# DYNAMIC DISTRIBUTED DIMENSIONAL DATA MODEL (D4M) DATABASE AND COMPUTATION SYSTEM

*Jeremy Kepner, William Arcand, William Bergeron, Nadya Bliss, Robert Bond, Chansup Byun, Gary Condon, Kenneth Gregson, Matthew Hubbell, Jonathan Kurz, Andrew McCabe, Peter Michaleas, Andrew Prout, Albert Reuther, Antonio Rosa, Charles Yee*

MIT Lincoln Laboratory[1], Lexington, MA

## ABSTRACT

A crucial element of large web companies is their ability to collect and analyze massive amounts of data. Tuple store databases are a key enabling technology employed by many of these companies (e.g., Google Big Table and Amazon Dynamo). Tuple stores are highly scalable and run on commodity clusters, but lack interfaces to support efficient development of mathematically based analytics. D4M (Dynamic Distributed Dimensional Data Model) has been developed to provide a mathematically rich interface to tuple stores (and structured query language "SQL" databases). D4M allows linear algebra to be readily applied to databases. Using D4M, it is possible to create composable analytics with significantly less effort than using traditional approaches. This work describes the D4M technology and its application and performance.

***Index Terms***— associative array, database, tuple store, linear algebra, fuzzy algebra

## 1. INTRODUCTION

Modern database analysis in the areas of healthcare, internet search, finance, and network security are outgrowing the capabilities of current technologies. The increasing size of the data (doubling every year), the increasing diversity of the data (e.g., web pages, documents, audio, images, and video), and the increasing complexity of the operations (e.g., ingestion, scanning, link analysis, and importance scoring) all present tremendous challenges.

The standard approach for handling the increasing size of data is to increase the storage capacity at the cost of increasing the time it takes to access any particular data item. Solving this problem requires a transparent mechanism (e.g., distributed arrays) for adding computation and network bandwidth as storage capacity is increased.

The standard approach for handling the increasing diversity of data is to increase the number of databases/tables at the cost of increasing the effort required to make the data available to users. Solving this problem requires a general mechanism (e.g., tuple stores) for adding a wide variety of data into a single database table.

The standard approach for handling the increasing complexity of operations is to increase the size of the functions at the cost of increasing the effort required to build these functions. Solving this problem requires a composable mechanism (e.g., multi-dimensional associative arrays) for creating operations of increasing complexity without increasing their relative effort.

A final challenge is that the above problems are anti-correlated. Addressing one problem will typically result in the other issue becoming more difficult. For example, adding computing and network resources to a database system increases the difficulty of adding more diverse data and adding more complex operations. Thus, a viable solution must address all of the issues simultaneously.

The goal of the Dynamic Distributed Dimensional Data Model (D4M) is to combine the advantages of distributed arrays, tuple stores, and multi-dimensional associative arrays to create a database and computation system that solves the challenges associated with increasing data size, data diversity and operation complexity. Our prototype implementation of D4M has demonstrated simultaneous improvement in all of these dimensions when compared to current standard approaches (e.g., Java + SQL).

## 2. D4M ARCHITECTURE

D4M addresses the challenges presented in the previous section by using a layered software architecture that addresses each challenge in its own layer (Fig. 1). The top layer consists of composable associative arrays that provide a one-to-one correspondence between database queries and linear algebra. Associative arrays can be both the input and output of a wide range of database operations and allow complex operations to be composed with a small number of statements. The middle layer consists of several parallel computation technologies (e.g., pMatlab [1,2], MatlabMPI [3], and gridMatlab [4]) that allow associative arrays to be

[1] This work is sponsored by the Department of the Air Force under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

distributed efficiently across a parallel computer. Furthermore, the pieces of the associative array can be bound to specific parts of one more databases to optimize the performance of data insertion and query across a parallel database system. The bottom layer consists of databases (e.g., HBase [5] and Accumulo [6]) running on parallel computation hardware. D4M can use any type of database. D4M can fully exploit the power of databases that use an internal sparse tuple representation (e.g., a row/col/val triple store) to store all data regardless of type. The D4M approach provides several advantages and improvements over existing methods, specifically: D4M Represents complex database operations and queries as composable algebraic operations on associative arrays; D4M Provides distributed arrays for parallel database operations; D4M Transparently handles diverse data types using a tuple store.

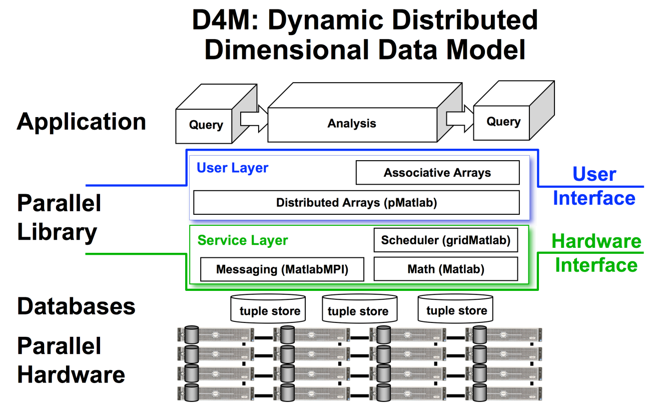

**Fig. 1**. D4M Matlab prototype architecture. At the top is the user application consisting of a series of query and analysis steps. In the middle is the parallel library that hides the parallel mapping of the operations. On the bottom are the databases (typically tuple stores) running on parallel computing hardware.

## 3. ASSOCIATIVE ARRAYS

Associations between multidimensional entities (tuples) using number/string keys and number/string values can be stored in data structures called associative arrays. For example, in two dimensions an associative array entry might be

```
A('alice ','bob ') = 'cited '
```

or

```
A('alice ','bob ') = 47.0
```

The above tuples have a one-to-one correspondence with the triple store representations

```
('alice ','bob ','cited ')
```

and

```
('alice ','bob ',47.0)
```

Constructing complex composable query operations can be expressed using simple array indexing of the associative array keys and values, which themselves return associative arrays. For example

| | |
|---|---|
| `A('alice ',:)` | alice row |
| `A('alice bob ',:)` | alice and bob rows |
| `A('al* ',:)` | rows beginning with al |
| `A('alice : bob ',:)` | rows alice to bob |
| `A(1:2,:)` | first two rows |
| `A == 47.0` | all entries equal to 47.0 |

The composability of associative arrays stems from the ability to define fundamental mathematical operations whose results are also associative arrays. Given two associative arrays A and B, the results of all the following operations will also be associative arrays

```
A + B    A – B    A & B    A|B    A*B
```

Associative array composability can be further grounded in the mathematical closure of semirings (i.e., linear algebraic "like" operations) on multi-dimensional functions of infinite strict totally ordered sets (i.e., sorted strings). In addition, many of the linear algebraic properties of fuzzy algebra can also be directly applied: linear independence [7], strong regularity [8], and uniqueness [9].

Finally, associative arrays can represent complex relationships in either a sparse matrix or a graph form (Fig. 2). Associative arrays are a natural data structure for performing both matrix and graph algorithms. Such algorithms are the foundation of many complex database operations across a wide range of fields [10]. Measurements using the D4M prototype indicate that these algorithms can be implemented with significantly less coding effort when compared standard approaches [11].

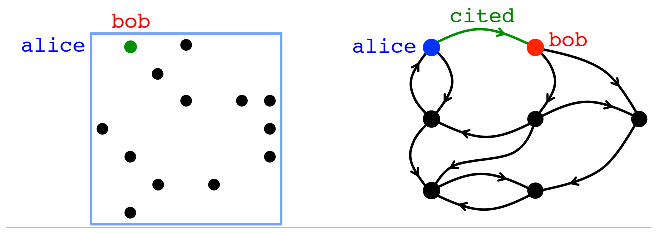

**Fig. 2**. A sparse associative 2D array and its graph dual.

## 4. DISTRIBUTED ARRAYS

Distributed arrays provide a simple mechanism for writing efficient parallel programs [1]. When an algorithm is run in parallel, the same algorithm (or code) is run by every instance of the program (on one or more processors) with a unique identity number $P_{ID} = 0 \ldots N_P\text{-}1$. This execution model is referred to as the Single-Program Multiple-Data (SPMD) computation model. In distributed array programming, it is necessary to map the tuples of an array onto a set of $P_{ID}$s. This mapping process allows each $P_{ID}$ to know which tuples of a distributed it "owns." In pMatlab this process is supported by adding a "map" object to the construction of an array. A map contains information on the method used to assign tuples to $P_{ID}$s. Thus it is possible to formally separate the execution of a program from how it is mapped onto a parallel processor. The primary benefits of this approach are simplicity (making a serial program a parallel program can be done with just a few lines of code) and performance (distributing arrays across processors allows each processor to easily work on data that is in its own memory).

## 5. TUPLE STORES

Web pages, documents, audio, images, and video all produce very different kinds of data. Traditional databases require different tables to handle these data. Tuple stores handle all of this data by treating them all as key/value pairs. This greatly simplifies the design of the database and allows for significant performance improvements.

For example, consider a traditional database table where each row represents the keywords in the document. Column names of this table might be “keyword1”, “keyword2”, … To find a row with a particular keyword entry requires a complete scan of the table or the construction of an index of all the entries. In a row/col/val triple store each row represents a document and the column keys can be the actual keywords themselves.

The Hadoop [12] distributed file system (HadoopDFS) is an open source distributed file system modeled on the Google file system [13]. Hadoop is a replicated block based distributed filesystem optimized for handling very large blocks and is well suited for managing large files that are larger than a typical storage device. HBase [5], Accumulo, and other “Big Table Like” databases leverage the HadoopDFS by modeling Google Big Table [14]. These databases are designed for data mining applications that do little read-modify-write and where statistical consistency is more than adequate. Under these the relaxed restrictions there is potential to get a sizable increase in performance. These databases are typically “NoSQL” databases that have their own custom interfaces.

D4M associative arrays provide a one-to-one mapping onto the tables in a tuple store that makes complex manipulations simple to code. Storing both the table and its logical transpose in the database allows for all rows and columns to be searched efficiently without the need to build specialized indexes. D4M associative arrays can make both the insertion and retrieval of data from transpose pairs transparent to the user.

## 6. TECHNOLOGY COMPARISON

D4M provides a database and computation system that combines composable associative arrays, distributed arrays, and tuple stores in an integrated manner. Table 1 compares various other technologies having aspects of D4M. Note that while there are many distributed array technologies, none – except D4M – implement an associative array. Likewise, D4M is the first implementation of multi-dimensional numeric associative arrays and the first implementation of composable associative arrays. Finally, D4M is the only associative array technology that can take advantage of the features of a tuple store.

**Table 4.** Technology Comparison. D4M uniquely supports composable multi-dimensional associative arrays on parallel computers and tuple store databases. Key: SQL (System Query Language), MPI (Message Passing Interface), HPF (High Performance Fortran), UPC (Universal Parallel C), VSIPL++ (Vector, Signal and Image Processing Library).

| Feature | Perl | SQL | HBase | MPI | HPF | UPC | VSIPL++ | pMatlab | **D4M** |
|---|---|---|---|---|---|---|---|---|---|
| Associative Array | | | | | | | | | |
| 1D | X | X | | | | | | | X |
| 2D | X | X | | | | | | | X |
| String key/value | X | X | | | | | | | X |
| Numeric key/value | | X | | | | | | | X |
| Composable query | | | | | | | | | X |
| Composable compute | | | | | | | | | X |
| Tuple Store | | | X | | | | | | X |
| Parallel Client | | | X | X | X | X | X | X | X |
| Distributed array | | | | | X | X | X | X | X |

## 7. APPLICATION EXAMPLE

To illustrate the use of D4M consider a facet search on the document keyword table A shown in Fig. 3. In this context facet search selects the subset of documents containing a set of keywords and then computes the histogram of all the keywords in this document subset. Facet search is particular useful in helping a user build searches by providing guidance as to the most popular keywords as their search narrows. Facet search is a highly dynamic query because it is not possible to compute the histograms of all sets of keywords in advance.

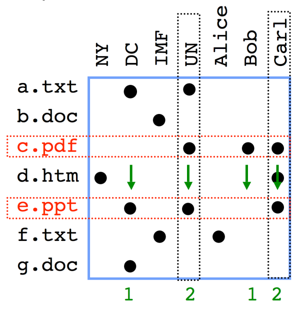

**Fig. 3**. Facet Search in D4M. Table A stores a list of documents and their keywords. Selecting keywords `UN` and `Carl` indicate that the documents `c.pdf` and `e.ppt` contain both. Selecting the documents `c.pdf` and `e.ppt` and summing the occurrences of their keywords retrieves the facets `DC` and `Bob`.

Facet search in D4M begins with choosing keywords in the table

```
x = 'UN '       and       y = 'Carl '
```

Next, all documents that contain both of those keywords are found

```
B = (sum(A(:,[x y]),2) == 2)
```

Finally, the distribution of keywords in that set of documents is computed

```
F = transpose(B) * A(row(B),:)
```

This complex query can be performed efficiently in just two lines of D4M code that perform two database queries (one column query and one row query). If the underlying table is a transpose table pair, then both of these queries are be performed efficiently in a manner that is completely transparent to the user. Implementing a similar query in Java and SQL takes hundreds of lines of code.

## 8. PERFORMANCE

Graph500 [15] is derived from the Graph Analysis benchmark [16] and is reflective of a new class of graph based computations. For large problems Graph500 becomes a database benchmark. Specifically, the initialization phase measures the insert rate into a database. The performance of D4M is measured by implementing the Graph Analysis benchmark. Figs. 4 and 5 shows the insert performance of D4M and D4M+Accumulo. These results are consistent with the performance of the native Java interface and deliver near the theoretical performance limits of the hardware.

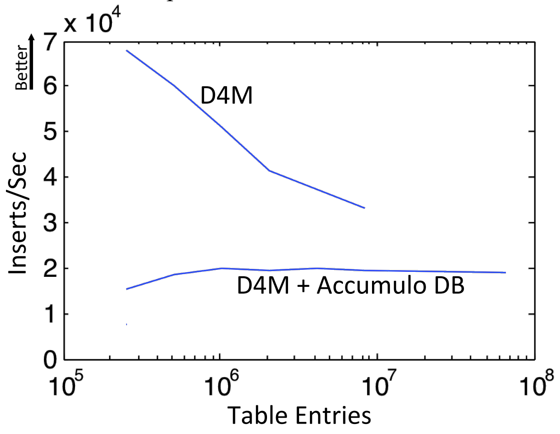

**Fig. 4**. Single process D4M and D4M+AccumuloDB insert rates (top) versus the number of table entries. Curves are shown for standalone D4M (in memory) and D4M connected to the Accumulo database (in storage).

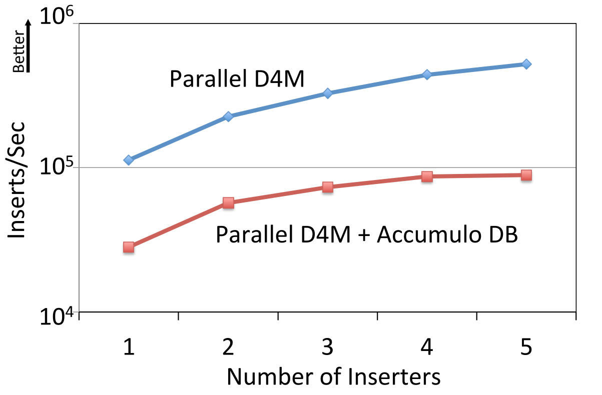

**Fig. 5**. Parallel D4M and D4M+AccumuloDB insert rates (top) versus number of insert processes on a single node system. Curves are shown for standalone D4M (in memory) and D4M connected to the Accumulo database (in storage).

## 9. CONCLUSIONS

Tuple store databases are a key enabling technology for larges scale data analysis. Tuple stores are highly scalable and run on commodity clusters, but lack interfaces to support efficient development of mathematically based analytics. D4M has been developed to provide a mathematically rich interface to tuple stores (and structured query language "SQL" databases). Using D4M, it is possible to create composable analytics with significantly less effort than using traditional approaches.

## 10. REFERENCES

[1] J. Kepner, *Parallel Matlab for Multicore and Multinode Computers*, SIAM Press, Philadelphia, 2009.

[2] N. Bliss and J. Kepner, "pMatlab Parallel Matlab Library," *International Journal of High Performance Computing Applications: Special Issue on High Level Programming Languages and Models*, J. Kepner and H. Zima (editors), Winter 2006 (November).

[3] J. Kepner and S. Ahalt, "MatlabMPI," *Journal of Parallel and Distributed Computing*, vol. 64, issue 8, August, 2004

[4] N. Bliss, R. Bond, H. Kim, A. Reuther, and J. Kepner, "Interactive Grid Computing at Lincoln Laboratory," *Lincoln Laboratory Journal*, vol. 16, no. 1, 2006.

[5] Apache HBase http://hbase.apache.org/

[6] Apache Accumulo http://incubator.apache.org/accumulo/

[7] J. Plavka, "Linear Independences in Bottleneck Algebra and their Coherences with Matroids," *Acta Math. Univ. Comenianae*, vol. LXIV, no. 2, pp. 265–271, 1995.

[8] P. Butkovic, "Strong Regularity of Matrices - a Survey of Results," *Discrete Applied Mathematics*, vol. 48, pp. 45-68, 1994.

[9] M. Gavalec and J. Plavka, "Simple Image Set of Linear Mappings in a Max–Min Algebra," *Discrete Applied Mathematics*, vol. 155, pp. 611 – 622, 2007.

[10] J. Kepner and J. Gilbert (editors), *Graph Algorithms in the Language of Linear Algebra*, SIAM Press, Philadelphia, 2011.

[11] B.A. Miller, N. Arcolano, M.S. Beard, N.T. Bliss, J. Kepner, M.C. Schmidt, and P.J. Wolfe, "A Scalable Signal Processing Architecture for Massive Graph Analysis," submitted.

[12] HadoopDFS http://hadoop.apache.org/hdfs/

[13] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *26th IEEE Symposium on Mass Storage Systems and Technologies*, 3-7 May, 2010.

[14] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A Distributed Storage System for Structured Data," *ACM Transactions on Computer Systems*, Volume 26 Issue 2, June 2008.

[15] Graph500 http://www.graph500.org

[16] D. Bader, K. Madduri, J. Gilbert, V. Shah, J.y Kepner, T. Meuse, and A. Krishnamurthy, "Designing Scalable Synthetic Compact Applications for Benchmarking High Productivity Computing Systems," CT Watch, Vol 2, Number 4A, November, 2006.