SLAC-TN-11-020

Using SciDB to Support Photon Science Data Analysis

Jacek Becla (becla@slac.stanford.edu) Daniel L. Wang (danielw@slac.stanford.edu)
Kian-Tat Lim (ktl@slac.stanford.edu)

October 31, 2011

1 Introduction

Array data analytic systems like SciDB hold great potential to accelerate processing data from SLAC's Linac Coherent Light Source and other experiments. SciDB is unique in its ability to integrate storage and processing of array data efficiently, providing both space-efficient storage and out-of-memory efficient parallel array processing. We describe a recent effort to leverage SciDB to store and process LCLS data. The work includes development of software to import data into SciDB, subsequent benchmarks, and interactive manipulation of data in SciDB.

1.1 Advantages and disadvantages of Integrating with SciDB

The main advantages of a SciDB-based system for large-scale scientific data are the normalization of access and analytics of array data and a scalable parallel engine for analytics. There are relatively few systems for serving array-based data. One such system is OPeNDAP (which provides subsetting, slabbing, metadata, and aggregation functionality), but most data centers serve arrays as formatted files through HTTP or FTP, sometimes with a web-based frontend for staging data. SciDB provides data access along with array manipulation functions that provide mathematical processing in addition to aggregation and subsetting. SciDB manages data distribution among a cluster of machines, providing the scalability to serve large datasets that do not fit on single machines, and distributed processing, so data sets may be processed in parallel on multiple nodes. The scalability features, in particular, are not found in other systems. Matlab, a proprietary mathematical software package, provides an extensive environment for performing mathematical manipulation, but does not address large scales of data, although a parallel implementation is available to speed up expensive computational tasks.

The main disadvantages of using SciDB for big science are concerned with the relative immaturity and young age of SciDB. Only a couple SciDB releases have been made, and no large projects have committed to adopting it, although several have been experimenting and evaluating it. Thus it is a tool unfamiliar to scientists, with a user community in its infancy. Its mathematical and scientific processing library is basic and limited, owing to the system's youth and the lack of a development community to implement more functions. Matlab, for example, has had over 20 years as a commercial product to accumulate an extensive array/matrix manipulation library, much of

Prepared for the Department of Energy under contract number DE-AC02-76SF0015

which is user-written. Other disadvantages are the lack of a method of organizing arrays (e.g., in a hierarchy) and managing array metadata.

In general, SciDB has considerable potential. Its data access capabilities are adequate and its scalability is not readily found in any other system. These capabilities are generally re-implemented with custom code in large science projects. Its scalability is extremely valuable. No packages exist for parallel mathematical processing at large scale, so users are forced to implement their own distributed, parallel software, which is widely recognized by software professionals as difficult, or integrate with one of the few parallel mathematical libraries, which is somewhat easier though still non-trivial. The use of SciDB to perform parallel data manipulation should significantly lower the barrier for large-scale data science.

1.2 Comparison to existing practices

SciDB's user languages (AFL[1] and AQL[2]) are languages specialized for array processing and array querying. In photon science, analytics are typically done using either a general-purpose programming language like C++, Python, or Java, or, after a data-import, a mathematics package such as Matlab or R. Using a general-purpose language allows advantages such as the ability to directly implement advanced algorithms, but carries disadvantages such as worrying about memory management and other programming concepts. Thus scientists write a small amount of general-purpose code to extract the interesting data and load them into Matlab or R, where they can express their analytics with more ease. Unfortunately, such mathematical packages are known to have limited scalability in data size, so scientist must carefully control how much data are processed at a time.

SciDB's approach has the distinct advantage of providing processing capability at large data scales. It storage and computational capacity scales as large as the largest data sets (now, and in the future). Though mathematical functionality is currently limited, a compellingly extensive library should arise as scientists adopt it for use on their own data.

2 Integration work

2.1 Data loading tools

We have implemented plugins for SciDB that allow loading of arrays from HDF5 (and FITS[3]) files. The code is publicly available at: `https://github.com/wangd/SciDB-HDF5`.

The HDF5 loader has been tested to successfully load arrays from HDF5 files from SLAC's LCLS detectors (nested: 1D array of 2D array; 1D array of multiple variables). We have tested loading large image arrays (an array of 42,247 16-bit 1024×1024 images) and simple compound arrays (an array of 42,247 records of 4 32-bit fields). The FITS loader has been tested to load image arrays from astronomical image simulation FITS files. Despite involving less than a week's effort, FITS development contributed measurably to improving the structure and robustness of the overall loader plugins.

The plugin interface was designed to be simple and to minimize scientist effort. The plugin can be loaded with a SciDB AFL query: `load_library('loadhdf');` using SciDB's `iquery` tool. Once the plugin is loaded, arrays from HDF5 files can be loaded using a single AFL function: `loadhdf`.

```
loadhdf(<target array name>, <HDF5 file>,
        <path to HDF5 array>);
```

[1] Array Functional Language, a lower-level language with function-syntax

[2] Array Query Language, a high-level declarative language

[3] FITS support was developed to test and to validate the more general portions of the HDF5 loading code.

The <target array name> is the name of the SciDB array that will be created (or overwritten) and loaded. <HDF5 file> is the path to the HDF5 file on the server's filesystem, and <path to HDF5 array> is the path to the particular array with in the HDF5 file (which typically contains a hierarchically-organized and named set of arrays). As an example, consider:

```
loadhdf('image', '/home/danielw/sxrcom10-r0232.h5',
        '/Run0/Inst2/Camera1/Dev2/image');}
```

In this example, "image" is the name of the target array, "/home/danielw/sxrcom10-r0232.h5" the path to the HDF5 file on the filesystem, and "/Run0/Inst2/Camera1/Dev2/image" the path of the desired array within the HDF5 file.

After this completes, we can inspect the array using standard SciDB commands:

```
AFL% show(image);
    [("image<int1:uint16 NOT NULL>
          [unnamed0=0:1023,1024,0,
           unnamed1=0:1023,1024,0,
           unnamed2=0:92233720368547758 07,1,0]")]
AFL% count(image);
    [(44299190272)]
```

We note that while the third dimension is large, its value indicates the maximum bound in that dimension, rather than the array's non-empty size.

The loader appears to be efficient and I/O (rather than CPU) limited. On a dual AMD Opteron 270 machine, the `top` (table of processes) program generally indicated 10% or less CPU spent in loading (source HDF5 file on one SATA disk and SciDB data storage on another SATA disk), and the SciDB process spent most of its time in `iowait`. For the 1024x1024x42247 array of int16 in our 29GB HDF5 file (the array consumes ≈95% of the file), the loader spent 69 minutes and 18.946 seconds, loading into a SciDB storage that was roughly 180GB afterwards. This averages to about 5MB/s read bandwidth and 46MB/s write bandwidth.

The FITS loader can be used with similar syntax as the HDF5 loader. Further details are in the `README` file in the source code.

2.2 Array manipulation using SciDB

We describe some basic commands to manipulate arrays in SciDB's AFL. The following examples assume an image array named "image3" with overall dimensions of 1024×1024×3, where each image is 1024×1024 in size with 3 recorded time steps.

2.2.1 Simple averaging

Average Compute an average over the whole array.

```
AFL% avg('image3');
    [(33.816)]
```

Average by time step Compute a time-series average.

```
AFL% avg('image3', int1, unnamed2);
    [(33.8467),(33.8863),(33.7149)]
```

Here, "int1" is the desired "attribute" in the array cell and "unnamed2" is the desired dimension to retain after averaging. The attribute specification is required when the dimension specification is used.

Transpose Transpose an array, reversing its dimensions.

```
AFL% transpose(image3);
```

Transposing affects an array in-place. Users should explicitly create a copy of the array before transposing if the original should be retained. Transposing may be helpful to re-order dimensional layout and improve performance for certain array-slicing operations.

Other operations In addition to "avg", SciDB also supports "min," "max," "stdev," and "sum" operations, which obey a similar syntax. More operations can be found in SciDB's AFL documentation.

2.3 Science-specific plugins

We began implementation of a plugin for array convolution but have not yet completed it at this time. Our LCLS collaborators noted that a histogram function would be useful, but we have not implemented it since it is promised for a future version of SciDB[4]. While SciDB is young, we have observed its ability for parallel data manipulation[5], and the SciDB plugin interface should be easier than other methods (like MPI) of achieving parallel analytics across cores and machines. SciDB's own parallel-capable implementations of functions and operators are good examples of what is capable in the plugin interface.

2.4 Science tasks

Due to the shortage of documented LCLS data analytics, it is difficult to illustrate the equivalent commands in SciDB. Since LCLS data is unique and only available in the last couple of years, scientists have not publicly documented any processing tasks for its data. Late in our work, we have collected a basic diagnostic process, but it contains primarily rotation and simple visualization processing, and its commonality to other users is unclear.

Since projects are disincentivized to share knowledge since they compete for research funding, and LCLS has no peers in the types of data it can collect, there has not been a push to standardize its interfaces and processing techniques. We speculate that even basic techniques are already yielding good results. As similar light sources are built, we expect increased collaboration, standardization, and documentation which should clarify where SciDB (or other array engines) could be used and their distinct advantages.

3 Visualization

SciDB does not yet have any built-in visualization tools, and visualization tools have not yet built support for a SciDB back-end. However, the plain text output from the `iquery` client is easily imported and plotted in Python. Figure 1 contains a plot of image statistics using the code in Section 7.

[4]Paradigm4 Inc. is planning on releasing a commercial closed-source plugin implementing a suite of mathematical functions.

[5]Under certain conditions in a standalone SciDB installation. We have not tested multi-node operation.

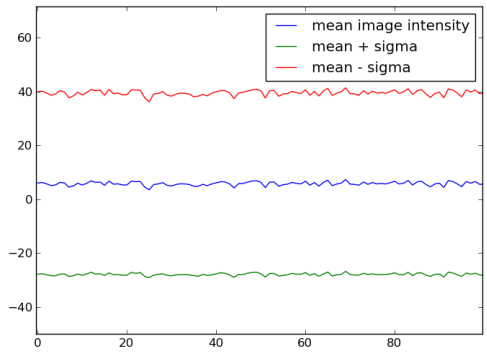

Figure 1: Plot of mean and standard deviation using sample script

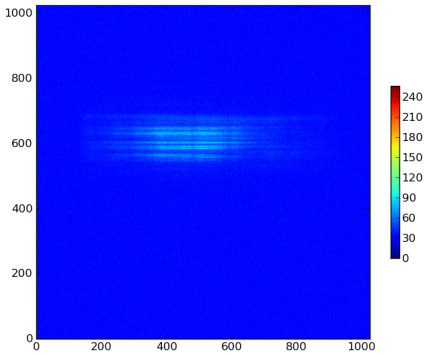

Figure 2: Image at t=0

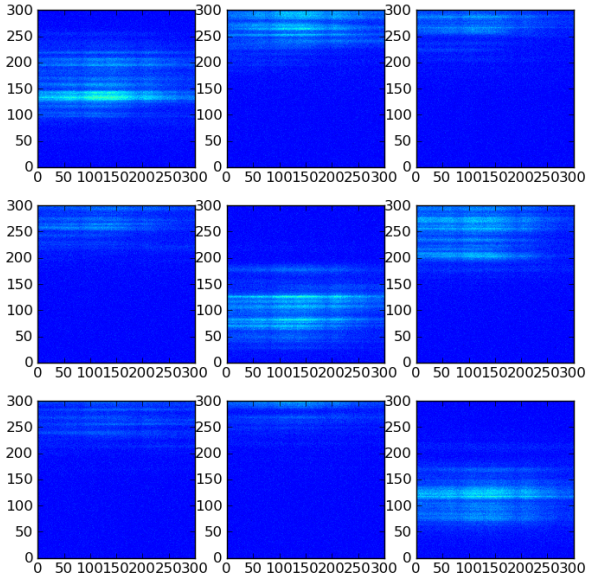

Figure 3: Image at t=0:8

Image plots (Figures 2 and 3) and can also be constructed.

4 Benchmarks

To get an idea of SciDB's performance, we have plotted execution times for queries computing simple standard deviations on an image array arrays sliced in various ways. The plots were done with two different dimension orderings, in order to quantify the effect of different storage layouts. Two dimension orderings were used, and the standard deviation for different slicings (similar to SQL "GROUP BY" permutations) were computed.

Figure 4 plots the query execution times when the dimensions are ordered as they are after loading

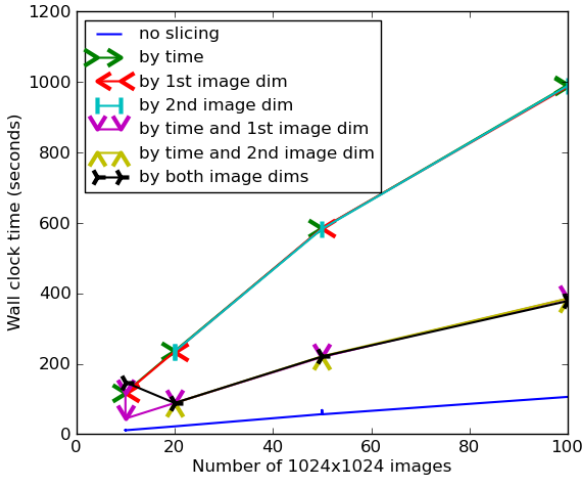

Figure 4: Image array performance

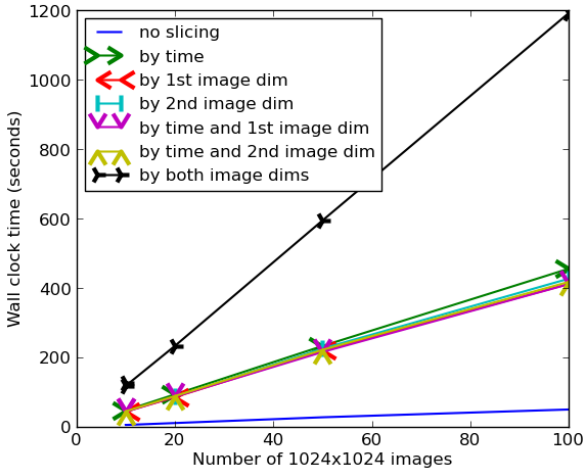

Figure 5: Transposed image array performance

directly from HDF5–1024×1024×44247. Figure 5 plots the same queries against the transposed array. The general form of the test query is:

AFL% stdev(subarray(image,0,0,0,1023,1023,9), int1, unnamed1);"

A subsetting operation is performed to limit the number of images, and the standard deviation is computed for a particular attribute ("int1", the only SciDB attribute in this case), sliced to retain the dimension "unnamed1", which is the second array dimension (of unnamed0, unnamed1, unnamed2: loader-generated dimension names since HDF5 does not explicitly name its array dimensions).

From the two figures, it is clear that all forms of slicing carry a significant penalty. Slicing in the latter dimensions is more costly. Comparing the performance when slicing by time in both figures, we can see how slow it is in the original array layout (where it is the last dimension), and how much faster it is for the transposed array (where it is the first dimension). Both plots show a linear scaling in execution time with respect to the data size, regardless of slicing parameters, although the original array displays some gradual leveling off. We have not tested for array sizes large enough to speculate on the degree of leveling off (which would indicate greater processing efficiency for larger array sizes).

This benchmarking was designed to characterize SciDB's performance at computing statistics over arrays. Although we have only tested `stdev` in detail, we expect that the other aggregation operators (`avg, min, max, stdev,` and `sum`) will have similar performance. Simple operations like retrieving hyperslabs via the `subarray` operator seemed efficient and did not appear to consume significant CPU time in our experience.

SciDB is still under heavy development and we expect its performance to change significantly within the next year.

5 Related work

This work has sparked interest in using SciDB to process data in HDF5 files. There are two related efforts, both aimed at implementing in-situ functionality so that SciDB can process external non-SciDB array data without an intermediate loading step. Nicholas Malitsky is leading an effort for HDF5 in-situ and Miguel Branco (École Polytechnique Fédérale de Lausanne), FITS.

Malitsky aims to use SciDB as a storage engine in the Experimental Physics and Industrial Control System (EPICS). An in-situ HDF5 backend to SciDB would allow a more seamless integration of SciDB's analytics with existing HDF5 data for querying and manipulation within EPICS.

6 Acknowledgements

We would like to thank our LCLS collaborators: Marc Messerschmidt, Andrei A. Salnikov, Mikhail S. Dubrovin, Igor Gaponenko, and Amedeo Perazzo for their valuable help in guiding and evaluating this work. We would also like to thank Alex Poliakov of Paradigm4 and Roman Simakov at the Russian Academy of Sciences for their technical help in developing the loader code.

7 Appendix: Python script for visualizing

The following sample script was used to produce a simple plot of average and standard deviation for the first 100 images.

```
#!/usr/bin/env python
#
#
# Purpose: make some plots from data in SciDB loaded from LCLS HDF5 data
#
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as cls

from subprocess import Popen, PIPE

def doQuery(query):
    iqueryBin = './iquery'
    proc =  Popen([iqueryBin, "-aq", query], stdout=PIPE)
    return eval(proc.communicate()[0])

def readStats(arrName):
    q1 = "stdev(%s, int1, unnamed0);" % arrName
    q2 = "avg(%s, int1, unnamed0);" % arrName

    stdev = doQuery(q1)
    avg = doQuery(q2)
    return [stdev, avg]

def makeFigure() :
    fig  = plt.figure(figsize=(8,8), dpi=80,
                      facecolor='w',edgecolor='w',frameon=True)
    return fig

def plotStdev():
    [avg,stdev] = readStats("subarray(image,0,0,0,100,1023,1023)")
    p = makeFigure()
```

```
    a1 = np.array(avg)
    s1 = np.array(stdev)
    plt.plot(a1, label="mean image intensity")
    plt.plot(a1 - s1, label="mean + sigma")
    plt.plot(a1 + s1, label="mean - sigma")
    plt.legend()

plotStdev()
plt.show()
```