An Exploration of SciDB in the Context of Emerging Technologies for Data Stores in Particle Physics and Cosmology

D Malon, P van Gemmeren and J Weinstein
Argonne National Laboratory, 9700 S Cass Ave, Lemont, IL 60439, USA

E-mail: malon@anl.gov

Abstract. Traditional relational databases have not always been well matched to the needs of data-intensive sciences, but efforts are underway within the database community to attempt to address many of the requirements of large-scale scientific data management. One such effort is the open-source project SciDB. Since its earliest incarnations, SciDB has been designed for scalability in parallel and distributed environments, with a particular emphasis upon native support for array constructs and operations. Such scalability is of course a requirement of any strategy for large-scale scientific data handling, and array constructs are certainly useful in many contexts, but these features alone do not suffice to qualify a database product as an appropriate technology for hosting particle physics or cosmology data. In what constitutes its 1.0 release in June 2011, SciDB has extended its feature set to address additional requirements of scientific data, with support for user-defined types and functions, for data versioning, and more. This paper describes an evaluation of the capabilities of SciDB for two very different kinds of physics data: event-level metadata records from proton collisions at the Large Hadron Collider (LHC), and the output of cosmological simulations run on very-large-scale supercomputers. This evaluation exercises the spectrum of SciDB capabilities in a suite of tests that aim to be representative and realistic, including, for example, definition of four-vector data types and natural operations thereon, and computational queries that match the natural use cases for these data.

1. Introduction

While the high energy physics community employs databases extensively for a wide range of purposes, the community has often chosen to develop its own in-house technologies to address many database functionality and persistence requirements that the computer science research community and the commercial marketplace have also endeavoured to address. There is, for example, a sizable literature on temporal databases spanning many decades, but the LHC experiments have developed their own "interval of validity" databases (cf. COOL) for managing time-varying conditions and calibrations, as did the high energy and nuclear physics experiments of earlier generations. Another example is the development of technologies for database content caching (cf. FroNTier), developed for Fermilab Tevatron experiments and now employed by LHC experiments. This tendency toward homegrown solutions is not restricted to databases and persistence technologies (nor indeed to high energy physics): high energy physics experiments have also developed their own configuration and release management tools, distributed data management infrastructure, workload management systems, and more.

This observation is not intended as a criticism—it is often true that available off-the-shelf technologies do not quite meet the requirements of high energy physics, or of other large-scale scientific enterprises—but it does suggest that from time to time it is appropriate to reassess the capabilities of current and emerging technologies and their potential utility to high energy physics.

Database technologies in particular have not traditionally been a good match for many large-scale data purposes in high energy physics. The LHC experiments process and analyze event data at a scale of many tens of petabytes (and soon, hundreds of petabytes), but even relatively small applications such as event-level metadata systems, at a scale of tens of terabytes, challenge the capabilities of relational databases. An event-level metadata infrastructure such as the ATLAS TAG database requires scalable support for variable-length structures (such as the properties of photons, electrons, muons, and jets, whose numbers vary from event to event), support for combinatorial queries (e.g., looking for pairs of oppositely-charged leptons with transverse energy above a threshold), and decoding (possibly bit-level decoding) of time-varying information (e.g., trigger decisions, where the trigger menus and hence the meanings of the bits may vary from run to run). All of these things are possible in principle and implemented in practice with relational databases, but they may be awkward to implement, or difficult to support in a scalable fashion. In this paper we take the opportunity to assess whether and how at least one state of the art development in scientific databases, SciDB, might be applicable to high energy physics data handling, doing so first in what is in some ways the simplest and smallest-scale event data setting in experimental particle physics, event-level metadata, then expanding these considerations to data produced by large-scale cosmological simulations.

2. Databases and SciDB: Some History

Even as the high energy physics (HEP) database "software stack" for LHC-era experiments has gradually emerged, the capabilities of commercial products and software from outside the community have been improving. Database technologies have evolved in a number of relevant ways. Column-oriented data storage, matching the access patterns of many kinds of analysis, is now available in several products. Temporal database technologies have grown in capability and range of applicability, and distributed caching is common in the web era. More importantly, high energy physics data scales are no longer rare: the data scales of Google and Amazon, and the data generated by the pervasive use of social media, now make HEP data challenges look tractable, though the access and analysis patterns are of course different in their demands. It is natural, then, to ask, whether there emerging database technologies that the HEP community should be tracking.

Motivated in part by the requirements emerging from the Large Synoptic Survey Telescope (LSST), the first XLDB (for extremely Large DataBase) workshop was convened in October 2007. At that meeting and at a followup meeting in March 2008, a consensus emerged that there are requirements common to a number of large-scale scientific enterprises that a collaboration between the academic and commercial database communities might be able to address. Shortcomings identified in current offerings included the lack of integrated analytical processing capabilities, the need for arrays, which are fundamental in science but difficult to emulate in relational databases, insufficient provenance and version control management, lack of internal support for uncertainty and error bars, an operations mismatch with SQL, and more. The commonality of such requirements led to the launch of the SciDB project, which seeks to address "the need for a scalable open-source data management with integrated analytics to support scientific research on massive datasets." In June 2011, the SciDB team released software that could be considered Version 1.0 of a product intended to address these common needs.
It is this product that is the subject of the explorations described in this paper.

3. Features of SciDB

Key features of SciDB are its array-oriented data model, its support for versioning and provenance, its shared-nothing architecture to allow massively parallel computations, its support for user-defined types and functions, and its native support for uncertainty.

3.1. Array Data Model

SciDB uses an array data model and supports multi-dimensional, nested arrays, which are often a natural representation for spatially- or temporally-ordered data. Array cells contain records with tuples of multiple attributes and/or one or more arrays. All cells in an array have the same attributes and data types. Arrays can have multiple named dimensions represented by contiguous integer values. SciDB supports unbounded dimensions in basic arrays. SciDB arrays may be sparsely populated and can have jagged edges, allowing each row/column to have a different dimensionality (see Figure 1). SciDB supports two separate classes of "missing" information, and allows user-definable, context-sensitive treatment of these cell values.

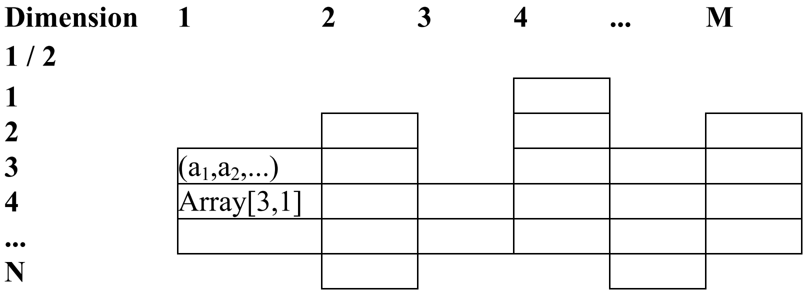


Figure 1. A two-dimensional SciDB array with jagged edges.

Data in a SciDB array is not updateable, but new array data can be appended to a SciDB database and the results of a SciDB query can be written back to storage. This behaviour matches the typical scientific use case in which data is read-only with updates being handled by versioning.

3.2. Preserving ordering helps complex computations

Unlike relational databases, which store their data in tables without row or column ordering, SciDB uses an array data model, with implicit ordering. This ordering allows for the notion of "adjacency," which is desirable for many scientific domains (see Figure 2). In many science applications, the order of data is not simply given by its representation, but is fundamental to the semantics; e.g., for sensor data, cells that are adjacent in terms of cell indices are physically adjacent, and for collider data, event ordering may correspond to temporal sequencing during data taking.

Relational Database

I	J	Value
0	0	32.5
1	0	90.9
2	0	42.1
3	0	96.7
0	1	46.3
1	1	35.4
2	1	35.7
3	1	41.3
0	2	81.7
1	2	35.9
2	2	35.3
3	2	89.9
0	3	53.6
1	3	86.3
2	3	45.9
3	3	27.6

SciDB

I / J	0	1	2	3
0	32.5	46.3	81.7	53.6
1	90.9	35.4	35.9	86.3
2	42.1	35.7	35.3	45.9
3	96.7	41.3	89.9	27.6

Figure 2. In SciDB arrays adjacency of cells is preserved, whereas relational tables do not support ordering.

3.3. Data Storage

In SciDB array dimensions can be “chunked” and data of each chunk can be stored on individual nodes. As each node has processing and storage capabilities, this partitioning allows shared-nothing operation. Chunk “overlaps” can be defined, so that certain operations involving neighboring cells are possible without communication among nodes (see Figure 3).

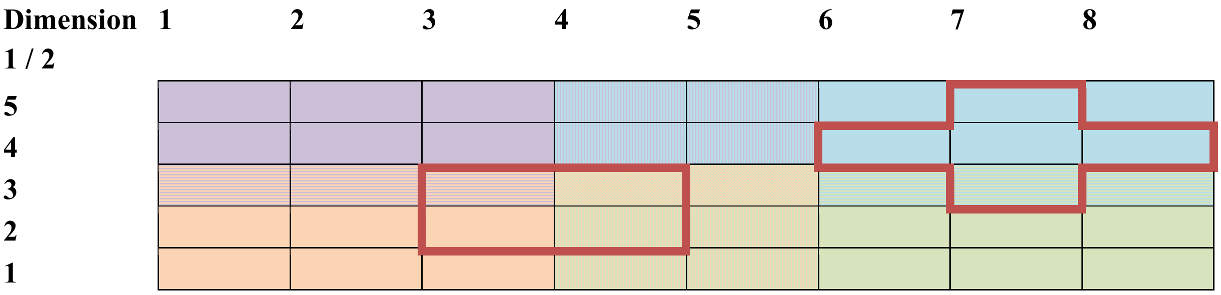


Figure 3. A two-dimensional SciDB array with chunking (indicated by the cell background color) and overlaps in both dimensions. Thanks to the overlap, both cell clusters (shown by the red line) can be processed by a single node without the exchange of data.

3.4. Architecture

The architectural concept of SciDB is built on a shared-nothing cluster of up to thousands of nodes on commodity hardware. A single runtime supervisor is used to dispatch queries and to coordinate the execution among the nodes' local executors and storage managers (see Figure 4).

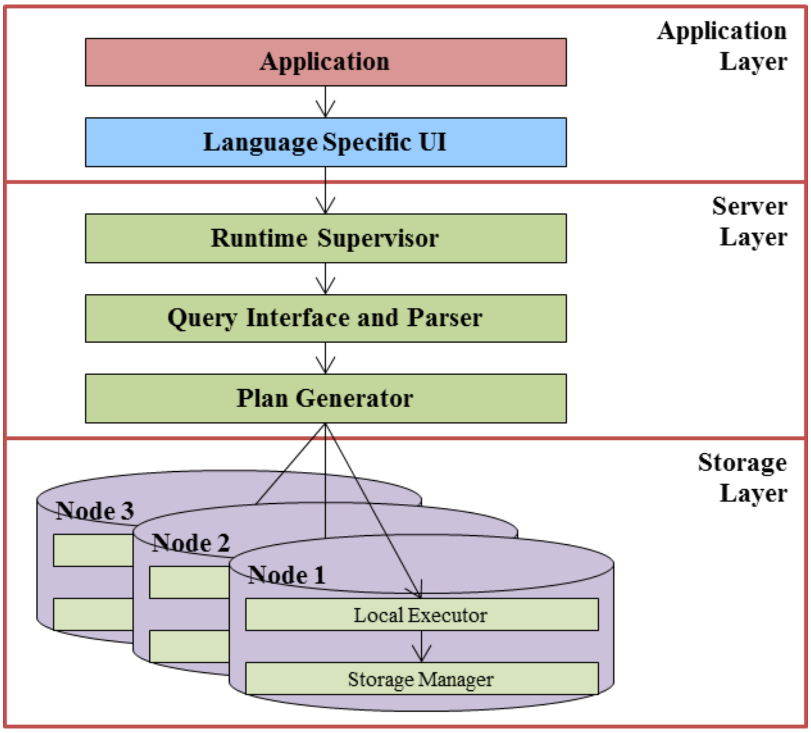


Figure 4. SciDB implements a shared-nothing architecture.

The array query language defined in SciDB refers to complete arrays agnostic to whether they are distributed or not. The SciDB plan generator optimizes queries for efficient data access and processing, taking into account the physical distribution of data among the nodes. The generated query plan is then run by a node's local executor, using its storage manager to access the data. The SciDB runtime supervisor coordinates the execution of the query.

4. Query Language and Operators

SciDB supports queries using either Array Query Language (AQL), a declarative SQL-like language extended for working with array data, or Array Functional Language (AFL), a functional language for internal implementation of operators.

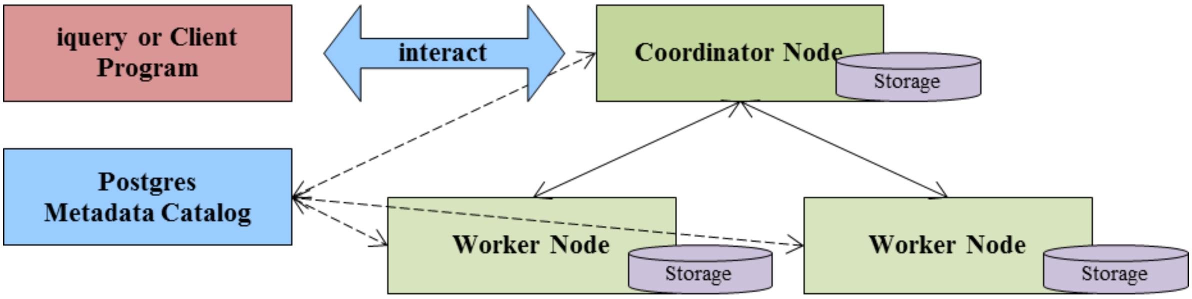


Figure 5. SciDB query plan execution

Queries can be executed using the SciDB command line interface iquery as:

```
iquery -p <port> "<queryString>"
```

SciDB also provides a Python API to execute queries from within programs:

```
# Import scidb interface
import scidbapi as scidb
# Connect to the scidb database at host:port
db = scidb.connect(<host>, <port>)
# Execute a query string using AFL
result = db.executeQuery(<queryString>, 'afl')
```

SciDB provides a number of array-specific operators (e.g., Aggregate, Apply, Filter, Join, Regrid, Subsample), and linear algebraic and matrix operations (e.g., clustering, covariance, linear and logistic regression) and the language is extensible with Postgres-style user-defined functions. Interfaces to other packages (Matlab, R, …) will be provided in a future release.

4.1. Example Query in AQL

The following AQL query creates a new one-dimensional array called AttributeList. Each cell will store tuples with RunNunber, EventNumber, DetFlags, IsSimulation and MissingET members, which are of uint32, bool or float data type. The only index of the array is called TAGID; the query specifies a chunk size of 1000 elements with an overlap of 10:

```
CREATE ARRAY AttributeList<RunNumber: uint32,
    EventNumber: uint32,
    DetFlags: uint32,
    IsSimulation: bool,
    MissingET: float> [TAGID=0:99999, 1000, 10]
```

The second example creates a two-dimensional Jet array using TAGID and JETID as indices, without chunking for JETID:

```
CREATE ARRAY Jet<EmFrac: float,
    Eta: float,
    Phi: float,
    Pt: float> [TAGID=0:99999, 10, 0, JETID=0:5, 6, 0]"
```

4.2. Array Functional Language (AFL)

The following AFL query filters out entries in the AttributeList array for which the EventNumber is greater than 12345:

```
filter (AttributeList, EventNumber > 12345)
```

The filter operator returns an array of the same dimension as the input array AttributeList, marking all cells in the input that do not satisfy the predicate expression as “empty.”

5. SciDB and Proton Collider Experiments?

Array-oriented operations with support for overlapping chunks, a major focus of initial SciDB development, is of inherent interest to a number of scientific disciplines, and in certain domains such features could be key. In proton collider experiments, though, such features are not determining factors in evaluating the suitability of an event data management product. The underlying parallel execution architecture, however, and a design explicitly intended for scalability to multi-petabyte stores, as well explicit support for sparse data, may be of genuine interest. A design with a view of transactions that is aware of the write-once nature of most scientific data is also relevant to efficient large-scale collider physics data handling. SciDB has plans in the near future to support *in situ* data, i.e., to allow access to and querying of data without translating and importing it into a SciDB-specific storage format, so that in principle bulk event data could be in a format known and proven within the community.

Initially, SciDB plans for *in situ* data support include HDF5 and possibly NetCDF plug-ins, but there is no reason that, say, a ROOT storage backend could not be accommodated. Since most collider physics experiments use ROOT as their principal event store technology for all but RAW data, such native support, with SciDB contributing a value-added layer, could be of great interest. Support for user-defined types and operators could be quite useful, as scientific data types and operations are often at best awkward to support in relational databases. An infrastructure for provenance tracking and management, though primitive in initial SciDB planning, is an important advantage. Support for uncertainty (e.g., error bars) in data is interesting in principle, though experimental particle physics has necessarily elaborate and sophisticated strategies for error estimation and bounds that may make such built-in features of academic interest at best.

SciDB's shared-nothing architecture on commodity clusters is definitely interesting by comparison with the purchase of special-purpose hardware for certain commercial database products. There are other non-database or alternative-database technologies and even relational databases that also take this approach, such as MySQL Cluster, which provides shared-nothing clustering capabilities for the MySQL database management system. There are also several products that provide array extensions to SQL—and while this is a worthwhile idea for some data, it may not be immediately apparent how this capability helps in the storage and management of bulk collider physics event data.

While event data stores may constitute the largest data stores in experimental particle physics, they are not the only ones, and it is also appropriate to consider whether technologies such as SciDB might be suitable for detector control system and conditions information (and such investigations are indeed underway), or for important subsets of an event data store, such as an event-level metadata system. The latter provides the principal setting for our investigations of SciDB.

6. SciDB for high energy physics data

The scope of the evaluation undertaken in the project described in this paper has been:

1. To install and configure first real release of SciDB, R11.06;
2. To provide machinery to upload data from multiple source types, including ROOT files;
3. To exercise the user interface and operators;
4. To explore tradeoffs in data organization;
5. To evaluate sparse versus dense data representations of identical data;
6. To provide user-defined types and functions in support of scientific data types.

6.1. Event-level Metadata from Proton Collisions at LHC

ATLAS records key quantities describing properties of proton collision events as event-level metadata called TAGs, which are useful in event selection. TAG data include, for example, event identification information, triggers passed, and particle properties for variable numbers of jets, photons, electrons, muons, and taus, along with sufficient information to allow navigation to file-resident raw and derived event data. Variable-length array data members are not a natural fit to relational databases: while they can be supported they often have large performance penalties.

TAG data from 2011 ATLAS runs were chosen for early tests because the data structure is relatively simple, and event-level metadata records are already hosted both in files and in relational databases. The TAG format is almost identical to highly derived physics data used for analysis, so that results for testing SciDB on TAG data are applicable to analysis data as well.

6.2. Cosmological Simulations

Data associated with cosmological simulations can challenge the capabilities of the largest leadership computing facilities. Still, these data often have a very simple structure, for example describing N-body problems with 8 floating point variables for each particle (position, velocity, mass, id):

> *x[Mpc], v_x[km/s], y[Mpc], v_y[km/s], z[Mpc], v_z[km/s], particle mass[M_sol], particle tag*

Important derived data products are often no more complex. Dark matter halo catalogs, for example, store halos as:

> *halo_mass[10^10 M_sol], x[Mpc], y[Mpc], z[Mpc], vx[km/s], vy[km/s], vz[km/s]*

Published reference datasets used in comparisons of cosmological simulations were uploaded in our tests. These simulations used 256^3 particles. Because of their structure, these data sets are natural candidates for exploiting three-vector constructs for position and velocity.

7. User-defined types and operators

A commonly-encountered drawback to the use of relational databases for scientific data is the lack of support for scientific data types and operators. The native array support in SciDB is an important step toward addressing this deficiency, but an important requirement for scientific data handling is the ability to support user-defined types and operators. Event selection in proton collider experiments, for example, requires identifying events with oppositely-charged leptons above an energy threshold, and this is a combinatorial operation on the variable number of muons and electrons produced in any given collision event. Spatial operations—for example, finding particles or jets whose trajectories have a particular geometric relationship (say, within a certain cone)—are another common requirement.

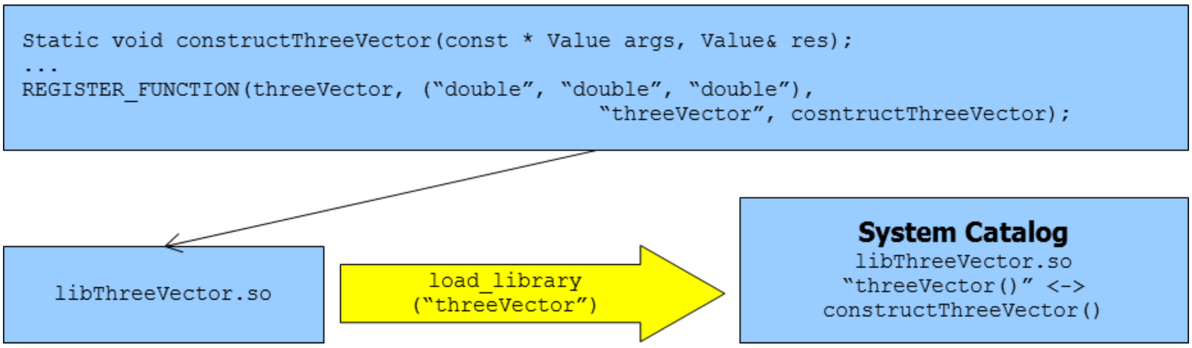
To explore the current capabilities of SciDB to support user-defined types and operators, several types and operators were defined and tested, including a user-defined three-vector type, a Lorentz vector, rotation operators, Lorentz transformations, and a number of related geometric operators.

The following example conveys the flavor of these constructions and these studies:

```
apply(AttributeList, Jet, threeVector(JetPt1, JetEta1, JetPhi1))
sort(AttributeList, angleBetween(Jet1, threeVector(1,1,1)))
```

The first line is a Jet constructor, building a Jet as a three-vector constructed from three separate floating point values in each AttributeList entry (transverse momentum plus the attributes to construct a directional vector {eta, phi}. The second line sorts the elements of AttributeList according to the angle between a contained Jet and a fixed vector (artificially chosen here as (1,1,1)).

As usual when introducing a new type into a system, the tedious work is not defining the data members (a struct of three doubles or three floats, say); rather it is in defining all the necessary supporting operators (constructors from values and from strings, assignment operators, input and output operators, and so on), and in registering the type so that it is recognized and its appropriate associated operators can be invoked by the system into which it is being introduced. The following figures illustrate how this happens in SciDB. The essential complexity of the task is no different than that of introducing a corresponding user-defined type in the underlying C or C++ language.



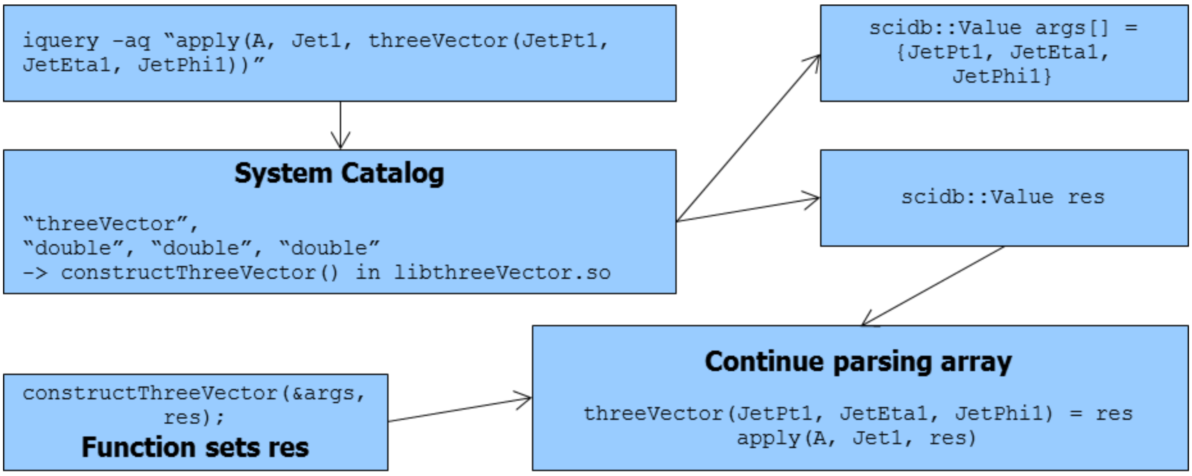
```
Static void constructThreeVector(const * Value args, Value& res);
...
REGISTER_FUNCTION(threeVector, ("double", "double", "double"),
                                   "threeVector", cosntructThreeVector);
```

```
libThreeVector.so
```

```
load_library
("threeVector")
```

```
libThreeVector.so
"threeVector()" <->
constructThreeVector()
```

Figure 6. Machinery in support of user-defined types: declaration and construction



```
iquery -aq "apply(A, Jet1, threeVector(JetPt1,
JetEta1, JetPhi1))"
```

```
"threeVector",
"double", "double", "double"
-> constructThreeVector() in libthreeVector.so
```

```
scidb::Value args[] =
{JetPt1, JetEta1,
JetPhi1}
```

```
scidb::Value res
```

```
threeVector(JetPt1, JetEta1, JetPhi1) = res
apply(A, Jet1, res)
```

```
constructThreeVector(&args,
res);
```

Figure 7. Machinery in support of user-defined types: operator invocation

8. Conclusion and Outlook

SciDB has advanced considerably in functionality and in robustness since its earliest incarnations. Much of the early development in SciDB has been aimed at supporting array data models, both in storage and in operations, and most testable features are of this flavor. For many sciences, this focus is entirely appropriate: the product is designed to be well suited to image data, with LSST as a major target client. Support for array data structures, for array partitioning (chunking) across processing nodes, for overlap specification, and for a range of array operations, all have solid implementations. The advantages of SciDB for high energy physics data that do not have as natural an array organization as spatial data are less pronounced, and there are an increasing number of alternative data hosting technologies that also support efficient shared-nothing processing. For proton collision data and for cosmological simulation data, SciDB's support for user-defined data types and operators are of genuine interest, and significantly decrease the complexity of query formulation by physicists. Increasingly,other requirements of large-scale scientific data handling are being addressed, including improved support for user-defined types and operators, for *in situ* data, for provenance tracking, and more, making SciDB a technology worth tracking for the high energy physics computing community.

Acknowledgments

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute.

9. References

[1] root.cern.ch

[2] www-conf.slac.stanford.edu/xldb/

[3] The SciDB Development team, "Overview of SciDB, Large Scale Array Storage, Processing and Analysis," SIGMOD'10 Conference, 2010.

[4] Borne K, Becla J, Davidson I, Szalay A, and Tyson J, 2008, "The LSST Data Mining Research Agenda," AIP Conf. Proc. 1082, pp 347-351.

[5] www.scidb.org/about/history.php

[6] Ronstrom M and Thalmann L, "MySQL cluster architecture overview, High availability features of MySQL cluster", MySQL Technical White Paper, 2004.

[7] O. Brüning et al., "LHC Design Report, v.1-3", CERN-2004-003-V-1 to CERN-2004-003-V-1-3.

[8] ATLAS Collaboration, "ATLAS Detector and Physics Performance Technical Design Report," CERN-LHCC-1999-14 and CERN-LHCC-1999-15.

[9] Malon D, 2005, "What your next experiment's data will look like: Event stores in the Large Hadron Collider era," Int. J. Mod. Phys. A20 pp 3871-3873.

[10] Van Gemmeren P and Malon D, 2009, "The event data store and I/O framework for the ATLAS experiment at the Large Hadron Collider," IEEE Int. Conf. on Cluster Computing and Workshops, p. 1.

[11] Malon D, Van Gemmeren P, Cranshaw J, and Schaffer A, 2006, "Sailing the petabyte sea: Navigation infrastructure in the ATLAS event store," Computing in High Energy and Nuclear Physics, Mumbai, India.

[12] Malon D, Cranshaw J, and Karr K, 2006, "A flexible, distributed event-level metadata system for ATLAS," Computing in High Energy and Nuclear Physics, Mumbai, India.

[13] Cranshaw J, Goosens L, Malon D, McGlone H, and Viegas F, 2008, "Building a scalable event-level metadata service for ATLAS," J. Phys. Conf. Ser. 119 072012.

[14] Cranshaw J, Cuhadar-Donszelmann T, Gallas E, Hrivnac J, Kenyon M, McGlone H, Malon D, Mambelli M, Nowak M, Viegas F, Vinek E, and Zhang Q, 2010, "Event selection services in s

[15] Van Gemmeren P and Malon D, 2010, "Event metadata records as a testbed for scalable data mining", J. Phys. Conf. Ser. 219 042057.