REPORT FROM THE SciDB WORKSHOP

J Becla[*1] *and K-T Lim*[2]

Stanford Linear Accelerator Center, Menlo Park, CA 94025, USA
[*1] *Email: becla@slac.stanford.edu*
[2] *Email: ktl@slac.stanford.edu*

ABSTRACT

A mini-workshop with representatives from the data-driven science and database research communities was organized in response to suggestions at the first XLDB Workshop. The goal was to develop common requirements and primitives for a next-generation database management system that scientists would use, including those from high-energy physics, astronomy, biology, geoscience and fusion, in order to stimulate research and advance technology. These requirements were thought by the database researchers to be novel and unlikely to be fully met by current commercial vendors. The two groups accordingly decided to explore building a new open source DBMS. This paper is the final report of the discussions and activities at the workshop.

Keywords: Database, XLDB

1 ABOUT THE WORKSHOP

The workshop was held March 30 through April 1, 2008, in Asilomar, CA. The workshop organizing committee was composed of Jacek Becla, Mike Stonebraker, David DeWitt, and Kian-Tat Lim. Participation was by invitation only in order to keep the discussion focused.

Scientific community representatives, in alphabetical order:

- Astronomy: Kirk Borne (GMU), Robert Lupton (Princeton), and Alex Szalay (JHU)
- Biology: Gordon Anderson (PNL)
- Fusion: Tim Frazier (LLNL)
- Geoscience: James Frew (UCSB)
- HEP: Gregory Dubois-Felsmann (SLAC) and Dirk Duellmann (CERN)

Database research community representatives:

- Academia: David DeWitt (Univ. of Wisconsin), Jignesh Patel (Univ. of Michigan), and Mike Stonebraker (MIT)
- Industry: Tom Barclay (Microsoft), Mike Carey (BEA), Guy Lohman (IBM), and Chris Olston (Yahoo!)

"Unclassified" participants:

- Jacek Becla (SLAC), Kian-Tat Lim (SLAC), and Oliver Ratzesberger (EBay)

2 SCIENTIFIC REQUIREMENTS

This report captures the key results of the workshop, summarizes the discussions that led to them, and documents the resulting action items. It does not attempt to reproduce the exact flow of the highly interactive, *ad hoc* conversations that occurred during the workshop.

One major result of the workshop was a set of requirements that a database management system should meet in order to support the storage and analysis needs of several fields of data-intensive science over the next decade. These requirements were distilled from wish lists presented by each field and from more in-depth analysis of specific desired features. This future science-oriented database management system will be referred to as "SciDBMS."

Summary of Requirements for SciDBMS

Adoption

- Open source with a stable developer community.

Scalability and Performance

- Scalability up to hundreds of petabytes using parallelized single queries on commodity hardware.
- Fault tolerance with intra-query failover.
- The performance, extensibility, and compression of a column store, but also with efficient "`SELECT *`" (or "`SELECT` many columns").

Interfaces

- SQL (with appropriate extensions, if needed).
- An object-relational mapping layer for external applications.
- User-defined functions and stored procedures expressed in familiar procedural languages operating on row cursors or post-ORM object cursors that have ordering and grouping properties.
- Partial results, query progress indicator, and query pause/restart/abort.
- Pre-execution query cost estimate, preferably in wall-clock time.

Features

- No transactions needed for the largest tables, but atomic multiple-table batch appends are needed. Metadata tables may need transactions.
- Support for spatial and temporal operations.
- Support for arrays as a first-class column type.
- Lightweight support for "error-bar" uncertainty of data elements by associating error columns with data columns and adding an "approximately equal" operator.
- Cheap one-to-one and one-to-many joins when both tables are partitioned on the join key.
- Versioning of tables and code, including the ability to tag or label sets that go together.
- Support for provenance of data elements, including querying and import of provenance.
- A resource management system including CPU and disk quotas.
- Support for tiered storage: migration from faster media to slower media (e.g. flash, fast disk, slow disk).

Less Important Features

- Column grouping: asserting for efficiency that certain columns will always be used together.
- Access controls and perhaps namespaces for sets of columns in wide tables.
- Support for probabilistic uncertainty for nominal values.
- XQuery-like engine for querying hierarchical data.

Explanation of Requirements

Adoption

SciDBMS must be open source if it is to achieve adoption in the scientific community. Many projects have been burned by over-dependence on a single commercial vendor. Source escrow is insufficient, as it still leaves ongoing support of the software in a questionable state. Open source alone is also insufficient, however. A robust community of developers must be present to ensure that the software will not languish in obscurity.

Scalability and Performance

SciDBMS must be able to scale to databases of hundreds of petabytes, with individual tables measured in trillions of rows. These sizes are already part of the requirements for the Large Synoptic Survey Telescope (LSST) database, which will store trillions of observations of tens of billions of astronomical objects. It must be able to parallelize single queries, rather than simply allowing multiple queries to execute in parallel, as many sciences will have queries that touch large fractions of the data but must still return results on interactive timescales. Achieving this performance must be possible on commodity hardware; proprietary hardware leads to cost and support constraints that are undesirable for large scientific projects. Since it is using commodity hardware, the system must be fault tolerant. Since some queries will still have long execution times, the ability to continue a running query if a node fails, rather than restarting it from the beginning, is necessary.

Column stores are extremely interesting as a model for SciDBMS for several reasons. They provide performance advantages for queries that operate on a small subset of columns; such queries are expected to make up a significant fraction of scientific workloads. Column stores provide easy extensibility of tables, with low-cost addition and deletion of columns. These operations are frequent in research-oriented databases where schemas are not fully understood in advance. Column stores also promise much greater compression of data on disk, providing advantages both in cost and in performance due to reduced I/O. SciDBMS will require all of these advantages. On the other hand, scientists may need to extract many columns from a wide table to perform complex analyses. Accordingly, SciDBMS must also allow efficient “`SELECT *`” or at least “`SELECT` many columns” queries.

Fault tolerance will likely require storage of two copies of the data. Fortuitously, the second copy might be just what is needed to support a separate “`SELECT *`” view. Requiring more than two copies is less desirable.

Interfaces

SciDBMS must provide a standard SQL interface, with appropriate extensions, if needed. The astronomy community has become familiar with SQL and is able to express complex analytical operations in the language. Many fields are using relational databases and SQL for querying metadata, such as experimental conditions, sometimes within more user-friendly tools. Overall, it provides a common baseline for declarative relational queries.

Beyond SQL, SciDBMS should provide an object-relational mapping layer that external applications can use to process data. This layer will simplify access to the database and allow scientists to think in terms of scientifically-relevant objects instead of normalized rows. Full object support, e.g. with pointers, is likely not needed; mapping to hierarchical structures should be adequate. The mapping layer should have interfaces to at least C++, Java, and Python.

SciDBMS should allow users to define functions and stored procedures in familiar procedural languages such as those mentioned above. These procedures should be able to operate on row cursors or post-ORM object cursors that have ordering and grouping properties. The goal here is to allow scientists to write procedures that compute values dependent on multiple rows, such as looking for significant changes in time series, in a familiar fashion, without having to learn new syntax such as windowing functions. Users should be allowed to specify that the function or procedure is distributive and can be executed in parallel. Since the function or procedure is user-written code, it may have errors, so exceptions must be handled reasonably, and the user must be able to debug the code easily. The functions and procedures must be able to share scans of large data tables with other simultaneously-running queries, particularly when interacting with tiered storage (see below).

The above functions and procedures will essentially provide a workflow capability within the database engine. One of the dangers of this embedding is that using such features ties projects closely to the DBMS chosen. Most large scientific projects span decades, often requiring migrations from one system to another as technology and products change. Such migrations become more difficult as the amount of non-portable code increases. On the other hand, giving the database engine control of the workflow not only permits movement of computation to data that is essential for peta-scale datasets, but also allows the engine to track provenance more simply and accurately (see below).

Many queries to SciDBMS are expected to run for substantial periods of time. It should be possible to pause such long-running queries, restart them, and abort them. These queries should generate partial results so that the user can immediately determine if they are computing useful values and abort them if not. Also, the user needs to get feedback from the system about query progress.

A similar requirement that helps to enable efficient exploration of the data is that SciDBMS should provide a reasonably accurate estimate of the cost of a query before it is executed. This estimate would be most useful if it is in terms of wall-clock time.

Features

The large tables in scientific databases are, like commercial data warehouses, typically read-only after they have been loaded. Instrumental observations should never be updated; derived data is typically not updated in place to ensure reproducibility. As a result, transaction capabilities should not be needed for the largest tables. Instead, what is needed for loading these tables are batch appends that are atomic across multiple tables. Concurrent access to these large tables by a single writer and multiple readers is required; dirty reads are acceptable. Other tables, particularly DBMS-internal metadata tables, may need transactions in order to maintain consistency.

SciDBMS must provide efficient support for spatial and temporal operations. Astronomy and earth sciences operate on two- or three-dimensional spatial grids, often using a plethora of spherical coordinate systems. Transformations between those coordinate systems need not be embedded in the database, but operations relating records using spatial information, such as finding near-neighbors, are essential. Nearly all sciences need to deal with time series of data; it is frequently necessary to understand relationships between consecutive elements in time or analyze entire sequences of observations.

All sciences need to work with non-scalar values like vectors and arrays. SciDBMS must provide support for arrays as a first-class column type.

All sciences must deal with observations and derived data that have inherent uncertainties. The simplest form of this associates each data element with an “error bar.” typically given as a standard deviation. SciDBMS must at a minimum allow an error column to be similarly associated with each data column. Using such a column is easiest if an “approximately equal” operator is provided. This operator would be used in `WHERE` and `JOIN` clauses. Such an operator will need to be able to take a third parameter to indicate the width of the error bar in terms of the number of standard deviations specified by the error column, i.e. the “3” in “± 3 sigma.” More complex uncertainty operations require detailed understanding of correlations between errors and so were thought to be best left to application code, rather than embedded in the database engine.

Each column should be able to be associated with an appropriate unit of measure. SciDBMS should prevent operations on incommensurate units. Ideally, support would be provided for converting conformable units, like that provided by the Unix *units* utility.

Scientists often need to annotate data, adding information based on their own analyses. In some cases these annotations can be columns in a shared table, but in other cases they are most easily thought of as columns in a separate table joined to a primary one. It should be cheap to do one-to-one and one-to-many joins between such annotation tables and the main table when both tables are partitioned on the join key. Enabling the storage of these annotation columns in separate tables could be used to give the equivalent of per-user “namespaces” for columns, can allow differing access control parameters for different sets of columns, and enables such common scientific tasks as classifying an observation in multiple simultaneous ways, with probabilistic weightings of and different attributes attached to each classification.

SciDBMS should allow versioning of both tables and code elements like the procedures described above. The ability to tag or label sets of tables and code that go together is also desirable. The versioning model of the Subversion version control system is one that scientists felt familiar with and believed would be adequate. These capabilities will allow scientific applications to generate reproducible results. They will also make simpler the management of the process of applying new algorithms to old data, a frequent occurrence in all sciences.

Along with versioning, SciDBMS must provide support for recording the provenance of data elements. Provenance information allows users to determine the answers to questions such as:

- What operations led to the creation of this element?
- What operations used this element?

- What data elements were used as input to this operation?
- What data elements were created as output from this operation?

In order to answer these queries, the database will likely need to log all operations that modify data, including creation, deletion, and updates of data elements. Since operations on data will also occur outside the database, it must be possible to import provenance from external systems when loading data. The attributes of an operation that need to be recorded for provenance purposes include not only what was done but also the state of the system at the time. This system state information may be as detailed as operating system patch levels, compiler versions, etc.

Large data-intensive science projects often have diverse user bases, in some cases extending to the general public. To support such environments, it is essential that SciDBMS incorporate a complete resource management system that controls the CPU, disk, and other resources available to the database, including the ability to set quotas.

It will not always be possible or desirable to store all data on the fastest media. Many existing scientific systems provide support for tiered storage, in which data may be migrated from fast disk to slow disk to tape, and maybe also in the other direction to flash memory in the future. SciDBMS must also support this capability.

Less Important Features

There are several features that were considered to be less important or more speculative, but still potentially interesting for science.

In several cases, scientists know that certain columns will always be used together. For example, spatially-oriented fields like astronomy and geoscience will typically have two coordinates, a latitude and longitude or right ascension and declination that are always used as a pair. For efficiency, it may be desirable to indicate this column grouping to the database.

If annotation columns are added directly to a main table rather than placed in separate joined tables as described above, the ability to have access controls and perhaps separate namespaces for sets of columns within a single wide table would be desirable.

Classification probabilities, or probabilistic uncertainties associated with nominal values, are a common element stored in scientific databases, but the use of such probabilities is more complex. In some cases, thresholds will be applied; in others, the probabilities may be used as weights for aggregations. There does not yet appear to be a consensus on the operations and uncertainty model required for using these probabilities.

High-energy physics in particular has traditionally stored its largest datasets in a hierarchical form, rather similar to an XML document, that corresponds closely to the way that physicists think about the data. Other sciences, while amenable to pure-SQL relational queries over their data, may also have more hierarchical views of it, particularly for analyses coded in programming languages as described above. An XQuery-like engine for querying hierarchical data, which could still be stored in a “shredded” relational form, might be interesting as a way of providing a familiar data model while still reducing the amount of code that needs to be written to perform an analysis.

SCIENTIFIC DATABASE CUSTOMERS

Database researchers present at the workshop believed that scientific database requirements are on the bleeding edge of what database management systems can provide but that industrial users will soon catch up in terms of the complexity of their analytics and may need similar features from a DBMS. It was also believed that science itself, particularly as it becomes more data-intensive, is a sufficiently large and growing database market to support a science-oriented DBMS.

During the discussions it became clear that the HEP community has significant experience with handling large-scale data but that it is unlikely to seriously drive the requirements for a near-term SciDBMS both because of the timing of their system deployments and because of the additional complexities of satisfying their community’s historical preference for code-driven rather than query-driven operations on complex, hierarchical data.

On the other hand, other sciences are poised to adopt large scientific databases in the near future. Astronomy, in particular the LSST project, perhaps presents the most significant immediate opportunity. While other fields are rapidly approaching similar scales, the requirements and needs of LSST appear to be a superset of those of other scientific communities.

DEVELOPMENT PLANS

The database researchers present agreed that the requirements presented by the scientific representatives are reasonable and that methods for addressing them are sufficiently well-understood so as not to pose insurmountable research problems. People familiar with large commercial DBMS vendors did not feel that those companies would take on the task, however. As a result, building a new system optimized for data-intensive science is necessary.

A startup company was proposed as the best vehicle for building this system. This company would lead the open source development of SciDBMS, building a working prototype in about a year and productizing it the next. Its primary source of revenue, as for others based on open source, would be from support contracts. The company would likely be located in Silicon Valley to take advantage of the developer talent in the area.

In order to obtain startup funding from the venture capital community, market interest from the scientific community would need to be demonstrated. Sufficient “buy-in” could be shown if there are 2–3 “lighthouse” customers seriously interested in using the product and if science is willing to commit 1–2 full-time-equivalent developers to the project. The personnel portion of this requirement could be assisted by grants from funding agencies such as the Department of Energy’s Advanced Scientific Computing Research program.

There was some discussion as to which code base to use as a starting point. Systems mentioned included PostgreSQL and its derivatives such as Greenplum and EnterpriseDB, Vertica, BerkeleyDB (SleepyCat), Hadoop, MonetDB and MySQL. Some believed it would be easiest to start from PostgreSQL, but this remains to be evaluated.

ACTION ITEMS

- Produce a 10-15 page document sketching a proposed design for SciDBMS [lead = Mike Stonebraker; due end of April].
- Solidify science participation. This includes identifying at least two projects (“lighthouse” customers) interested in using the SciDBMS and willing to work closely with the development team, as well as assessing the level of interest and participation by other potential users [lead: Jacek Becla; due April 20].
- Organize fund raising [lead = Mike Stonebraker; due May 10, after the previous item].
- Discuss the possibility of collaboration with industry (eBay, Google and others) [lead: Jacek Becla; due mid-May, after the design sketch is developed].