rank" and "avg_rank" operator and SciDB connector on a sample series

To retrieve the 80th percentile from the sample series, the 4th smallest value has to be found. SciDB connector uses two operators to derive the lower and upper bound ranking of a specific value to finally determine that 7 is the 4th smallest value.

6. Ensemble aggregation of GEFS

HTTP request:

```
http://URL/execute_query?id=#&query=aggregate(project(between(GEF
S, M_idx, 0, F_idxlower, Y_idxlower, X_idxlower, M_idx,
ensembleCount-1, F_idxupper, Y_idxupper, X_idxupper),
attributeName), aggregationType(attributeName), F_idx, Y_idx,
X_idx)&save=(float)
```

In this query, first a 4D sub array with smaller X and Y span, one forecast step and all ensembles is extracted (As is mentioned before, the model run dimension only has one value). After it, "aggregate" operator is used to aggregate all ensemble values for the sub grid. The output is thus a 3D spatiotemporal sub array.

Results returned by above functions are read and transferred as binary arrays with read_bytes function of Shim, then they are stored in the HydroNET in-memory objects on Windows virtual machine. In the end, the data access API exports objects to JSON files.

6 Benchmark test and analysis

In fact, what are benchmarked are two databases. Basically, a database is considered to consist of two essential parts, one is data stored on disks while the other is the management layer, i.e. functionalities to manage data. For NetCDF, its management layer is NetCDF connector which is embedded in HydroNET-4. SciDB is a database originally, and Shim is the interface to communicate with SciDB which utilizes its own binary format to store data. In this research an enhanced control layer, i.e. SciDB connector is also included in HydroNET-4. That is to say, HydroNET-4 is the interface for both storage systems, which assures fair measurements for querying despite that query is processed in different places (Figure 5.1). In the following parts, details of data storage in NetCDF and SciDB are described. Then final testing results and analysis are presented.

6.1 Data storage

6.1.1 Files in 64-bit offset format

Although in the introduction chapter, it is mentioned that NetCDF files in classic format are expected to be benchmarked, in practice, 64-bit offset format is tested. For one thing, MPE and GEFS datasets are stored in 64-bit offset format originally and conversion can take much time. For another, what is actually the research interest is the contiguous storage structure of NetCDF and 64-bit offset format indeed applies this structure.

For MPE dataset, each NetCDF file contains four time steps constituting one hour, so totally 4000 x 4000 x 4 cells are included in a file. A folder contains 1722 such NetCDF files and files start recording from 23: 45 on 31st August, 2013. So if query is concerned with four time steps starting at 23: 45 on 31st August, only one NetCDF file would be read. But if data in the first hour of September were retrieved, then 2 NetCDF files would be opened. Apparently, such divergence of query response will be weakened when the time series extracted composes fairly large number of time steps.

As is indicated in Table 3.3, each GEFS file only contains one model run, 20 ensembles, 40 forecast steps, so totally 800 grids of size 360 x 181 are stored in a single file. As to benchmark test, only one file is stored in the folder and is used for query. This is because, for forecast dataset, it is always the latest forecast data that are intensively concerned. The model run time of the test sample file is 6:00 15-05-2014 and it includes the forecast for the following 10 days every 6 hours. The file size is 1.55 GB.

6.1.2 Files in NetCDF-4 format

With NetCDF connector, all NetCDF files in 64-bit offset are converted into NetCDF-4 format with

same data types and metadata. NetCDF-4 format provides more possibilities for storing data considering chunk size and compression setting. Lee, et al. (2008) showed the crucial impact of employing diverse chunk sizes and compression settings on file reading and writing rate. However, the reading in that investigation refers to reading the whole file at once and selectivity is not taken into account.

In this research, no advanced features from NetCDF-4 data model such as group are involved. In addition, the duration of this study cannot allow more chunk size experiments which is a trivial work. On the other hand, effect of chunk size has been investigated by Lee, et al. (2008). They provide a general recommendation that making chunk size exactly match the variable size can be an excellent choice for NetCDF-4 file management. The recommendation is adopted (Table 6.1). Compression of NetCDF-4 utilizes DEFLATE algorithm, a variation of LZ77 method (Ziv & Lempel, 1977).

Data store name	Chunk size (X x Y x Time)	Single file size
NetCDF4_ MPE_C2	4000 x 4000 x 1	250 MB
NetCDF4_ MPE_C2_C (compression)	4000 x 4000 x 1	3 MB

Table 6.1. Storage information for MPE NetCDF files in NetCDF-4 format

For each data store, a single file includes four time steps and the start time of each whole store is 0: 00 on 1st September, 2013, i.e. 15 minutes later than NetCDF files in 64-bit offset format. The whole dataset covers data collected for the entire September of 2013 and 2 folders are established to hold the two data stores. So every folder has 720 files in NetCDF-4 format. It should be mentioned that with SDS library, NetCDF-4 files are created with all dimensions set to unlimited. This implementation actually does not have influence on query since chunked storage structure implements identical chunk storage with unlimited and limited dimension setting.

The GEFS forecast data produced by the model on 6:00, 15th May, 2014 is stored into 4 NetCDF-4 files including 2 chunk sizes and 2 compression settings. Table 6.2 lists specific parameters for these files. Chunk size 360 x 181 x 1 x 20 x 1 is selected to keep consistency with SciDB GEFS arrays created for benchmarking. In the S5 scheme, one chunk only contains a grid, which is a decision comes from recommendations of earlier researches as well.

Data store name	Chunk size (X x Y x Forecast x Ensemble x Modelrun)	Single file size
NetCDF4_GEFS_S3	360 x 181 x 1 x 20 x 1	1.55 GB
NetCDF4_GEFS_S3_C (compression)	360 x 181 x 1 x 20 x 1	654 MB
NetCDF4_GEFS_S5	360 x 181 x 1 x 1 x 1	1.55 GB
NetCDF4_GEFS_S5_C (compression)	360 x 181 x 1 x 1 x 1	561 MB

Table 6.2. Storage information for GEFS NetCDF files in NetCDF-4 format

6.1.3 SciDB arrays

Scalability coping with increasing data is a crucial factor for evaluating data management and query systems, especially databases. Scalability of SciDB is tested with MPE data due to its large size of uncompressed storage. 5 storage levels (Table 6.3) are established. For the first three levels, i.e. tiny, small and medium, each contains 12 arrays (Table 6.4 ~ 6.6) recording the identical data but with diverse chunk size and compression settings. The array here refers to the key array MPE which is

used for query. For last two array levels (Table 6.7 ~ 6.8), due to the large size, it is inapplicable to load the identical data into 12 array versions, which takes lots of time. So only two array versions with compression and non-compression but the same chunk size are stored at each level.

Array level	MPE rainfall rate data stored	Time step count	Original size of files in 64-bit offset format
Tiny	First 2 hours of 1st September, 2013	8	488 MB
Small	First 6 hours of 1st September, 2013	24	1.3 GB
Medium	1st September, 2013	96	5.7 GB
Large	7 days from 1st to 7th September, 2013	672	40 GB
Very large	30 days of September, 2013	2880	171.6GB

Table 6.3. Five levels of MPE array in SciDB

In the following parts, storage details for arrays at different levels are presented.

MPE tiny arrays

Array name	Chunk size (Time x Y x X)	Chunk count	Average chunk storage size	Total storage size
SciDB_MPE_C1_tiny	4 x 4000 x 4000	2	20M	40.1M
SciDB_MPE_C1_C_tiny (compression)	4 x 4000 x 4000	2	5.5M	11.1M
SciDB_MPE_C2_tiny	1 x 4000 x 4000	8	5M	40.1M
SciDB_MPE_C2_C_tiny (compression)	1 x 4000 x 4000	8	1.4M	11.1M
SciDB_MPE_C3_tiny	4 x 800 x 800	50	0.8M	40.2M
SciDB_MPE_C3_C _tiny (compression)	4 x 800 x 800	50	0.2M	11.2M
SciDB_MPE_C4_tiny	1 x 800 x 800	200	0.2M	40.2M
SciDB_MPE_C4_C_tiny (compression)	1 x 800 x 800	200	56.6K	11.3M
SciDB_MPE_C5_tiny	4 x 100 x 100	3200	13K	41.6M
SciDB_MPE_C5_C_tiny (compression)	4 x 100 x 100	3200	3.7K	11.8M
SciDB_MPE_C6_tiny	1 x 100 x 100	12800	3.3K	42.3M
SciDB_MPE_C6_C_tiny (compression)	1 x 100 x 100	12800	1K	12.8M

Table 6.4. Storage information for MPE arrays with different chunk size and compression settings at the tiny level

MPE small arrays

Array name	Chunk size (Time x Y x X)	Chunk count	Average chunk storage size	Total storage size
SciDB_MPE_C1_small	4 x 4000 x 4000	6	19.3M	115.9M
SciDB_MPE_C1_C_small (compression)	4 x 4000 x 4000	6	5.3M	32.1M
SciDB_MPE_C2_small	1 x 4000 x 4000	24	4.8M	115.9M
SciDB_MPE_C2_C_small (compression)	1 x 4000 x 4000	24	1.3M	32.1M
SciDB_MPE_C3_small	4 x 800 x 800	150	0.8M	116.3M
SciDB_MPE_C3_C _small (compression)	4 x 800 x 800	150	0.22M	32.4M
SciDB_MPE_C4_small	1 x 800 x 800	600	0.2M	116.4M
SciDB_MPE_C4_C_small	1 x 800 x 800	600	55K	32.8M

(compression)				
SciDB_MPE_C5_small	4 x 100 x 100	9600	13K	120.2M
SciDB_MPE_C5_C_small (compression)	4 x 100 x 100	9600	3.6K	34.2M
SciDB_MPE_C6_small	1 x 100 x 100	38400	3.2K	122.2M
SciDB_MPE_C6_C_small (compression)	1 x 100 x 100	38400	1K	37.1M

Table 6.5. Storage information for MPE arrays with different chunk size and compression settings at the small level

MPE medium arrays

Array name	Chunk size (Time x Y x X)	Chunk count	Average chunk storage size	Total storage size
SciDB_MPE_C1_medium	4 x 4000 x 4000	24	20.4M	489M
SciDB_MPE_C1_C_medium (compression)	4 x 4000 x 4000	24	5.7M	136M
SciDB_MPE_C2_medium	1 x 4000 x 4000	96	5.1M	489M
SciDB_MPE_C2_C_medium (compression)	1 x 4000 x 4000	96	1.4M	136.2M
SciDB_MPE_C3_medium	4 x 800 x 800	600	0.82M	490.8M
SciDB_MPE_C3_C _medium (compression)	4 x 800 x 800	600	0.23M	137.1M
SciDB_MPE_C4_medium	1 x 800 x 800	2400	0.2M	491M
SciDB_MPE_C4_C_medium (compression)	1 x 800 x 800	2400	58K	139M
SciDB_MPE_C5_medium	4 x 100 x 100	38400	13.2K	506.6M
SciDB_MPE_C5_C_medium (compression)	4 x 100 x 100	38400	3.8K	145.2M
SciDB_MPE_C6_medium	1 x 100 x 100	153600	3.4K	514.7M
SciDB_MPE_C6_C_medium (compression)	1 x 100 x 100	153600	1K	157.3M

Table 6.6. Storage information for MPE arrays with different chunk size and compression settings at the medium level

MPE large arrays

Array name	Chunk size (Time x Y x X)	Chunk count	Average chunk storage size	Total storage size
SciDB_MPE_C4_large	1 x 800 x 800	16800	0.18M	2.98G
SciDB_MPE_C4_C_large (compression)	1 x 800 x 800	16800	51K	864M

Table 6.7. Storage information for MPE arrays with different compression setting at the large level

MPE very large arrays

Array name	Chunk size (Time x Y x X)	Chunk count	Average physical chunk size	Total storage size
SciDB_MPE_C4_vlarge	1 x 800 x 800	72000	0.2M	13.7G
SciDB_MPE_C4_C_vlarge (compression)	1 x 800 x 800	72000	58K	3.88G

Table 6.8. Storage information for MPE arrays with different compression setting at very large level

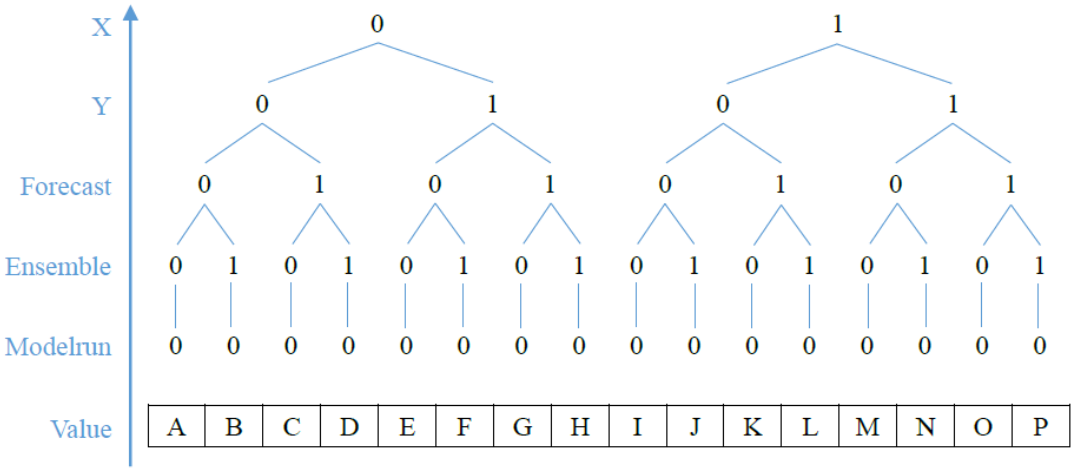
Analogous to MPE data stored in NetCDF-4 format, the starting time step of MPE dataset in SciDB is 0: 00 01-09-2013, i.e. 15 minutes later than 64-bit offset files. Compared with MPE data stored in 64-bit offset format, array storage size can be 10 times smaller from information listed in Table 6.4~6.8. This is thanks to SciDB's automatic run length encoding (RLE) for all data stored. In fact, SciDB's indexing strategy is based on RLE. If DEFLATE compression is executed on RLE encoded data, the storage size of an array experiences another decline with a factor of 4. When the array is populated with data, SciDB creates fixed-size metadata such as starting physical address for each chunk constituting an array. So arrays with small chunks utilize more space for metadata. Besides, due to RLE and 98% values of MPE equal to 0, splitting large chunk into many small chunks increases the storage size of chunks excluding metadata as well. For example, suppose a chunk of size 1000 x 1000 containing all 0 values has storage size of 1 KB (excluding per-chunk metadata). If it is split into 100 chunks with size 100 x 100, then each chunk will occupy more than 1 KB/100, i.e. 0.01 KB. So per-chunk metadata together with RLE are the reasons why Table 6.4 ~ 6.6 show with the decrease of chunk size, total storage of array grows.

MPE data are actually rasters organized along the temporal dimension and every 15 minutes, a raster is generated. Setting the chunk size of temporal dimension equal to 1 keeps consistent with data generating process. Additionally, although chunks are not implemented for 64-bit offset format yet the chunk size of temporal dimension of NetCDF-4 files equals to 1, every NetCDF file indeed elaborates 4 time steps. Consequently, at tiny, small and medium level, some arrays employ 4 as the chunk size in temporal dimension in order to keep the similarity of storage between SciDB and NetCDF at a certain level. It should be noted that, the order of dimensions used to structure data inside chunks of SciDB differs from that of NetCDF-4 files. This is caused by different specifications of storage scheme used by SciDB and NetCDF. Since chunks created for storing MPE data have 2D or nearly 2D shape, i.e. temporal dimension equal to 1 or 4, it is considered the dimensions order does not make too much difference.

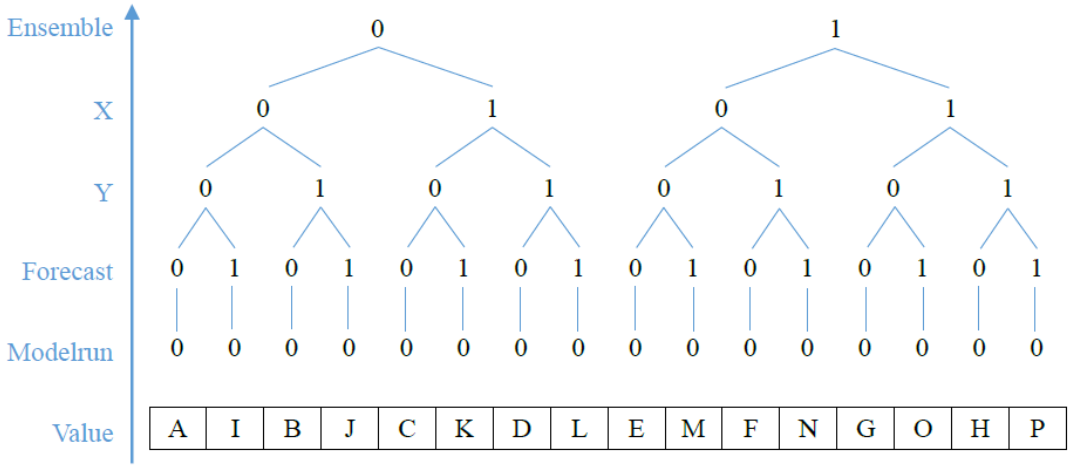
1 x 800 x 800 is settled as the chunk size for MPE array at large and very large level, which becomes the decision after initial testing of SciDB. Basically, as will be shown in section 6.2.1, the performance of time series extraction degrades a lot if large chunk size such as 4 x 4000 x 4000 or 1 x 4000 x 4000 is adopted above tiny level. On the other hand, small chunks present the best scalability with time series extraction, however they can cause huge overload in memory. As a matter of fact, the first time to load MPE data at very large scale, 1 x 100 x 100 is put into practice. After loading 756 time steps, the memory usage of SciDB after restarting increases 1.3 GB. This is due to the fact that SciDB loads in-memory metadata of every array after starting up. Such metadata are tuples as <InstanceID, ArrayID, ChunkID, VersionID> used to locate chunks from the data file on the disk and large amount of chunks can make the metadata in memory bloat. So chunk size 1 x 800 x 800 is actually a compromise.

The unique point of GEFS dataset is its high dimensionality. It is the fact that the order of dimensions to organize data determines how data are stored on the disk. Figure 6.1 presents storage of a sample dataset with 1 modelrun, 2 ensembles, 2 forecast steps, 2 Y and X values. The modification of the order of dimensions to structure data totally changes data storage on the disk, which might have

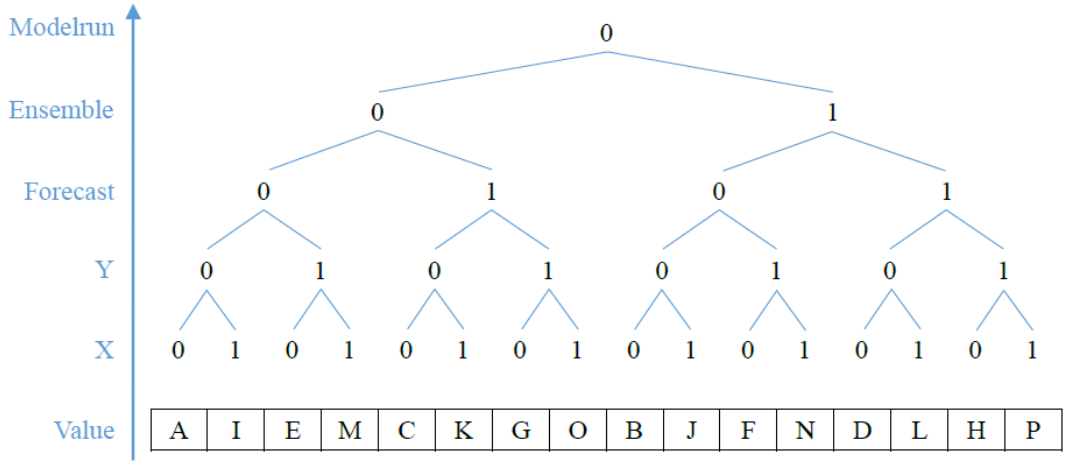
significant influence on query performance.



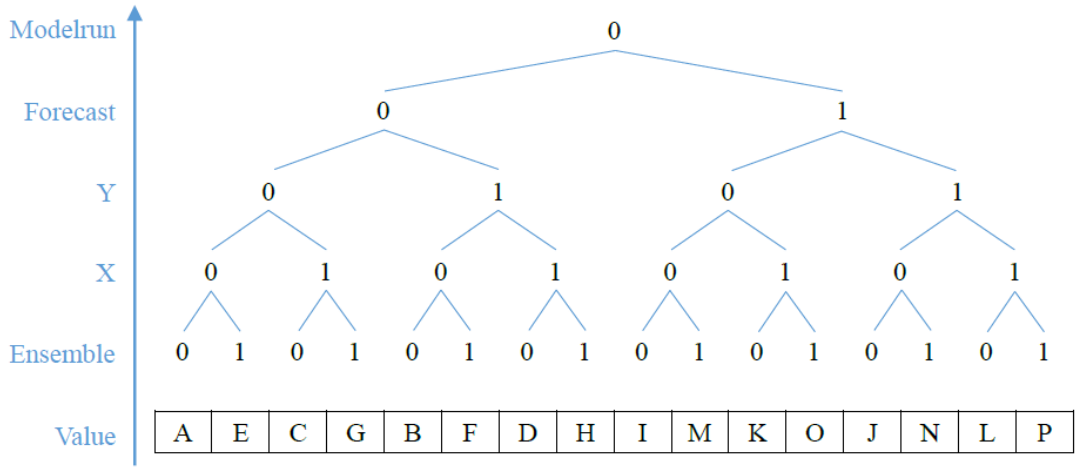
a. Data storage with dimension order Modelrun (M), Ensemble (E), Forecast (F), Y, X



b. Data storage with dimension order M F Y X E



c. Data storage with dimension order X Y F E M



d. Data storage with dimension order E X Y F M

Figure 6.1. The storage of a sample dataset conforming to 4 GEFS schemas.

In SciDB, several schemes (Table 6.9) are implemented to store GEFS dataset for benchmarking.

In Table 6.9, S1, S2, S3 and S4 represent 4 data schemes, i.e. structures for GEFS storage caused by modifying the order of dimensions. S1, S2, S3 and S4 conform to the storage structures shown in Figure 6.1a, b, c and d respectively. The chunk sizes of these 4 arrays keep at the same modest level, i.e. 1 value in the model run and forecast dimension, 20 values in the ensemble dimension, 181 values in the Y dimension and 360 values in the X dimension. To fully investigate whether data organization inside the chunk has an influence on query performance, GEFS dataset is additionally stored into one chunk but with 4 different dimension orders, i.e. C1, C2, C3 and C4. The compression again refers to DEFLATE method. All arrays store the same 8 attributes (Table 5.3). Although model run dimension ranges from 0 to 675, the array only contains the 675th model run step.

Array name	Dimension part in the schema	Chunk size	Chunk count	Average chunk storage size	Total storage size
SciDB_GEFS_S1	[M_idx=0:675,1,0,E_idx=0:19,20,0,F_idx=0:39,1,0,Y_idx=0:180, 181,0,X_idx=0:359,360,0]	1 x 20 x 1 x 181 x 360	40	6.7 MB	268 MB
SciDB_GEFS_S1_C (Compression)	[M_idx=0:675,1,0,E_idx=0:19,20,0,F_idx=0:39,1,0,Y_idx=0:180, 181,0,X_idx=0:359,360,0]	1 x 20 x 1 x 181 x 360	40	2.16 MB	86.4 MB
SciDB_GEFS_S2	[M_idx=0:675,1,0,F_idx=0:39,1,0,Y_idx=0:180,181,0,X_idx=0:359, 360,0,E_idx=0:19,20,0]	1 x 1 x 181 x 360 x 20	40	6.2 MB	247 MB
SciDB_GEFS_S2_C (Compression)	[M_idx=0:675,1,0,F_idx=0:39,1,0,Y_idx=0:180,181,0,X_idx=0:359, 360,0,E_idx=0:19,20,0]	1 x 1 x 181 x 360 x 20	40	2 MB	79 MB
SciDB_GEFS_S3	[X_idx=0:359,360,0,Y_idx=0:180,181,0,F_idx=0:39,1,0,E_idx=0:19, 20,0,M_idx=0:675,1,0]	360 x 181 x 1 x 20 x 1	40	6.2 MB	246.5 MB
SciDB_GEFS_S4	[E_idx=0:19,20,0,X_idx=0:359,360,0,Y_idx=0:180,181,0,F_idx= 0:39,40,0,M_idx=0:675,1,0]	20 x 360 x 181 x 1 x 1	40	5.8 MB	232.4 MB
SciDB_GEFS_C1	[M_idx=0:675,1,0,E_idx=0:19,20,0,F_idx=0:39,40,0,Y_idx=0:180, 181,0,X_idx=0:359,360,0]	1 x 20 x 40 x 181 x 360	1	268 MB	268 MB
SciDB_GEFS_C2	[M_idx=0:675,1,0,F_idx=0:39,40,0,Y_idx=0:180,181,0,X_idx=0:359 ,360,0,E_idx=0:19,20,0]	1 x 40 x 181 x 360 x 20	1	246.7 MB	246.7 MB
SciDB_GEFS_C3	[X_idx=0:359,360,0,Y_idx=0:180,181,0,F_idx=0:39,40,0,E_idx= 0:19,20,0,M_idx=0:675,1,0]	360 x 181 x 40 x 20 x 1	1	246.5 MB	246.5 MB
SciDB_GEFS_C4	[E_idx=0:19,20,0,X_idx=0:359,360,0,Y_idx=0:180,181,0,F_idx= 0:39,40,0,M_idx=0:675,1,0]	20 x 360 x 181 x 40 x 1	1	250 MB	250 MB

Table 6.9. Storage information for GEFS arrays with different data scheme and compression settings.

6.2 Query benchmarking

To start benchmarking, the configuration file of SciDB 14.3 has to be edited. This includes the setting of some crucial parameters such as maximum amount of memory that can be occupied by SciDB and the SciDB instance number. The SciDB instance refers to an independent SciDB group of processes. For benchmarking, 3 GB is set as the maximum memory usage while only one instance is set up. Appendix C provides the whole configuration file.

For every query executed, its specific record is profiled into the log generated by HydroNET-4. The time spent on executing the corresponding function such like spatiotemporal selection of MPE in SciDB connector can then be extracted from the log. For a specific data store, i.e. NetCDF files or SciDB array, a query is executed 20 times discontinuously. Suppose there are 4 data stores, D1, D2 D3 and D4, to execute the query Q 20 times is to perform the program below,

```
For (int i = 0; i < 20; i++ )
  {
      Query(D1);
      Query(D2);
      Query(D3);
      Query(D4);
  }
```

There is no disruption such as reboot system in between. The final result of a query response time is the average of the middle 12 records with largest 4 and smallest 4 records removed for each data store. It should be noted that the query response time of SciDB arrays also includes HTTP communication between two virtual machines and query parsing in the Shim. To measure them, an empty array is created in SciDB, and a testing function is developed in SciDB connector which sends the request,

http://URL/execute_query?id=#&query=consume(emptyArray)

The query is to scan all data stored in the empty array. By executing the empty query 20 times, the distribution of response time is depicted in a Whisker chart (Table 6.3).

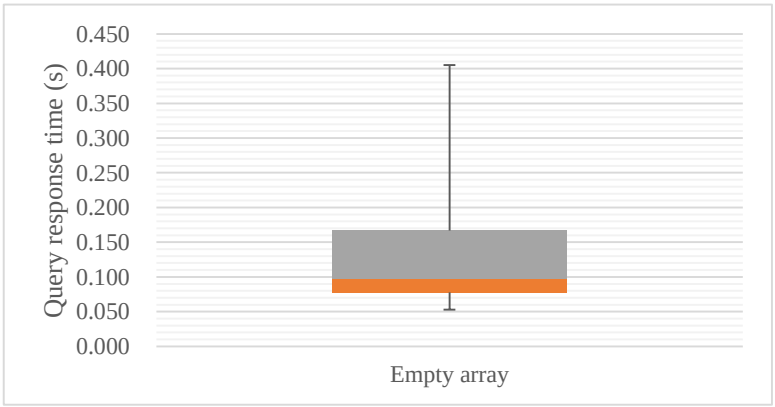


Figure 6.2. Distribution of the empty query response time for SciDB. Horizontal line breaks are maximum, 75th percentile, median, 25th percentile and minimum.

Figure 6.2 indicates that most empty query time (Q_e) measurements are between 0.05 s to 0.2 s.

Compared with a normal query like selecting a sub grid at a certain time step from MPE array, the empty query takes much less processing time in SciDB. It also has no cost on storing query result on binary file disk, reading the binary file and sending query result to Windows virtual machine. So by subtracting Q_e from a normal query response time, the database processing time, writing and reading time from disk, result transferring time through HTTP can be acquired. The largest query result is from aggregation function of MPE dataset at the Netherlands scale and totally 11868 float values are returned. Since Shim stores it as float array in the binary file, the file size is thus about 46 KB. The typical hard drive transferring speed is 147 MB/s. So the total reading and writing time cost is below 0.001 s. Besides, it is tested that with current benchmark architecture, HTTP communication between two virtual machines does not go into external network, but instead, internal stacks. So the result transfer takes place in memory, which should take little time. So the major part from the subtraction is the database processing time.

On the NetCDF side, a normal query time measured only refers to data processing, e.g. read sub grid. Consequently, in principle, by subtracting Q_e from SciDB measurements, the time cost of a query for NetCDF and SciDB are comparable. However, since Q_e also fluctuates and as later shown, a normal query response time can be below 0.1 s, so Q_e can only be used as a reference for discussion.

Due to large amount of benchmark tests, all query benchmarking figures are put into Appendix G.

6.2.1 Query performance on MPE dataset

Figure 6.3 shows locations used in queries on MPE dataset.

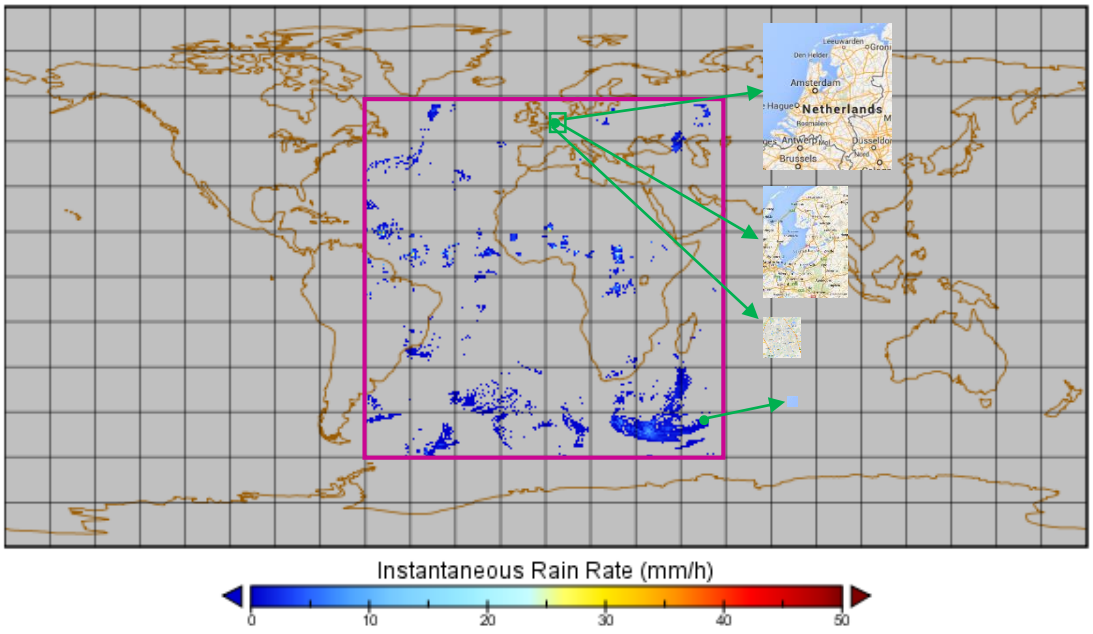


Figure 6.3. MPE rainfall rate map for the one third of the world (purple box) at 01:00 01-09-13. Four green objects refer to query areas, i.e. from top down, the Netherlands, northern part of the Netherlands, Delft (a rectangle area) and a spot location in the Indian Ocean.

Sub grid selection

With this query, two different sub grids are selected. The first sub grid covers Delft, a town in the Netherlands, and it corresponds to 12, i.e. 4 x 3 cells in MPE dataset. The second sub grid covers northern part of the Netherlands containing 2880, i.e. 60 x 48 cells. For both sub grid queries, the scalability of NetCDF files in 64-bit offset format and NetCDF-4 format, and SciDB arrays is tested.

For all levels, Delft sub grid returns all zero values, i.e. no rainfall at that time. While except tiny, small and large level for which the query results only contain zero values, selecting sub grid of northern part of the Netherlands presents 20% and 25% non-zero values at medium and very large scale respectively.

Figure 6.4 presents the average performance for retrieving the sub grid covering the northern part of the Netherlands. On the whole, NetCDF-4 without compression shows its superiority over other solutions. NetCDF 64-bit offset option ranks the second place and it is then followed by various SciDB arrays with different chunk sizes and compression settings. However, the additional noise such as HTTP communication included in SciDB measurements should not be underestimated. Considering the range of noise shown in Figure 6.2, SciDB arrays can probably achieve the same performance as 64-bit offset store especially the arrays with small chunk sizes. But NetCDF-4 store will still be the fastest solution. In addition, the effect of chunk size in temporal dimension is not significant comparing SciDB_MPE_C1 and SciDB_MPE_C2 or SciDB_MPE_C3 and SciDB_MPE_C4, etc. Most likely, it is because the chunk size gap for temporal dimension, i.e. 1 and 4 is insufficient to observe performance variance. When DEFLATE compression is applied on NetCDF-4 storage, the query performance degrades severely and it takes around 40 times longer to extract the sub grid than normal NetCDF-4 files. This is mainly caused by decompressing process in memory. While the compression of SciDB does not have significant influence because as is mentioned, SciDB builds index on RLE encoded data and the compressing ratio introduced by DEFLAET, i.e. a secondary compression is not high.

The figure clearly presents that for all data solutions, query processing scales well, and that is performance keeps at a constant level as data size gets larger. For NetCDF files, the reason lies in the B-tree index built by NTFS file system. Basically, the whole MPE dataset is split into separate files each of which contains 4 time steps. All the files are arranged into one folder and the NTFS file system then builds B-tree index for all the files. So if a grid at a certain time step is queried, through the B-tree, HydroNET-4 API locates the corresponding file and then retrieves the grid. As to SciDB arrays, the fine stability is thank to the metadata structure loaded in the memory which functions as index for all chunks belonging to an array. Through ArrayID and ChunkID in the metadata, SciDB retrieve right chunk and then extract grid requested.

Retrieving Delft sub grid takes less time than the grid of northern part of the Netherlands on the whole (Appendix G). However, the performance of 64-bit offset is an exception and it takes more time to retrieve Delft sub grid. This is probably owing to the unstable performance of HydroNetCDF library as is mentioned in Chapter 5. Basically, the library is copied from HydroNET-3 system and integrated into HydroNET-4 without further polishing.

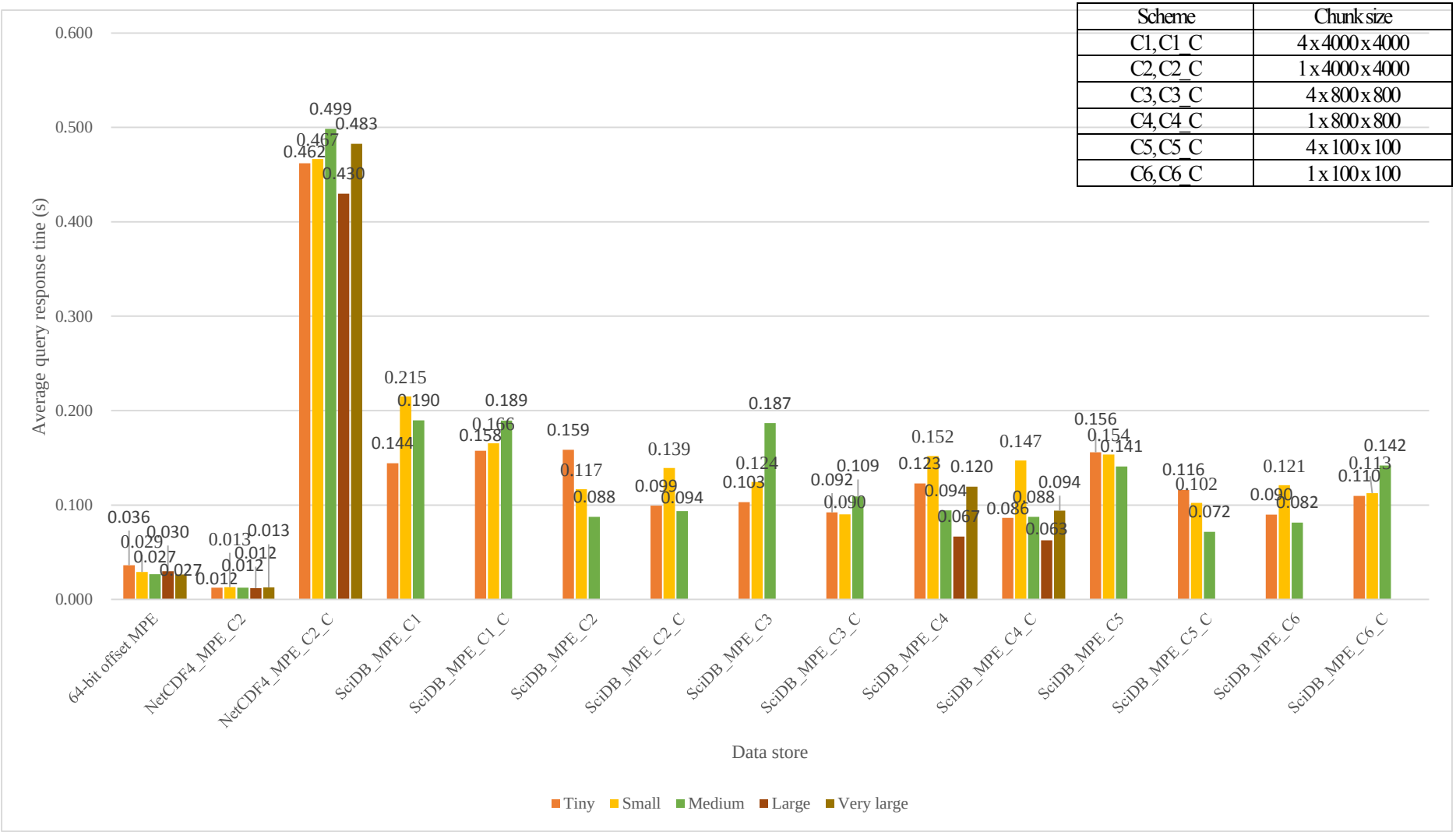


Figure 6.4. Performance of diverse data solutions for retrieving the grid covering northern part of the Netherlands at one time step.

Time series extraction

As is mentioned in the problem statement, time series extraction is the shortcoming of NetCDF's contiguous storage structure. To get more details, three test groups are developed. For all groups, the location used for query is a point, i.e. one cell located in the Indian Ocean (Figure 6.3). The spot location of Delft is not used (Section 3.2) because in September of 2013, Delft has very little rainfall. On the one hand, hydrologists are more interested in precipitation. On the other hand, SciDB utilizes RLE for storage which implies all zero values incorporate much compression and this may then have specific influence on query performance. The single location in the Indian Ocean however experiences more rainfall, i.e. around 4 hours precipitation on average every day in September, 2013.

The first test is to retrieve a time series consisting of 8 time steps from 64-bit offset files, NetCDF-4 files and SciDB arrays at five levels. At the tiny and very large levels, values retrieved are all positive numbers while the result at the small level has 63% zero values. There are half zero values included in the query result at both medium and large levels. Specific benchmark graphs can be checked in Appendix G. To conclude, all data solutions scale well like sub grid selection. NetCDF-4 without compression have the best overall performance while 64-bit offset solution is around 6 times slower. Taking account of noise enrolled in the SciDB measurements, SciDB arrays can achieve the same level as 64-bit offset option. SciDB arrays with smaller chunk sizes present superiority over arrays with large chunk sizes. When chunk size is below 4 x 800 x 800, query performance for SciDB arrays does not show obvious gap. Once again the DEFLATE compression of NetCDF-4 causes huge negative effect, making the query executed 200 times longer than normal NetCDF-4 solution. But the compression effect is insignificant for SciDB arrays.

The second test group is to extract time series of various lengths from NetCDF and SciDB arrays at the medium level. Table 6.10 lists basic configuration for this test,

Length of time series, i.e. time step count	Data store tested	Start time step	End time step
8	64-bit offset, NetCDF4_C2, NetCDF4_C2_C, MPE medium arrays	2013-09-01T16:00:00	2013-09-01T18:00:00
24	64-bit offset, NetCDF4_C2, NetCDF4_C2_C, MPE medium arrays	2013-09-01T00:30:00	2013-09-01T06:30:00
96	64-bit offset, NetCDF4_C2, NetCDF4_C2_C, MPE medium arrays	2013-09-01T00:00:00	2013-09-02T00:00:00

Table 6.10. Configuration of second test of time series extraction

The 8-step time series extracted has 50% values equal to 0. The ratios of zero values in the query results increase to 63% and 67% for the 24-step and 96-step time series queries respectively. Figure 6.5 shows the average query response time for the second group of time series extraction. Due to large values of NetCDF-4 with compression, corresponding bars are not visualized. In this test, query ranges, i.e. time steps the query concerned are different and it is natural that it takes more time to select a time series with larger length. To better understand scalability of solutions, the query response time corresponding to 24-step and 96-step query are divided by 3 and 12 respectively to get the average time to retrieve time series elaborating 8 time steps (Figure 6.6).

Figure 6.6 indicates that NetCDF-4 without compression and SciDB arrays with chunk size C5 and

C6 are the fastest solutions for extracting time series at the medium level. SciDB arrays with large chunk size but no compression and 64-bit offset store are located at the same performance level. The negative effect of compression on SciDB arrays with chunk size C1 and C2 is significant. While for small chunk sizes, compression does not present negative impact. Such a pattern implies that effect of DEFLATE compression in SciDB has correlation with chunk size and most likely, with the decrease of chunk size, negative effect of compression declines as well. The DEFLATE compression of NetCDF-4 on the other hand still causes severe degradation of query performance. The figure also shows that MPE_C1 and MPE_C2 are slower than other SciDB arrays, which can imply that indexing chunks works more efficient than indexing values inside a chunk for SciDB. This is derived because for array with large chunk size, SciDB seeks the sub grid from one chunk while if chunk size is small, first relevant chunks should be located and then internal index work to find target cells. Figure 6.6 clearly indicates that NetCDF-4 and SciDB arrays with small chunk sizes have favorable scalability. That is, as data searched increases, the time to extract a time series consisting of 8 steps on average reduces. For other data stores, the scalability keeps at a constant level.

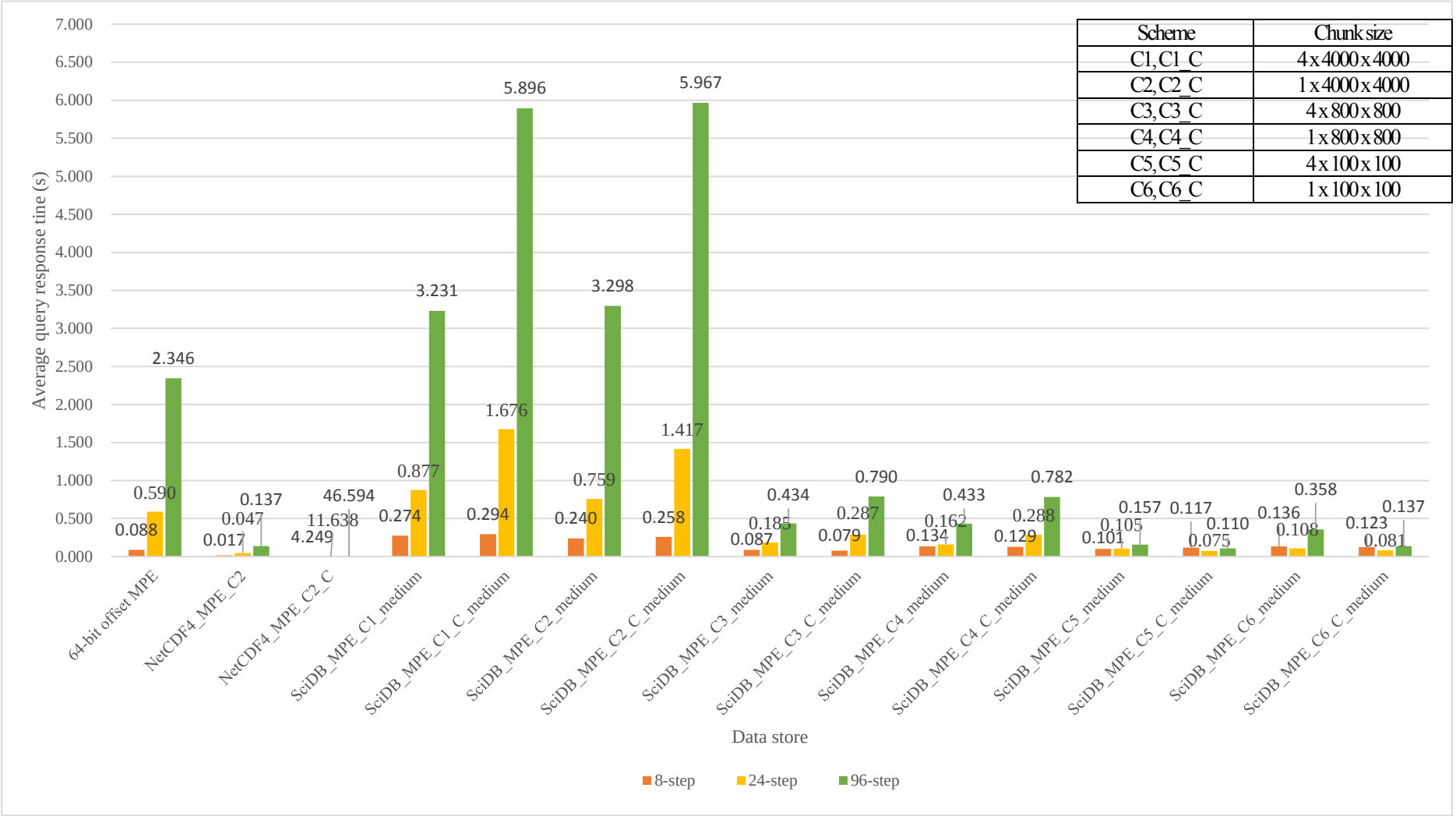


Figure 6.5. Performance of diverse data solutions at medium level for retrieving time series of different lengths from a single location in the Indian Ocean.

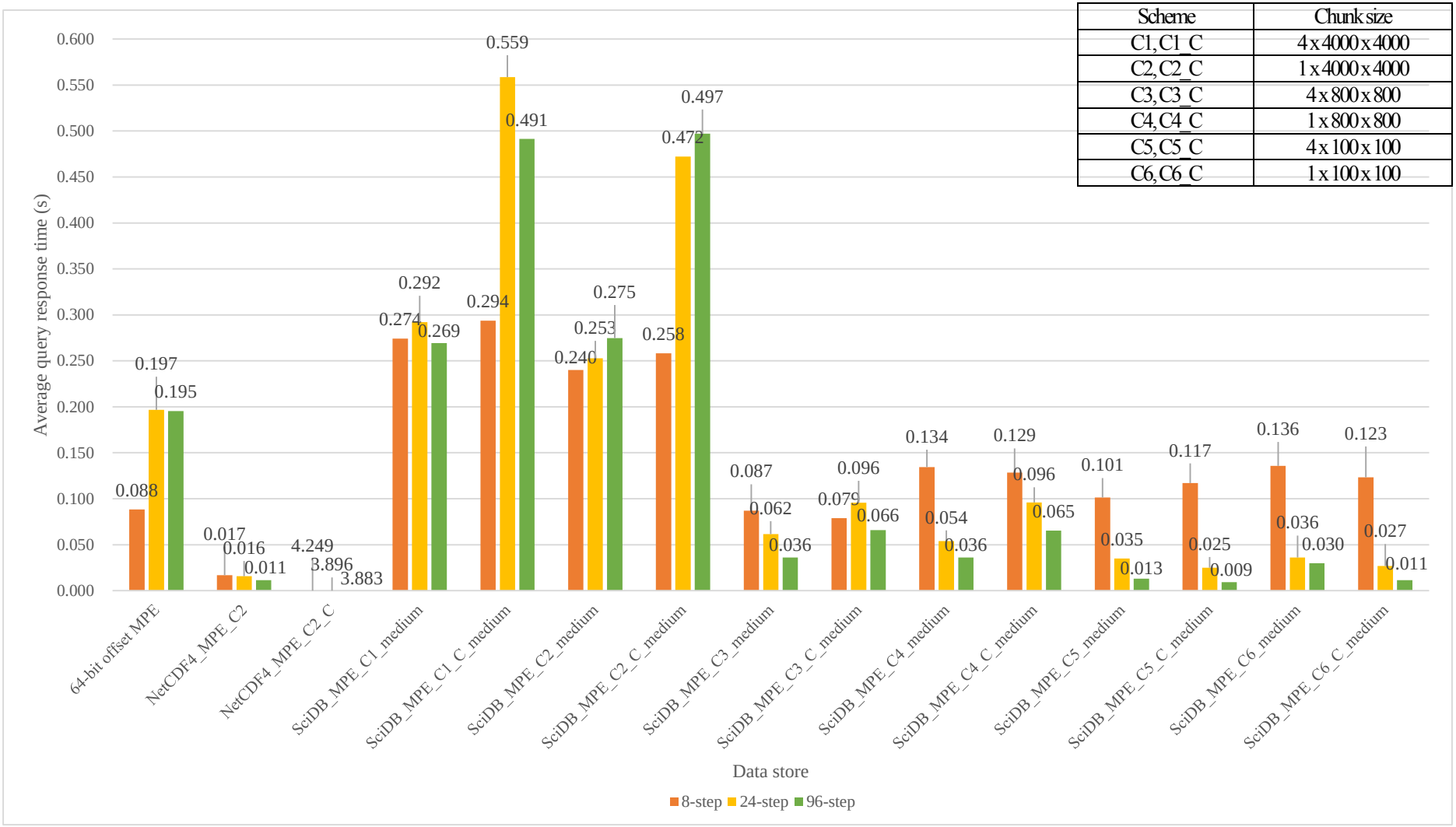


Figure 6.6. Aligned performance of diverse data solutions at medium level for retrieving time series of different lengths from a single location in the Indian Ocean.

The third group focuses on large dataset and extraction of time series of different lengths is further explored at the very large level. Parameters of the test is listed in Table 6.11.

Length of time series, i.e. time step count	Data store tested	Start time step	End time step
8	64-bit offset, NetCDF4_C2, MPE very large arrays	2013-09-27T10:15:00	2013-09-27T12:15:00
24	64-bit offset, NetCDF4_C2, MPE very large arrays	2013-09-07T13:30:00	2013-09-07T19:30:00
96	64-bit offset, NetCDF4_C2, MPE very large arrays	2013-09-21T05:00:00	2013-09-22T05:00:00
672	64-bit offset, NetCDF4_C2, MPE very large arrays	2013-09-07T00:00:00	2013-09-14T00:00:00
2880	64-bit offset, NetCDF4_C2, MPE very large arrays	2013-09-01T00:00:00	2013-10-01T00:00:00

Table 6.11. Configuration of third test of time series extraction at very large level

The 8-step query returns all positive values while there are 46 percent zero values in the 24-step query result. For the time series extraction with last three lengths, the ratios of zero in the query result are all around 85%. As the previous test group, to clearly show the scalability of diverse solutions, query response time is aligned to 8-step case (Figure 6.7). Through aligning, i.e. averaging process, the noise for SciDB measurements becomes very small for the 672-step and 2880 step cases and can thus be omitted.

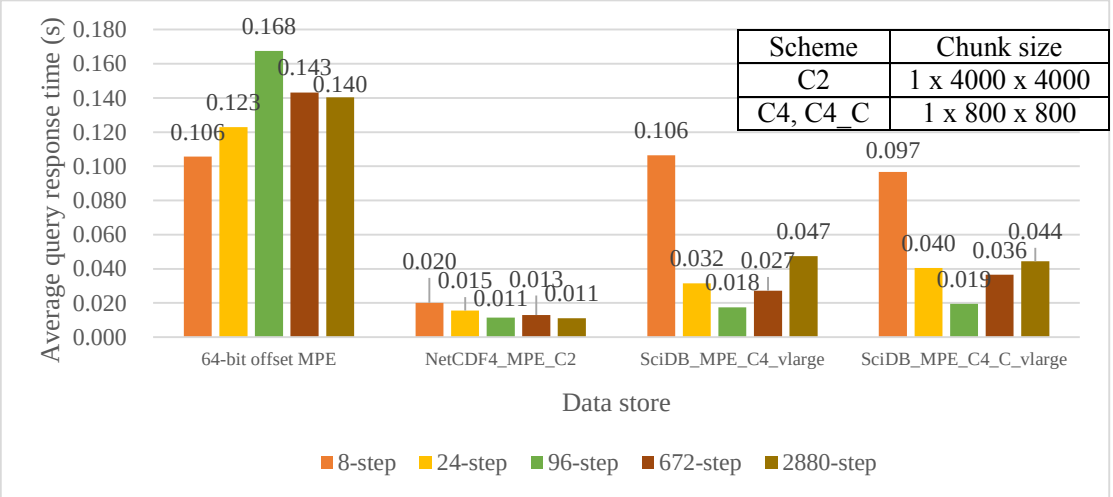
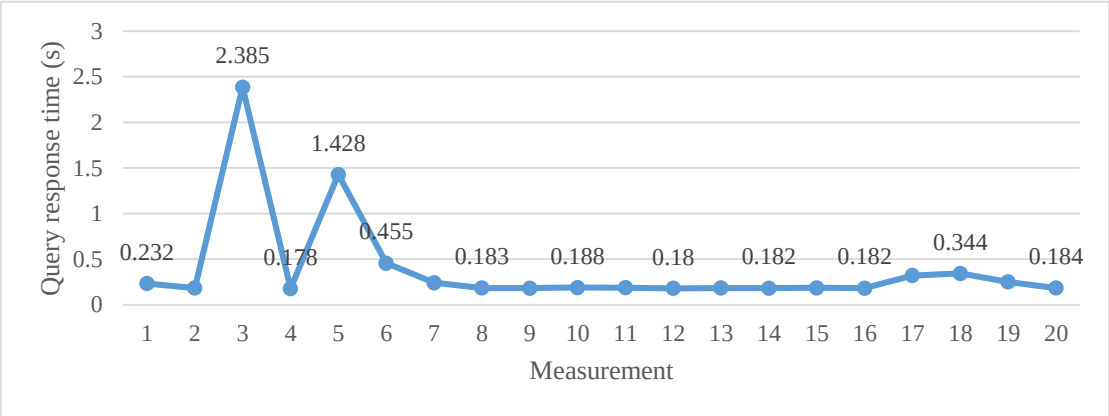


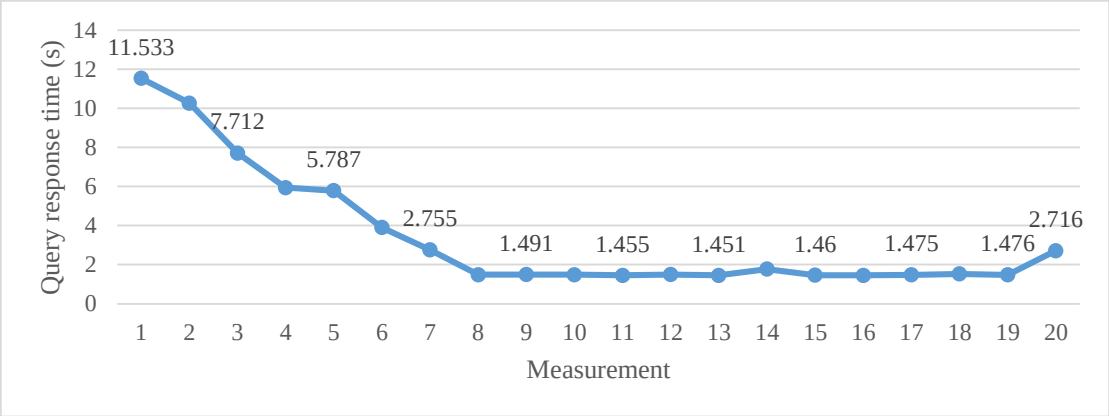
Figure 6.7. Aligned performance of diverse data solutions at very large level for retrieving time series of different lengths from a single location in the Indian Ocean.

Due to large query response time of compressed NetCDF-4 solution in initial tests, it is excluded from benchmarking. According to the figure, NetCDF-4 solution keeps the best performance. And extracting time series is very expensive from NetCDF 64-bit offset files, which is the original problem for this research. 64-bit offset store is nearly 10 times slower than NetCDF-4 for time series extraction at the very large level. The NetCDF-4 option once again presents the pattern that the time on average to extract time series decreases when more data is searched. While the 96-step seems to undermine such a judgment. By doing an additional 5658-step time series extraction on MPE data of two months, for which the average query response time is smaller than the 2880-step case, previous conclusion is verified. From the raw test records, it is found that the NetCDF-4's favorable

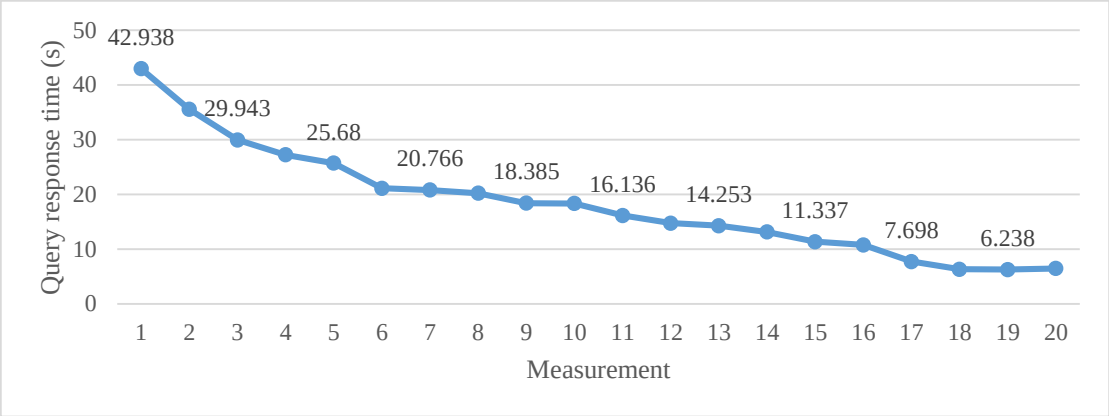
scalability is due to Windows caching mechanism since after the first query execution, response time decreases to a certain level dramatically. Besides, the DEFLATE compression on SciDB array does not bring significant effect on query performance.



a. Extracting 96-step time series from SciDB_MPE_C4_vlarge 20 times



b. Extracting 672-step time series from SciDB_MPE_C4_vlarge 20 times



c. Extracting 2880-step time series from SciDB_MPE_C4_vlarge 20 times

Figure 6.8. Raw query response time measurements of three time series extraction on SciDB_MPE_C4_vlarge array.

The 96-step query turns out to be a trough for 2 SciDB arrays in Figure 6.7. To explore the reason, original measurement records are checked (Figure 6.8).

Figure 6.8b and 6.8c indicates that the average query response time calculated by removing largest 4 and smallest 4 values still takes some large "outliers" into account. For instance, in Figure 6.8 b, the 5th, 6th, 7th and 20th measurements are elaborated for averaging as well. The initial idea to remove extreme values is to keep the remaining measurements at a constant level such as the 0.18 s level in Figure 6.8 a. However, the calculation fails to function on extracting time series with large lengths where query response time declines gradually. The same benchmark test is executed using script on SciDB server to exclude possible noise, but analogous pattern is observed. In essence, this is a result from caching process of SciDB. During the whole process for executing the query 20 times on a SciDB array, for example, SciDB_MPE_C4_vlarge, relevant data such as chunks are cached into memory gradually instead of all buffered into memory after executing the same query 2 or 3 times.

If the stable levels, i.e. 1.47 s and 6.3 s are used as the average execution time of 672-step and 2880-step time series extraction on SciDB_MPE_C4_vlarge, and analogous operation is applied to SciDB_MPE_C4_C_vlarge, then Figure 6.7 is modified to Figure 6.9 in which the scalability of SciDB arrays presents the pattern as is expected to some extent.

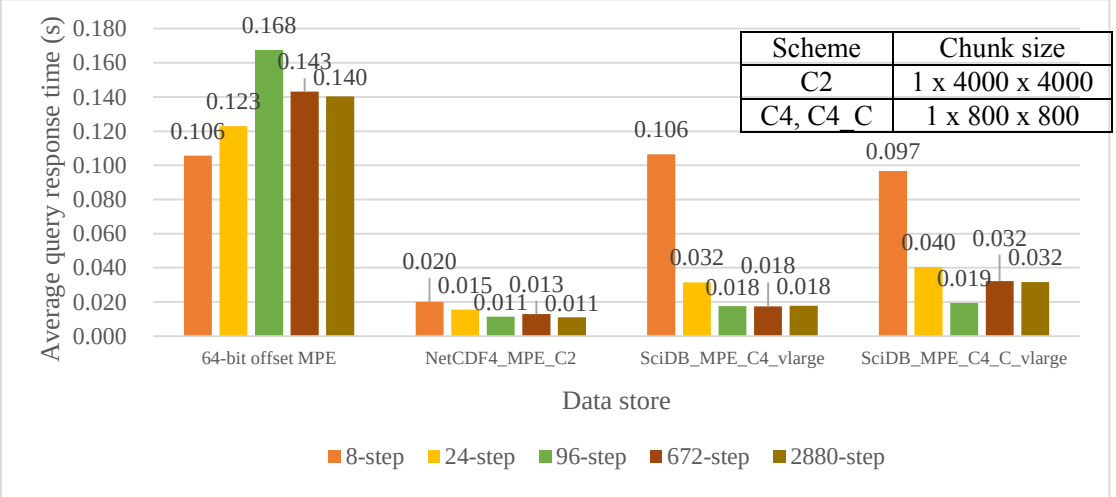


Figure 6.9. Aligned performance of diverse data solutions at very large level for retrieving time series of different lengths from a single location in the Indian Ocean with modification of SciDB measurements

Pyramid query

Pyramid query is not tested since this is an unfair comparison. SciDB does not provide functions to build pyramid. With "aggregate" operator of SciDB, it is possible to generate pyramids on the fly. While pyramids are built and stored along with creations of NetCDF files in HydroNET-4, and query is then processed on the pre-stored data. It is possible to pre-store pyramids in SciDB but in this case, pyramid query becomes grid sub selection of another dataset, i.e. MPE data in curse resolution. This has no added value to the research. As a matter of fact, it is more worthwhile to compare querying pyramids and querying its original dataset, which is not the focus of this study.

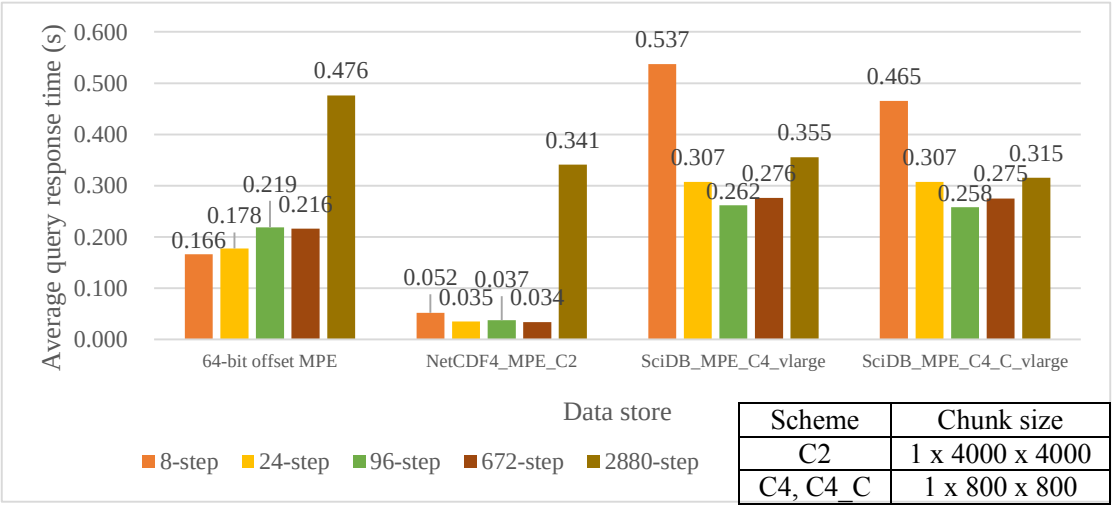
Average calculation

For average calculation of MPE dataset, focus is put on data at very large level. The input for this query is a 3D spatiotemporal cube and the output is a 2D array sharing the same spatial extent as the input. So basically average is performed on time series at all locations inside the spatial range. In this benchmarking, spatial extent is the whole Netherlands covering 11,868 cells (129 x 92). The lengths on temporal dimension of the input cube vary from 8 to 2880. Specific parameters for testing is listed in Table 6.12.

Number of time steps for averaging	Data store tested	Start time step	End time step
8	64-bit offset, NetCDF4_C2, MPE very large arrays	2013-09-27T10:15:00	2013-09-27T12:15:00
24	64-bit offset, NetCDF4_C2, MPE very large arrays	2013-09-07T13:30:00	2013-09-07T19:30:00
96	64-bit offset, NetCDF4_C2, MPE very large arrays	2013-09-21T05:00:00	2013-09-22T05:00:00
672	64-bit offset, NetCDF4_C2, MPE very large arrays	2013-09-07T00:00:00	2013-09-14T00:00:00
2880	64-bit offset, NetCDF4_C2, MPE very large arrays	2013-09-01T00:00:00	2013-10-01T00:00:00

Table 6.12. Configuration of average calculation on MPE data at very large level

The first query, i.e. average over 8 time steps return all 0 values. Averaging 96 time steps leads to 99% 0 values in the result. While the queries concerning 24, 672 and 2880 time steps return all positive values. Query result is aligned into 8-step case, i.e. response time of queries concerned with more than 8 time steps is divided by certain factors to get the time on average to do aggregation on 8 time steps (Figure 6.10). Through such an aligning process, the noise involved in SciDB measurements for averaging 672 and 2880 time steps become very small and can be neglected.



Scheme	Chunk size
C2	1 x 4000 x 4000
C4, C4_C	1 x 800 x 800

Figure 6.10. Aligned performance of diverse data solutions at very large level for average calculation with different time steps at the Netherlands scale.

Figure 6.10 shows that for average calculation at the very large scale, NetCDF-4 holds the best performance on the whole. 64-bit offset file solution comes after while SciDB arrays take more time to finish. Performance of the normal SciDB array and its compressed version do not vary much.

The execution of average calculation is composed of two phases for NetCDF file solutions. First phase is to extract the related spatiotemporal sub cube from the whole MPE data store and this takes place in NetCDF connector. Then aggregation is performed on selected data in the Processor (Figure 5.1). HydroNET-4 log records time measurements for both phases and it is found that the aggregation process is several hundred times faster than sub selection. But the combination of operator "aggregate" and "between" (Section 5.4.2) for average calculation in SciDB results in much more overhead than only sub selection. Figure 6.11 shows performance of only selecting gird covering the Netherlands in 2880 time steps at very large scale. The query response time is divided by 360 to align to 8-step case.

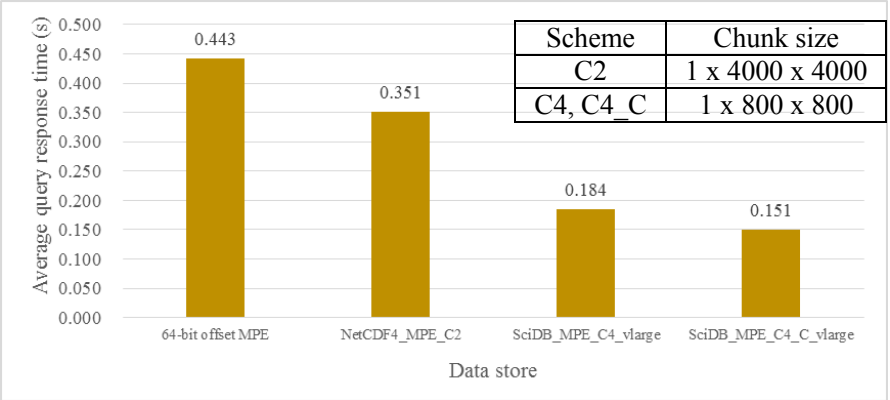


Figure 6.11. Aligned performance of diverse data solutions at very large level for extracting Netherlands grids in 2880 time steps

By comparing performance of SciDB arrays in Figure 6.10 and 6.10, it can be seen that an additional aggregating operation, query response time is doubled for SciDB arrays. The specific reason is unknown but a possible solution to improve this is to select sub array inside SciDB and then calculate average in the Processor, which can significantly fasten query processing of SciDB solutions.

Figure 6.10 also presents an noticeable issue for NetCDF solutions and that is when average is performed on 2880 time steps, the query response time suddenly increases several times especially for NetCDF-4 solution. Such a pattern is also shown in Figure 6.11, i.e. selection of grids at 2880 time steps. By checking raw average aggregation measurements of NetCDF-4 store, all 20 measurements keep at nearly the same level, which entails Windows cached little relevant data for the aggregation query. Additional tests are performed to work out reason. By calculating average with 2880 time steps at Netherlands scale with NetCDF-4 store 20 times consecutively, the average query response time is around 12 s and this corresponds to 0.033 s in Figure 6.10. However, if the query is executed as below (the same way as is used for benchmarking),

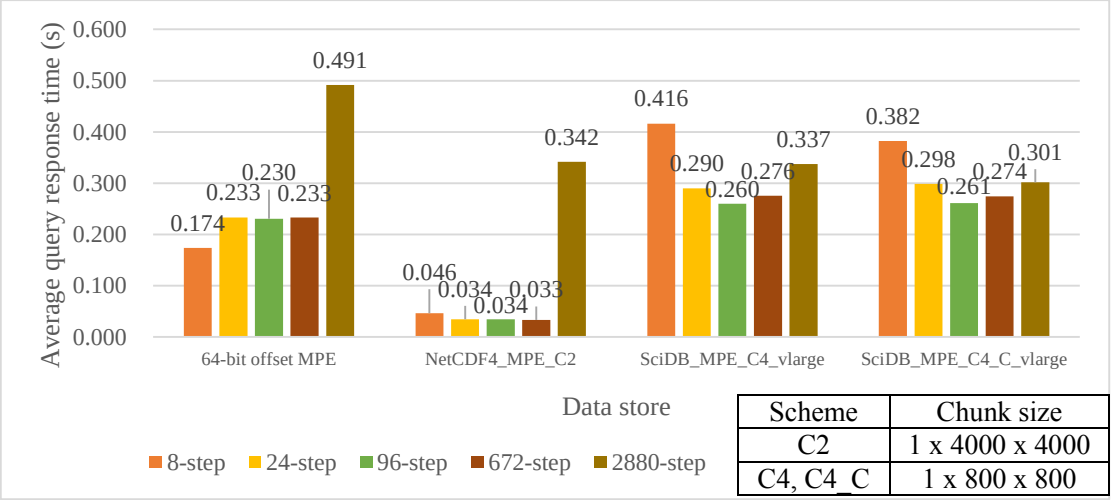
```
For (int i = 0; i < 20; i++ )
  {
      Query(64bitoffset);
      Query(NetCDF-4);
  }
```

Then there is no performance gain for NetCDF-4 and thus it can be deduced that the caching of 64-bit offset store flushes the cached data of NetCDF-4 files. Average calculation with less time steps

results in smaller query results of sub selection, which is probably the reason why cache flushing does not occur. Figure 6.10 indicates that SciDB provides a better caching strategy. Besides, the reason of the trough in the middle of SciDB arrays in Figure 6.10 is explained in time series extraction.

Maximum calculation

The only difference between maximum and average calculation is the change of aggregation type. Other configurations remain the same. Ratio of zero values in the query results of maximum calculation is also identical to that of the average calculation.



Scheme	Chunk size
C2	1 x 4000 x 4000
C4, C4_C	1 x 800 x 800

Figure 6.12. Aligned performance of diverse data solutions at very large level for maximum calculation with different time steps at the Netherlands scale.

Figure 6.12 visualizes benchmarking results of maximum calculation. Compared with average calculation, all solutions take similar amount of time to process the query. The reason for NetCDF solutions is understandable and as is as mentioned, additional statistical calculation costs fairly small amount of time compared to sub data selection. While data selection of both aggregation queries is identical. SciDB on the other hand, calculates maximum slightly faster than average. Interpretation of other patterns can be referenced in the part of average calculation.

6.2.2 Query performance on GEFS dataset

Figure 6.13 presents locations used in queries on GEFS dataset.

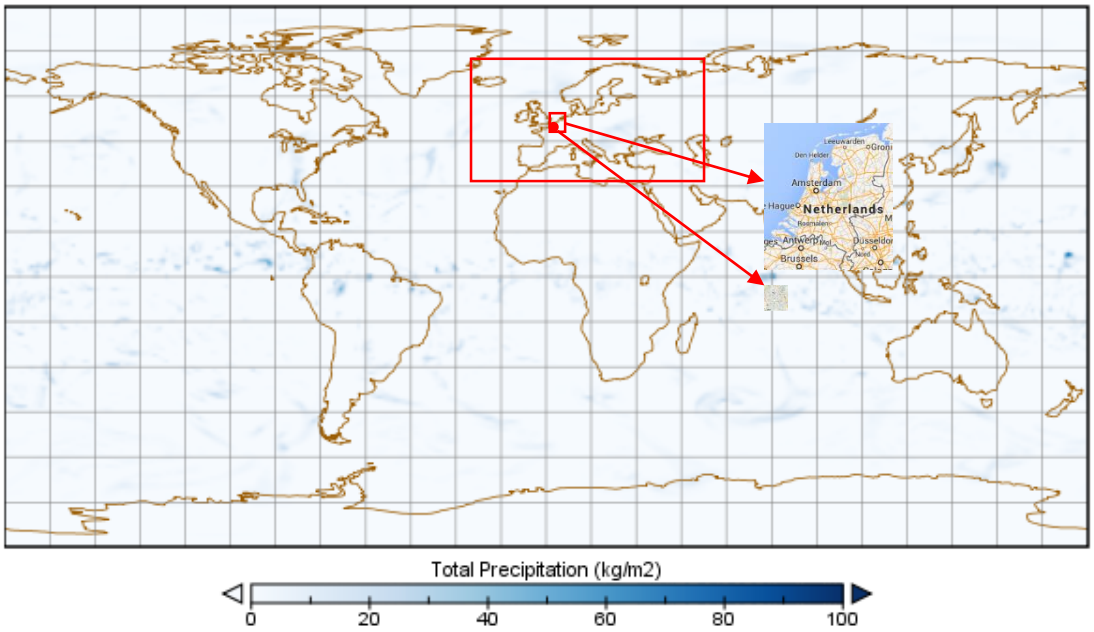


Figure 6.13. GEFS precipitation map calculated with first ensemble for the whole world at 12:00 15-05-14. Three red objects refer to query areas, i.e. from outside inward, Europe, the Netherlands and Delft (a spot location)

Effect of order of dimensions in the schema on query performance

As is mentioned, the order of dimensions to organize attribute in the array can change the data storage on the disk (Figure 6.1), which may have much influence on query performance. Two queries (Table 6.13) are established to investigate such an effect on SciDB_GEFS_S1, SciDB_GEFS_S2, SciDB_GEFS_S3and SciDB_GEFS_S4.

Conceptual query	Query for implementation (AFL)
Select total precipitation for all ensembles in Delft at 0:00, 20-05-2014	project(between(SciDB_GEFS_S1,675,0,17,38,184,675,19,17,38,184),APCP)
Calculate ensemble mean of total precipitation in Delft at 0:00, 20-05-2014	aggregate(project(between(SciDB_GEFS_S1,675,0,17,38,184,675,19,17,38,184),APCP),avg(APCP),X_idx,Y_idx,F_idx)

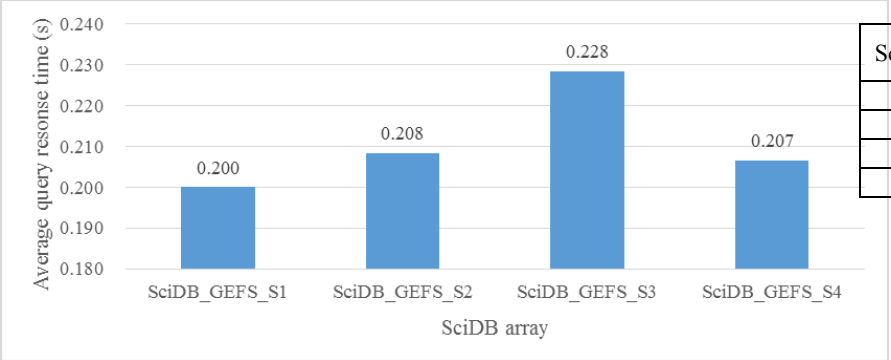
Table 6.13. Two queries used to test the influence of dimensions order on query performance

Table 6.13 only lists AFL command for implementing queries on SciDB_GEFS_S1, for other schemas, the order of numbers inside the "between" operator should be modified. Delft corresponds to one cell in GEFS dataset. It is commonly believed that retrieving values which are stored close to each other on the disk should be faster than values stored discontinuously. So considering dimensions order of 4 arrays above, SciDB_GEFS_S1 should take less time to execute two queries than SciDB_GEFS_S2 while SciDB_GEFS_S4 should response faster than SciDB_GEFS_S3. Besides, SciDB_GEFS_S1 and SciDB_GEFS_S4 should both response more quickly than either

SciDB_GEFS_S2 or SciDB_GEFS_S3. Benchmark tests are then performed locally on SciDB server with bash script (Appendix F). Basically, the first query is executed on the 4 arrays and this is followed by the second query run on the 4 queries. Such 8 query executions form one workflow which is then implemented 10 times. It should be noted that with basic configuration file in Appendix D, an additional line "mgr-cache-size=0" is added to disable caching of SciDB. In addition, in the testing script, the command

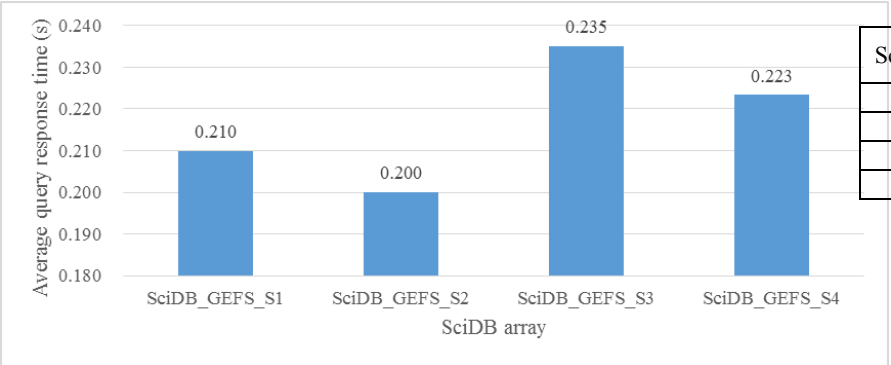
```
sh -c 'echo 3 >/proc/sys/vm/drop_caches'
```

is executed to clean Linux cache after each query run. Through procedures above, for a specific query and an array, 10 measurements mostly keep at constant level. By calculating average for 10 measurements without the largest 2 and smallest 2 values, for all arrays, the query performance is provided in Figure 6.14. The first query returns all positive values and so does the aggregation query.



Scheme	Dimensions order
S1	M E F Y X
S2	M F Y X E
S3	X Y F E M
S4	E X Y F M

a. Selecting all ensembles of total precipitation in Delft at one forecast step



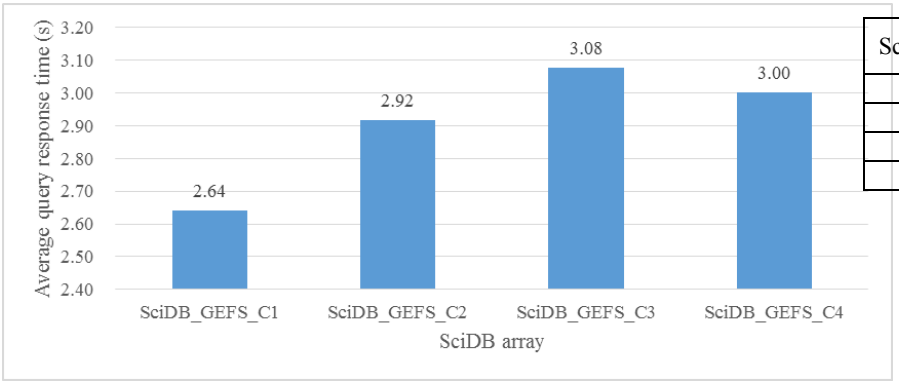
Scheme	Dimensions order
S1	M E F Y X
S2	M F Y X E
S3	X Y F E M
S4	E X Y F M

b. Calculating ensemble mean of total precipitation in Delft at one forecast step

Figure 6.14. Query performance of 4 GEFS arrays with identical moderate chunk size but different dimensions order in schemas

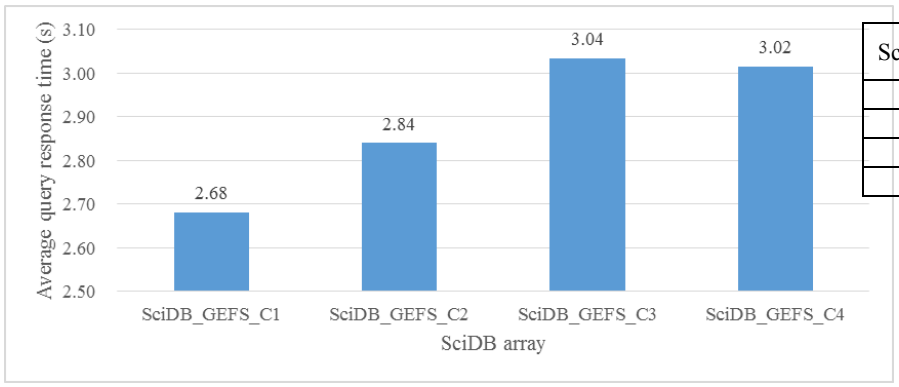
Figure 6.14a presents the pattern as is expected, i.e. SciDB_GEFS_S1 outperforms SciDB_GEFS_S2 by 8 ms, while SciDB_GEFS_S3 is slower than SciDB_GEFS_S4 by 21 ms. Also SciDB_GEFS_S1 and SciDB_GEFS_S4 rank the first two places. However, the gap is still not significant due to the short query execution time. For ensemble mean query, an exception arises that SciDB_GEFS_S1 on average costs 10 ms more to process than SciDB_GEFS_S2. In addition, response time of SciDB_GEFS_S2 is shorter than SciDB_GEFS_S4. To exclude the effect of chunk size on performance, GEFS dataset is managed into 4 other arrays, namely, SciDB_GEFS_C1, SciDB_GEFS_C2, SciDB_GEFS_C3 and SciDB_GEFS_C4 (Table 6.9). Each array has one chunk.

Their dimension orders conform to 4 SciDB_GEFS_S arrays. By testing same benchmarks again with another script (Appendix F), average performance is then depicted in Figure 6.15.



Scheme	Dimensions order
C1	M E F Y X
C2	M F Y X E
C3	X Y F E M
C4	E X Y F M

a. Selecting all ensembles of total precipitation in Delft at one forecast step



Scheme	Dimensions order
C1	M E F Y X
C2	M F Y X E
C3	X Y F E M
C4	E X Y F M

b. Calculating ensemble mean of total precipitation in Delft at one forecast step

Figure 6.15. Query performance of 4 GEFS arrays composed of only one chunk but different dimension order in schemas

By storing GEFS dataset into one chunk and perform the two queries, effect of dimensions order on query performance becomes more significant, especially from the gap between SciDB_GEFS_C1 and SciDB_GEFS_C2. However, the exception still exists that SciDB_GEFS_C2 in which ensemble values in a single location at one forecast step are stored discontinuously responses faster than SciDB_GEFS_C4 where ensemble values are stored successively for both queries. There are indeed some other factors of storage on disk that has been changed with the modification of dimensions order, for example, the starting physical storage address of 20 ensemble values in a single location at a certain forecast step. Generally speaking, the experiments to explore the influence of dimensions order to arrange data storage on query performance present some patterns as are expected, but because of high dimensionality of GEFS dataset which adds much complexity to the mapping from dimensions order to data organization on disk, some patterns are still not fully understandable. Nevertheless, dimensions order can indeed cause significant change of query performance, as gaps between diverse arrays shown in Figure 6.15. As a matter of fact, the analysis of dimensions order influence is more complex for SciDB due to RLE. Assume the 20 ensemble values extracted in first query are all equal to 0 (this is not the case in fact), then with C1 and C4 scheme, they are stored as

'200', i.e. 20 zero values. However, with C2 and C3 scheme, 20 zero values are separated and it is likely that RLE may not achieve the best performance since neighbors of each zero value are unequal to 0. This causes different storage size of 4 arrays (Table 6.9) and introduces more uncertainty to query performance.

In the following, due to relatively better performance in initial tests (Figure 6.14), SciDB_GEFS_S1 and SciDB_GEFS_S2 are used for benchmarking. On the NetCDF side, due to current implementation of NetCDF connector in HdyroNET-4, it is only possible to write NetCDF files with dimensions order in S3 schema. But NetCDF-4 files can be created with different chunk size. At last, in addition to 64-bit offset solutions, two NetCDF-4 schemes are tested together with their compressed versions (Table 6.2).

Forecast time series extraction

This query is to extract 20 ensembles of precipitation in Delft, a spot location at all forecast steps. 56% data in the query result equal to 0. Figure 6.16 shows the benchmark result.

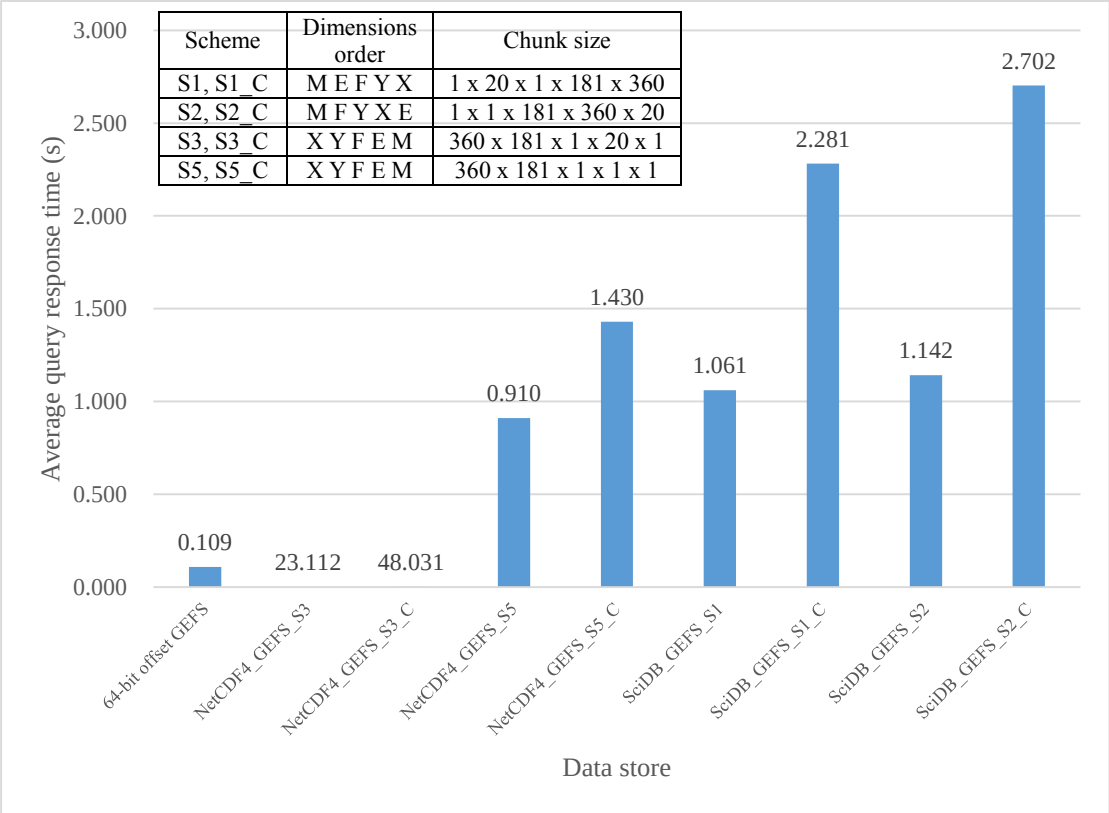


Figure 6.16. Query performance on extracting GEFS forecast time series of total precipitation

In Figure 6.16, performance of NetCDF-4 stores with schema S3 are not visualized for their high values. On the whole, 64-bit offset solution performs the best and is 9 time faster than NetCDF-4 with smaller chunk size which comes the second. The outstanding performance of 64-bit offset solution is majorly owing to small data size of GEFS dataset. SciDB arrays without compression take more time to return query results but the noise included in the measurements cannot be neglected. With HTTP communication time removed, SciDB_GEFS_S1 and SciDB_GEFS_S2 can approach the performance of 64-bit offset solution closely. SciDB_GEFS_S1 takes less time than

S2 to process the query while the gap is insignificant. For their compressed versions, i.e. SciDB_GEFS_S1_C and SciDB_GEFS_S2_C, the difference becomes more obvious, and these result from modification of dimensions order. The compression of SciDB arrays doubles the query response time compared with non-compressed arrays. This is also the case for NetCDF4_GEFS_S3_C. The compression on NetCDF4 S5 store however does not cause so much degradation. Compared with the intolerable performance decrease caused by compression of NetCDF-4 in MPE benchmark tests, 2 reasons lead to amelioration of query performance of NetCDF-4 compressed stores. For one thing, the compressing ratio of GEFS dataset is 2, much less than 83 of MPE dataset. For another, the caching of compressed NetCDF-4 store also contributes and this is significant for NetCDF4_GEFS_S5_C by checking raw testing records.

The high response time of NetCDF-4 S3 stores is due to little caching and the explanation is provided in MPE aggregation benchmark. That is the buffered data of NetCDF4 S3 stores are flushed by caching of other NetCDF stores. NetCDF-4 with smaller chunk size, i.e. scheme S5 can benefit from caching and therefore the chunk size of NetCDF-4 can influence the behavior of caching as well. The caching pattern of NetCDF-4 files also applies to GEFS queries later.

Percentile calculation

This query is to calculate the 80th percentile of total precipitation for all 40 forecast steps in Delft, a spot location. Query result contains 35% 0 values and the rest are diverse values above 0.

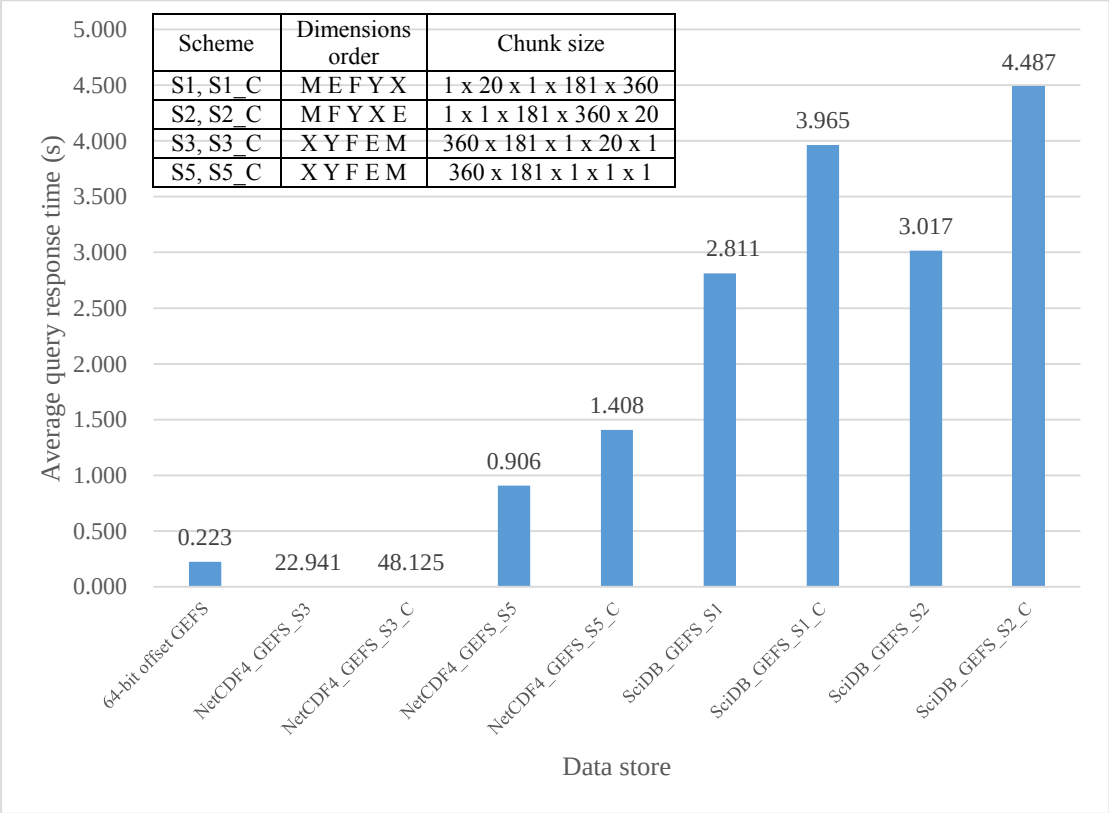


Figure 6.17. Query performance on GEFS total precipitation 80th percentile calculation

Figure 6.17 indicates that the 64-bit offset data solution holds the fastest query processing record. It is then followed by NetCDF-4 S5 stores and SciDB solutions. NetCDF-4 S3 solutions rank the last.

The overall pattern is analogous to forecast time series extraction. However, the complex implementation of percentile calculation in SciDB (Section 5.4.2) cause much overhead on query performance. Compared with only sub selection shown in Figure 6.16, the compound percentile function triples the time cost. While on the other hand, after selecting relevant data in NetCDF files for the percentile calculation in NetCDF connector, these data are transferred into Processor to derive the 80th percentile. The calculating in the Processor takes 0.001 s to finish. So an improvement for SciDB solutions is to only execute sub selection in SciDB but deliver the job of percentile calculation to the Processor, then the performance can be boosted. In addition, dimensions order has slight influence on query performance of SciDB arrays. Reasons of other interesting characters in the figure are interpreted in the previous query.

Ensemble mean calculation

Ensemble mean calculation of a cell is to average the attribute values belonging to all ensembles in that cell. So one cell contains one ensemble mean value. In this test, two locations are used for precipitation ensemble mean calculation at all 40 forecast steps. One is the Netherlands containing 20 cells (5 x 4) and the other Europe scope has 3888 cells (72 x 54) in GEFS dataset. For the query on Netherlands, 32% of the returned values are 0. While for the Europe scope, 18% values retrieved are zero values. Testing results are provided in Figure 6.18.

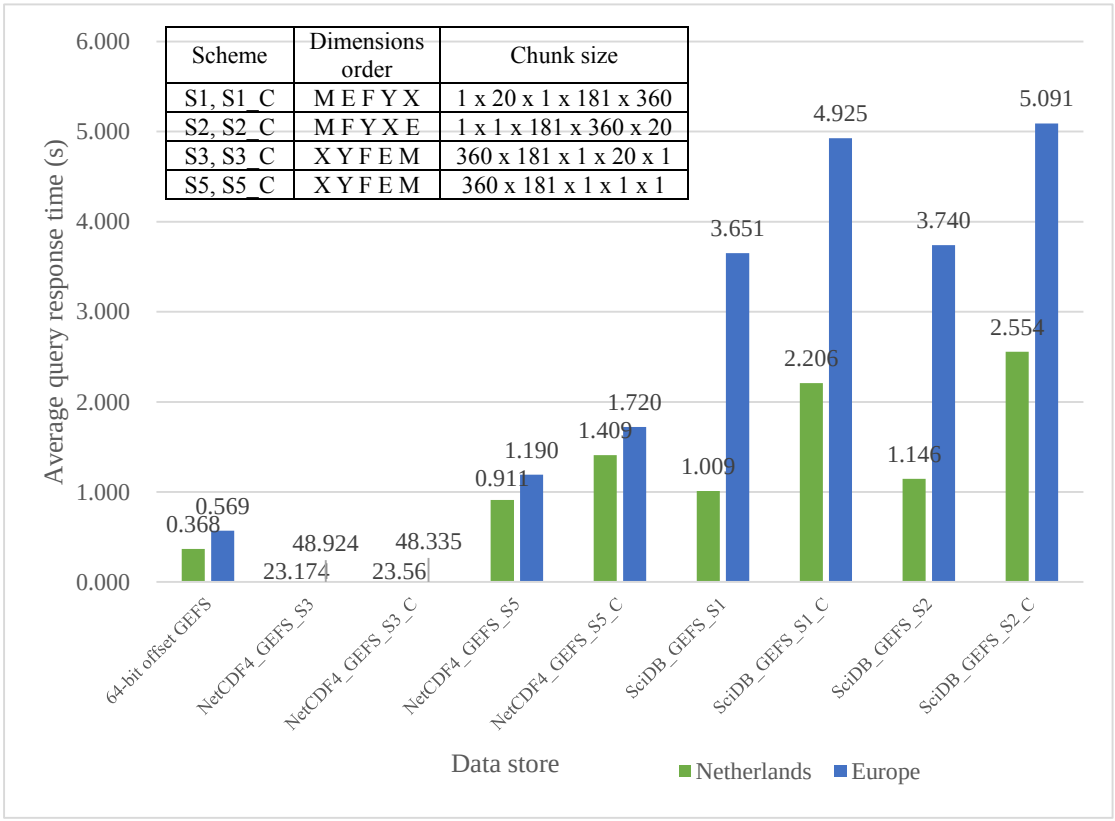


Figure 6.18. Query performance on ensemble mean calculation of GEFS total precipitation

For the calculation of either query area, the pattern of performance shown in Figure 6.18 keep similarity to previous two GEFS tests. Compared with percentile calculation, ensemble mean calculation at the Netherlands scale costs 0.001 s in the Processor while at the Europe scale, time cost in the Processor increases to 0.03 s. The effect of dimensions order is insignificant for SciDB

arrays. Thanks to caching, compression of NetCDF4 S5 store cause less negative impact on query performance compared with SciDB arrays. NetCDF-4 solutions with larger chunk size, i.e. S3 do not benefit from caching undermined by query execution on other data stores.

NetCDF4 S5 data stores present the best scalability and with the query area increases 200 percent, the query response time only grows 20 to 30 percent. The query processing time of 64-bit offset solution experiences a 54 percent increase. With little caching, NetCDF-4 S3 data stores cost twice the time to on the Europe scope compared to the Netherlands scope. The scalability of SciDB solutions rank the last. It should be noted that the realization of ensemble mean calculation in SciDB connector is a hybrid function using two AFL operators, i.e. “aggregate” and “between” rather than only one operator. The added complexity might be a reason for the worse scalability.

6.3 Overall evaluation

To investigate the third sub research question, i.e. for the multidimensional array database, if queries on different dimensions can achieved the same performance level, specific benchmark tests are devised. In MPE benchmark tests, the sub grid selection of northern part of the Netherlands returns 2880 values while the extraction of time series in a spot location in the Indian Ocean at the very large level also includes 2880 values. So through comparing related data solutions on processing these two queries (Table 6.14), some conclusions can be drawn.

Data store	Sub grid selection (s)	Time series extraction (s)	Ratio
64-bit offset MPE	0.03	50.54	1685
NetCDF4_MPE_C2	0.01	3.95	395
SciDB_MPE_C4_vlarge	0.12	17.07	142
SciDB_MPE_C4_C_vlarge	0.09	15.97	177

Table 6.14. Performance of queries targeted at different dimensions

Table 6.14 clearly indicates that spatial selection is much faster than temporal selection, especially from the large ratio of 64-bit offset solution. All the large ratios presents that spatial dimension are superior dimensions on which queries can be processed more efficiently. However the priority of dimensions is indeed reflected in the storage scheme adopted, essentially, the chunk size and dimensions order. For MPE data stored in NetCDF-4 and SciDB arrays, the chunk size on temporal dimension always equals to 1 while chunk sizes of spatial dimensions range from 100 to 4000. In this way, MPE storage schemas adopted indeed give superiority to spatial dimensions. Besides, the order of dimensions in each chunk cannot remedy inferior temporal dimension since the only possibility is to exchange X and Y dimension which cannot benefit time series extraction. It can be imagined that decreasing chunk sizes of spatial dimensions while extruding the chunk size of temporal dimension can improve the performance of time series extraction. Thus a cubic chunk size is needed to judge query performance on different dimensions fairly.

In GEFS benchmark tests, storage schemes with cubic chunk sizes are implemented, i.e. 4 SciDB_GEFS_S arrays without compression. Each chunk has 1 modelrun and forecast value but incorporates 20 ensembles, 181 latitudes and 360 longitudes. When exploring the effect of dimensions order on query performance, two queries targeted at ensemble dimension are used for

testing. It is the fact that executing the same queries on an array with a different dimension order can be considered equivalent to executing queries targeted at another dimension on the original array (For easy understanding, one can think each chunk contains 20 ensembles, 20 longitudes and 20 latitudes). The test results show that exchanging the order of ensemble, longitude and latitude dimension inside each chunk has insignificant influence on query performance. That is equivalent to say, with the scheme utilizing modest and cubic chunk sizes, there is no priority on any dimension inside the chunk and query performance on different dimensions enrolled in one chunk can locate at the same level. Even though later SciDB solutions with large chunks, i.e. 4 SciDB_GEFS_C arrays show notable effect of dimensions order, which may prove the superior dimensions do exist, but in most cases, modest chunk size is employed due to better query capability. As can be seen from Figure 6. 13 and 6.14, the solutions with moderate chunk sizes can be 10 times faster than large chunk solutions for processing two queries. In addition, significant variation of performance of arrays with large chunk size can be caused by RLE of SciDB, as has been analyzed.

It is thus inferred that for SciDB arrays, to make dimensions have fair roles in query processing, 2 conditions should be satisfied. One is the chunk size is modest. The other is the chunk sizes on all dimensions are equal, i.e. hypercube. Nonetheless, more benchmarking tests should be implemented to verify such an inference.

In the end, as the research is concerned with management and query performance of large hydrologic datasets, a comprehensive evaluation of different data solutions is provided in Table 6.15.

Data solution		64-bit offset	NetCDF-4	NetCDF-4 DEFLATE compression	SciDB array	SciDB array DEFLATE compression
Management	Data loading	5	4	3	1	1
	Storage	1	1	5	3	4
	Scheme transformation	1	1	1	4	4
Management overall score		7	6	9	8	9
Query	MPE sub grid selection	4	5	1	3	2
	MPE time series extraction	2	5	1	4	3
	MPE average calculation	4	5	1	3	3
	MPE maximum calculation	4	5	1	3	3
	GEFS forecast time series extraction	5	4	2	3	1
	GEFS percentile calculation	5	4	3	2	1
	GEFS ensemble mean calculation	5	4	3	2	1
Query overall score		29	32	12	20	14
Compound score (management * 0.33 + query * 0.14)		6.48	6.57	4.71	5.52	5.00

Table 6.15. Overall evaluation of diverse data solutions for managing and querying large hydrologic datasets

Data solutions refer to combined data managing and querying systems. So 64-bit offset, NetCDF-4

with and without compression actually represent systems composed of HydroNET-4 interfaces together with diverse NetCDF file stores. As is mentioned in the beginning of this chapter, such a hybrid NetCDF system is in essence a database. SciDB array solutions include the SciDB database and the enhanced interface, SciDB connector embedded in HydroNET-4. The criteria listed are confined for this research, some general strengths of databases like parallel, security and transaction support are not taken into account. The maximum score of each criterion such as data loading and MPE sub grid selection is 5 while the minimum is 1. The total final compound score equals to 10 with management and query performance sharing a half respectively. The specific calculating formula is provided in the table.

Table 6.15 indicates that NetCDF-4 without compression scores the highest among all solutions. It is then followed by 64-bit offset solution. Two SciDB solutions come after while NetCDF-4 with compression ranks the last. The remarkable scores of NetCDF-4 and 64-bit offset solutions lie in their outstanding query performance.

Regarding data management, 3 aspects are considered. Original MPE and GEFS data are in GRIB2 format and converted to 64-bit offset format by Hydrologic Research for daily use. So the 64-bit offset solution gains 5 for data loading since files have already existed. With NetCDF and SciDB connectors, 64-bit offset NetCDF files are transformed to other files. The loading process of SciDB constituting several steps (Section 5.4.1) and very slow. Loading 170 GB MPE data of one month into SciDB_MPE_C4_vlarge (14 GB) and SciDB_MPE_C4_C_vlarge (4 GB) took one and a half days to finish. While it cost half a day to import same amount of MPE data into NetCDF-4 files including the compressed version. Transforming the GEFS data of one model run into NetCDF files without compression cost 3.6 minutes while time cost increases to 4.9 minutes when DEFLATE is applied to NetCDF-4. For both MPE and GEFS dataset, NetCDF-4 files without compression occupy same disk space as 64-bit offset files, which is the reason why both of them are scored 1. In regard to MPE data, due to RLE and possible DEFLATE compression, storage size of SciDB arrays can be 10 times smaller. Moreover, SciDB provides generic “redimension” and “repart” operators to support scheme transformation. However, according to practical experience, these two operators are inefficient. NetCDF solutions have no support on transforming data storage scheme. To change dimensions order or chunk size for example, specific functions have to be developed in NetCDF connector. Besides, new query functional modules have to be developed to adapt to new data schemes accordingly.

As to query performance, it should be noted that SciDB solutions include some noise such as HTTP communication in the query response time. For MPE aggregation queries, NetCDF-4 with compression take too much time to finish, so it is excluded from MPE aggregation benchmarking. Nonetheless, it should score the lowest in query performance. Recalling GEFS benchmark tests, two NetCDF-4 schemes, i.e. S3 and S5 are tested however the NetCDF-4 S3 solutions spent longest time to process queries due to cache flushing. The score is based on the assumption that caching of NetCDF-4 can work effectively on GEFS queries, i.e. NetCDF-4 S3 solution, then NetCDF-4 solutions get higher score than SciDB solutions. Another point is that two regions are used for ensemble mean calculation, and the score is derived according to the average of query performance on both areas.

7 Conclusions and future work

7.1 Summary

The whole research is summarized with respect to research questions as follows.
Main research question:
Can a multidimensional array database process frequently implemented queries faster than NetCDF solutions for large hydrological datasets?

Within the scope of this research, the answer is negative. Through interviews, queries and datasets frequently processed by water experts are collected. After query classification and designing, specific queries and datasets used for benchmarking are determined. 9 criteria are established to compare multidimensional array databases and SciDB is selected for benchmarking. After constructing testing environment in HydroNET-4 system, NetCDF solutions and SciDB solutions are benchmarked and analyzed. In Table 6.15, diverse data solutions are scored with respect to 7 queries tested. And the overall score indicate that the solution of NetCDF-4 without DEFLATE compression and 64-bit offset solution outperform SciDB solutions with and without DEFLATE compression. However, this conclusion is constrained by several points,

1. Since the benchmark environment is the HydroNET-4 system built on a Windows virtual machine, a SciDB connector is developed to communicate with SciDB on Linux virtual machine through HTTP requests. As a result, measurements of SciDB query response time include additional time cost like HTTP communication and data transfer. Such noise can indeed influence the judgment of favorable data solutions for specific queries. One notable example is MPE sub grid selection, by removing the noise, SciDB solutions can even be faster than 64-bit offset solution. Fortunately, the noise cannot change the final conclusion that NetCDF solutions are preferable due to the large gap of overall query performance (Table 6.15).
2. As the benchmark results present, the chunk size plays a crucial role in query performance but chunk sizes tested for SciDB arrays are limited. Besides, for MPE tests, chunks are all of 2D or near 2D shape, i.e. chunk size on temporal dimension equal to 1 or 4. Performance of SciDB as well as NetCDF-4 solutions with real cubic chunks are still unknown. According to experience and understanding acquired from SciDB tests, it is inferred that cubic chunk size can boost SciDB's performance on MPE time series extraction and two aggregation calculation.
3. A fundamental issue of this research is concerned with the concept "large dataset". It is thought that "large" concerns two aspects. One is the semantic context, i.e. application domain. The other is technical indicator. More specifically, as big data is bloating all the time and benchmarking always ends at a certain level of data size, large dataset should cause bottlenecks of software or hardware employed. From two aspects, GEFS data of size 1.55 GB is indeed small. The MPE satellite dataset at very large level is 170 GB stored in 64-bit offset files. Its spatial resolution is 3.3 km and more importantly, the temporal resolution is 15 minutes, very high among satellite data products. Also as the tests reveal, MPE data at large and very large level causes long caching process for SciDB in aggregation queries. Caching has little effect for NetCDF-4 files at very large level due to flush. The intolerable slow data loading of SciDB also

arises. So despite that 98% of MPE values are 0, it is thought that MPE data achieve the property of large hydrologic dataset.

4. Parallelization is the current trend to solve big data problems. NetCDF libraries until now have limited support for reading with multiple threads while SciDB is applying both shared memory and shared nothing architecture. Some operators are optimized to utilize multithreads with one core. While combination of operators and some algebra operators can redistribute data and execute queries on all logic nodes, i.e. all SciDB instances (http://www.scidb.org/forum/viewtopic.php?f=11&t=1441). In this research, such a strength of SciDB is not tested.

Taking account of data management such as loading and storage space, Table 6.15 actually presents a more comprehensive comparison among diverse data solutions. NetCDF-4 without compression still ranks the first place but a bit surprisingly, 64-bit offset solution is just behind NetCDF-4 solution with little gap. As developers of Hydrologic Research explain, this is thanks to constant optimization of the data flow to read 64-bit offset files. For example, a single file is kept open until all reading tasks are completed. Such a research does not deny SciDB which is a relatively new product and still needs to be optimized. Nonetheless, combining the research results, a general suggestion for data management might be that before importing all data into specific DBMSs, spend some time on learning if there is possibility to improve available file based solutions.

4 sub research questions:

1. *What datasets and queries should be used to fully assess the performance between a multidimensional array database and NetCDF solutions?*

Datasets and queries for benchmarking should be frequently processed by hydrologists. In total, 6 water specialists are interviewed among whom half come from academia while the other half are from industry. Multiple dimensions and large volume are the interests when selecting datasets. In the end 3 dimensional MPE satellite data and 5 dimensional GEFS model forecast data are chosen for tests. Conceptual queries collected from experts are classified and redesigned to conform to datasets. 8 queries are designed elaborating sub grid selection, time series extraction, pyramiding and aggregation. But in the final benchmarking, majorly due to different implementations of NetCDF solution and SciDB, pyramiding is not tested.

From the consultancy, more insight on queries and datasets on which hydrologists are working has been acquired. Since current hydrologic tasks increasingly deal with more than one data sources, queries based on two or more different datasets are executed frequently, like data quality check. Besides, the consultancy also indicates that spatial operation, as another query class can be a daily work for hydrologic experts. This is understandable since natural hydrological data are normally recorded in grids while information of urban environment are stored as vector maps. Many applications related to human and natural interaction need certain kind of spatial operation such as intersection. These two queries are omitted by researchers before and should receive more attention for comprehensive benchmarking.

2. *Which specific multidimensional array database should be used for tests?*

A definition of multidimensional array database is proposed in this thesis that it refers to a database of which the abstract model for data management and query is multidimensional array consisting of dimensions and attributes. According to this definition, array database like Rasdaman and SciDB, commercial DBMS for OLAP such as Essbase and hybrid system UFI are all specific implementations (Section 4.1). The research then focuses on Rasdaman and SciDB. By establishing 9 criteria such as compression support and NetCDF loader, these two solutions are compared with evidence retrieved from literature, forum, source code, etc. But no practical tests are involved for assessment.

Through grading all criteria, SciDB is eventually chosen mainly due to lossless compression support in its community version. Also it scores higher in knowledge support included in maintenance criterion. Recalling the whole research progress, the SciDB forum indeed keeps input (Appendix E). However, the lack of .Net API and NetCDF importer for SciDB adds much additional development work in the research, which took several months to finish.

3. *For the multidimensional array database, is the performance in handling queries on different dimensions at the same level using one data storage schema?*

The answer is yes, i.e. if the SciDB array schema utilizes modest and hypercubic chunk size, then dimensions have equal roles in query processing. But this is derived from analysis indirectly from benchmark tests. Specific explanation is provided in the first part of Section 6.3. In brief, arrays with cubic and modest chunk sizes are created for storing GEFS data. But these arrays vary in the order of dimensions to organize data structure. By testing two queries, the result shows that exchanging the order of longitude, latitude and ensemble dimensions inside each chunk has slight influence on query performance. Based on this, it is further derived that if the chunk size is identical on each dimension and total chunk size is modest, say, containing around 1 million cells, a level recommended by SciDB developers, there is no superior dimension on which queries can be processed faster. More tests are needed for verification, of course. Anyway, according to the tests on dimensions order effect, it is thus suggested that the research emphasis on chunked storage structure should move to designing appropriate chunk sizes or smartly indexing chunks instead of data structure and indexing inside chunks.

4. *Does data compression in the multidimensional array database have an impact on the query performance?*

SciDB implements RLE for data storage and that is to say, data have been compressed when stored into SciDB arrays. SciDB constructs indexing strategy based on RLE encoded data. Other compression types are also supported on top of the RLE encoded data. And when decompressing, data return to the RLE state. In the tests, additional DELFATE compression is applied and compared with normal RLE encoded version to learn the influence of DELFATE compression on query performance.

Through benchmark tests, it is found that the DEFLATE compression on SciDB arrays can either have negative effect or no effect on query performance. In addition, from the MPE time series extraction, the performance of DEFLATE compressed SciDB arrays presents correlation with chunk size. That is, the negative impact of DEFLATE compression on query decreases as chunk size reduces. Although specific reasons for such a correlation still need to be investigated, it is considered that the overhead of uncompressing large chunks inside memory is the origin.

7.2 Extension of current research

In regard to existing flaws as well as more interests inspired by the research, extension work can be conducted. Typical directions are:

1. Chunk model. Previous researches (Lee et al., 2008) and this study demonstrate that the chunk size is perhaps the most crucial factor for efficiency of chunked storage structure. On the other hand, the effect of chunk size is unpredictable and for example, at which chunk size a specific query performance experience significant tuning point. It is expected from analysis with cubic and modest chunk sizes, sub grid selection and time series extraction can consume same amount of time but this still needs to be clarified. Regarding this, intensive experiments can draw specific conclusions on chunk size impact while more generic understanding relies on theoretical support. A possible way is to propose a model concerning important factors such as hardware configuration, data size and query habits to predict the performance of chunks. After this, the model should be verified by comprehensive benchmark tests.
2. "Hot" and "cold" query tests. Normal approaches used to measure the average query response time include two types. "Hot" test means that the same query is executed on each data store consecutively several times and then average time is calculated. Caching effect is elaborated. As the opposite, "cold" test purges the cache every time before executing each query. Lower boundary of query response time comes from "hot" test while upper boundary depends on "cold" test. The method used in this research however is neither of them (Section 6.2). It is a state close to "hot", i.e. cache is not cleaned for a specific data store but can be flushed by caching of other data stores. Honestly, none of these 3 approaches reflect the reality. A more scientific way might be through analyzing query logs recorded by Hydrologic Research for instance to learn what users query in reality and try to simulate those scenarios for testing.
3. Query performance with less memory. Caching is influential on query processing and both NetCDF and SciDB solutions benefit much from it. However, the cached data cannot exceed the capacity of main memory. It is therefore interesting to learn what will happen if data queried is too large to fit into the memory. This research does not reach such an edge. Earlier experience indicate that server query performance degradation will take place when memory is full for databases. But the situation for file based solution is not clear yet.
4. Parallelization. Parallel data loading of SciDB with the tool loadcsv.py presents better performance according to extra tests. But as is mentioned previously, the parallel query processing of SciDB is not explored in the research. Also the research demonstrates the favorable capability of NetCDF solutions without parallelization. Then a potential topic in the future is to compare parallel query performance between SciDB and NetCDF solutions. Besides, as data loading is indeed a problem for SciDB, effort can also be put on parallel loading

techniques.

7.3 Dimension and multidimensional data management

Dimension and attribute are introduced when discussing multidimensional data storage in Chapter 2. There, only some samples of what can be dimensions and attributes are presented without clearly distinguishing these two concepts. In essence, on the abstract level for understanding the world, there is no difference between dimension and attribute. Sounds, temperature and precipitation are different dimensions of information acquired from the world. Location and time are also dimensions related to life. These dimensions can also be called attributes. On the abstract level, it does not make sense to distinguish these two concepts.

The distinction happens in the implementation and in both NetCDF and SciDB, such a distinction between dimension and attribute exists. A simple example, a 2D grid recording precipitation show in Figure 7.1 is used for the explanation.

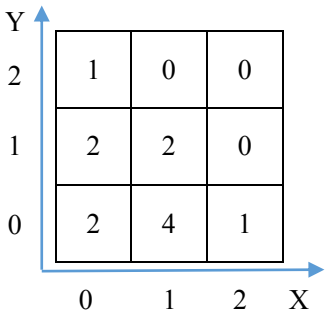


Figure 7.1. Precipitation Map (Data scheme 1)

In most cases, attribute is the precipitation data which is stored on the disk. Dimensions have not to be necessarily stored, they are used as indexes and can be reflected in the order of how precipitation data is arranged on the disk. However it is possible to transform the attribute into dimension and vice versa (Figure 7.2).

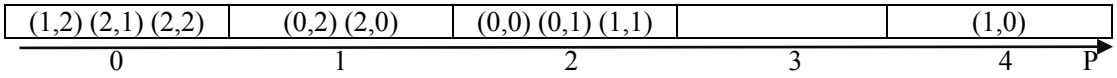


Figure 7.2. Precipitation map (Data scheme 2)

In Figure 7.2, precipitation is regarded as the only dimension while the original (x, y) tuples are stored as attributes. Sometimes multiple attributes are stored in the same grid layer, e.g. precipitation, temperature and humidity in one dataset, which can then be transformed into another multidimensional dataset with precipitation, temperature and humidity as dimensions. This transformation from dimension to attribute indeed makes sense. In fact, two types of queries are frequently referred to, one is based on dimensions, i.e. sub-selection and the other is based on attribute values. With the first data scheme (Figure 7.1), the first type of query can be efficiently executed while for second type of query, data scheme 2 has the superiority. The conversion between

dimension and attribute also implies the transformation of the two types of query.

Now that on the abstract level, there is no difference between attributes and dimensions, a logical thinking is to implement this in the data model, i.e. an all dimensional data model. In this model, original attributes are realized as dimensions and what is then stored in each cell of the multidimensional array is the information indicating whether the corresponding combination of dimension values exists or not. Such information can be expressed by 1 bit, which entails using 0 to represent non-existence while 1 refers to existence. However, it should be noted that the precipitation example presented above is an ideal case. In reality, measurements such as precipitation are normally floating numbers and their range can even be unlimited, which can result in huge overhead to implement an all dimensional data model. In other words, the resultant array can be very sparse, a few 1 values filled in a very large multidimensional cube. From this point of view, the all dimensional data model cannot bring any benefits. The implementation of such a model requires more understanding.

Even with current dimension-attribute storage scheme, smart use of dimension can reach the performance gain. In this research, the way NetCDF files of MPE dataset is managed, i.e. whole dataset is split into many single files indeed adds an additional dimension which is the index of files. The original contiguous data storage is split into separate data blocks on the disk thanks to the additional dimension. Though this is some work has to be done because of the file size limitation of 64-bit offset format. Nevertheless, the query processing can then benefit from B-tree index of NTFS system functioning on the additional dimension introduced.

Besides, a dimension always has meaning such as ensemble, time as well as the artificially created dimension, the file index for example. The semantics of file index dimension refers to every file consisted of 4 time steps. And due to the semantic property of dimensions, it is possible to embed additional information or knowledge by adding artificial dimensions to original datasets, which implies a datasets can contain more information than the original information it conveys. For example, on the one hand, there is a rectangle image covering the coastal area of The Hague and on the other hand, a precipitation map covering the same area needs to be stored. From the image, the land type, i.e. sea or mainland can be derived. So it is possible to add an additional dimension, i.e. land type to the precipitation map and split the map into two chunks according to land type dimension. In this way, the knowledge of land type is reflected in the data storage structure of the precipitation map, i.e. the separation of chunks. So if the image of The Hague is deleted, there is still relevant information in the precipitation data storage. But the impediment lies in recovery of the land type information. From two chunks, it is possible to derive an underlying dimension composed of two members, but it is difficult to deduce that the dimension is land type. The burden cannot simply be moved to metadata which becomes too heavy then.

By the way, the knowledge embedded in the data structure is sort of like DNA where most information is expressed in permutation and combination of base pairs rather than the 4 bases…

References

Baumann, P. (1999). A database array algebra for spatio-temporal data and beyond. In *Next Generation Information Technologies and Systems* (pp. 76-93). Springer Berlin Heidelberg.

Baumann, P., Dehmel, A., Furtado, P., Ritsch, R., & Widmann, N. (1998). The multidimensional database system RasDaMan. In *ACM SIGMOD Record* (Vol. 27, No. 2, pp. 575-577). ACM.

Baumann, P., Furtado, P., Ritsch, R., & Widmann, N. (1997). The RasDaMan approach to multidimensional database management. In *Proceedings of the 1997 ACM symposium on Applied computing* (pp. 166-173). ACM.

Baumann, P., & Stamerjohanns, H. (2014). Towards a Systematic Benchmark for Array Database Systems. In *Specifying Big Data Benchmarks* (pp. 94-102). Springer Berlin Heidelberg.

BCS. (2012). The Universal File Interface (an in-depth presentation). Retrieved from http://www.barrodale.com/UFIinDepth.pdf

BCS. (2014). Universal File Interface (UFI): querying large files without database loading. Retrieved from: http://www.barrodale.com/universal-file-interface-ufi

Brown, P. G. (2010). Overview of SciDB: large scale array storage, processing and analysis. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (pp. 963-968). ACM.

Cohen, S., Hurley, P., Schulz, K. W., Barth, W. L., & Benton, B. (2006). Scientific formats for object-relational database systems: a study of suitability and performance. In *ACM SIGMOD Record* (Vol. 35, No. 2, pp. 10-15). ACM

Colliat, G. (1996). OLAP, relational, and multidimensional database systems. In *ACM SIGMOD Record* (Vol. 25, No. 3, pp. 64-69). ACM

Cornillon, P., Gallagher, J., & Sgouros, T. (2003). OPeNDAP: Accessing data in a distributed, heterogeneous environment. *Data Science Journal*, 2(5), 164-174.

Cudre-Mauroux, P., Kimura, H., Lim, K. T., Rogers, J., Madden, S., Stonebraker, M., ... & Brown, P. (2012). SS-DB: A standard science DBMS benchmark. Retrieved from: http://www-conf.slac.stanford.edu/xldb10/docs/ssdb_benchmark.pdf

Hahn, K., Reiner, B., Höfling, G., & Baumann, P. (2002). Parallel query support for multidimensional data: inter-object parallelism. In *Database and Expert Systems Applications* (pp. 820-830). Springer Berlin Heidelberg.

Hartnett, E., & Rew, R. K. (2008). Experience with an enhanced NetCDF data model and interface for scientific data access. In *24th Conference on IIPS*. American Metereological Society, New Orleans C (Vol. 7).

Idreos, S., Groffen, F., Nes, N., Manegold, S., Mullender, K. S., & Kersten, M. (2012). MonetDB: Two decades of research in column-oriented database architectures. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 35, 40-45.

InterSystems. (2014). Using Caché Globals. Retrieved from http://docs.intersystems.com/documentation/cache/20141/pdfs/GGBL.pdf

Kenan Technologies. (1996). An Introduction to multidimensional database Technology. Retrieved from http://www.fing.edu.uy/inco/grupos/csi/esp/Cursos/cursos_act/2003/DAP_SistDW/Material/ken96.pdf

Lane, D. M. (2013). Introduction to Statistics. Retrieved from http://onlinestatbook.com/Online_Statistics_Education.pdf

Lee, C., Yang, M., & Aydt, R. (2008). NetCDF-4 Performance Report. Retrieved from http://www.hdfgroup.org/pubs/papers/2008-06_netcdf4_perf_report.pdf

Manegold, S., Kersten, M. L., & Boncz, P. (2009). Database architecture evolution: Mammals flourished long before dinosaurs became extinct. *Proceedings of the VLDB Endowment*, 2(2), 1648-1653.

Oracle. (2008). Oracle Essbase Database Administrator's Guide. Retrieved from http://docs.oracle.com/cd/E12825_01/epm.111/esb_dbag/frameset.htm?dinconc.htm

Oracle. (2014). GeoRaster Developer's Guide, 12c Release 1 (12.1). Retrieved from http://docs.oracle.com/cd/E16655_01/appdev.121/e17894.pdf

Pedersen, T. B., & Jensen, C. S. (2001). Multidimensional database technology. *Computer*, 34(12), 40-46.

Rasdaman. (2013). Rasdaman query language guide, version 9.0. Retrieved from http://www.rasdaman.org/export/eb24d6243de082c64a898a28277cc4cb6d623f06/manuals_and_examples/manuals/doc-guides/ql-guide.pdf

Rew, R., Hartnett, E., & Caron, J. (2006). NetCDF-4: Software implementing an enhanced data model for the geosciences. In *22nd International Conference on Interactive Information Processing Systems for Meteorology, Oceanograph, and Hydrology*.

SciDB. (2014). SciDB User's Guide. Retrieved from http://scidb.org/HTMLmanual/14.3/scidb_ug/index.html

Seering, A., Cudre-Mauroux, P., Madden, S., & Stonebraker, M. (2012). Efficient versioning for scientific array databases. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference* (pp. 1013-1024). IEEE.

Stonebraker, M., Brown, P., Poliakov, A., & Raman, S. (2011). The architecture of SciDB. In *Scientific and Statistical Database Management* (pp. 1-16). Springer Berlin Heidelberg.

Stonebraker, M., & Cetintemel, U. (2005). One size fits all: An idea whose time has come and gone. In *Proceedings of the 21st International Conference on Data Engineering* (pp. 2–11).

Su, Y., & Agrawal, G. (2012). Supporting user-defined subsetting and aggregation over parallel netcdf datasets. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium* (pp. 212-219). IEEE

Suijker, P., Alkemade, I., Kodde, M.P., & Nonhebel, A. (2014). *User requirements massive point clouds for esciences (wp1)* (Technical Report). Retrieved from TU Delft library: http://repository.tudelft.nl/view/ir/uuid%3A351e0d1e-f473-4651-bf15-8f9b29b7b800/

Unidata. (2014). In Unidata's NetCDF website. Retrieved from http://www.unidata.ucar.edu/software/netcdf/

Unidata. (2011). NetCDF Users' Guide. Retrieved from http://www.unidata.ucar.edu/software/netcdf/docs/netcdf/

Van Oosterom, P., Marinez-Rubi, O., Horhammer, M., Geringer, D., Ravada, S., Tijssen, T.,…& Goncalves, R. (2014). *Massive point cloud data management: design, implementation and execution of a point cloud benchmark* [Paper under review]. Delft, the Netherlands: Delft University of Technology, Faculty of Architecture.

Xie, Q. J. (2008). Oracle Spatial, Raster Data. In *Encyclopedia of GIS* (pp. 826-832). Springer US.

Zhang, Y., Kersten, M., & Manegold, S. (2013). SciQL: array data processing inside an RDBMS. In *Proceedings of the 2013 international conference on Management of data* (pp. 1049-1052). ACM.

Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, *23*(3), 337-343.

Appendix A: Questionnaire for expert consultancy

Q1: What kind of queries do you execute most?
(Query mainly refers to queries executed with SQL, or simple calculations that can be expressed using SQL. Complex computation based on data, for instance data assimilation is not the query. Some query examples are provided below,
E.g.1 aggregation, from monthly rainfall to yearly rainfall
E.g.2 selection of subset, select data from 2001 to 2010 from original dataset which ranges from 1990 to 2013
E.g.3 Some soil moisture datasets have a flag field which indicates the accuracy of the data, so if the data value exceed the accuracy threshold, then this data value should not be selected and used.
Above queries are just examples, do not be constrained by these examples.)

Q2: For queries you mentioned above, which datasets are they based on? How many dimensions do these datasets have?
(Dimension refers to the property to confine the data of interest. For example, in a 5 dimensional precipitation forecast dataset, the data of interest is of course the precipitation. While it contains latitude and longitude dimensions, a temporal dimension, a perturbation dimension which indicates the ensembles (normally 20 times). And the last dimension is the time you run the forecast model.)

Q3: Which format (which version, for example NetCDF has version 1 to 4) is used to store the datasets you mentioned or are they stored in the database?
(Besides, you may also provide some general remarks about database or data formats used by hydrologists)

Q4: What problems did you encounter with executing queries using the data format or database?
(Are they efficient for running queries? Are the data retrieved expected by you, i.e. may be the data returned contains only part of the data you wanted?)

Q5: Do you think big data (mainly refer to big size) is a problem or will the big data become a problem in hydrologic domain in the future?
(Have you ever thought about it?)

Q6: Do you have experience with parallel query processing or data compression techniques which can improve the query efficiency?

Appendix B: Records of interview

Interview with Dr.ir.Steele-Dunne from water resources management group in TU Delft.

Q: I am now doing a research on comparison of querying efficiency between a multidimensional database and NetCDF. And as an expert of remote sensing and soil moisture, you must have a lot experience dealing with satellite data or other relevant datasets, so which queries do you execute most frequently?

A: I usually process satellite datasets in which observations are within swaths and outside those swaths, there is no data. So in a NetCDF file which encodes a whole array, which can be soil moisture for example, only cells within a swath have observation values and other cells contain null value. A frequent task I do is to retrieve those cells which hold values and then start interpretation. Using NetCDF, each cell should be checked and then cells containing non-null values can be extracted, which takes some time. But since the datasets I use can be utmost several GBs, querying efficiency is not a big issue. In addition, for some soil moisture products, they also include a flag layer indicating whether soil moisture values in corresponding cells contain high errors. So another job we do is to exclude those cells which do not satisfy quality constraints.

Well, since my job is mainly about interpreting the data and then demonstrating the physics behind these data, not so many queries are performed. But there is a project called Digital Delta in which our group is also involved and its aim is to integrate different kinds of hydrologic datasets on a platform and then serve as a portal for users to download data. Perhaps you can find queries which are executed most or other relevant information about querying on the website of that project.

Q: OK, then which datasets are often used for you analysis?

A: Remote sensing data, like TRMM, GRACE or SMOS. Besides, I still recommend you the Digital Delta project where you can find popular datasets as well.

Interview with Dr.ir.Siek from Hydrologic Research B.V.

Q: As an expert of hydroinformatics, you must have lots of experience of querying hydrologic data. So what kind of queries do you often execute?

A: Hydrologists typically have various kinds of observed data to be processed and analyzed. My current work mostly relates to hydraulic and hydrologic modelling. For these modelling, I often program my own scripts to read and process the data which are stored in formats that are simple to operate like .txt or gis files. As the data is mostly not stored in a database, some queries are not really required. For instance, a historic precipitation time series collected by rainfall gauges is a crucial input for a typical hydrologic model for calibration. If the calibration period is 30 years, a single txt file is used to store precipitation time series of 30 years recorded by each gauge. For missing values, there is a flag column in the file indicating no value for specific time steps which is 15 minutes. The program can recognize this and run without errors popping up.

Q: Michael, you have talked something about the data you use but I want more details. So how large are the datasets, how many dimensions do they have? Do you also process data stored in NetCDF or database?

A: For hydrologic modelling I mentioned, the model input data are evaporation, precipitation, runoff, etc. The size could in total be as large as 500 MB for one model. There are just three columns for these datasets: date time, input variable and a flag indicating missing value. For my current work, I seldom use NetCDF data or retrieve data from database.

Q: What do you think of big data from a hydrologist point of view? And do you have experience with techniques like compression or parallel processing to improve performance of data processing?

A: I do not think large raw data is a problem, but the data analysis of it could create a very large data. From my experience, hydrologic raw data size increases linearly. That is to say, this year the data size is 500 MB for example, next year, it can be more or less double to 1000 MB, assuming they have the same data structure and spatiotemporal resolutions.

I never tried data compression nor parallel processing, but the memory block data retrieval was used to foster the highly-required computation for model calibration. Personally, I think compression takes more time due to the required compressing and decompressing processes, but this depends on case by case. On the other hand, parallel computing has a great potential to speed up the large data processing.

Interview with Prof. Wang from school of hydrology and water resources, Hohai University

Q: As a professor specialized in hydrologic remote sensing and hydrometeorology, you must have some experience with executing queries on various hydrologic datasets. And which type of query do you run most frequently?

A: Data analysis is a very important part in our research, but we do not explicitly execute queries in databases. Instead, some professional softwares like R and Matlab are normally applied for these work.

We mainly use two kinds of data. First type of data is ground-based observations, mainly precipitation or discharge data stored in txt or excel files. For these data, we can perform trend analysis which majorly include two types. One is the change of total value. Take precipitation as an example, the data we use are daily based and we want to know whether the rainfall increases or decreases over 60 years. Then these daily precipitation records should be summed into yearly data for analysis. The other trend analysis type is the change of extreme value, for instance, how the maximum precipitation changes during decades of years. For precipitation of each year, peaks, that is maximum rainfall values should be averaged. After it, statistical methods can be performed on the averaged extreme rainfall value of each year to detect the trend.

The second type of data are products processed from satellite imageries, for example, MODIS. There are many MODIS products freely available on line, such as land surface temperature, NDVI and NPP (Net Primary Productivity), but data quality is not guaranteed. So we need to select data based on the data quality. But the selection is more or less empirical, depending on the purpose and the cloud coverage for the area of interest. For instance, at daily time interval, we may select land surface temperature products with less than 10% no-value data which indicate cloud contamination for a given catchment. After the quality control, we can use the products for some further analysis, such as estimating soil moisture or evapotranspiration.

Q: How many dimensions do these datasets have and do you have experience with NetCDF or databases?

A: For precipitation and discharge data itself, it is only one dimensional. But there are many stations normally, so that is also a three-dimensional problem, i.e., two spatial dimensions and one time dimension. For satellite's products, normally they contain two spatial dimensions and that is longitude and latitude. Normally we should analyze the temporal changes of what we observe from satellite images, so that is also a three-dimensional issue.

Except time series stored in ASCII files, remote sensing products we use are often stored in GeoTiff or IMAGE formats which are easy to process with mature proffessional softwares like ArcGIS. We do not often use NetCDF. Also, my group seldomly apply databases. There are standards for storing precipitation or flow data in China like which attributes should be recorded and how the metadata should be like. With these standards, it is convenient to manage data in databases. While for research, we usually store data in a casual and specific way to achieve high processing efficiency and

databases do not have the priority.

Q: Do you think big data will become a headache for hydrologists in the future? And do you have experience with compression or parallel processing techniques to improve efficiency of data processing?

A: Whether big data will become a problem or not depends on the aim and application. For my research, for example, mostly we focus on a certain area such as a basin. Even though in the future, data volume can get larger and larger, but for analysis in a certain area which only takes limited kinds of datasets as input, we do not require very high computing power. But for data centres which provide different datasets, I do think big data will be an issue.

As to compression, some remote sensing data have already been compressed and there are corresponding softwares to interpret these data. So there is no need to further compress them. For time series, we do not implement compression.

Parallelization is a potential technique. For instance when combining distributed hydrologic modelling with EnKF algorithm, the iteration process indeed costs lots of time and parallel structure would improve the performance.

Interview with Ir. Commandeur from Hydrologic Research B.V.

Q: I know you are one of the main developers of HydroNet, a hydrologic information system provides hydrometeorological services. So can you tell me types of queries frequently executed by you or your clients?

A: HydroNET is the core product of our company and through it, we provide services for hydrologic experts to perform their own analysis. Besides that, we also do our own hydrologic projects. On the whole, queries can be categorized into several types.

1. Sub-selection of original datasets. For example, select the rainfall data of Delft from a rainfall raster of the world. Sub-selection of time series is also included.
2. Extraction of time series from a set of rasters at different time steps.
3. Statistical operations. These includes simple mathematic calculations such as averaging or summing. A particular statistical problem is to determine extreme events from datasets. Taking precipitation as an example, this means locating pixels of maximum precipitation values from a spatial-temporal rainfall cubic dataset.
4. Determination of percentile for forecast datasets. Normally a set of ensembles are used in a forecast model to produce forecast datasets like precipitation forecast data. Before we use these forecast data, a specific percentile, e.g. 80^{th} percentile of the dataset has to be calculated to filter out large values.
5. Subtraction one layer from another layer. For example, to calculate the quantity of water held by the ground. We should subtract evapotranspiration data from precipitation data. A problem exists that the resolution of involved datasets may be different, so techniques like interpolation should be performed before the subtraction.

Q: What datasets are involved in the queries you mentioned and how many dimensions do these datasets have?

A: Queries are based on datasets of different dimensions. These datasets include,

1. Time series recorded by gauges like precipitation, discharge and temperature. They are one dimensional datasets.
2. Precipitation data derived from Doppler radar and a step further, calibrated radar precipitation data using data recorded by gauges. They include three dimensions, x, y and time.
3. Satellite data products like precipitation and soil moisture. And they are also consisted of two spatial dimensions and a temporal dimension.
4. Forecast datasets like GEFS precipitation forecast data including five dimensions, longitude, latitude, time, perturbation and model run time.
5. Orbiting satellite observation datasets. Data are recorded for different swaths and have x, y two dimensions.

Q: Which formats are these datasets stored with and do you also retrieve data from databases?

A: I usually get multidimensional datasets stored in GRIB 1/2, HDF5, NetCDF, BUFR and ASCII. To manage these different datasets, our company developed a NetCDF-1 system which can transform all these formats into NetCDF-1. Then our analysis and services can all be constructed on NetCDF. We also retrieve time series data from MySQL database constructed by data providers.

Q: What problems did you encounter with running queries either using the database or NetCDF?

A: Extraction time series from spatial layers takes a lot of time using NetCDF-1. Another problem with requesting data from MySQL database is that it is impossible to input the spatial coordinate of a station to get the recorded time series. You can only retrieve the data with station ID.

Q: Nowadays, big data is an issue facing many disciplines. As a water-ICT experts, what is your opinion about big data? Have you ever try compression or parallel processing techniques which may improve data processing efficiency?

A; Scientists and engineers have already proposed smart approaches to solve big data problem like cloud computing and distributed computing. For hydrology, it is not difficult to build workbench in cloud to tackle hydrologic big data issue. We have already use multi-threads approaches based on GPU to run hydrodynamic models and achieved performance gain. For data compression, I think it can help improve the query efficiency because I/O transfer is the most time consuming part for executing queries and compression can decrease the data transferred from disk to caches. .

Interview with Ing. Van der Wielen from Hydrologic B.V.

Q: You are the hydro-ICT engineer from Hydrologic and I learnt that you were constructing web services for providing water information. So could you tell me hydrologic queries executed most frequently either by clients or you?

A: I run several types of queries most frequently and they are,

1. For time series data like precipitation data collected by specific gauges, we do sub-selection and summation.
2. Extracting time series from grid layers.
3. Subtraction of accumulated data. Instead of storing variable at specific intervals, we store the accumulated values. For example, for rainfall observed every five minutes, the original dataset may be 5, 5, 5..., while we store accumulated data for convenience of certain queries and accumulated dataset should then be 5, 10, 15... So a query can be what is the amount of rainfall for a certain hour and this is concerned with subtraction of the dataset at start and end time steps.
4. Combining data of two different grids. An example is actual precipitation deficit, which is precipitation minus evaporation. The precipitation grid is in a different projection from the evaporation grid, which means the grids need to be intersected. Each cell in the evaporation grid needs to know the percentage of intersecting with precipitation grid cells. Calculating / retrieving this intersection is currently the most intensive task. Later we do summation of the combined data along temporal dimension.
5. Intersecting polygons with grids, a spatial operation.
6. We are also working on a new application where colors will be applied to polygons depending on the underlying values based on intersection of polygons and grids.

Q: What are the datasets these queries based on? How are these datasets stored?

A: As is elaborated partly earlier, datasets are,

1. Time series data for certain locations, 2D including time and location.
2. Radar data, 3D including x, y and time.
3. Results from hydrologic model, 4D including x, y, time and model run date
4. Forecast data, 5D including x, y, time, ensemble and model run date

We use MySQL to store some point time series data while for nearly all grid data, we apply NetCDF classic format.

Q: What problems did you encounter with executing queries using the data format or database?

A: The biggest problem right now is intersecting grids from different projections. Actually intersection operation is not very serious. But problem lies that we store intersections in cache, which is of course volatile and data are removed when restarting the API, for example. And this is

not efficient. Besides, as is mentioned earlier, although using accumulated grids can bring convenience for some scenarios, it also has drawbacks. For instance, once a time layer is added which is originally missing will lead to the recalculation of values at all following time steps. Another problem is that currently our WMS service can only read the complete grid rather than the extent to which the user is zoomed in. This may cause problems with large grids.

Q: Do you think big data will become a problem in hydrologic domain in the future? We focus big data size here.

A: Yes, big data will be a challenge for us because we will soon have to deal with very large data sets: meteorologic data of Yemen containing 1.2 billion cells of 30x30m and evaporation data in the Netherlands at 8x8m cell size.

As a matter of fact, big data will not be the issue if the data is static. However, in reality, normally dynamic dimensions like time and ensembles are involved, which then cause the problem.

Q: Do you have experience with parallel query processing or data compression techniques which can improve the query efficiency?

A: Accumulated grids as mentioned before has an enormous performance gain for some scenarios, but it has shortcomings. I have experience with multi-threads calculation using CUDA, which helps solve intensive calculations on data in memory. But I think bottleneck lies in the hard drive speed, not memory speed. I don't use parallel query processing and compression techniques. While I guess compression will speed up data retrieval from the hard drive, but it depends on the speed of the decompression algorithm. There's probably a break-even point somewhere depending on the scale of the retrieved data size and also the CPU speed and HDD speed.

In the near future we will develop a scaling algorithm to create pyramids, i.e. coarser versions of the original data set. If a user is zoomed out to a large extent, he will see the coarse data rather than the fine data. So preprocessing pyramids can be a technique to improve query efficiency.

Reply from Ir. Villa Real in IBM using questionnaire

Q1: What kind of queries do you execute most on hydrological/multidimensional data?
(Query mainly refers to queries executed with SQL, or simple calculations that can be expressed using SQL. Complex computation based on data, for instance data assimilation is not the query. Some query examples are provided below,
E.g.1 aggregation, from monthly rainfall to yearly rainfall
E.g.2 selection of subset, select data from 2001 to 2010 from original dataset which ranges from 1990 to 2013
E.g.3 Some soil moisture datasets have a flag field which indicates the accuracy of the data, so if the data value exceed the accuracy threshold, then this data value should not be selected and used.
Above queries are just examples, do not be constrained by these examples.)

A1: The input data we have is comprised of various NetCDF files, each of which representing a specific feature of the area of interest (e.g. elevation data, soil type, land use, initial soil saturation conditions). Most of these grids are processed once (we crop them to the extents of interest, sanitize them to make sure cells have values that make sense, etc) and used many times. The only flag which we utilize is the so-called "NODATA", that masks out certain parts of the grids from the simulation. Anyhow, given that our tools deal with NetCDF directly we do not issue any SQL queries. Rather, we simply read the files and skip the areas which are not of interest to us during runtime.

Q2: For queries you mentioned above, which datasets are they based on? How many dimensions do these datasets have?
(Dimension refers to the property to confine the data of interest. For example, in a 5 dimensional precipitation forecast dataset, the data of interest is of course the precipitation. While it contains latitude and longitude dimensions, a temporal dimension, a perturbation dimension which indicates the ensembles (normally 20 times). And the last dimension is the time you run the forecast model.)

A2: Besides the single-variable NetCDF files mentioned above, we too have precipitation grids produced by WRF. Each of these grids is given as NetCDF and variables have typically a resolution of 1x90x90 – meaning it's a 1-dimensional grid with a resolution of 90x90. We care about 5 variables at most, whereas the original NetCDF file comes with more than 200 variables. To save space we remove the variables that we do not care about with the NCO tools, and then transfer the modified files to the computer(s) that will utilize them. It's worth mentioning that rather than having multi-dimensional precipitation variables, we have multiple NetCDF files, each of which containing data for a given simulation time step.

Q3: Which format (which version, for example NetCDF has version 1 to 4) is used to store the datasets you mentioned or are they stored in the database?
(Besides, you may also provide some general remarks about database or data formats used by hydrologists)

A3: We use NetCDF version 3 for now, but are intending to upgrade to NetCDF 4 for larger datasets.

Q4: What problems did you encounter with executing queries using the data format or database? (Are they efficient for running queries? Are the data retrieved expected by you, i.e. may be the data returned contains only part of the data you wanted?)

A4: All problems that we had so far had to do with two things: (a) parsing NetCDF files can be tricky because of the lack of official ways (i.e. flags) to tell if a file is turned upside down or not, and (b) the NetCDF format has several limitations on the size of the grids that it can hold, even when 64-bit support is enabled and used. Regarding performance we have found in Parallel NetCDF a great way to improve I/O, although there are some considerations that need to be addressed in the software to properly use it.

Q5: Do you think big data (mainly refer to big size) is a problem or will the big data become a problem for hydrologists in the future?
(Have you ever thought about it?)

A5: It already is a problem today, especially for those working with large domains (say, a whole continent) or with high resolution data such as LiDAR. Satellites are becoming more accessible than ever, and they also have increased resolution as compared with former products (e.g. VIIRS, LANDSAT). The combination of the various datasets used in hydrological models is therefore a problem that needs to be addressed somehow at the storage level and from an I/O perspective.

Q6: Do you have experience with techniques like parallel query processing or data compression which can improve the query efficiency?

A6: I have been using Parallel NetCDF (which relies on MPI infrastructure) with a degree of success. There is still work going on to improve performance using different load balancing techniques, but we were able to have dramatic performance gains with its use.

With regard to data compression, I have evaluated HDF5 with different compression settings. The impact of the on-the-fly decompression was significant, though, making it prohibitive to operational flood forecasting. I have also attempted to save the static data in a compressed file system called SquashFS, but the very same observation with regard to performance apply.

It seems to me that Huffman-based compression algorithms are not the way to go when both space and performance need to be optimized. One possible approach is to employ the use of file systems with de-duplication support. De-duplication works at the block layer, meaning that two if two blocks are the same then it will be saved only once in the hard disk. There is no performance degradation to read the file back, because there is nothing to decompress. I have not tested such file systems in the realm of hydrology yet, however. My experience with that comes from evaluation of storing dozens of virtual machine images on said file systems.

Appendix C: Two HydroNET-4 data structures

```
Grid{
    GridDescription{
        CellHeight,
        CellWidth,
        Columns,
        Rows,
        XLL,
        YLL
    },
    Start date,
    End date,
    Interval,
    Projection,
    Variable{
        Values
    },
    Data type,
    NoDataValue,
    Availability,
    Quality
}

TimeSeries{
    Location{
        X, Y
    },
    Start date,
    End date,
    Interval,
    Projection,
    Variable{
        Values,
        Dates,
        Qualities,
        Availabilities
    },
    Data type,
    NodataValue
}
```

Appendix D: Configuration file of SciDB

```
[hydronet]
server-0=localhost,0
db_user=hydrologic
db_passwd=hydroresearch
install_root=/opt/scidb/14.3
pluginsdir=/opt/scidb/14.3/lib/scidb/plugins
logconf=/opt/scidb/14.3/share/scidb/log4cxx.properties
base-path=/home/scidb/data
tmp-path=/tmp
base-port=1239
interface=eth0
enable-catalog-upgrade=true
max-memory-limit=3072
```

Appendix E: Communicating records with SciDB team

Issue	Web link	Time
NetCDF loader for SciDB	http://www.scidb.org/forum/viewtopic.php?f=13&t=1309	18/03/2014
SciDB interface on Windows	http://www.scidb.org/forum/viewtopic.php?f=13&t=1337	17/04/2014
Inserting data records into SciDB with AFL	http://www.scidb.org/forum/viewtopic.php?f=11&t=1348	06/05/2014
Retrieving storage details of chunks and arrays from SciDB	http://www.scidb.org/forum/viewtopic.php?f=13&t=1368	11/06/2014
Possibility to load CSV file directly into a final multidimensional array without one dimensional load array	http://www.scidb.org/forum/viewtopic.php?f=11&t=1378	24/06/2014
Specific implementation of SciDB operator “filter”, “between” and “subarray”	http://www.scidb.org/forum/viewtopic.php?f=13&t=1388	14/07/2014
Deleting array versions, which causes significant overload when load large datasets	http://www.scidb.org/forum/viewtopic.php?f=11&t=1393	16/07/2014
Importing 5D ensemble forecast data into SciDB	http://www.scidb.org/forum/viewtopic.php?f=11&t=1397.	24/07/2014
Tool to observe SciDB resource usage in real time	http://www.scidb.org/forum/viewtopic.php?f=11&t=1387	30/07/2014
Increasing memory usage of SciDB after starting up, a problem with utilization of small chunk size for storage	http://www.scidb.org/forum/viewtopic.php?f=11&t=1437	02/09/2014
SciDB instance setting, run length encoding for storage and influence of using unlimited dimension on query performance	http://www.scidb.org/forum/viewtopic.php?f=11&t=1441	06/09/2014
Influence of the order of dimension on SciDB array storage structure and query performance	http://www.scidb.org/forum/viewtopic.php?f=11&t=1449	29/09/2014
Chunk size and total array storage size	http://www.scidb.org/forum/viewtopic.php?f=11&t=1455	07/10/2014

Appendix F: Bash scripts for GEFS test on dimensions order effect

Script 1

```
#!/bin/bash
#Test on SciDB_GEFS_S1, SciDB_GEFS_S2, SciDB_GEFS_S3 and
SciDB_GEFS_S4

for (( i=1; i<=10; i++ ))
do
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(project(between(GEFS_S1,675,0,17,38,184,675,19,17,38,184
),APCP))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(project(between(GEFS_S2,675,17,38,184,0,675,17,38,184,19
),APCP))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(project(between(GEFS_S3,184,38,17,0,675,184,38,17,19,675
),APCP))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(project(between(GEFS_S4,0,184,38,17,675,19,184,38,17,675
),APCP))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(aggregate(project(between(GEFS_S1,675,0,17,38,184,675,19
,17,38,184),APCP),avg(APCP),X_idx,Y_idx,F_idx))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(aggregate(project(between(GEFS_S2,675,17,38,184,0,675,17
,38,184,19),APCP),avg(APCP),X_idx,Y_idx,F_idx))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(aggregate(project(between(GEFS_S3,184,38,17,0,675,184,38
,17,19,675),APCP),avg(APCP),X_idx,Y_idx,F_idx))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(aggregate(project(between(GEFS_S4,0,184,38,17,675,19,184
,38,17,675),APCP),avg(APCP),X_idx,Y_idx,F_idx))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
```

```
done
```

Script 2

```
#!/bin/bash
#Test on SciDB_GEFS_C1, SciDB_GEFS_C2, SciDB_GEFS_C3 and
SciDB_GEFS_C4

for (( i=1; i<=10; i++ ))
do
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(project(between(GEFS_C1,675,0,17,38,184,675,19,17,38,184
),APCP))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(project(between(GEFS_C2,675,17,38,184,0,675,17,38,184,19
),APCP))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(project(between(GEFS_C3,184,38,17,0,675,184,38,17,19,675
),APCP))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(project(between(GEFS_C4,0,184,38,17,675,19,184,38,17,675
),APCP))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(aggregate(project(between(GEFS_C1,675,0,17,38,184,675,19
,17,38,184),APCP),avg(APCP),X_idx,Y_idx,F_idx))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(aggregate(project(between(GEFS_C2,675,17,38,184,0,675,17
,38,184,19),APCP),avg(APCP),X_idx,Y_idx,F_idx))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(aggregate(project(between(GEFS_C3,184,38,17,0,675,184,38
,17,19,675),APCP),avg(APCP),X_idx,Y_idx,F_idx))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
    /usr/bin/time -f "%e" -a -o GEFS_localtest iquery -anq
"consume(aggregate(project(between(GEFS_C4,0,184,38,17,675,19,184
,38,17,675),APCP),avg(APCP),X_idx,Y_idx,F_idx))"
    sudo sh -c 'echo 3 >/proc/sys/vm/drop_caches'
done
```

Appendix G: Benchmark figures

MPE benchmark figures

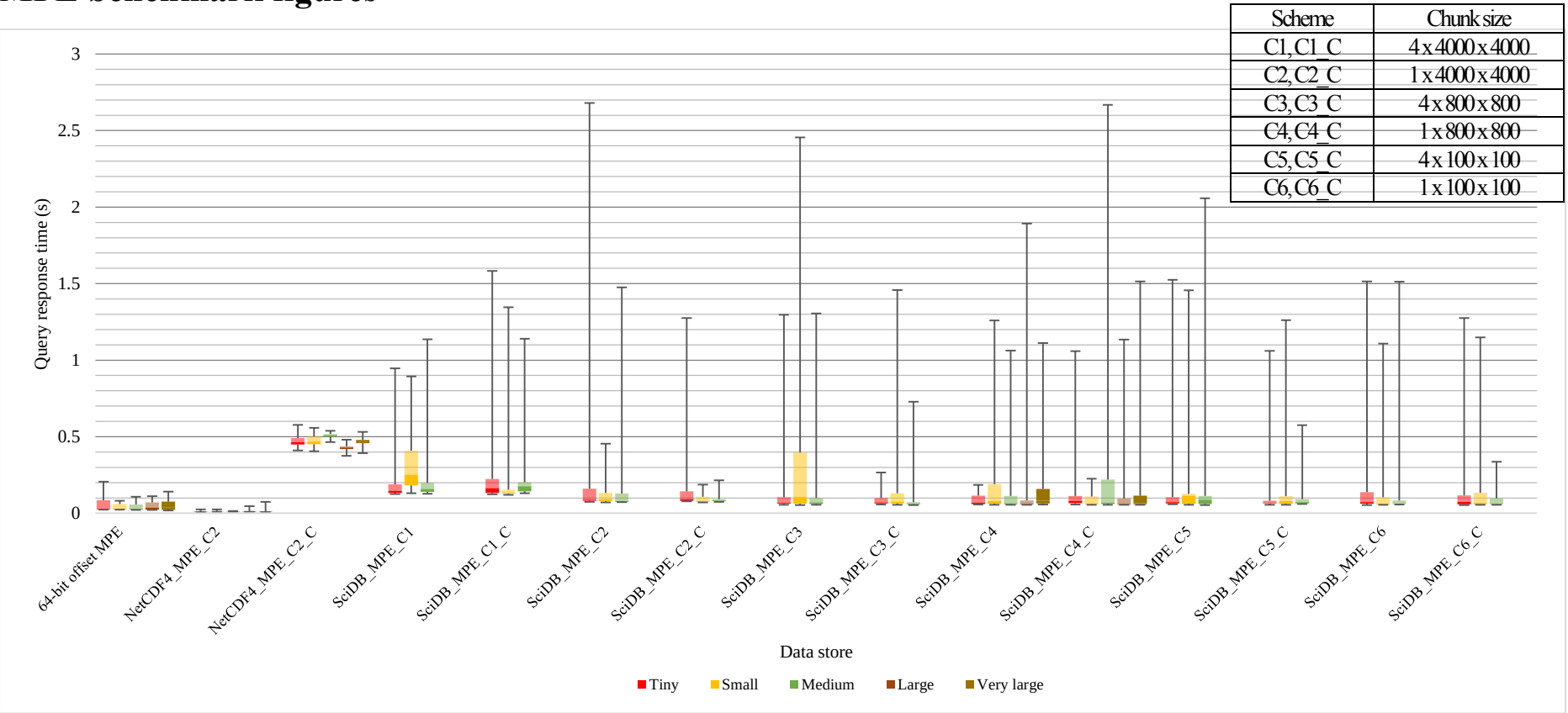


Figure G1. Distribution of 20 benchmark measurements of each data solution for retrieving the grid covering Delft at one time step.

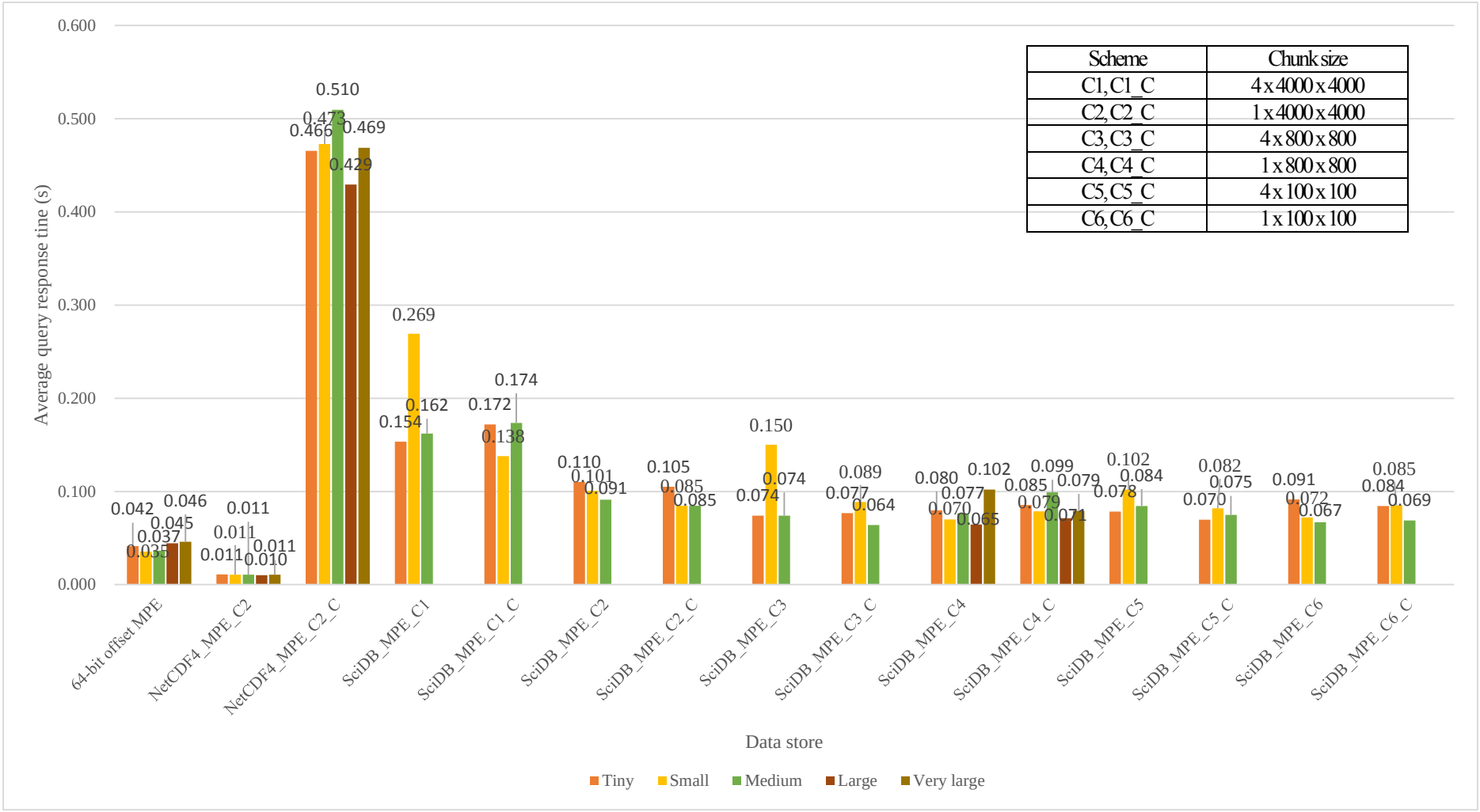


Figure G2. Performance of diverse data solutions for retrieving the grid covering Delft at one time step

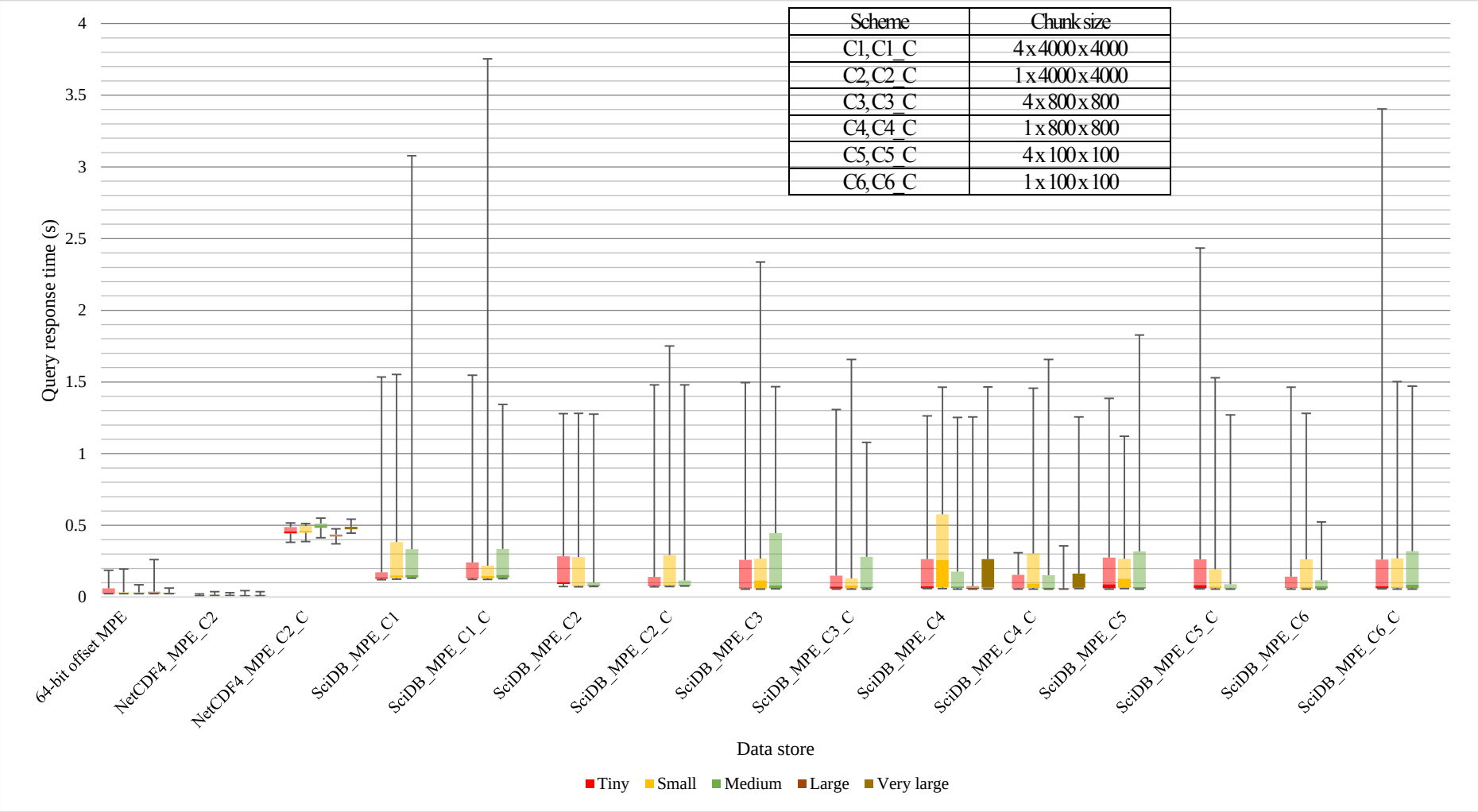


Figure G3. Distribution of 20 benchmark measurements of each data solution for retrieving the grid covering northern part of the Netherlands at one time step

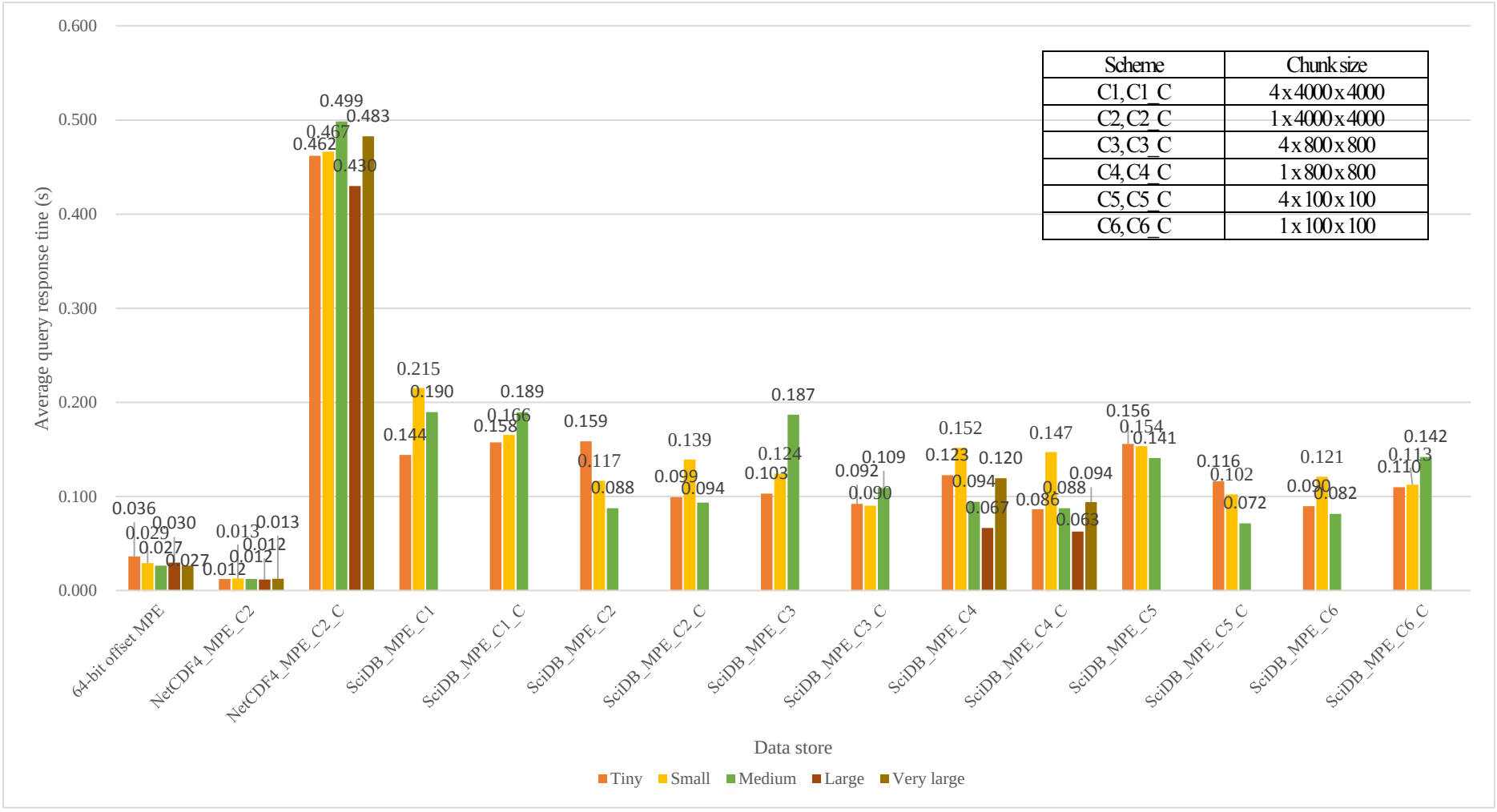


Figure G4. Performance of diverse data stores for retrieving the grid covering northern part of the Netherlands.

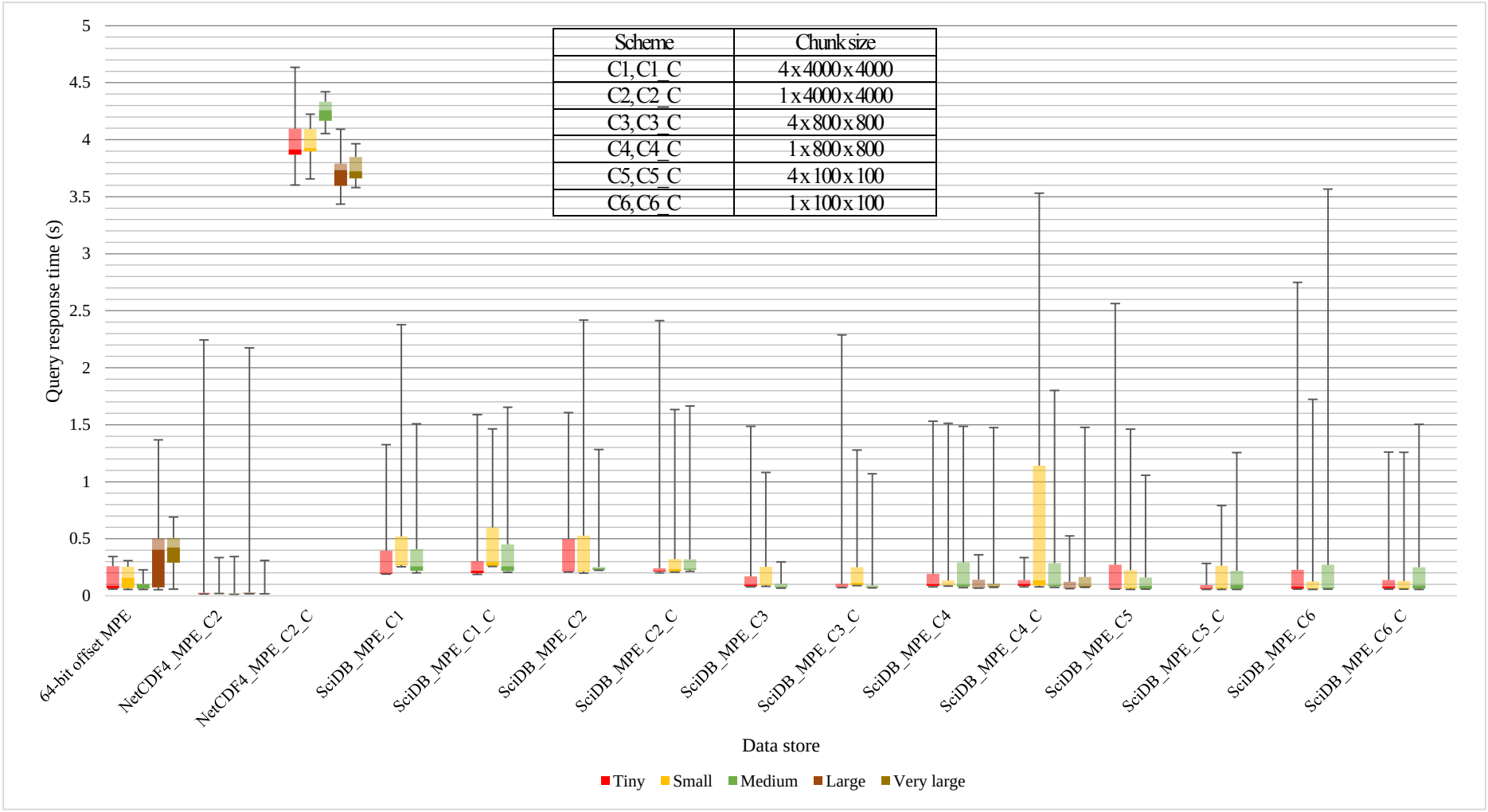


Figure G5. Distribution of 20 benchmark measurements of each data solution for retrieving 8-step time series from a spot location in the Indian Ocean.

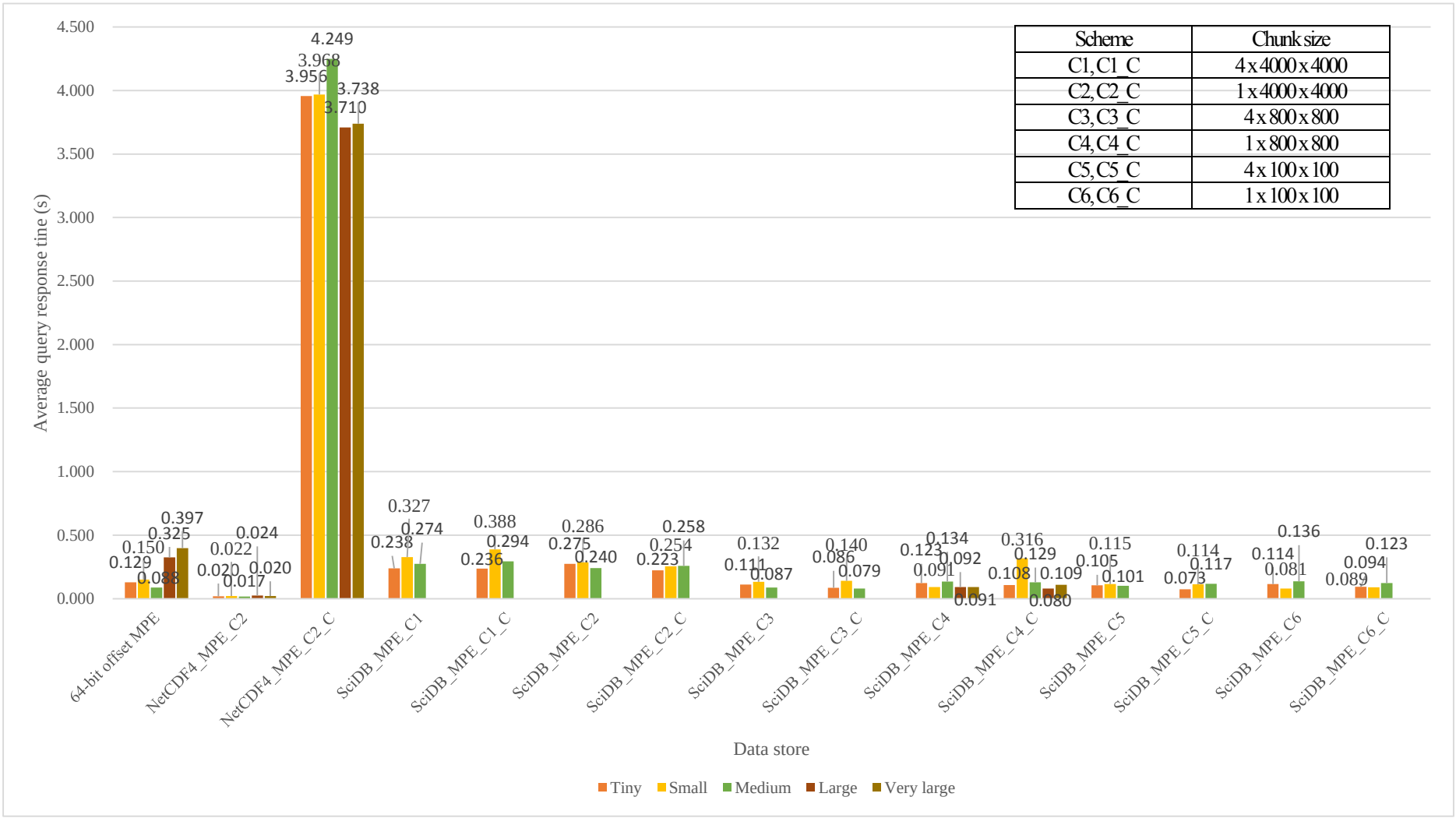
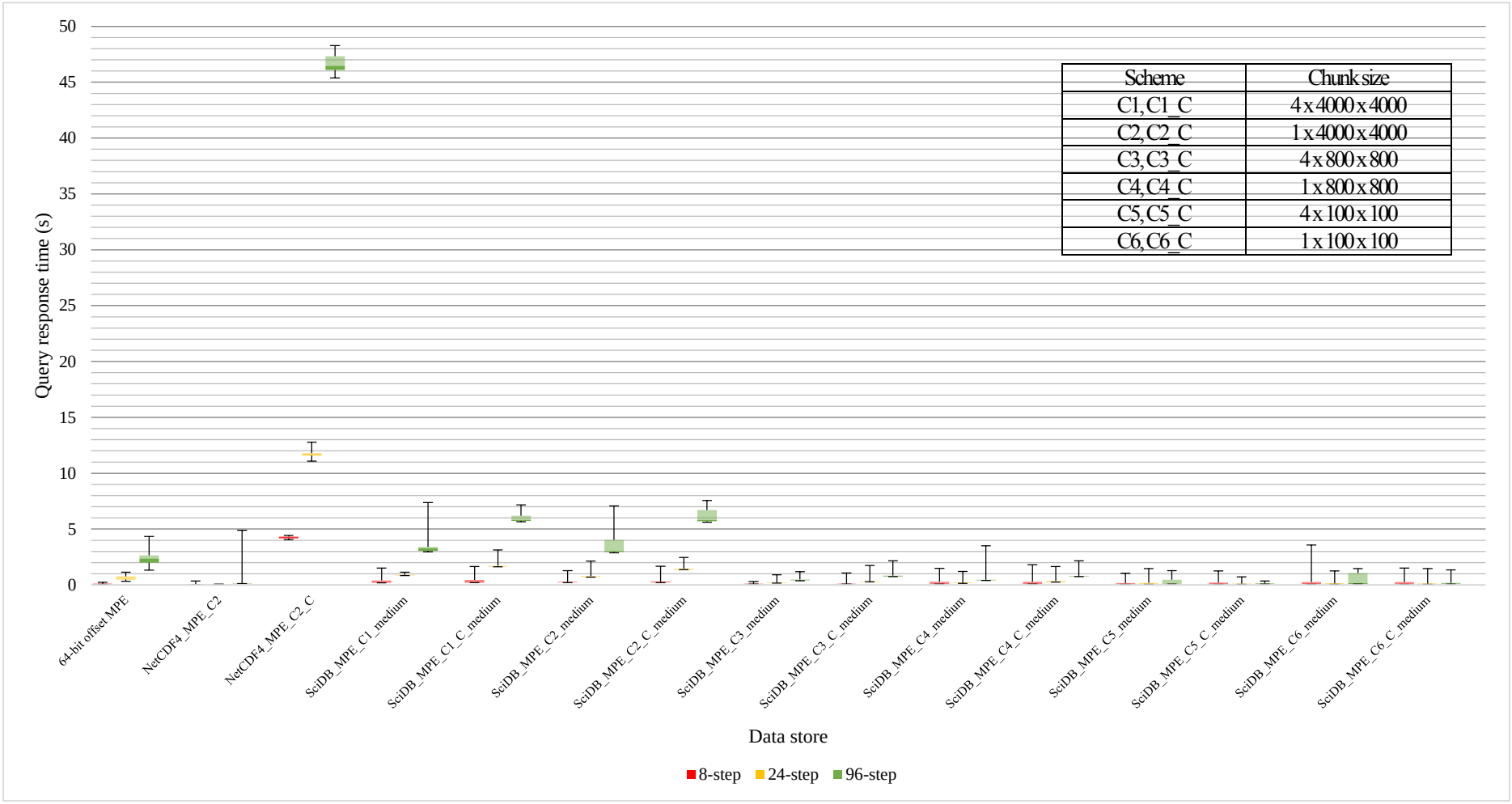


Figure G6. Performance of diverse data stores for retrieving 8-step time series from a spot location in the Indian Ocean.



Scheme	Chunk size
C1, C1_C	4 x 4000 x 4000
C2, C2_C	1 x 4000 x 4000
C3, C3_C	4 x 800 x 800
C4, C4_C	1 x 800 x 800
C5, C5_C	4 x 100 x 100
C6, C6_C	1 x 100 x 100

Figure G7. Distribution of 20 benchmark measurements of each data solution at medium level for retrieving time series of different lengths from a spot location in the Indian Ocean.

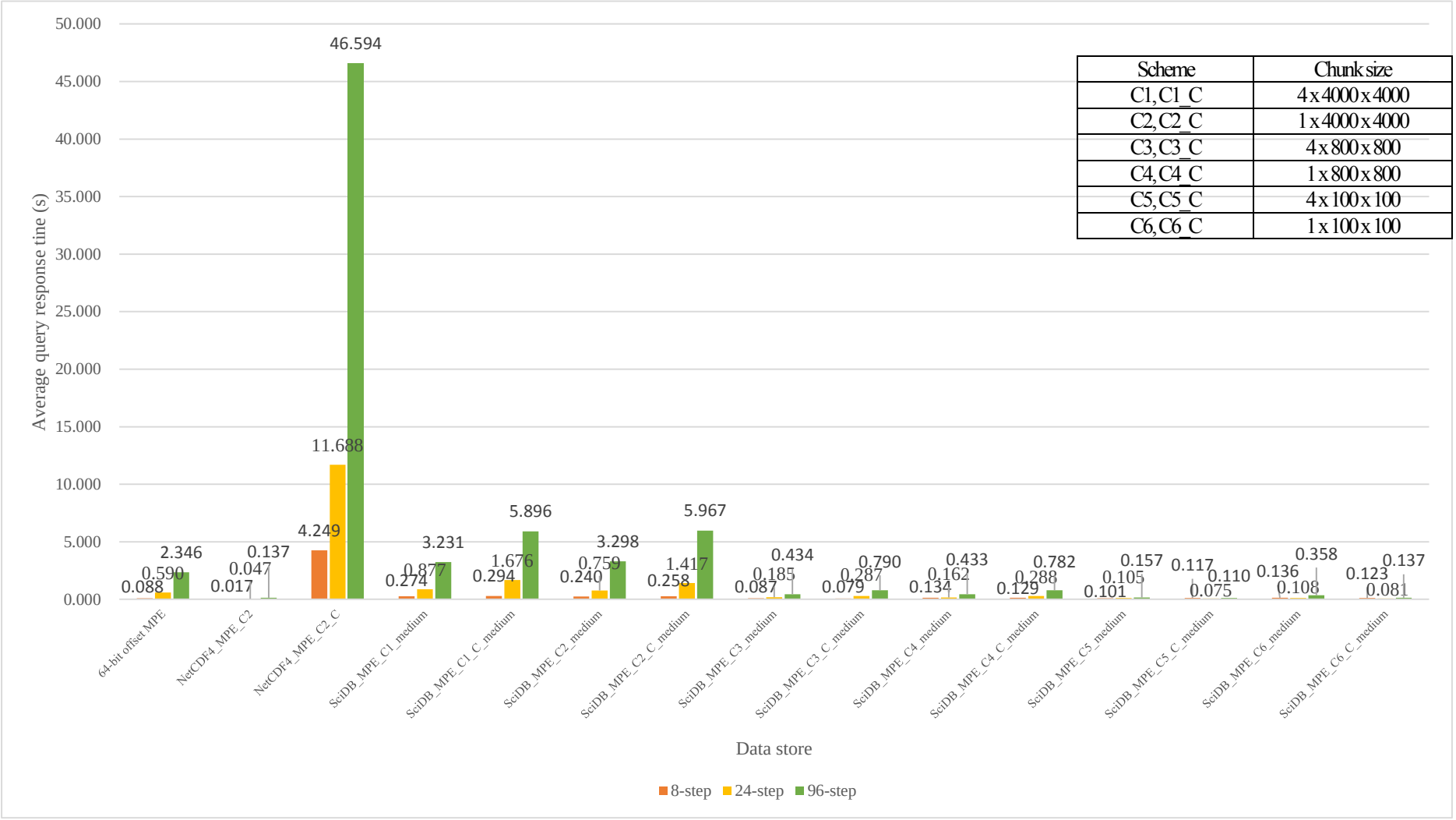


Figure G8. Performance of diverse data solutions at medium level for retrieving time series of diverse lengths from a spot location in the Indian Ocean.

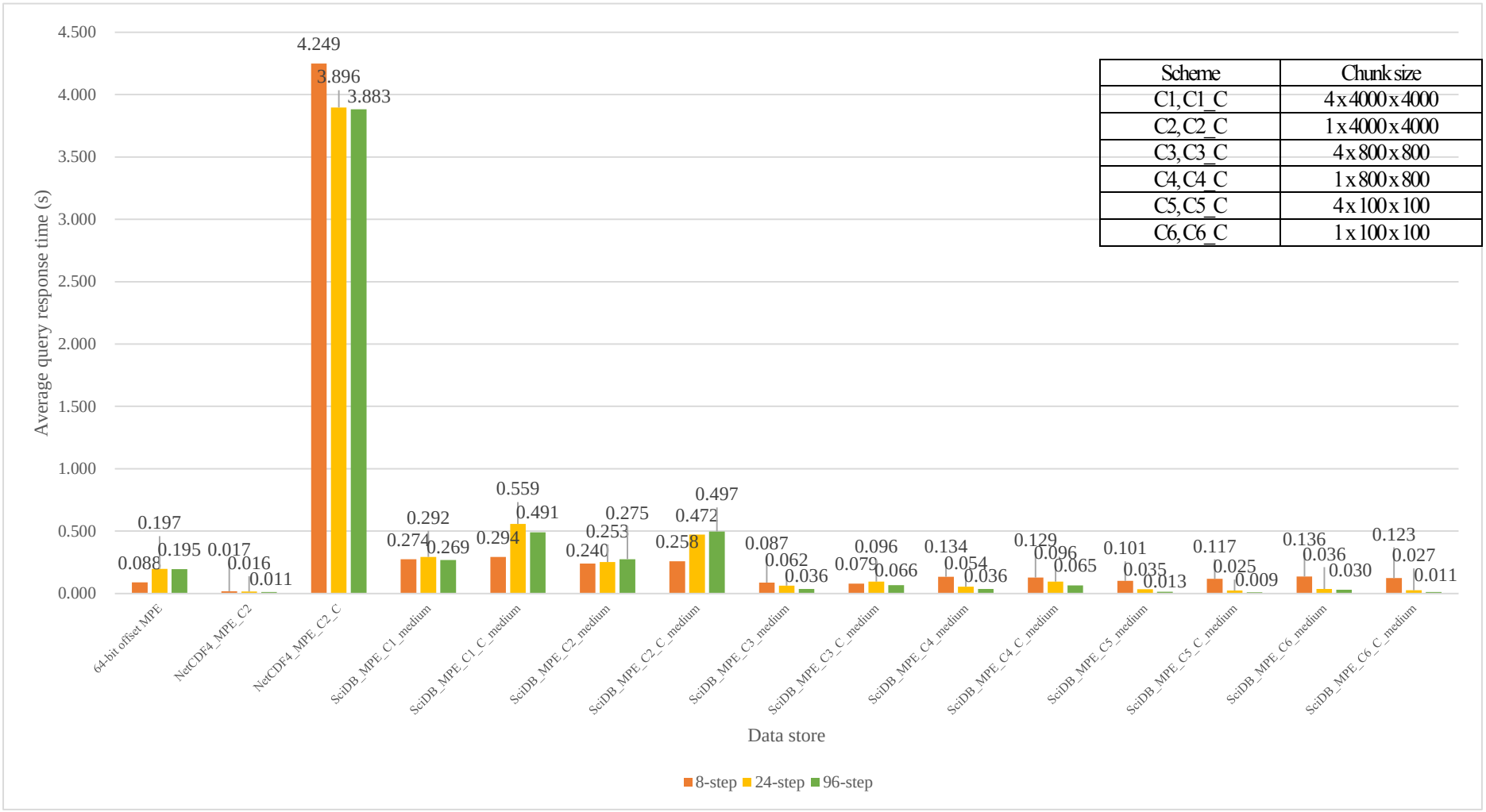


Figure G9. Aligned performance of diverse data solutions at medium level for retrieving time series of different lengths from a spot location in the Indian Ocean.

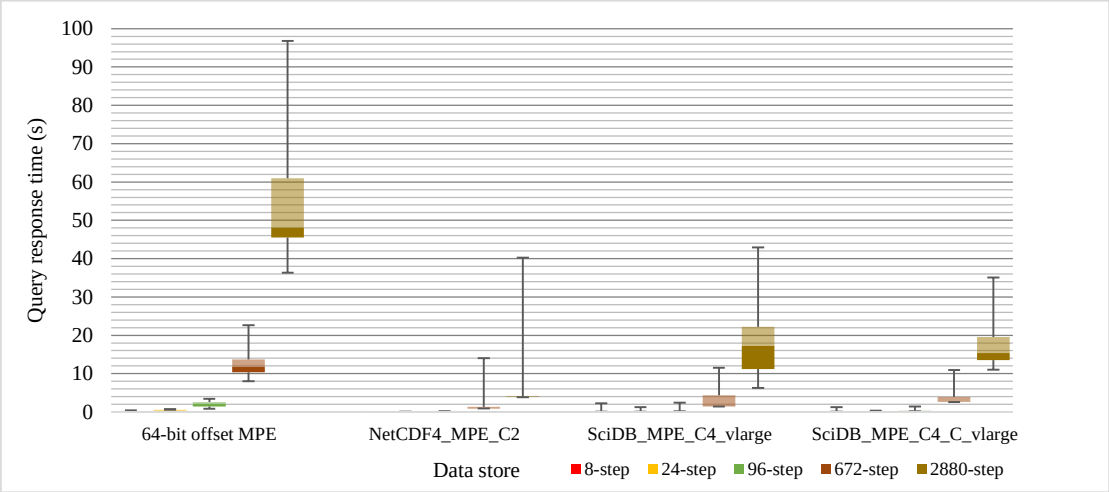


Figure G10. Distribution of 20 benchmark measurements for each data solution at very large level for retrieving time series of different lengths from a spot location in the Indian Ocean

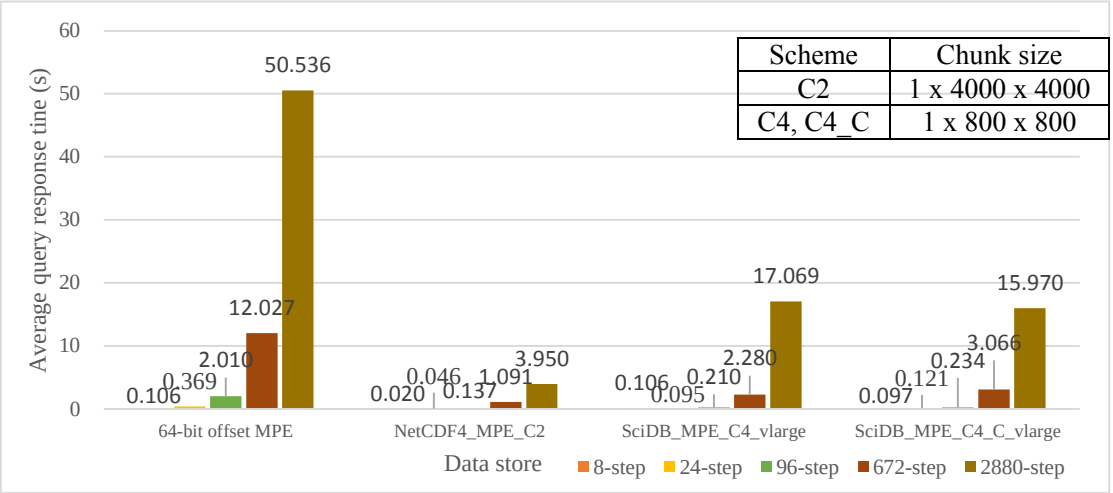


Figure G11. Performance of diverse data solutions at very large level for retrieving time series of different lengths from a spot location in the Indian Ocean

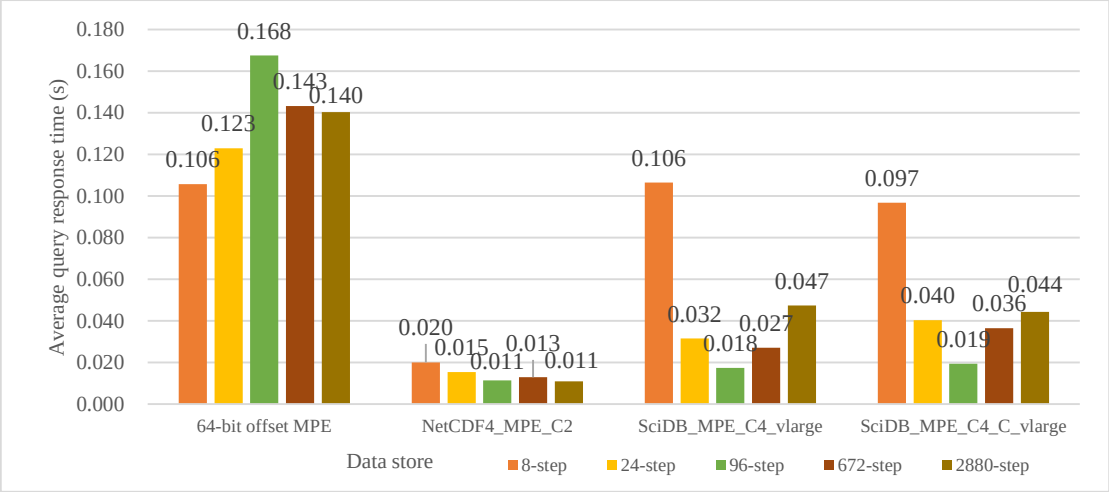


Figure G12. Aligned performance of diverse data solutions at very large level for retrieving time series of different lengths from a spot location in the Indian Ocean

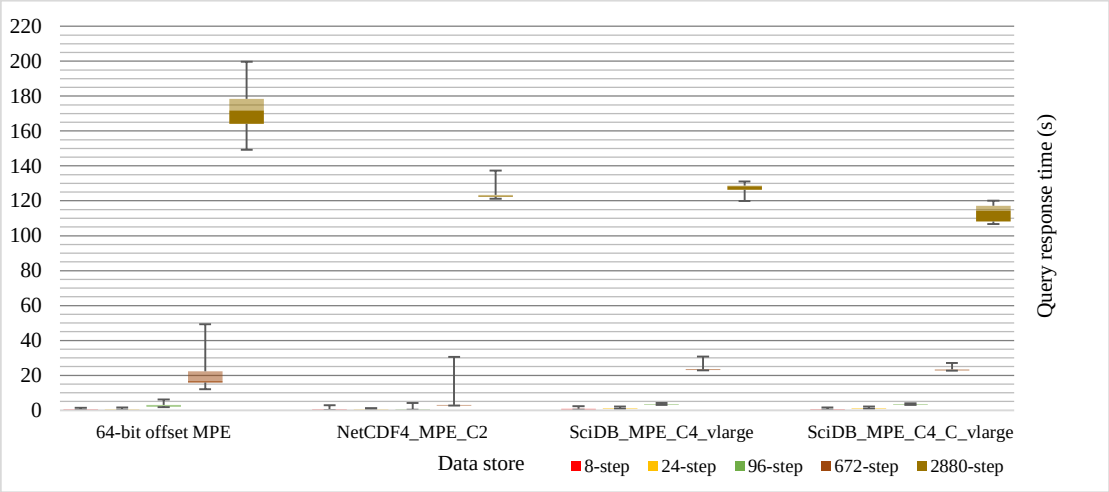


*Figure G13. Distribution of 20 benchmark measurements for each data solution at very large level for **average** calculation with different time steps at the Netherlands scale*

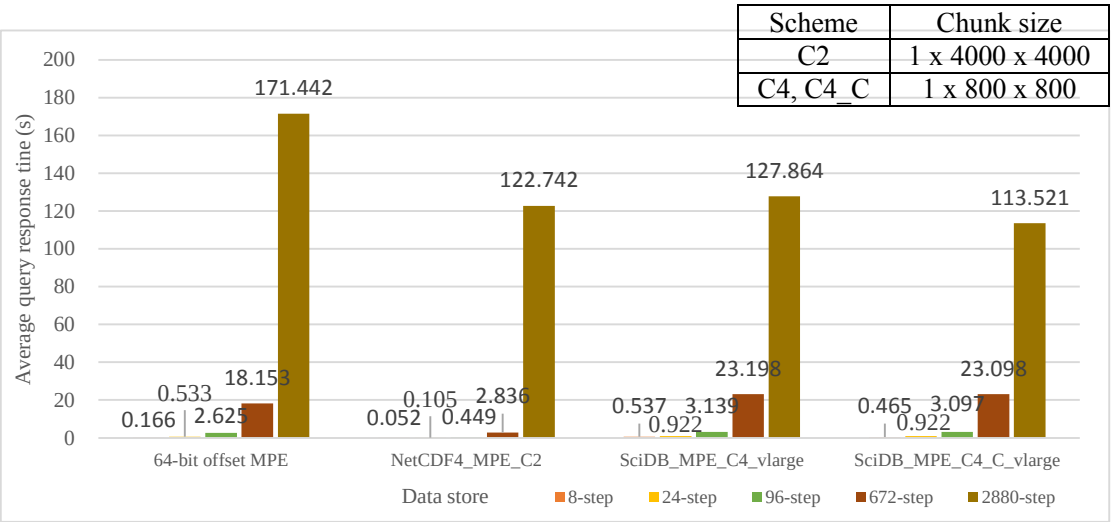


*Figure G14. Performance of diverse data solutions at very large level for **average** calculation with different time steps at the Netherlands scale*

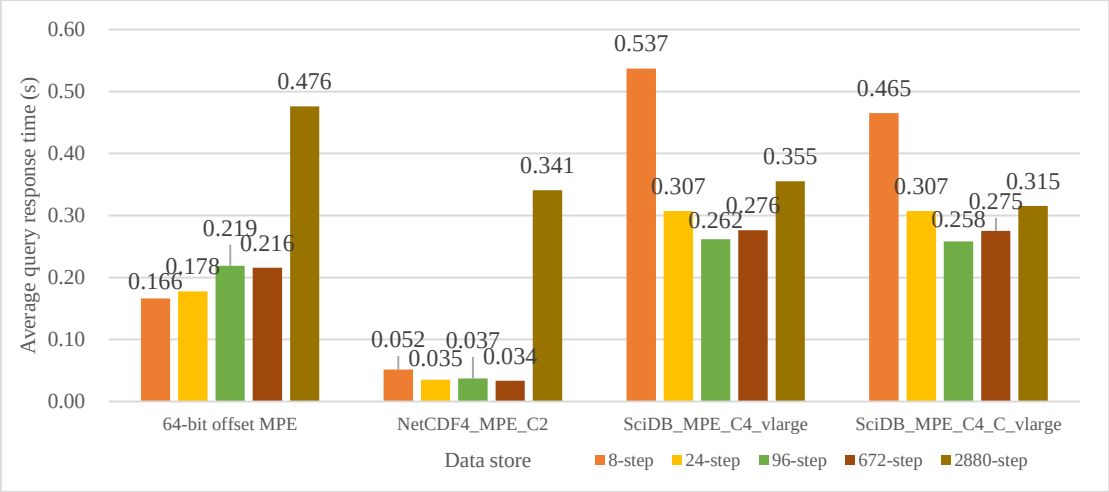


*Figure G15. Aligned performance of diverse data solutions at very large level for **average** calculation with different time steps at the Netherlands scale*

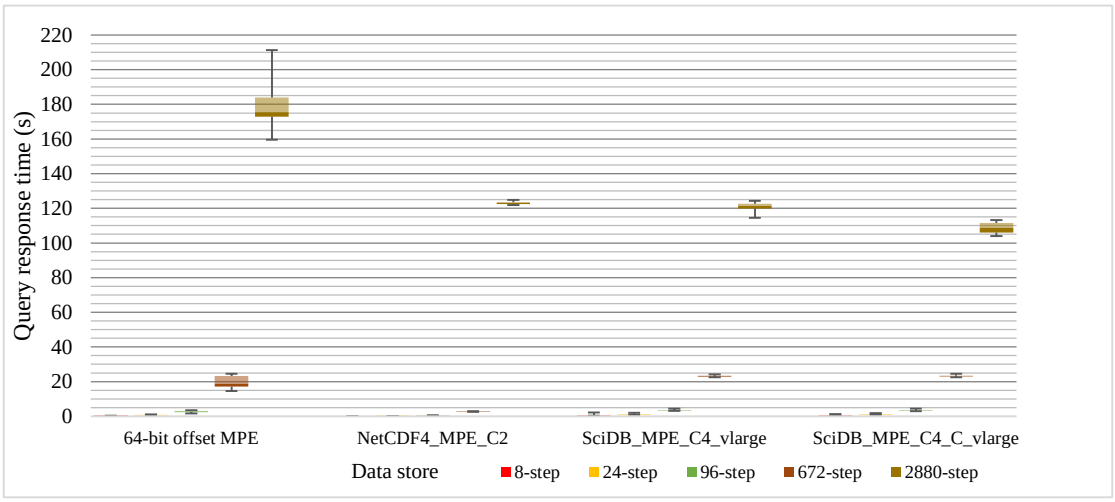


*Figure G16. Distribution of 20 benchmark measurements for each data solution at very large level for **maximum** calculation with different time steps at the Netherlands scale*

Scheme	Chunk size
C2	1 x 4000 x 4000
C4, C4_C	1 x 800 x 800

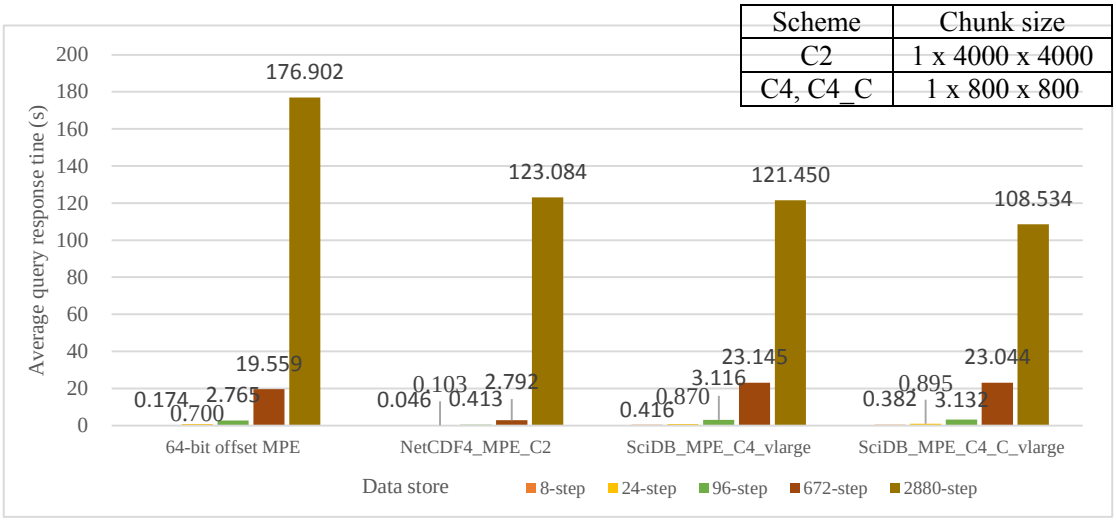


*Figure G17. Performance of diverse data solutions at very large level for **maximum** calculation with different time steps at the Netherlands scale*

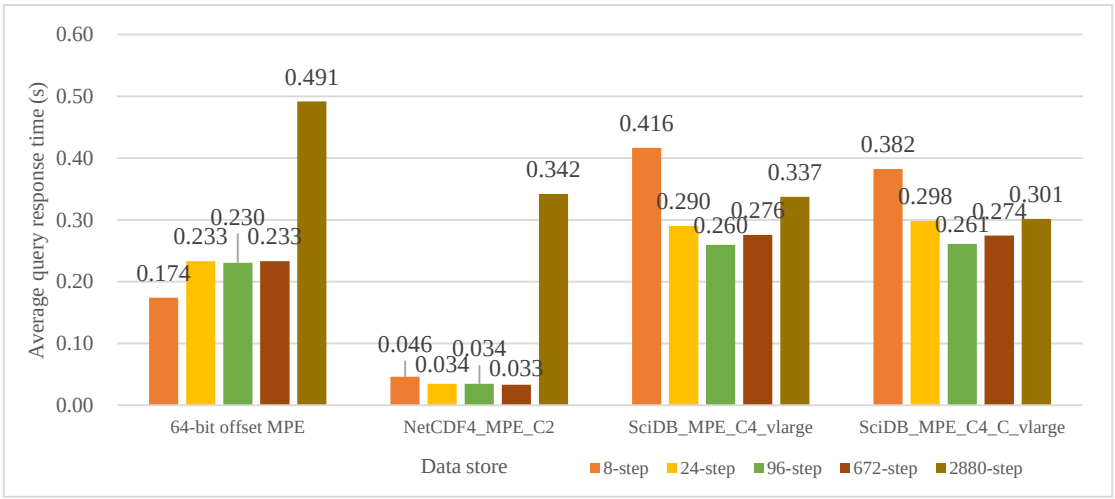


*Figure G18. Aligned performance of diverse data solutions at very large level for **maximum** calculation with different time steps at the Netherlands scale*

GEFS benchmark figures

(Note: only box plots with 25th and 75th percentile of measurements are shown, average performace are ploted in Figure 6.14 to 6.18)

Scheme	Dimensions order	Chunk size
S1	M E F Y X	1 x 20 x 1 x 181 x 360
S2	M F Y X E	1 x 1 x 181 x 360 x 20
S3	X Y F E M	360 x 181 x 1 x 20 x 1
S4	E X Y F M	20 x 360 x 181 x 1 x 1

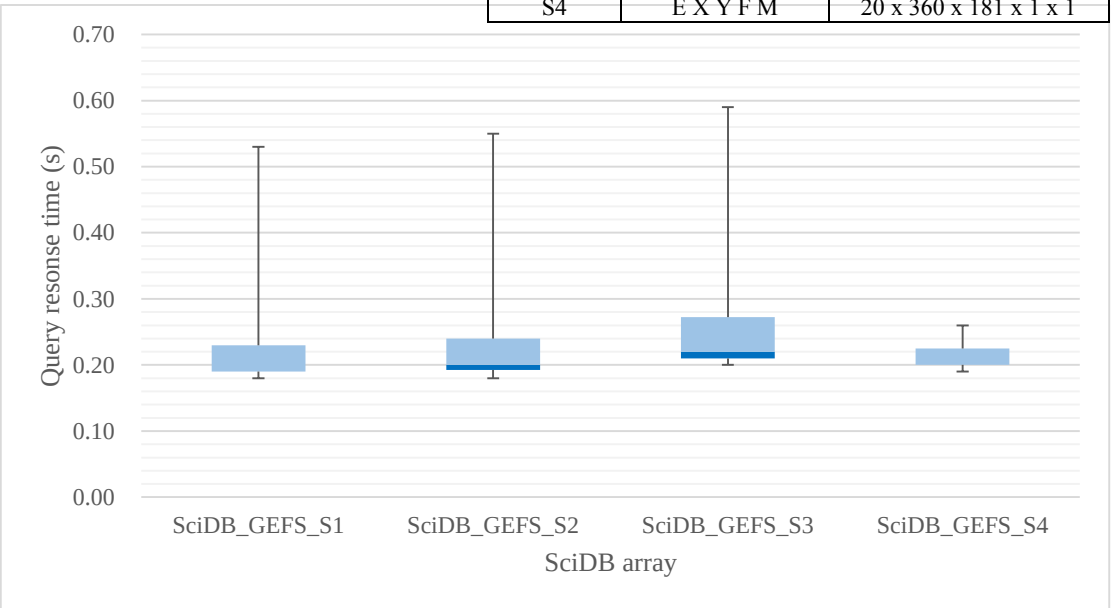


*Figure G19. Distribution of 10 benchmark measurements tested locally for selecting all ensembles of total precipitation in Delft at one forecast step (**Modest** chunk size)*

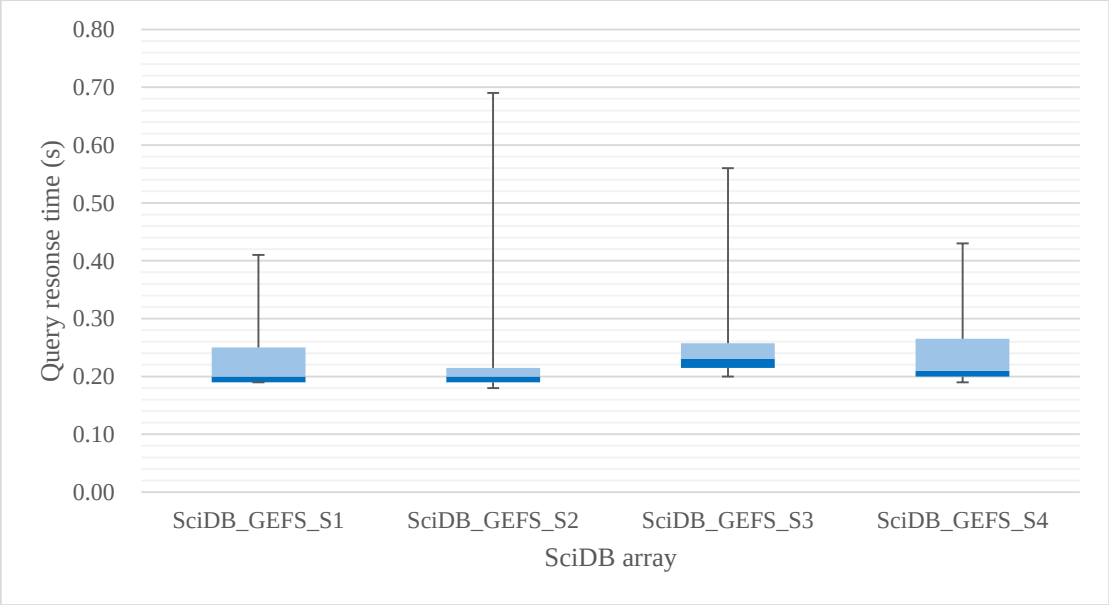


*Figure G20. Distribution of 10 benchmark measurements tested locally for calculating ensemble mean of total precipitation in Delft at one forecast step (**Modest** chunk size)*

Scheme	Dimensions order	Chunk size
C1	M E F Y X	1 x 20 x 40 x 181 x 360
C2	M F Y X E	1 x 40 x 181 x 360 x 20
C3	X Y F E M	360 x 181 x 40 x 20 x 1
C4	E X Y F M	20 x 360 x 181 x 40 x 1

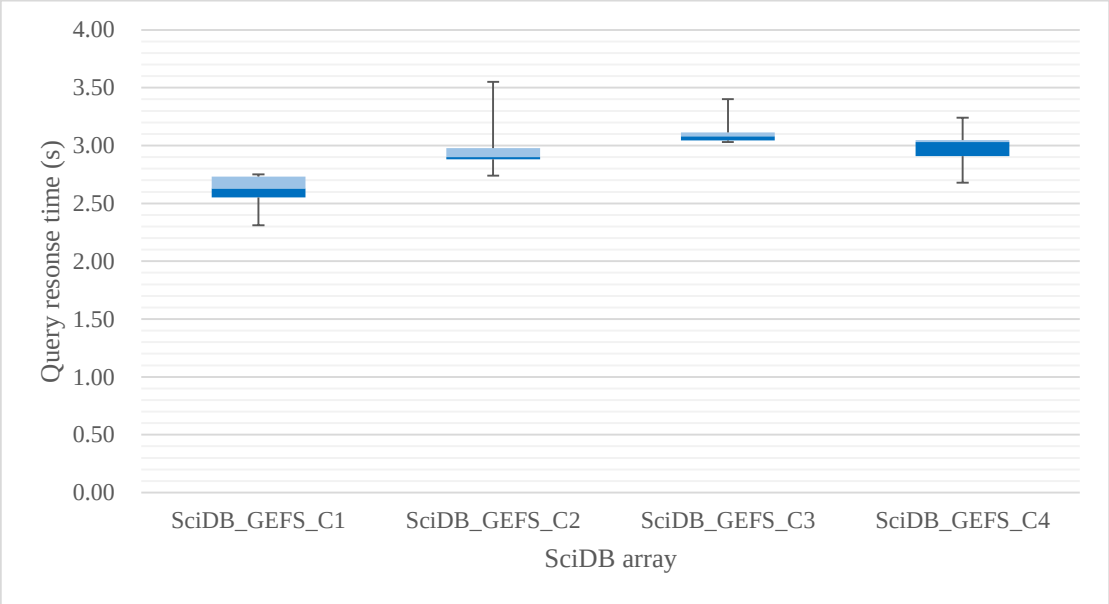


*Figure G21. Distribution of 10 benchmark measurements tested locally for selecting all ensembles of total precipitation in Delft at one forecast step (**Large** chunk size)*

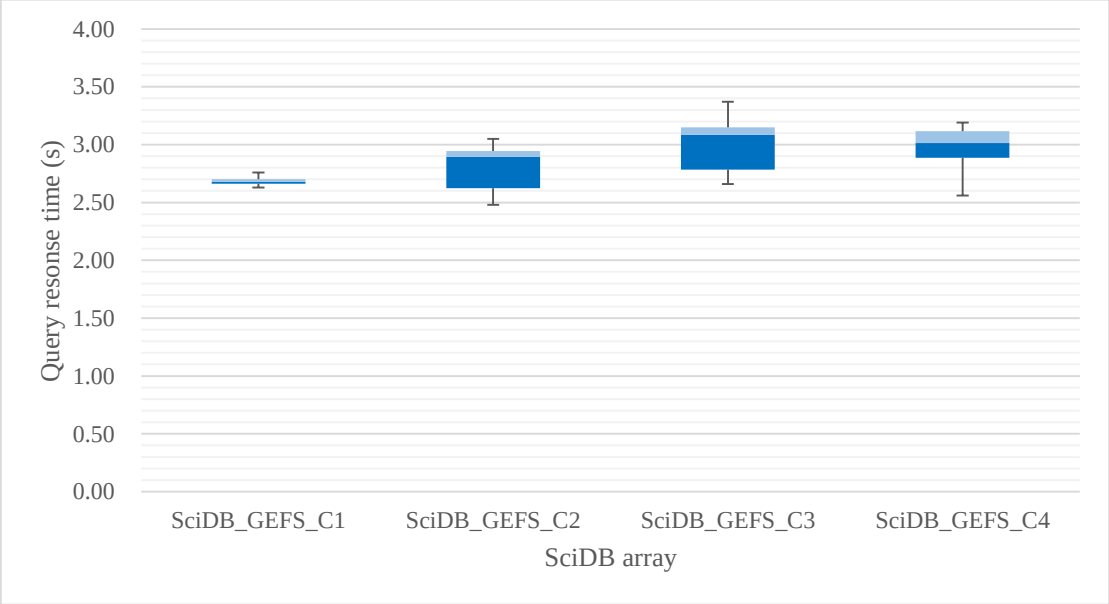


*Figure G22. Distribution of 10 benchmark measurements tested locally for calculating ensemble mean of total precipitation in Delft at one forecast step (**Large** chunk size)*

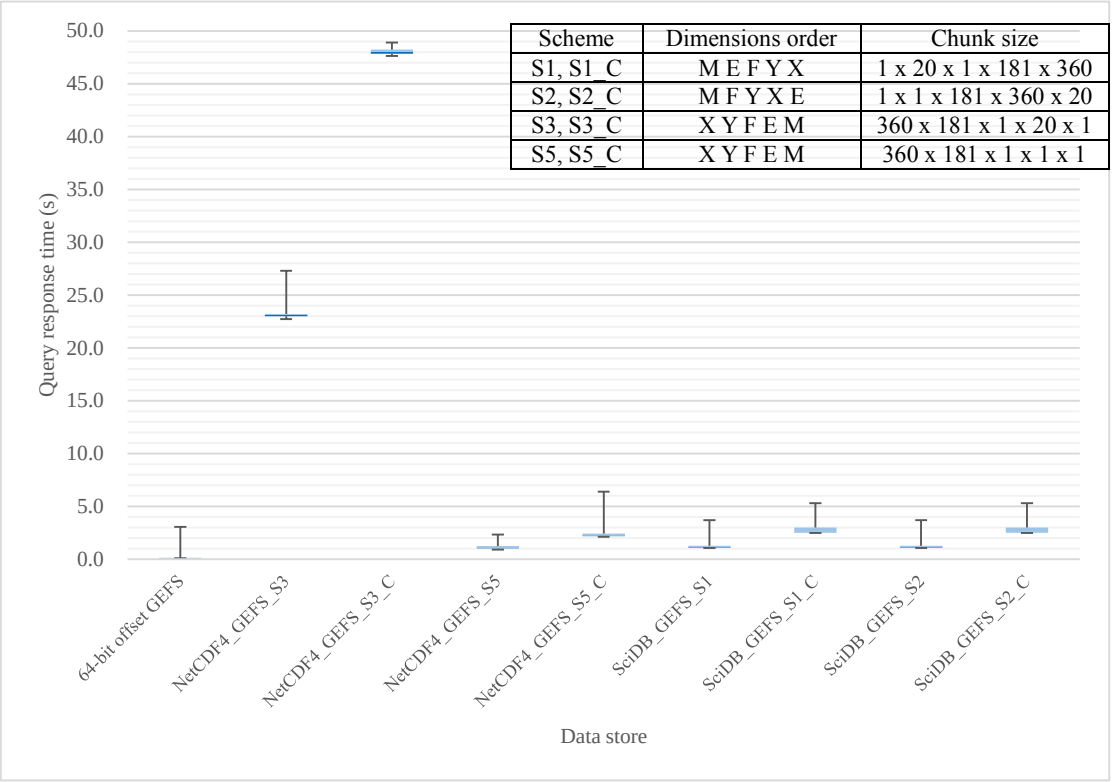


Figure G23. Distribution of 20 benchmark measurements of diversion solutions for extracting GEFS forecast time series of total precipitation

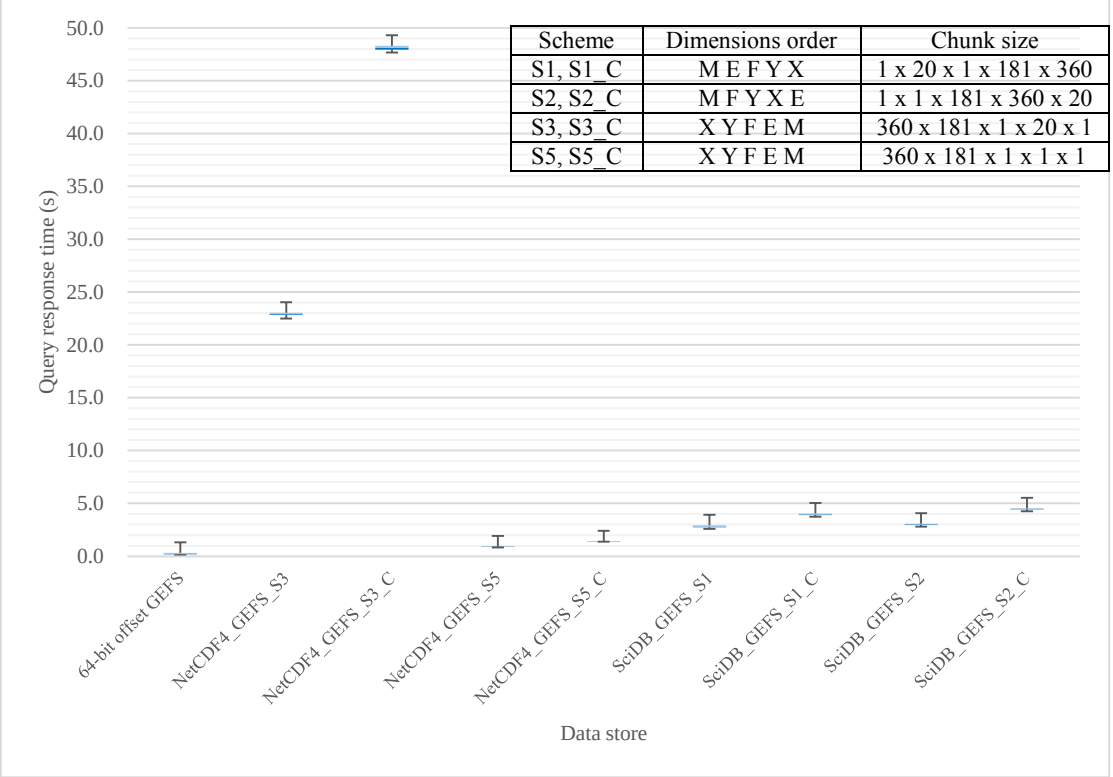


Figure G24. Distribution of 20 benchmark measurements of diversion solutions for GEFS total precipitation 80th percentile calculation

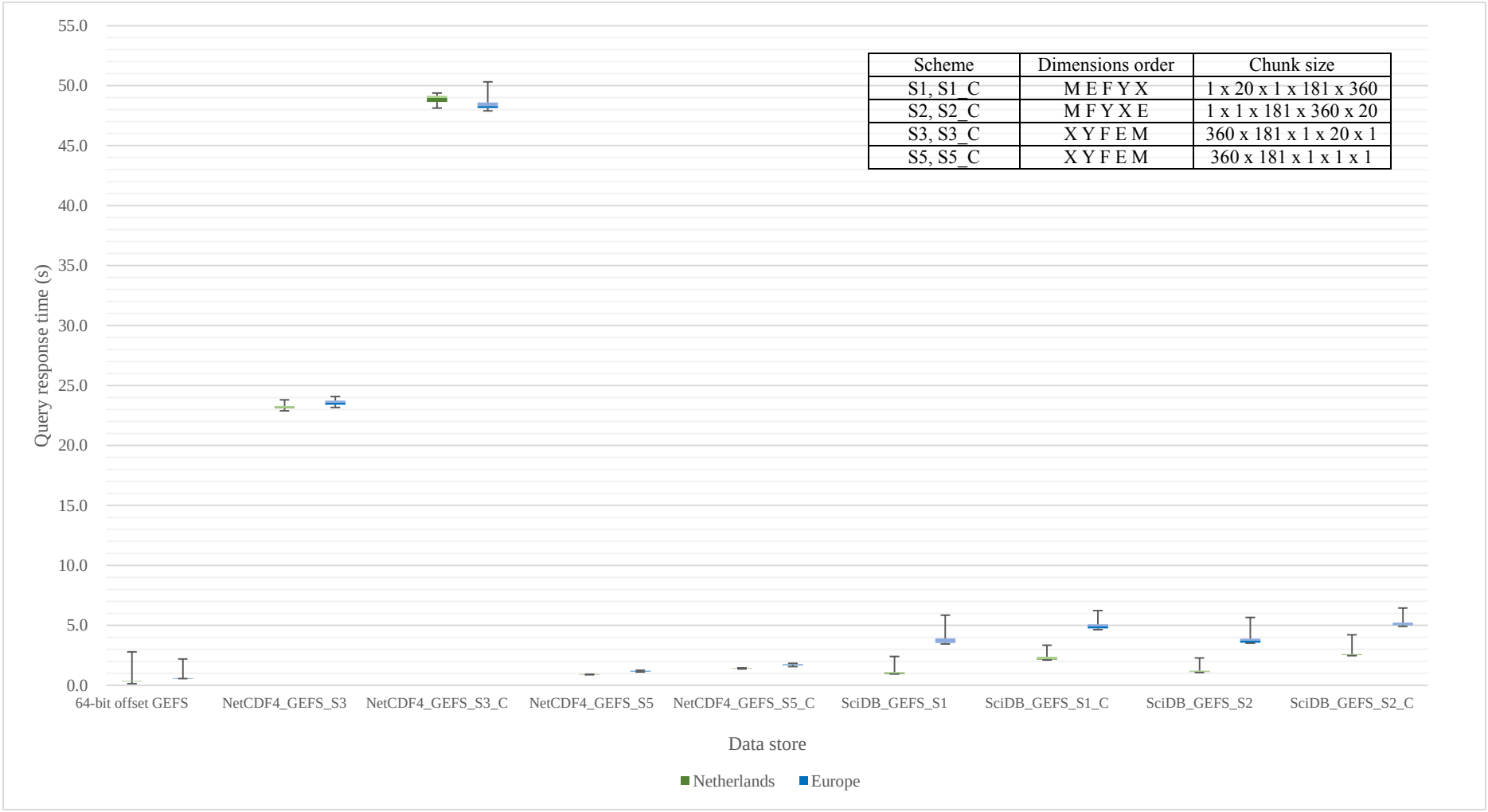


Figure G25. Distribution of 20 benchmark measurements of diversion solutions for ensemble mean calculation on GEFS total precipitation