

A Demonstration of SciDB: A Science-Oriented DBMS

P. Cudre-Mauroux MIT
H. Kimura Brown U
K.-T. Lim SLAC
J. Rogers Brown U
R. Simakov NIISI RAS
E. Soroush UW
P. Velikhov NIISI RAS
D. L. Wang SLAC
M. Balazinska UW
J. Becla SLAC
D. DeWitt Microsoft
B. Heath VoltDB
D. Maier PSU
S. Madden MIT
M. Stonebraker MIT
S. Zdonik* Brown U

ABSTRACT

In CIDR 2009, we presented a collection of requirements for SciDB, a DBMS that would meet the needs of scientific users. These included a nested-array data model, science-specific operations such as regrid, and support for uncertainty, lineage, and named versions. In this paper, we present an overview of SciDB's key features and outline a demonstration of the first version of SciDB on data and operations from one of our lighthouse users, the Large Synoptic Survey Telescope (LSST).

1. INTRODUCTION & BACKGROUND

The XLDB-1 workshop in October 2007 brought together a collection of big science and commercial Internet users with extreme data base requirements. The users complained about the inadequacy of current commercial DBMS offerings. Although the DBMS community has been working on science data bases for years and has even built prototypes (e.g. Sequoia 2000 with Postgres [4], Paradise [3], and extensions to MonetDB [8]), some of which are successfully being used today (e.g., the Sloan Digital Sky Survey [16]), all these systems still lack important features to meet the needs of increasingly data rich sciences.

To address this shortcoming, a collection of science users and DBMS researchers met at Asilomar in March 2008 to define requirements for a new type of scientific DBMS. The results were presented at the XLDB-2 workshop in October 2008, and the reaction from science users was extremely positive. These ideas, presented at CIDR 2009 [15], included the following constructs: (1) A nested array data model; (2) Science-specific primitive operations, such as regrid; (3) Provenance; (4) No-overwrite storage; (5) "In situ" data

*Author affiliations expanded: Massachusetts Institute of Technology (MIT), Brown University (Brown U), SLAC National Accelerator Laboratory (SLAC), Scientific Research Institute for Systems Studies of the Russian Academy of Sciences (NIISI RAS), University of Washington (UW), Portland State University (PSU).

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '09, August 24-28, 2009, Lyon, France

Copyright 2009 VLDB Endowment, ACM 000-0-00000-000-0/00/00.

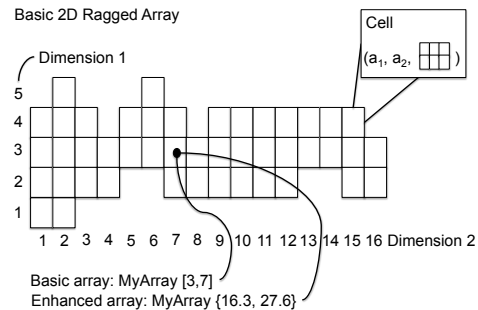


Figure 1: Example 2D ragged array. Each cell contains two scalar attributes, a_1 and a_2 , plus another 2D array. The MyArray instance has been enhanced with an extra coordinate system.

(i.e., the DBMS should process data without requiring that it be loaded); (6) Named versions; and (7) Uncertainty.

In late 2008, the project embarked on building SciDB, a new science-oriented DBMS, embodying these ideas. At VLDB 2009, we will demonstrate version 1 of SciDB on imagery from the Large Synoptic Survey Telescope (LSST) [9], showing queries that find space objects (e.g., galaxies) and a visualization of query results.

The rest of this paper is organized as follows. In Section 2, we briefly discuss SciDB's data model and query facilities, followed, in Section 3 by other capabilities. Section 4 discusses novel aspects of our implementation. Section 5 describes our VLDB demonstration.

2. DATA MODEL & QUERY FACILITIES

2.1 Data Model

Arrays are a natural data model for a significant subset of science users (specifically astronomy, oceanography, fusion, remote sensing, climate modeling, and seismology). Seemingly, biology and genomics users want graphs and sequences. They will be happy with neither a table nor an array data model. Chemistry users are in the same situation. Lastly, users with solid modeling applications want a mesh data model [7] and will be unhappy with tables or arrays. The net result is that one size will not fit all, and science users will need a mix of specialized DBMSs.

Our project uses an array data model, primarily because it makes a considerable subset of the community happy and is easier to build than a mesh model. We support a multi-

dimensional, nested array model with array cells containing records, which in turn can contain components that are multi-dimensional arrays.

Specifically, basic arrays can have any number of named dimensions as illustrated in Figure 1. Each dimension has contiguous integer values between 1 and N (the high-watermark). Each combination of dimension values defines a cell. Every cell has the same data type(s) for its value(s), which are one or more scalar values, and/or one or more arrays. It is acceptable to create a basic array, which is unbounded in one or more dimensions.

SciDB will support Postgres-style user-defined functions [13] (methods, UDFs), which must be coded in C++, the implementation language of SciDB. As in Postgres, UDFs can internally run queries and call other UDFs. We will also support user-defined aggregates and table functions, again Postgres-style.

UDFs can serve to enhance arrays: any function that accepts integer arguments can be applied to the dimensions of an array to produce an array with a new coordinate system, formed by transposition, scaling, translation, or other transform of the original arrays dimensions. These new arrays can be indexed by their enhanced dimensions. Some arrays are irregular, i.e., they do not have integer dimensions, and some are defined in a particular co-ordinate system (e.g., Mercator geometry). Enhancing arrays with more complex UDFs can support both types of coordinate spaces. When dimensions have ragged edges (e.g., in a 2D array, each row has a different number of columns), we can enhance a basic array with a shape function. A shape function is a UDF that specifies the outline or dimensionality of a ragged array.

To support irregular and ragged arrays, a sophisticated storage manager is required. As noted in Section 4, our initial prototype does not include all of these features.

2.2 SciDB Operators

SciDB is novel in that it has no built-in operators. Instead all operators are UDFs; some are provided with the initial system and the remainder are provided separately using the extension facility.

2.2.1 Structural Operators

The first operator category creates new arrays based purely on the structure of their inputs. In other words, these operators are data-agnostic. The simplest example here is an operator that we call Subsample. Subsample takes two inputs, an array A and a predicate over the dimensions of A, which defines a subslab of the array. The output will always have the same number of dimensions as the input, but will generally have a smaller number of dimension values.

Other structural operators include add dimension, remove dimension, concatenate, and cross product.

2.2.2 Content-Dependent Operators

The next category involves operators whose result depends on the data that is stored in the input array. A simple example of this kind of operator is Filter. Filter takes an array A and a predicate over the data values that are stored in the cells of A. It returns an array with the same dimensions as A, with the original cell value if the predicate evaluates to true, and with NULL if the predicate evaluates to false.

Other examples of data-dependent operators include Apply UDF and Project.

2.2.3 Operators with Both Structural and Content-Dependent Forms

Some operators come in both structural and data-dependent forms. The principal example is Join. Join can be performed either on indices or data values (or possibly both). In the case of a Join on an m-dimensional array and an n-dimensional array that involves only k index attributes from each of the arrays in the join predicate, the result will be an $(m+n-2k)$ -dimensional array with concatenated cell tuples wherever the predicate is true.

In the case of a Join on an m-dimensional array and an n-dimensional array that involves only data attributes in the join predicate, the result will be an $(m+n)$ -dimensional array with concatenated cell tuples wherever the predicate was true.

3. OTHER SCIDB FEATURES

3.1 No Overwrite

Most scientists are adamant about not discarding any data. If a data item is shown to be wrong, they want to add the replacement value and the time of the replacement, retaining the old value for provenance (lineage) purposes. As such, they require a no-overwrite storage manager, in contrast to most commercial systems today, which overwrite the old value with a new one.

Postgres [13] contained a no-overwrite storage manager for tables. In SciDB, no-overwrite is even easier to support. Specifically, arrays can be optionally declared as updatable. All arrays can be loaded with new values. Afterwards, cells in an updatable array can receive new values. To support this concept, a history dimension must be added to every updatable array. Our design supports delta compression to minimize the amount of space required for historical data.

3.2 Grid Orientation

LSST expects to have 55 petabytes of raw data. It goes without saying that a DBMS that expects to store LSST data must run on a cloud (grid) of shared-nothing [12] computers. Conventional DBMSs such as Teradata, Netezza, DB2, and Vertica have used this architecture for years, employing horizontal partitioning of tables as in Gamma [2].

It would be easier to use a fixed partitioning scheme in SciDB. However, there will be a class of applications that cannot be load-balanced using such a tactic. These include ragged and irregular arrays as noted earlier, and applications where data ingest is not uniform over time. Hence, in SciDB we allow the partitioning to change over time: e.g., a first partitioning scheme is used for time less than T and a second partitioning scheme for time greater than T.

3.3 “In Situ” Data

A common complain from scientists is “I am looking forward to getting something done, but I am still trying to load my data”. Put differently, the overhead of loading data is high and may dominate the value received from using the DBMS. As such, SciDB must be able to operate on in situ data, without requiring a load process. Our approach to this issue is to define a self-describing data format and then write adaptors to various popular external formats, for example HDF-5 [10] or NetCDF [11]. If an adaptor exists for the user’s data or if he is willing to put it in the SciDB format above, then he can use SciDB without a load stage.

3.4 Integration of the Cooking Process

Most scientific data comes from instruments observing a physical process of some sort. For example, in remote sensing applications, imagery is collected from satellite or airborne observation. Such sensor readings enter a cooking process whereby raw information is transformed into finished information. Cooking entails converting sensor data into standard types, correcting for calibration information, correcting for cloud cover, etc. SciDB will naturally support cooking inside the DBMS, by simply loading the raw data and then using UDFs and data manipulation operations to perform the cooking process. Of course, users can still perform external cooking, if they desire.

3.5 Named Versions

A requirement of most science users is the concept of named versions. With named versions, a scientist can make application-specific changes to a small portion of an array, retaining the rest of the array unchanged. Clearly, one wants to support versions without paying the cost of a copy for unchanged data. In SciDB, versions are stored as a delta off their parents and thus consume essentially no space, until application-specific updates are applied.

3.6 Provenance

A universal requirement from scientists is repeatability of data derivation. Scientists wish to be able to recreate any array A, by remembering how it was derived. For a sequence of processing steps inside SciDB, one merely needs to record a log of the commands that were run to create A. The search requirements for this log are then: (1) For a given data element D, find the collection of processing steps that created it from input data. (2) For a given data element D, find all the downstream data elements whose value is impacted by the value of D.

Recording the log is straightforward. The hard part is to create a provenance query language and an efficient implementation. Although one could use Trio [1] as an exemplar, we are concerned about the space cost of recording item-level derivations. In earlier work [15], we presented an alternate proposal that required minimal additional space. However, to support the search functionality noted above, this solution requires running non-trivial queries. An interesting research issue is to find a solution that can easily morph between a minimal storage solution and the Trio solution.

3.7 Uncertainty

Essentially all scientific data is imprecise, and without exception science researchers have requested a DBMS that supports uncertain data elements. Of course, current commercial RDBMSs are oriented toward business users, where there is a much smaller need for this feature. Hence, commercial products do not support uncertainty.

In talking with many science users, there was near universal consensus on requirements in this area. They requested a simple model of uncertainty: normal distributions for data elements. In effect, they requested error bars (standard deviations) and an executor that would perform interval arithmetic when combining uncertain elements.

Hence, SciDB will support uncertain x for any data type x that is available in the engine. Of course, this requires two values for any data element (value and error), rather than one. However, every effort will be made to effectively code

data elements in an array, so that arrays with the same error bounds for all values will require negligible extra space.

There is another implication of uncertain data, exemplified by the data base design for the PanSTARRS telescope project. Since observation of objects entails some error, the PanSTARRS DBAs have identified the maximum possible error, P . To ensure that joins can be performed without moving data elements, they have grid partitions overlap by P . In this way, PanSTARRS can implement uncertain join without moving objects between nodes.

More generally, SciDB will allow an array to be repartitioned across grid nodes, such that the partitions overlap. The mechanism to support this functionality is to run a UDF at each site that holds array values. This UDF operates on array indexes and maps them to a collection of sites.

3.8 Open Source

It appears impossible to get any traction in the science community unless the DBMS is open source. The main reasons why the scientific community dislikes closed-source software include (a) a need for multi-decade support required by large science projects, (b) an inability to recompile the entire software stack at will and (c) difficulties with maintaining closed-source software within large collaborations encompassing tens or even hundreds of institutions. As such, SciDB is an open source project. Because the science community wants a commercial strength DBMS, a non-profit foundation (SciDB, Inc.) has been organized to manage the development of the code.

4. VERSION 1 OF SCIDB

We demonstrate the first version of SciDB at this year's VLDB. SciDB Version 1 implements basic arrays as defined in Section 2 along with a large collection of operators, all supported by a general purpose UDF framework. Moreover, Version 1 includes a well-developed local storage manager, which organizes each local array into a collection of rectangular buckets, defined by a stride in each dimension. Hence, within a node an array partition is divided into variable size rectangular buckets. An R-tree [6] keeps track of the size of the various buckets.

Optimization of the storage management layer includes deciding when to change the partitioning criteria between sites and when to merge disk buckets into larger ones.

It is clear that compression must be an integral part of any science DBMS. We compress each storage block individually with a scheme that best suits its data as well as the user's requirements. Blocks are compressed when written and decompressed when accessed. An executor that operates on compressed data will be worked on in the future. We choose a compression scheme for each block individually, based on an objective function that takes into account encoding time, decoding time and compression ratio.

We currently support several lossless algorithms including Null Suppression, Run-Length Encoding, Lempel-Ziv, Delta Encoding and Adaptive Huffman. We also added our own multidimensional extensions for Run-Length Encoding and Adaptive Huffman. Null Suppression has been generalized to subtract out a background pattern, a likely occurrence in scientific data such as LSST data.

Version 1 has two mechanisms to pass data between operators. If both operators act on array cells, one at a time, then the system can use conventional relational pipelining.

When UDFs operate on whole arrays, results are passed via the storage system, rather than by pipelining.

The optimizer in Version 1 is quite primitive. A SciDB parse tree contains a collection of operators in a tree structure. Many of these operations do not commute. Hence, our ability to optimize is limited to paths between non-commuting operators. Our goal is to parallelize as many UDFs as possible. Operators that can be pipelined are trivial to parallelize. Others, for example picking stars out of raw telescope imagery, deal with an array as a whole. However, they can be parallelized by noting the maximum size of a star and then overlapping partitions by this amount. Hence, we have implemented a version of the exchange operator [5] that will produce overlapping partitions.

To simplify application development, SciDB Version 1 also includes advanced C++ language bindings similar in spirit to LINQ¹: application developers access SciDB arrays as if they were local array objects. They express queries by invoking methods on these objects. Queries are evaluated lazily when applications access query results.

5. DEMONSTRATION

To focus attention on the needs of science users, we have written a science benchmark [14] that abstracts the requirements of LSST and is similar to the needs in other science domains. This benchmark records raw astronomy data that is observed by pointing a telescope at a particular portion of the sky. A function is then used to extract astronomy objects (stars, galaxies, and others) from images. Such observations have a variety of data elements, including an uncertain center position, an uncertain radius, and uncertain spectral properties. Cooking raw data into observations is a time consuming operation that should be parallelized. Successive imagery of the same sky co-ordinates will yield successive raw imagery and cooked observations. The last step of the cooking process is to group observations into ones that represent the same object at different points in time.

The benchmark includes this cooking process along with 9 queries posed on combinations of the raw and cooked data that mimic what LSST users might ask of a DBMS. For example, query 2 requires regridding the raw data for a specific time range such that the grid cells collapse 10::3. Then a user defined interpolation function is run on new and old observations to find the ones that have been lost. The easiest query in the benchmark filters observations on a specific predicate, while the hardest one requires computing the trajectory of an observation group and then finding pairs of trajectories that come within a certain distance of each other.

We demonstrate a multi-site SciDB system storing the above benchmark LSST data as basic arrays. Cooking is done inside the DBMS using UDFs for the two cooking stages. The executor includes parallel operators for 15 common operations, and is smart enough to run all 9 queries.

SciDB obtains maximum parallelism by strategically partitioning raw imagery, observations, and groups. This parallelism exploits our exchange operator with overlap.

Besides running the benchmark queries and cooking process, our demonstration includes a visualization of several terabytes of LSST space imagery and objects (e.g., galaxies, planets, etc) and allows users to pose a collection of ad-hoc queries over those objects: find galaxies in this region of the

sky and show me objects moving faster than velocity v . Our visualization includes the ability to pan through space and time with interactive response rates. These operations are implemented as queries against our prototype, demonstrating the performance of the SciDB engine.

We also show users the queries and application code that communicates with SciDB, with the goal of receiving feedback from potential users about what additional features they would like to see added to the system.

6. CONCLUSION

This paper has sketched the capabilities of SciDB and indicated which ones are operational in Version 1. The system is being constructed in stages by a distributed team of developers at SLAC, NIISI, MIT, Portland State University, Brown University, and the University of Washington. Over the next two years, we expect to have an increasingly sophisticated open-source system for the science community.

7. REFERENCES

- [1] Agrawal et. al. Trio: a system for data, uncertainty, and lineage. In *Proc. of the 32nd VLDB Conf.*, pages 1151–1154, 2006.
- [2] DeWitt et. al. GAMMA - a high performance dataflow database machine. In *Proc. of the 12th VLDB Conf.*, pages 228–237, 1986.
- [3] DeWitt et. al. Client-server paradise. In *Proc. of the 20th VLDB Conf.*, pages 558–569, 1994.
- [4] Dozier et. al. Sequoia 2000: A next-generation information system for the study of global change. In *Proc. of the IEEE Symp. on Mass Storage Systems*, pages 47–53, 1994.
- [5] G. Graefe and D. L. Davison. Encapsulation of parallelism and architecture-independence in extensible database query execution. *IEEE Trans. Softw. Eng.*, 19(8):749–764, 1993.
- [6] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. of the SIGMOD Conf.*, pages 47–57, 1984.
- [7] B. Howe. *GRIDFIELDS: Model-Driven Data Transformation in the Physical Sciences*. PhD thesis, Portland State University, 2007.
- [8] Ivanova et. al. MonetDB/SQL meets SkyServer: the challenges of a scientific database. In *Proc. of the 19th SSDBM Conf.*, page 13, 2007.
- [9] Kantor et. al. Designing for peta-scale in the LSST database. *Astronomical Data Analysis Software and Systems XVI*, 376:3–11.
- [10] National Center for Supercomputing Applications (NCSA). HDF5: API specification reference manual. <http://hdf.ncsa.uiuc.edu/>, 2004.
- [11] NETCDF user guide. <http://www.unidata.ucar.edu/software/netcdf/>.
- [12] D. A. Schneider and D. J. DeWitt. A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment. In *Proc. of the SIGMOD Conf.*, pages 110–121, 1989.
- [13] M. Stonebraker. The design of the POSTGRES storage system. In *Proc. of the 13th VLDB Conf.*, pages 289–300, 1987.
- [14] Stonebraker et. al. A standard science DBMS benchmark. (submitted for publication).
- [15] Stonebraker et. al. Requirements for science data bases and SciDB. In *Fourth CIDR Conf. – Perspectives*, 2009.
- [16] Szalay et. al. Designing and mining multi-terabyte astronomy archives: the Sloan Digital Sky Survey. In *Proc. of the SIGMOD Conf.*, pages 451–462, 2000.

¹<http://msdn.microsoft.com/en-us/vbasic/aa904594.aspx>