# SciDB: A Database Management System for Applications with Complex Analytics

*A description and discussion of the SciDB database management system focuses on lessons learned, application areas, performance comparisons against other solutions, and additional approaches to managing data and complex analytics.*

Historically, science users with big data problems often created their own solutions from the bare metal on up. Examples abound, including NASA's Mission to Planet Earth and the Large Hadron Collider (http://lhc.web.cern.ch). In 2007, when the data-management team at the Stanford Linear Accelerator Center (SLAC) was charged with managing data for the next "big science" astronomy project—the Large Synoptic Survey Telescope (LSST)—team members realized that they had to plan on ingesting nearly 100 petabytes of astronomy data. To discuss the challenge of coping with petascale information, one of the authors of this article (Jacek Becla) organized the 1st Extremely Large Data Base Workshop at SLAC in October 2007.[1]

Several science groups attended the workshop, including the Large Hadron Collider and LSST teams. Everyone involved realized that relational database management systems (RDBMSs) wouldn't work on their problems, but were reluctant to build a new data-management system. Several teams from large Web properties also attended the workshop and indicated that they, too, had petascale problems, that RDBMSs weren't a solution, and that they had the resources to build a new system. Several RDBMS vendors were also present; they were in denial that their systems wouldn't work on the classes of problems presented. Finally, two representatives from the database research community (David DeWitt and Mike Stonebraker, who is an article author) were present and said, in effect, that they had been trying (fairly unsuccessfully) to work with the science community for years, most recently on the Mission to Planet Earth. They were ready to help; that is, if the workshop participants could define functional requirements for a broadly applicable science DBMS, then they would try to build it. What followed was a sequence of small workshops to define what ultimately became SciDB. A previous article presents many of the design decisions.[2]

We're now delivering a production version of SciDB with data that can be distributed across the nodes of a Linux cluster. Performance has been improved markedly, documentation written, and a quality-assurance process has made the code base reliable. These tasks have been largely undertaken by a commercial entity, Paradigm4. The current code base can be downloaded from http://scidb.org.

1521-9615/13/$31.00 © 2013 IEEE
Copublished by the IEEE CS and the AIP

Michael Stonebraker
*Massachusetts Institute of Technology*
Paul Brown and Donghui Zhang
*Paradigm4*
Jacek Becla
*SLAC National Accelerator Laboratory*

## Other Solutions to Science Data Challenges

Before we delve into the inner workings of SciDB, let's discuss solutions routinely proposed for scientific data. We focus on the problems a science user would have with RDBMSs, using the Hierarchical Data Format (HDF) file system and Apache Hadoop software library.

### Science Data and RDBMSs Are Mismatched

Most science data isn't naturally modeled as relational tables. Consider, for example, satellite imagery. When you see such imagery, it's a 2D array with a value for each latitude–longitude cell. In some applications, time is a third dimension and sensor frequency could be a fourth dimension. Simulating array data on top of tables is an unnatural act and usually results in poor performance.

Arrays appear to be the natural data structure for satellite imagery, as well as for astronomy data (where the sensor collectors are pointed up rather than down). Similarly, high-energy physics data is a collection of time series data from spatial sensors. A similar organizational structure is seen in data from natural resource (oil and gas) exploration. Essentially, all simulations (computational fluid dynamics, climate modeling, and oceanography) entail cell-based partial differential equation models, which result in array data. Genomics databases are multidimensional arrays, where the sequence for a given human is one axis and the identifier of the person is a second axis. Other genomic data (SNP and microarray observations) look similar.

Even in biology and chemistry, where the natural data structure is a graph, it's entirely possible that arrays will be the chosen implementation. Specifically, consider a sparse array, where there's a non-null $(i, j)$ cell value if there's an arc from node $i$ to node $j$. As part of the SciDB project, we are exploring simulating graphs as sparse matrices. Our conjecture is that popular algorithms (minimum cut set, reachability, and so on) can be efficiently coded as sequences of linear algebra array operations. We expect to compare this mechanism against application-specific data structures for the GraphLab prototype (see www.graphlab.org), as well as native graph DBMSs.

Also, it's noteworthy that the Sloan Sky Survey[3] has been deployed on top of Microsoft SQLServer with excellent results. However, this project had the benefit of a world-class computer scientist (Jim Gray) doing database design and tuning. Even so, future requirements from the sky survey developers appear to necessitate array data structures. Also, the sky survey doesn't include the raw imagery from the telescope; it deals only with derived information on celestial objects.

Continuing with the satellite imagery example, the popular query operations subset the data in multiple dimensions and then perform transformation or "regridding." A reasonable transformation would be the conversion to a different coordinate system (such as from latitude–longitude to Mercator). Regridding would mean adjusting the size of cells and/or their centers in preparation for some sort of data fusion. These operations are especially painful on a relational simulation, and run dramatically slower than a native array implementation.

A second type of operation is linear algebra. Most complex analytics (curve fitting, regressions, machine learning, $k$-nearest neighbors, and so on) are best expressed as linear algebra operations over arrays. Obviously, we would want to be able to subset and/or transform the data and then run complex analytics, all without leaving a single data model or moving data around between subsystems.

It's clearly desirable to have a query language with science-appropriate operators, rather than the business data processing ones in SQL. Array Query Language (AQL), the query language for SciDB, has been designed with this objective in mind. Moreover, there is an effort underway to construct a standard array query language.

Science users are adamant that data should never be discarded. Even when data elements are wrong, the correction should not overwrite the original data, because previous models and published results might have used the previous datasets, and they must be preserved for provenance. In general, a no-overwrite storage system is required, and care must be taken to preserve the updates that created any new data element.

Furthermore, all science data is uncertain (that is, has error bars); this is in contrast to business data, which is usually precise (for example, my salary is an exact number). Hence, scientists universally want a DBMS that understands uncertain data. Lastly, RDBMSs have a single notion, null, to mean "data not present." Most science users need multiple types of nulls—for example, data not present but it's a "dropout," data not present but it will be coming, and so on. None of these features appear to be important to the vendors of relational systems, who are focused on the business market.

Any system operating at petascale will be distributed over many nodes in a local cluster as well

as geographically over remote clusters. Hence, hardware scalability is an absolute requirement. Any system that can't run over hundreds to thousands of nodes is dismissed.

For better or worse, the science community won't accept a proprietary code base. The general reasons seem to be twofold. First, scientists want control over their destiny. In other words, if they encounter a show-stopping bug and the vendor doesn't fix it quickly, they want the ability to fix it themselves. Second, the community has had bad experiences with the support from proprietary DBMS vendors.

In summary, RDBMSs seem inappropriate for the vast majority of science applications because they have the wrong data model, the wrong query language, lack important features, and don't provide scalability in an open source code base. The goal of SciDB is to do for the science community what RDBMSs accomplished in the business world.

### File Systems

The current gold standard for scientific data is file systems. This low-level interface has myriad problems. First, scientists routinely encode the date of the experiment, the sensor identifier, and other metadata in the file name. This makes searching the metadata problematic. Also, it's difficult to enforce standard naming conventions across a community of researchers. Metadata about experiments should be stored in some DBMS for easy search and management. Second, scientists who wish to operate on a subset of the data must write their own accessing routines. Similarly, if they want to combine their data with another dataset, they must write join routines. Hence, standard DBMS capabilities aren't available in a file system setting. Third, scientists often have to share their data with other researchers. As such, colleagues must know the exact layout of file data to use foreign files. This leads to scientists sharing access routines, written in a lower level language, which often won't compile in foreign systems. In general, file systems make data sharing difficult.

Although some science projects, notably the Large Hadron Collider, put their metadata in a DBMS, this solves only the first of the presented difficulties. The whole idea behind SciDB is to put all data in a DBMS and push science users to a higher level of abstraction. In other words: do for science data what RDBMSs did for business data.

### HDF

HDF is a file format and an interface specification at a bit higher level than the file system. It's nice in that it's an array-oriented facility and is popular in certain science domains. SciDB currently has a loader that converts HDF5 data into SciDB format, so users can access their data using the higher-power SciDB facilities. In addition, we envision being able to process HDF5 data "in situ," so that scientists who don't want to go through the effort of loading their data can still use SciDB capabilities, albeit at lower performance and without the transactional guarantees of a DBMS.

### Hadoop

Hadoop is often cited as the desirable solution for scientific data processing. Here, we consider three different use cases.

***Problems that look like "grep."*** This includes processing files sequentially looking for a pattern of interest. Most text processing fits this paradigm. In addition, performing transformations on streams of input data (called *extract*, *transform*, and *load* in the business world) is an equally parallelizable task. Hadoop is good at these "embarrassingly parallel" use cases.

***Problems that look like queries.*** This use case would encompass most anything written in Hive (see http://hive.apache.org). There has been considerable literature on Hadoop's performance shortcomings relative to parallel DBMSs.[4] In this case, we should expect Hadoop to be at least an order of magnitude slower than a DBMS, because of Hadoop's inefficient communication model. Hence, Hadoop is suitable for small pilot projects, but the scalability needs of production applications will pose serious issues.

***Problems that look like data analytics.*** These might be coded in R, Matlab, Mahout, Pregel, or some analytic front end for Hadoop. In this case, the same communication issues previously noted will make Hadoop an order of magnitude or more slower than alternatives, such as SciDB.

In general, pilots will be fine on Hadoop in the latter two use cases, but scalability issues are likely to ensue. SciDB has been designed for these sorts of applications with native array storage, a high-level query language, and efficient parallel execution on both data management and analytic queries.

## SciDB Architecture and Capabilities

This section describes a few of the major design decisions in SciDB, including the array data model, AQL, the storage organization for arrays on disk,

and the execution of queries. We describe how SciDB implements a no-overwrite store and manages multiple nodes in a computer cluster. We close with a discussion of our interface to the statistical package, R.

### Array Data Model

SciDB has a native array data model. Hence, the logical object accessible to users is an *N*-dimensional array, and not a table. Specifically, each cell in an array can have a vector of values (sometimes called a *composite data type*). Dimensions can be either the familiar integers found in most programming languages or user-defined types (strings, floats, latitude, longitude, and so on). Consider, for example, an array of sensor data specified in Figure 1 using the SciDB data definition facility.

This array has two dimensions, `SensorID` and `Timestep`. In each cell, there is a vector of three values for `Windspeed`, `Temperature`, and `Conditions`. For each value we must specify the object's data type, if it's not an integer. Note that we have suppressed the array bounds for simplicity, as well as storage parameters. Table 1 shows some sample data for this array.

Arrays with integer dimensions are divided into *storage chunks*, presently composed of a *stride* in each dimension. Hence, chunks are rectilinear regions of the array. Chunks are stored on disk in a format described later. With noninteger dimensions, SciDB must map the given dimensions onto integers using a B-tree, and then store the array as previously described. Last, SciDB must cope with "ragged arrays" that might model, for example, the ocean along a coastline. In this case, SciDB allows a "not valid" value for arbitrary collections of cells in any array.

The choice of whether data should be an attribute or a dimension is a logical database design problem that should be based on the expected workload to be processed. Moreover, SciDB supports schema migration, so attributes can be promoted to dimensions and dimensions deprecated to attributes. Also, attributes and dimensions can be added and removed.

```
CREATE ARRAY Sensor_Data
    < WindSpeed : double, Temperature : double, Conditions : string >
    [ SensorID (string), Timestep];
```

Figure 1. Creating an example array.

### AQL

RDBMSs standardized on SQL many years ago as a convenient access language to tabular data. In a similar vein, an array DBMS needs a query language. In early conversations with potential users about this, we got two types of reactions:

- Make it look like SQL, because that's what our programmers know.
- Make up an "operator language" by defining a collection of primitives, and then let me cascade them together to produce the result I'm seeking. Such a language would look a lot like APL from the 1980s.

Our AQL language satisfies the first group of users. Given `Sensor_Data` described previously, a user can find a subset of interest by a filter operation, as shown in Figure 2.

This logically creates a subarray of the original array satisfying the indicated predicate. In effect, a filter produces an array with the same "shape" as the original array but with a greater number of "not valid" cells. In contrast, a relational filter just produces a smaller table. Hence, AQL's semantics are a bit different than relational semantics.

An AQL user is free to use linear algebra operations. Figure 3 shows how to express a Pearson correlation between `Temperature` and `Windspeed`, and also indicates the standard SQL-like construct of allowing anything that resolves to an array to appear in the `from` clause.

In addition, AQL compiles into a functional language that we call *Array Functional Language* (AFL). Our second priority is to surface and document this interface, so that users in the second category from the list will have a procedural language to their liking.

**Table 1. The sample array.**

| SensorID | Step 1 | … | Step 500 | … | Step 5,000 |
|---|---|---|---|---|---|
| "A" | (12.5, 75.1, clear) | … | (11.5, 69.5, cloud) | … | (9.5, 65.2, rain) |
| "B" | (1.0, 55.2, cloud) | … | (0.5, 55.2, clear) | … | (2.0, 55.2, clear) |
| "C" | (10, 35.1, snow) | … | (12.5, 34.2, snow) | … | (12.5, 33.8, snow) |
| "D" | (0.5, 85.1, clear) | … | (1.0, 85.5, clear) | … | (0.5, 85.7, clear) |
| "E" | (15.2, 66.2, rain) | … | (7.9, 66.5, clear) | … | (12.7, 66.9, clear) |

```
SELECT S.Temperature
FROM Sensor_Data S
WHERE S.WindSpeed < 10 AND
S.Timestep BETWEEN 550 and 650;
```

Figure 2. Example Array Query Language (AQL) query.

```
SELECT *
From Pearson (S.Temperature, S.Windspeed)
```

Figure 3. Example AQL query with linear algebra.

For a full discussion of the query language, see the SciDB website (http://scidb.org).

### Storage Management

An array is stored on disk in fixed logical-size chunks. Moreover, if there are multiple cell values, then there are multiple physical arrays, one for each cell value. Hence, storage chunks contain values for only one field type. The desired size of such storage chunks is a few megabytes, so that a disk seek is amortized over a large number of bytes read.

To deal with "not valid" cells in an array, SciDB allocates an extra storage chunk for this information. This extra chunk contains a heavily encoded list of "not valid" cells, and no space is allocated in the data chunks for such cells. The SciDB executor reads this out-of-band chunk to decide how to interpret the values in each data chunk. Each data chunk then contains a list of valid values, which are stored in row-major order and run length encoded. Wound into this encoding scheme is an allowance for multiple null types. It's possible at execution time to remap the various null types into runtime values; for example, some can be mapped to null and some to zero. Over time, we expect to experiment with other compression schemes.

Also, SciDB optimizes CPU performance by performing vector processing, which was originally perfected in Monet.[5] Using this tactic, we set up a pipeline of "subchunks" and blasts through a whole collection of values, so the setup overhead is amortized over the execution of many values.

It should be clearly noted that this chunking strategy supports multidimensional queries in a straightforward way. Imagine, for example, a database of US persons with their home address as a (latitude, longitude) pair. In SciDB, we simply make latitude and longitude dimensions and then chunking automatically provides a 2D index, enabling the 2D searches popular on such data. In contrast, in an RDBMS, we would have to either choose a primary attribute for clustering, use a materialized view to provide two clusterings, or try to provide performance through secondary indexes. In any such scenario, SciDB will be noticeably faster and much simpler to manage.

However, the current design suffers from skew problems. For example, the density of people in Montana is several orders of magnitude lower than the density of people in Manhattan. Hence, fixed logical-size chunks might be highly variable in actual physical size. This skew in chunk size could cause performance problems, because the space on disk occupied by chunks can vary drastically. In addition, the setup time to process a chunk is significant, and in sparse chunks the time is amortized over only a few values.

To overcome this skew problem, we're experimenting with two different strategies. The first is to assemble sparse chunks together into "super chunks" that are stored together on disk. This will make disk blocks much more uniform in size. However, this approach won't deal with the CPU time required to perform setup for small chunks. Our second strategy is to move to variable size chunks, using some sort of hierarchical decomposition, such as quad or R trees.

### Query Execution

Query execution generates all the challenges of relational query execution, plus has added issues due to the array data model. For example, in AQL, it's possible to specify joins that align collections of

- dimensions in one array to those in a second one,
- dimensions in one array to collections of cell values in a second one, and
- cell values in one array to collections of cell values in a second one.

In the first case, if the two arrays are compatibly chunked, then SciDB executes a high-performance pair-wise chunk join. If the two arrays aren't chunked compatibly, then SciDB rechunks one array to the chunking scheme of the other. In the second case, SciDB redimensions the cell-side array to the dimension chunking scheme of the second array. Finally, in the third case, both arrays must be redimensioned. In effect, this strategy is a generalization of the merge-and-sort scheme popular in RDBMSs, because rechunk and redimension are effectively sorting in a multidimensional space. The current release doesn't implement a hash and join strategy, but we plan to

explore this join tactic. An article on our storage manager and executor is in preparation.[6]

Now, we turn our attention to the execution of linear algebra operations. It's common pragma that analysis packages like ScaLAPACK and ARPACK are considerably faster and more extensive than what can be freshly coded with ordinary effort. Moreover, the accuracy of these solutions has been thoroughly verified. Given that they've been tested and optimized over the years, it's difficult to be competitive without considerable tuning and quality-assurance effort.

Therefore, we adopted a *loose coupling* model for analytics. Hence, we run ScaLAPACK alongside SciDB on the same hardware. Whenever a matrix operation is performed, SciDB reformats the data to ScaLAPACK format and sends it to ScaLAPACK. Because essentially all ScaLAPACK operations run in $O(N ** 3)$ time and reformatting runs in $O(N ** 2)$ operations, the reformatting operation won't be the dominant cost—at least for large $N$.

However, this architecture creates a serious resource-management problem. In other words, there are two large software systems coexisting on the same hardware. We're presently working on a system to deal with such contention for resources.

In addition, Intel has released its Xeon-Phi chip (previously called *Knights Crossing* or *MIC*) that packs 62 cores on a chip, each a high-speed arithmetic processor. We expect to experiment with running ScaLAPACK on these chips and SciDB on regular Xeon ones. This sort of accelerator system may alter the resource consumption dramatically and make something other than ScaLAPACK CPU and memory the bottleneck. An article on this experimentation is in preparation.[7]

We should clearly note that SciDB offers huge advantages over ScaLAPACK or ARPACK in isolation—namely, the ability to mix and match analytics with data management in the same environment, without having to learn two systems or manually copy data back and forth.

### No-Overwrite Storage

As we mentioned before, scientists are adamant that they don't want to discard old data. Hence, SciDB allocates a special dimension for each array, called a *version*. This is an integer dimension, which begins at zero and monotonically increases with each update until the array is destroyed. Inserts and updates put new values at the appropriate version in the array, leaving the previous values intact. Then, deletes simply put a "not valid" value into deleted cells at the appropriate time. It's straightforward to tie our versions to wall-clock time or to version names.

At the physical level, each cell in a chunk is maintained as a backward delta of values, reflecting the fact that most applications typically want the current version of a cell value. AQL allows any user to query the database as of any point in time. Hence, time travel is supported, as originally conceived in Postgres.[8] When historical queries are run, SciDB must decode this chain of backward delta. At their discretion, users can delete or archive older data to manage database storage over time.

No-overwrite storage provides accurate provenance in conjunction with a log of update statements. This approach also supports database auditing, provides regulatory compliance, and enables result reproducibility.

### Shared-Nothing Design

To scale to petabytes of data, SciDB must run across collections of nodes in a cluster of machines. We follow the standard practice of partitioning the data across nodes. Hence, chunks are partitioned in one of several ways. Query execution can then proceed by running a local query in parallel across some collection of nodes followed by a scatter-gather data shuffle to rearrange data for the next collection of local operations. As such, this is quite similar to the query processing strategy of parallel RDBMSs.

To support parallel execution of nearest-neighbor queries, feature detection operations, and other data clustering operations, SciDB lets chunks overlap by a user-specified amount, intended to be the radius of the largest cluster or feature in the user's data.

### SciDB-R

In talking with many scientists, we find that R is by far the most popular statistics package and visualization tool. However, there's frustration that this package doesn't scale to multiple nodes in a cluster or to data that doesn't fit in the main memory. Moreover, R doesn't perform the data-management operations that usually precede a statistical one.

To alleviate this difficulty, we built an interface for R that lets R scripts access data residing in a SciDB database. Hence, R commands are exported to SciDB, where they can be run in parallel, and the answer is stored in SciDB for further processing. Subsequent R commands are similarly pushed to SciDB, along with other R data needed as operands. Hence, this produces a scalable version

```
CREATE ARRAY Swath_Sensor_Data
< Sensor_Value : double >
[Time(datetime), Latitude(lat), Longitude(long)];
```

Figure 4. Example SciDB array for modeling remote sensing satellite data.

of R, as well as one that provides standard data-management services directly though SciDB. A similar interface has been prototyped in other work.[9]

## Early Use Cases

We now discuss early use of SciDB in three different domains: satellite imagery, astronomy, and genomics.

### Satellite Imagery

Raw satellite imagery is best thought of as a wide piece of scotch tape that's wrapped repeatedly around the earth. In effect, the satellite is at a specific place above the earth and scans a "swath" of real estate beneath itself. As the satellite moves forward on its orbit, it traces out a wide piece of scotch tape, for each cell of which it records a collection of sensor readings at various frequencies. Hence, raw imagery is a 3D array, as Figure 4 shows.

In current practice, this imagery is "cooked" into various higher-level data products that are freely available from NASA. However, earth scientists face a cruel dilemma when the higher-level products don't quite meet their science requirements. These researchers can either learn a great deal about the (arcane) formatting of level-one (raw) imagery and then spend a lot of time coding their specific requirements, or they can live with the higher-level products that don't meet their science needs.

To provide earth scientists with a better system for Moderate Resolution Imaging Spectroradiometer (MODIS) imagery, we designed a software system to ingest level-one imagery into SciDB and then wrote the "cooking" algorithms used by NASA to produce the higher-level products. Hence, all of these tools are available in a database, and users can freely subset the data of interest to them, and then provide their own cooking as SciDB queries. In a recent article, we explained this system and showed how to create some popularly derived information with little effort through SciDB queries.[10]

### Astronomy Applications

Satellites look downward; telescopes look upward. Hence, LSST is pointing its telescope (think of it as a digital camera) at some portion of the sky and recording a 2D image. This is then "cooked" into observations of astronomical objects and trajectories of these objects over time. This application looks much like satellite imagery, and the SLAC data-management team is prototyping SciDB (along with other possible solutions) in preparation for a complete LSST deployment in a few years.

The Lyra astronomy project is charged with forming a common repository for observation data from multiple telescopes.[11] To do so, researchers must do a spatial join of two or more arrays of observations. However, there are obviously errors in recording the data due to fundamental uncertainty and calibration issues. Hence, Lyra must perform a "fuzzy" spatial join.

The Lyra team has coded its application on SciDB using spatial geometry as dimensions and a user-defined function, *cross-match*, which performs the fuzzy join calculation. With the use of overlapping chunks, discussed previously, cross-match can be efficiently executed as a pairwise chunk join. This operation was also coded in Postgres using its spatial indexing capability. On identical hardware, SciDB was approximately a factor of two times faster than Postgres.

### Genomics

Imagine constructing the complete genome for a single human—a few billion symbols long—from an alphabet of four options. The cost of sequencing a human is expected to drop to less than $1,000 within a year or two, and biologists expect thousands to hundreds of thousands of humans to be sequenced. The result is a gigantic 2D array. Moreover, the "holy grail" in this field is to have human disease characteristics as a third axis. Massive correlations then would yield genomic markers (if they exist) for diseases. SciDB is being prototyped in this application area quite aggressively.

An early SciDB user wanted to perform biclustering—a popular technique for finding clusters of possibly related genes—in a large microarray dataset. Specifically, he wanted to use a technique developed in other work,[12] which is based on a sparse singular value decomposition (SVD). This algorithm was coded as a user-defined function in SciDB, which had an inner loop of matrix multiply and transpose, and was called using SciDB-R. In addition, the same algorithm was coded in standard R.

Two results were evident. First, the SciDB implementation ran in parallel over many servers

and provided linear scalability. The R solution was, of course, limited to a single node. As such, SciDB could dramatically accelerate response time to large SVD problems. Second, we ran a 50,000 × 50,000 SVD computation; on 16 nodes in an Amazon Elastic Compute Cloud cluster, the computation, running in parallel, finished in 20 minutes. The computation couldn't be run in standard R, because the problem size exceeded the R matrix indexing limits.

### Commercial Applications

SciDB is also getting traction in many non-scientific domains, including analytics on location-based data, mass personalization of automobile insurance rates using in-car sensors, and manufacturing and industrial analytics on failure data for parts and subassemblies.

Science users made it crystal clear to us that they required a commercial-quality database system with high performance, good documentation, and rigorous quality assurance. There's no possibility of writing "industrial strength" system software in an academic research laboratory. Hence, we elected to follow the open source model pioneered by Red Hat, wherein a community edition is available without charge and without support, while an enterprise version is also available and is supported in the standard way. As such, capable science users can access a DBMS that meets their needs at no charge; more than a thousand users have downloaded the code to date from http://scidb.org. Users who want support can get it from the commercial company (Paradigm4).

With this dual model, Paradigm4 is coordinating SciDB development and doing much of the heavy lifting. There's a small (but growing) community of contributors to the code base from the science community. Paradigm4 is also staffing quality assurance and documentation.

In addition, there is a dedicated group of academic researchers working on issues concerning array databases. Dave Maier and Kian-Tat Lim are working on a standard array query language in conjunction with Martin Kersten, who is working on a SciQL layer on top of MonetDB, and Peter Baumann, who is addressing a raster data manager (Rasdaman) array DBMS. Magda Balazinska and Emad Soroush are working on discovering better chunking schemes, and their initial results have appeared in other work.[13] Mike Stonebraker and Donghui Zhang are working on a cost-based optimizer for the AQL language. Furthermore, we designed a benchmark that captures the essence of LSST requirements, and Philippe Cudre-Mauroux is working on running this benchmark on a variety of database engines.[14] Also, we're prototyping a version control system[15] and provenance solutions.[16] Much of this work is supported by the US National Science Foundation, and will hopefully lead to increasingly better array implementations.

At present, Paradigm4 has just released version 1 of SciDB. It's a fully functional array DBMS with good performance and high reliability. Tools for database design, a Web–client interface, and system monitoring haven't been constructed yet, so the user-interface and database administration are a bit clunky. Moreover, in version 2, we expect to improve current performance substantially. A parallel loader is available and support for HDF5 is in process. Finally, we expect to support the standard relational interfaces of Open Database Connectivity and Java Database Connectivity, so that we can connect to popular applications systems, especially visualization packages such as Tableau and Spotfire.

Our goal is to make SciDB the de facto standard for science applications and to bring the power of scientific data management and scientific analytics to the commercial and industrial worlds. CISE

## References


1. J. Becla and K.-T. Lim, "Report from the First Workshop on Extremely Large Databases," *Data Science J.*, vol. 7, 2008, pp. 1–13.
2. M. Stonebraker et al., "The Architecture of SciDB," *Proc. Scientific and Statistical Data Management Conf.*, Springer-Verlag, 2011, pp. 1–16.
3. A.S. Szalay et al., "Designing and Mining Multi-Terabyte Astronomy Archives: The Sloan Digital Sky Survey," *Proc. Sigmod Conf.*, ACM, 2000, pp. 451–462.
4. A. Pavlo and et al., "A Comparison of Approaches to Large-scale Data Analysis," *Proc. Sigmod Conf.*, ACM, 2009, pp. 165–178.
5. P.A. Boncz, S. Manegold, and M.L. Kersten, "Database Architecture Evolution: Mammals Flourished Long Before Dinosaurs Became Extinct," *Proc. Conf. Very Large Data Bases* (VLDB), VLDB Endowment, 2009, pp. 1648–1653.
6. J. Duggan, "Compression and Execution in SciDB," in preparation.
7. J. Dongarra et al., "A Composite Data Management and Linear Algebra Benchmark," in preparation.
8. M. Stonebraker: "The Design of the Postgres Storage System," *Proc. Conf. Very Large Data Bases*, Morgan Kaufmann, 1987, pp. 289–300.

9. P. Leyshock, "Agrios: A Hybrid Approach to Scalable Data Analysis Systems," Extremely Large Data Base Workshop, presentation, 2012; www-conf.slac.stanford.edu/xldb2012/talks/xldb2012_tue_LT06_Leyshock.pdf.
10. G. Planthaber et al., "EarthDB: Scalable Analysis of MODIS Data Using SciDB," *Proc. 1st ACM SIGSPATIAL Int'l Workshop on Analytics for Big Geospatial Data*, ACM, 2012, pp. 11–19.
11. A.V. Mironov et al., "The Multicolor 'Lyra' Photometric System for Variable stars and Halo Studies," 2010; http://arxiv.org/abs/1002.4644.
12. A. Bhattacharjee et al., "Classification of Human Lung Carcinomas by mRNA Expression Profiling Reveals Distinct Adenocarcinoma Subclasses," supplementary material, *Proc. National Academy Science*, vol. 98, no. 24, 2001, pp. 13790–13795; www.pnas.org/content/98/24/13790/suppl/DC1.
13. E. Soroush et al., "ArrayStore: A Storage Manager for Complex Parallel Array Processing," *Proc. Sigmod Conf.*, ACM, 2011, pp. 253–264.
14. P. Cudre-Mauroux et al., "SS-DB: A Standard Science DBMS Benchmark," submitted for publication.
15. A. Seering et al., "Efficient Versioning for Scientific Array Databases," *Proc. Int'l Conf. Data Eng.*, IEEE, 2012, pp. 1013–1024.
16. E. Wu et al., "A Demonstration of DBWipes: Clean as You Query," *Proc. Conf. Very Large Data Bases*, VLDB Endowment, 2012, pp. 1894–1897.



***Michael Stonebraker*** *is an adjunct professor at the Massachusetts Institute of Technology's Computer Science and Artificial Intelligence Laboratory, and codirector of the MIT Intel Science and Technology Center. His research interests include database research and technology, science-oriented and online transaction processing DBMSs, and scalable data curation. Stonebraker has a PhD in computer, information, and control engineering, University of Michigan. He was awarded the ACM System Software Award in 1992, given the Innovation award by the ACM SIGMOD in 1994, and elected to the National Academy of Engineering in 1997. He was awarded the IEEE John Von Neumann award in 2005. Contact him at stonebraker@csail.mit.edu.*

***Paul Brown*** *is employed by Paradigm4 as the architect for the SciDB platform. His research interests include scientific and statistical data management. Paul Brown received a BSc in computer science from the University of Queensland, St. Lucia, Australia. Contact him at pbrown@paradigm4.com.*

***Donghui Zhang*** *is the director of Software Development at Paradigm4. His research interests include spatial and temporal database indexing and query processing. Zhang has a PhD in computer science from the University of California, Riverside. He received a US National Science Foundation Career Award in 2004. Contact him at dzhang@paradigm4.com.*

***Jacek Becla*** *leads the Scalable Data Systems team at SLAC National Accelerator Laboratory. His research interests include high-energy physics, astronomy, and photon sciences, relating them to database technology for managing and analyzing their massive datasets. Becla has an MSc in electronics engineering from AGH University of Science and Technology, Krakow, Poland. He chairs the Extremely Large Database (XLDB) conference series, and leads the XLDB community. Contact him at becla@slac.stanford.edu.*