An Approach for Assessing Array DBMSs for Geospatial Raster Data

Janne Kovanen*, Ville Mäkinen† and Tapani Sarjakoski‡
Finnish Geospatial Research Institute,
National Land Survey of Finland
Email: *janne.kovanen@iki.fi, †ville.p.makinen@nls.fi, ‡tapani.sarjakoski@nls.fi

***Abstract*—The increasing quantity and use of high-resolution raster data has put its management in the forefront of development. In this paper, we describe an approach that can be used to assess the capabilities of Array Database Management Systems (DBMSs) regarding the management and processing of raster data. The paper presents a framework that can be used to compare the functionalities of Array DBMSs and benchmark them. The main feature of the framework is assessing functionality using both targeted test cases and benchmarking. This assessment is followed by leveraging the gained experiences to assess non-functionality using characteristics from existing quality models. The framework can be extended by further DBMSs, benchmarks and additional hardware resources. The assessment was first implemented for the community editions of SciDB and rasdaman. The study presents some key initial observations regarding the particular Array DBMSs.**

***Keywords*–array DBMS; software assessment; benchmarking; SciDB; rasdaman.**

I. Introduction

Array Database Management Systems (DBMSs) have been proposed as a solution for data that naturally – or with a meagre conversion – fits on a regular multi-dimensional grid. Their significance has especially been noticed in the era of the Big Data phenomenon. Climate data, high-resolution rasters and sensor time series represent data that may suit an array data model. For example, the pixels of a raster can be mapped to cells in the array, and the four axes of climate data can correspond to a geographic location, altitude and time.

Array-oriented management solutions have been available for several decades; for instance, the development of Hierarchical Data Format (HDF) [1] and NetCDF [2] data formats and libraries started in the late 1980s. Their contemporary implementations, however, only conceptually resemble their earliest versions.

In addition to the aforementioned machine-independent data formats, SciDB [3], rasdaman [4], Ophidia [5] and TileDB [6] represent modern, domain-independent solutions. Domain-specificity may be gained by using associated components (e.g., Petascope [7] for rasdaman or H5IM for HDF5), third-party interface layers (e.g., scidb4geo [8] for SciDB) or with application-specific solutions. An alternative to domain independence is to use ready-made raster-centric solutions, like PostGIS's raster type [9] or the GeoRaster feature of Oracle Spatial and Graph [10].

A. *Previous work*

The diversity of prospective systems makes analysing and comparing them a challenging task for both developers and management. To help with reasoning, comparison studies have been published. Merticariu, Misev and Baumann [11] compared the sequential performance of rasdaman, SciDB and SciQL [12] using randomly generated artificial dense eight-bit data. They came to the conclusion that, in general, their rasdaman implementation outperformed the others by one to two orders of magnitude if really small queries or data ingestions were not taken into account. Liu et al. [13] compared the performance of the file-based NetCDF-4 and SciDB regarding three-dimensional spatio-temporal rainfall data. Their study found the uncompressed NetCDF-4 to be more efficient than SciDB.

The published comparisons assess the different Array DBMSs concerning relevant functions; however, they primarily look at the systems from the performance point of view using benchmarking. The most extensive work on database benchmarking has been performed by the non-profit TPC [14]. Its benchmarks evaluate performance with use cases from different areas of business – like stock brokerage – and execution scenarios, which vary in parallelism and complexity. None of the TPC benchmarks are, however, relevant for array DBMS. The Standard Science DBMS Benchmark (SS-DB), instead, is a benchmark developed by Cudre-Mauroux et al. [15], which was designed with astronomy in mind but may be used as a generic benchmark for scientific 1D–3D array data. Cudre-Mauroux et al. used it to compare the performance of MySQL and SciDB, the outcome of which was that overall SciDB performed two orders of magnitude faster.

While the execution time is important, several other quality characteristics also affect whether an Array DBMS meets the needs of stakeholders. A plethora of models for software quality have been published (e.g [16]–[18]). A comparison of models is presented by Miguel [19]. The models have different purposes by which they can be classified as definition, assessment or prediction models of quality [20]. The quality of software can even be validated as being up to standards: the International Organization of Standardization (ISO) has a full series (ISO/IEC 25000) of standards for software quality and its evaluation. For instance, the standard ISO/IEC 25010:2011 [21] defines a dual model that splits quality into in-use quality and internal/external product quality. Figure 1 outlines the two-level characteristics included in the latter.

This paper presents an approach to assess array DBMS. In Section II, we propose a framework to assess them that includes the criteria used for evaluation and benchmarking. Next, in Section III, we describe how the framework was used to evaluate two different Array DBMSs, the used hardware and some of the initial key findings. Finally, in Section IV, we discuss some issues and conclude the paper.

II. The assessment method

In an optimal situation, a rigorous assessment can be performed based on a set of application-specific user requirements that represent the needs of stakeholders. However, in the case of Array DBMSs, the number of potential stakeholders is so high that it is impossible to determine all the requirements. Moreover, it is not enough to assess their functionality based

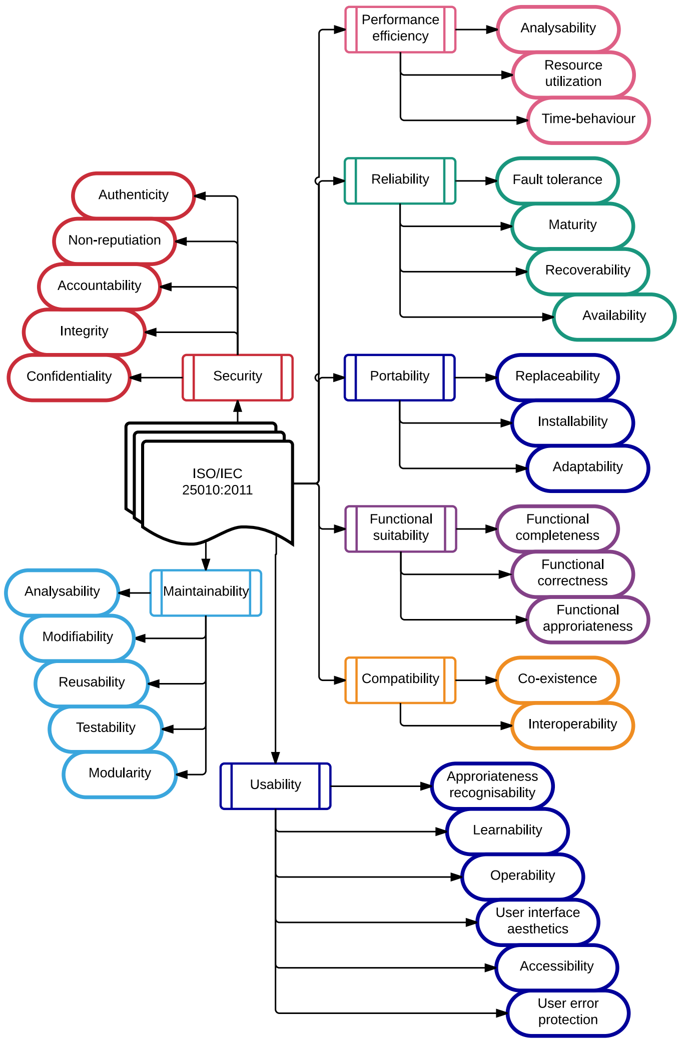

Figure 1. The characteristics and sub-characteristics of the ISO/IEC 25010:2011 product quality model.

on the documentation alone because of the rapid evolution that DBMSs are undergoing. For this reason, instead of using existing quality models as such, we divided the assessment into three parts: two parts concerning functionality and one part non-functionality.

Functionality is first reviewed against the documentation based on a set of criteria. Next, the found functionality is validated using an open web client. The client can be run from anywhere to verify that the statements regarding functionality are valid. The client is implemented using Jupyter Notebook [22], which is an open-source web application integrating live code, visualisations and accompanying text that uses Markdown language. The validation tests are written in Python and made small enough that they can be performed within a reasonable time limit. The test data is created on the fly by the web client or by using an array generator query. Hence, different instances of the same Array DBMS can be validated simply by changing the access parameters, like IP address, username and password.

Secondly, functionality is benchmarked on the server side. In this way, we can run long-lasting queries without worrying about network connection problems, and we do not need to speculate on how extensively the network transfers affect the timing. Nevertheless, benchmarking is affected by the chosen DBMS parameters and internal communication between nodes. As many users will not go through the burden of finding the optimal parameter combination, the benchmarking is run with the default settings. Bare bones results can be complemented by those gained from better performing configurations or external third-party software as long as they can be clearly distinguished and the default results are also included in the comparison.

The last part – evaluating non-functionality – is initiated after both parts concerned with functionality have been executed. This is to gain an initial understanding of the relevant quality characteristics (like reliability, maintainability and usability) through real use.

A. *The criteria for comparison*

Software, including Array DBMSs, can be assessed qualitatively using the experts of a particular field. Alternatively, the assessment may follow a predefined criteria list that scores several aspects of the software in a quantitative manner. The benefit of criteria is that they may be used by different people against diverse software. Criteria may be domain-specific or look at the software from a more general point of view. For example, the criteria of the Software Sustainability Institute [23] specialise in code quality, usability and overall sustainability.

Domain-specific criteria are based on the needs or concerns of a particular problem domain's potential stakeholders. Good criteria are also objective and unbiased; for example, no single software should be used as a reference. In the case of Array DBMSs, domain-specific requirements have been listed by Stonebraker et al. [24] and Xie [25]. The user concerns brought up by Stonebraker et al. are especially related to those raised by scientists and scientific data. Xie, on the other hand, looks at the requirements of a raster-specific DBMSs, which also apply in large part to generic array databases. Some of them are only relevant for spatial data though, like raster algebra and analytics, re-projection and cartographic modelling. As the most important characteristics, Xie picks out scalability and performance, and suggests that a database preferably has in-built analytics capabilities in order to achieve the required performance.

We split the criteria into functional and non-functional parts. The non-functional criteria first assess general software properties, like dependencies, hardware requirements, licensing, operating system support, source code access, means of installation, documentation, logging, memory-use, and error-handling. Next, other non-functional qualities are evaluated using existing quality models as a guideline.

The functional criteria are decomposed into 1) general DBMS capabilities, 2) a data model and schemas, and 3) a processing model. We also include geospatial capabilities as a domain-specific subgroup of the criteria. The general DBMS criteria assess the functionality that may be available with any type of data, like data compression, interfaces, support for accelerators and user-defined functions. The data model and schemas concentrate on array-specific capabilities, like restrictions of the data type, the density of data, the regularity of data bounds, uncertainty and multiple representation. The processing model looks at what operations the arrays can be used for – like aggregation, algebraic, bitwise, logical, moving window and string operations – but also transcendental functions, subsetting and joining. The criteria are then converted into questions to be answered in a consistent way; for example, the following questions are made regarding the array-specific dimensionality:

- What is the maximum number of dimensions?
- Is it possible to name the dimensions? Is it mandatory to name the dimensions? Can a dimension be renamed?
- Can a dimension be added or removed?
- Can the data be scaled by an integer for a dimension; that is, can a cell be duplicated a number of times before moving to the next cell?
- Can the bounds (lower/upper) of a dimension be changed after being defined?
- Can a dimension be used for time? Are time zones supported? Can time be treated as a continuous dimension or does it need to be treated as a discrete quantity; for example, if a time series is initially stored at one-second resolution, can data be added, that is defined, with millisecond accuracy in between old values?

B. *Benchmarking*

For benchmarking, we used two datasets, both licensed under CC BY 4.0 and having coordinates in the ETRS-TM35FIN projected coordinate reference system. KM10 [26] data represents the digital elevation model (DEM) of Finland in 10-metre resolution. It is composed of 1,509 GeoTiff files with a combined size of 9.7 GB. The data is two-dimensional with elevations being 16-bit floating point numbers, and it contains null values, which are coded -9,999. The benchmarking includes ingestion, export, cropping, operations (average, minimum), the moving average and simultaneous queries. The operations are performed on different sized square areas up to $60{,}000 \times 60{,}000$ cells. The moving average query is performed with different window sizes, ranging from 3×3 to 51×51 cells. The performance is measured by computing average values for cells within different windows. For each window size, several areas of different sizes, starting from 100×100 cells, are used. Regarding ingestion, it is performed in the format preferred by the DBMS; the time to translate to the format is not included in the timing.

The second dataset is CORINE [27]–[29], which represents the land cover classification of Finland for three different years (2000, 2006 and 2012). It is used for three-dimensional benchmarking. The resolutions of the original data are 25 and 20 metres, but in the benchmarking these are converted to a uniform five metres, allowing cell-by-cell comparisons to be made. The benchmarks include ingestion, counting the number of filtered cells and counting changes between two years. Filtering is done by area, timestamp and attribute value.

III. Evaluation case

We performed an assessment of the rasdaman community (version 9.5.0) and SciDB Community Edition (version 16.9). For rasdaman, we installed Petascope, the Semantic Coordinate Reference System Resolver (SECORE) [30] and their dependencies like Apache Tomcat [31] and PostgreSQL [32]. As the storage backend for rasdaman, we selected SQLite [33]. For SciDB, to store its metadata, we installed PostgreSQL.

A. *Software and hardware used in validation*

Concerning hardware, the evaluation was performed using an Infrastructure as a Service (IaaS) [34], where each node was composed of six virtual CPUs (2 GHz; 4096 KB cache from Intel Xeon E312xx (Sandy Bridge)) and 15.6 GB of RAM. The CPUs are over-committed and thus require the benchmarking to be run several times in order to give a good estimate. The network bandwidth between servers was validated to be 8 Gb/s with iPerf [35]. The servers have an 80GB root disk that is stored on a central storage system. Additional disks were added as required. The operating system was decided to be the latest version of Ubuntu, which was supported by the DBMS; version 16.04 was used with rasdaman and version 14.04 with SciDB.

Logging was set to warning level as the more detailed levels affected the running time. In the case of multiple rasdaman nodes (peers), each node was given its own replicated data source. The option of using a centralised data storage was not tested. Neither did the tests consider the option of splitting the data into separate arrays and distributing those to different nodes.

B. *Functional comparison*

We performed the functional comparison according to the presented criteria. As the information source, we used publicly accessible documentation. We also tried to use scientific literature, but it turned out to be too imprecise or it referred to planned functionality.

As we expected, the fast evolution of DBMSs seems to make it hard to keep the documentation up to date. It also turned out that developing both a commercial and a community version side by side is a really challenging task. For example, in SciDB, some functionality was marketed as being available in the community edition but actually was not. Meanwhile, some functionality of the enterprise edition was found in the community edition. Most troubling, however, was realising that the disclaimers of some code in the community edition required an enterprise license. This applied, for example, to the support of complex numbers, which is provided as a user-defined type.

Next, we created a validation client for each DBMS with Jupyter Notebook. In the clients, we used application-specific declarative languages, and the queries were passed to the DBMSs from their web interfaces. To access rasdaman, we used its web service that forwards requests in rasdaman query language (rasql). In the case of SciDB, we used shim [36]. It allows the execution of arrays managing queries using SciDB's Array Functional Language (AFL), which has a SQL-like syntax. However, we could not use operations defined in SciDB's Array Query Language (AQL), because it is not supported by shim.

Validating every functionality found in the documentation would have required us to implement a complete unit testing framework for both DBMSs. Hence, we chose to direct the testing towards the 1) basic functionality, 2) the most demanding functionality and 3) the most recent functionality. This approach paid off regarding finding problems. In particular, the late addition of null values in rasdaman turned out to be problematic. At worst, their use in data types corrupted the whole database. Without the validation, the problems of the functionality would not have been found. Moreover, isolating problem sources gave significant input for the non-functional assessment.

C. Performance evaluation

The performance evaluation was executed using bash scripting on the server side. For each DBMS, a single server instance was run. If the Array DBMS supported multiple nodes, then distributed instances were also tested; hence, we had three different configurations: sequentially on a node, parallel on a node and parallel on four nodes. We did not assess additional software, whether from the same author as the DBMS or a third-party, but acknowledged them and their potential in the functional comparison.

1) The KM10 benchmark: The KM10 data was stored in two dimensions (E, N) with an attribute containing the elevation as a floating point value. The data was stored without overlap. On SciDB, the chunk size was selected to be 2,400 $\times$ 1,200 cells and the history of array modifications was removed during ingestion.

On both DBMSs, the data was ingested from CSV files. For rasdaman, the files only contained elevation values, not coordinates. For SciDB, the files contained coordinates and elevations for the cells for which data existed. The uncompressed data sizes were 76.8 and 84.5 GB for rasdaman and SciDB respectively. Figure 2 represents the ingestion speed. For rasdaman, it includes the time (3,039 s) required to set the initial null values into blocks sized 10,000 $\times$ 10,000 cells. This had to be performed because otherwise the DBMS has zeros as null values, which is unacceptable in the case of a DEM.

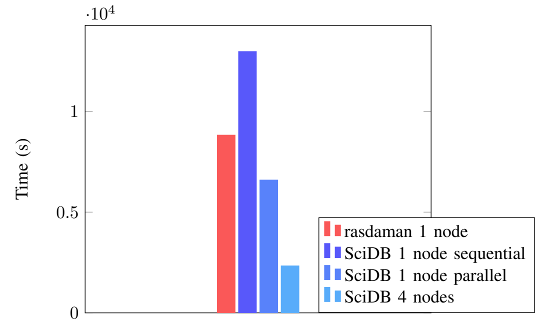

Figure 2. The ingestion time of the KM10 digital elevation model.

The data was exported from rasdaman using the CSV format, but in the case of SciDB we had to use its CSV+ format because the data contained empty cells. Initially, the servers had no swap memory, but it was added to enable testing rasdaman operations, because they failed when the RAM ran out. For example, rasdaman failed the 30,000 $\times$ 30,000 cell crop test without swap; still, not even an unlimited swap helped it in the execution of the 60,000 $\times$ 60,000 sized query. Figure 3 illustrates the time required for export from KM10.

In the moving window calculations, for each window size, the execution times with different analysis areas were scaled to comparable units and averaged out. The results are shown in Figure 4. The execution times with rasdaman showed little variation with respect to the size of the analysis area. On the other hand, the largest area analysed with rasdaman was 5,000 $\times$ 5,000 cells, whereas with SciDB 10,000 $\times$ 10,000 and 28,000 $\times$ 28,000 sized areas were analysed with single node and cluster installations respectively, with window sizes of up to 21 $\times$ 21 cells.

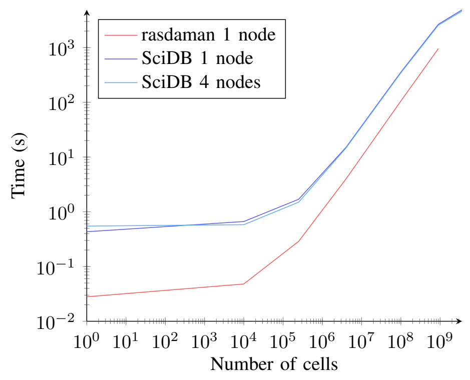

Figure 3. The time required for export from the KM10 digital elevation model.

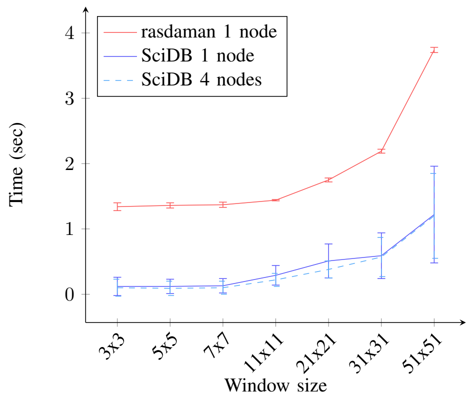

Figure 4. The time required for computing the moving window average for 10,000 cells with the KM10 digital elevation model.

2) The CORINE benchmark: On SciDB, the CORINE data was stored in three dimensions (E, N, $year$) and the land cover code was stored as a 16-bit unsigned integer. The chunk size was selected to be 2,000 $\times$ 2,000 $\times$ 1 cells. The data was stored without overlap. The history of array data was not stored.

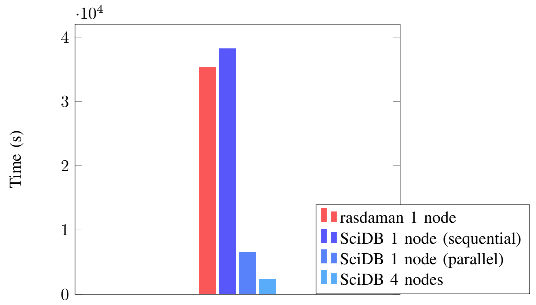

Figure 5. The ingestion time of the CORINE data.

The data ingestion of CORINE followed the same pattern as with KM10 (Figure 5). Between the DBMSs, the gap of sequential input got smaller than with the KM10. Similarly, the parallel and multi-node versions of SciDB performed better. A reason for this may be that in rasdaman the data was imported without using the scale function, because the source argument of the function must fit into the server's main memory, and according to the documentation, only nearest neighbour interpolation is supported for scaling. In SciDB, we used the xgrid function for scaling.

In both tests related to counting, the DBMSs behaved similarly: rasdaman handled smaller queries better, whereas SciDB performed faster with larger areas. The main difference that could be found was in the use of multiple processors and nodes. For example, SciDB was almost seven times faster regarding the largest area on one node regarding the query that counted the number of cells filtered by area, timestamp and attribute value (Figure 6). This correlates with the number of processors; likewise, with four nodes, SciDB became over three times faster than on one node.

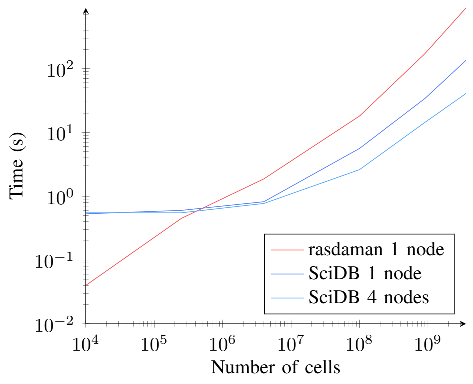

Figure 6. Counting the number of cells filtered by area, timestamp and attribute value from the CORINE data.

The difference between the DBMSs became larger when the complexity of the task grew. For example, SciDB was over 50 times faster in computing the changed cells between two timestamps from the CORINE data (Figure 7) regarding the largest query that rasdaman could manage.

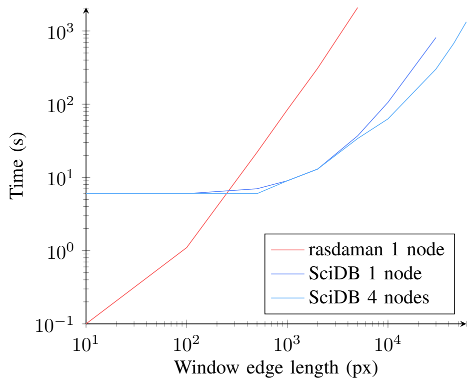

Figure 7. Counting the changed cells between two timestamps from the CORINE data.

IV. Conclusions, Discussion, and Future Work

In this paper, we addressed the assessment of Array DBMSs. We proposed performing the assessment in two consecutive steps: functional and non-functional steps. We also proposed executing the assessment of functionality by validating it against its documentation in an efficient manner, which especially targets error-prone areas but also evaluates the basic functionality. The used approach allows others to re-evaluate the assessed systems and to expand it to other Array DBMSs.

We also performed an initial trial of the approach on two Array DBMSs, which showed that the DBMSs have achieved a good level of functionality and performance. However, they struggle keeping up their documentation regarding both the languages and capabilities of their model. Inconsistency between the documentation and the behaviour reduce the usability and creditability of the systems.

The DBMSs are evolving at such fast pace that our approach will face the same challenge as the SS-DB benchmark: it has not been updated to work with the latest DBMS versions. A potential group of actors to keep the validation up to date are the authors of the DBMSs themselves. However, that creates a dilemma – will they be able to make an independent evaluation that disregards their agenda? We doubt this, mainly because the developers are already putting in effort to create unit tests and moving towards continuous integration which should reveal the problems that they have thought of.

If the authors of the Array DBMSs do not update the benchmarks and the validation scripts after making changes to the query languages or interfaces, our approach may require a TPC-kind of actor with sufficient resources to take ownership of the assessment. One candidate for this role is the Research Data Alliance (RDA) because Array DBMSs are well suited for research-focused data. However, the RDA (or any other party taking responsibility) will need a user pool to define the requirements to be evaluated and possibly even to develop test cases that the DBMSs can aim at passing. The creation of the test cases and the performance evaluation would be even simpler if the languages and interfaces used by the DBMSs become harmonised at some point.

Acknowledgement

The paper is based on the research carried out in the collaboration project ”Evaluation and demonstration of Array DBMSs using national geospatial data”, part of the EU-funded RDA EU3 project (grant agreement number 653194). The IaaS used in the work was funded by the Academy of Finland through ”Finnish Grid and Cloud Infrastructure” (urn:nbn:fi:research-infras-2016072533; grant number 283818).

References

[1] The HDF Group, “HDF group history,” https://support.hdfgroup.org/about/history.html, [accessed: 2018-01-23].

[2] Unidata, “NetCDF,” http://www.unidata.ucar.edu/software/netcdf/, Boulder, CO: UCAR/Unidata Program Center, [accessed: 2018-01-23].

[3] Paradigm4 Inc., “SciDB,” http://www.paradigm4.com, [accessed: 2018-01-23].

[4] rasdaman GmbH, “rasdaman,” http://www.rasdaman.org, [accessed: 2018-01-23].

[5] CMCC Foundation, “Ophidia,” http://ophidia.cmcc.it, [accessed: 2018-01-23].

[6] TileDB Inc., “TileDB,” http://tiledb.io, [accessed: 2018-01-23].

[7] A. Aiordăchioaie and P. Baumann, “PetaScope: An open-source implementation of the OGC WCS geo service standards suite,” in Scientific and Statistical Database Management: 22nd International Conference, SSDBM 2010, Heidelberg, Germany, June 30–July 2, 2010. Proceedings, M. Gertz and B. Ludäscher, Eds. Springer Berlin Heidelberg, 2010, pp. 160–168.

[8] M. Appel, “scidb4geo,” http://github.com/appelmar/scidb4geo, [accessed: 2018-01-23].

[9] “PostGIS,” http://postgis.net, [accessed: 2018-01-23].

[10] Oracle Corporation, “Oracle Spatial and Graph,” http://www.oracle.com/technetwork/database/options/spatialandgraph/, [accessed: 2018-01-23].

[11] G. Merticariu, D. Misev, and P. Baumann, “Towards a general array database benchmark: Measuring storage access,” in Big Data Benchmarking: 6th International Workshop, WBDB 2015, Toronto, ON, Canada, June 16-17, 2015 and 7th International Workshop, WBDB 2015, New Delhi, India, December 14-15, 2015, Revised Selected Papers, T. Rabl, R. Nambiar, C. Baru, M. Bhandarkar, M. Poess, and S. Pyne, Eds. Springer International Publishing, 2016, pp. 40–67.

[12] Y. Zhang, M. Kersten, M. Ivanova, and N. Nes, “SciQL: Bridging the gap between science and relational DBMS,” in Proceedings of the 15th Symposium on International Database Engineering & Applications, ser. IDEAS ’11. New York, NY, USA: ACM, 2011, pp. 124–133.

[13] H. Liu, P. van Oosterom, C. Hu, and W. Wang, “Managing large multi-dimensional array hydrologic datasets: A case study comparing NetCDF and SciDB,” Procedia Engineering, vol. 154, no. Supplement C, 2016, pp. 207–214, 12th International Conference on Hydroinformatics (HIC 2016) - Smart Water for the Future.

[14] TPC, “Active TPC benchmarks,” http://www.tpc.org/information/benchmarks.asp, [accessed: 2018-01-23].

[15] P. Cudre-Mauroux et al., “SS-DB: A standard science DBMS benchmark,” in 4th Extremely Large Databases Conference October 6-7, 2010, 2010, [accessed: 2018-01-23].

[16] B. W. Boehm, J. R. Brown, and M. Lipow, “Quantitative evaluation of software quality,” in Proceedings of the 2nd International Conference on Software Engineering, ser. ICSE ’76. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, pp. 592–605.

[17] J. A. McCall, P. K. Richards, and G. F. Walters, “Factors in software quality,” Rome Air Development Center, Air Force Systems Command, Griffiss Airforce Base, New York 13441, Tech. Rep. RADC-TR-77-369, November 1977.

[18] R. G. Dromey, “A model for software product quality,” IEEE Software, vol. 13, no. 1, 1995, pp. 33–43.

[19] J. P. Miguel, D. Mauricio, and G. Rodriǵuez, “A review of software quality models for the evaluation of software products,” International Journal of Software Engineering & Applications (IJSEA), vol. 5, no. 6, 2014, pp. 31–53.

[20] F. Deissenboeck, E. Juergens, K. Lochmann, and S. Wagner, “Software quality models: Purposes, usage scenarios and requirements,” in 2009 ICSE Workshop on Software Quality, 2009, pp. 9–14.

[21] “ISO/IEC 25010:2011. systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – system and software quality models,” International Organization for Standardization, Geneva, CH, Standard, 2011.

[22] Project Jupyter, “The Jupyter Notebook,” http://jupyter.org, [accessed: 2018-01-23].

[23] M. Jackson, S. Crouch, and R. Baxter, Software Evaluation: Criteria-based Assessment, https://www.software.ac.uk/sites/default/files/SSI-SoftwareEvaluationCriteria.pdf, 2011, [accessed: 2018-01-23].

[24] M. Stonebraker et al., “Requirements for science data bases and SciDB,” in CIDR 2009, 2009.

[25] Q. Xie, “The design of a high performance earth imagery and raster data management and processing platform,” ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XLI-B4, 2016, pp. 551–555.

[26] National Land Survey of Finland, “Korkeusmalli 10 m, 20.04.2017,” http://kartat.kapsi.fi, [accessed: 2018-01-23].

[27] Finnish Environment Institute, “CORINE maanpeite 2012, 20 m, 30.9.2014,” http://www.syke.fi/fi-FI/Avoin_tieto/Paikkatietoaineistot, [accessed: 2018-01-23].

[28] ——, “CORINE maanpeite 2006, 25 m, 15.6.2010,” http://www.syke.fi/fi-FI/Avoin_tieto/Paikkatietoaineistot, [accessed: 2018-01-23].

[29] ——, “CORINE maanpeite 2000, 25 m, 11.5.2010,” http://www.syke.fi/fi-FI/Avoin_tieto/Paikkatietoaineistot, [accessed: 2018-01-23].

[30] D. Misev, M. Rusu, and P. Baumann, “A semantic resolver for coordinate reference systems,” in Web and Wireless Geographical Information Systems: 11th International Symposium, W2GIS 2012, Naples, Italy, April 12-13, 2012. Proceedings, S. Di Martino, A. Peron, and T. Tezuka, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 47–56.

[31] The Apache Software Foundation, “Apache Tomcat,” http://tomcat.apache.org, [accessed: 2018-01-23].

[32] The PostgreSQL Global Development Group, “PostgreSQL,” http://www.postgresql.org, [accessed: 2018-01-23].

[33] SQLite Development Team, “SQLite,” http://www.sqlite.org, [accessed: 2018-01-23].

[34] CSC – IT Center for Science Ltd., “Pouta user guide,” https://research.csc.fi/pouta-user-guide, [accessed: 2018-01-23].

[35] “iPerf,” https://iperf.fr, [accessed: 2018-01-23].

[36] Paradigm4 Inc., “shim,” http://github.com/Paradigm4/shim, [accessed: 2018-01-23].