

# Taming Biological Big Data with D4M

Jeremy Kepner, Darrell O. Ricke, and Dylan Hutchison

The supercomputing community has taken up the challenge of “taming the beast” spawned by the massive amount of data available in the bioinformatics domain: How can these data be exploited faster and better? MIT Lincoln Laboratory computer scientists demonstrated how a new Laboratory-developed technology, the Dynamic Distributed Dimensional Data Model (D4M), can be used to accelerate DNA sequence comparison, a core operation in bioinformatics.



**The growth of large, unstructured data-**sets is driving the development of new technologies for finding items of interest in these data. Because of the tremendous expansion of data from DNA sequencing, bioinformatics has become an active area of research in the supercomputing community [1, 2]. The Dynamic Distributed Dimensional Data Model (D4M) developed at Lincoln Laboratory, and available at <http://www.mit.edu/~kepner/D4M>, has been used to accelerate DNA sequence comparison, which is a fundamental operation in bioinformatics.

D4M is an innovation in computer programming that combines the advantages of five processing technologies: triple-store databases, associative arrays, distributed arrays, sparse linear algebra, and fuzzy algebra. Triple-store databases are a key enabling technology for handling massive amounts of data and are used by many large Internet companies (e.g., Google Big Table). Triple stores are highly scalable and run on commodity computers, but lack interfaces to support rapid development of the mathematical algorithms used by signal processing experts. D4M provides a parallel linear algebraic interface to triple stores. Using D4M, developers can create composable analytics with significantly less effort than if they used traditional approaches. The central mathematical concept of D4M is the associative array that combines spreadsheets, triple stores, and sparse linear algebra. Associative arrays are group theoretic constructs that use fuzzy algebra to extend linear algebra to words and strings.

### Computing Challenges in Bioinformatics

In 2003, the cost of sequencing the first human genome was \$3 billion. The cost for sequencing has declined steadily since then and is projected to drop to \$1000 by 2013 [3]. The dramatic decrease in the cost of obtaining genetic sequences (Figure 1) has resulted in an explosion of data with a range of applications:

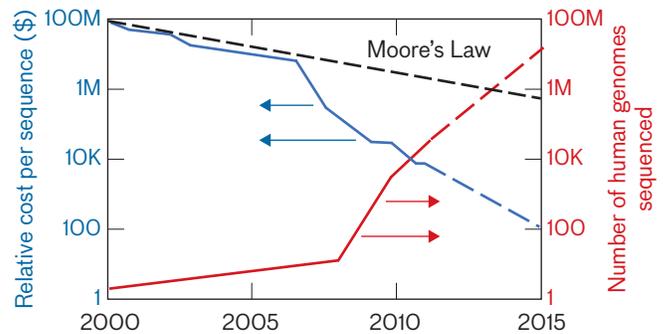
- Early detection and identification of bioweapons
- Early detection and identification of epidemics
- Identification of suspects from genetic samples taken from bomb components
- Determination of extended family kinship from reference and forensic DNA samples
- Prediction of externally visible characteristics from genetic samples

The computational challenges of these applications require Big Data solutions, such as Apache's Hadoop software, for research, development, and testing of algorithms on large sets of data. For example, the Proton II sequencer used by Lincoln Laboratory's biomedical engineers can generate up to 600 GB of data per day. In addition, energy-efficient solutions must be designed and tested for field deployment; for example, the handheld Oxford Nanopore sequencer can be connected to a laptop.

The computational requirements for bioinformatics rely on a variety of string matching, graph analysis, and database technologies. The ability to test and improve existing and new algorithms is limited by the computational requirements necessary to ingest and store the required data.

Consider the specific use case of identifying a specific bioweapon after victims experience the onset of symptoms. The current state of the art is well illustrated by the *E. coli* outbreak that took place in Germany in the summer of 2011 (Figure 2a). In this instance, while genetic sequencing played a role in confirming the ultimate source of the outbreak, it arrived too late to significantly impact the number of deaths.

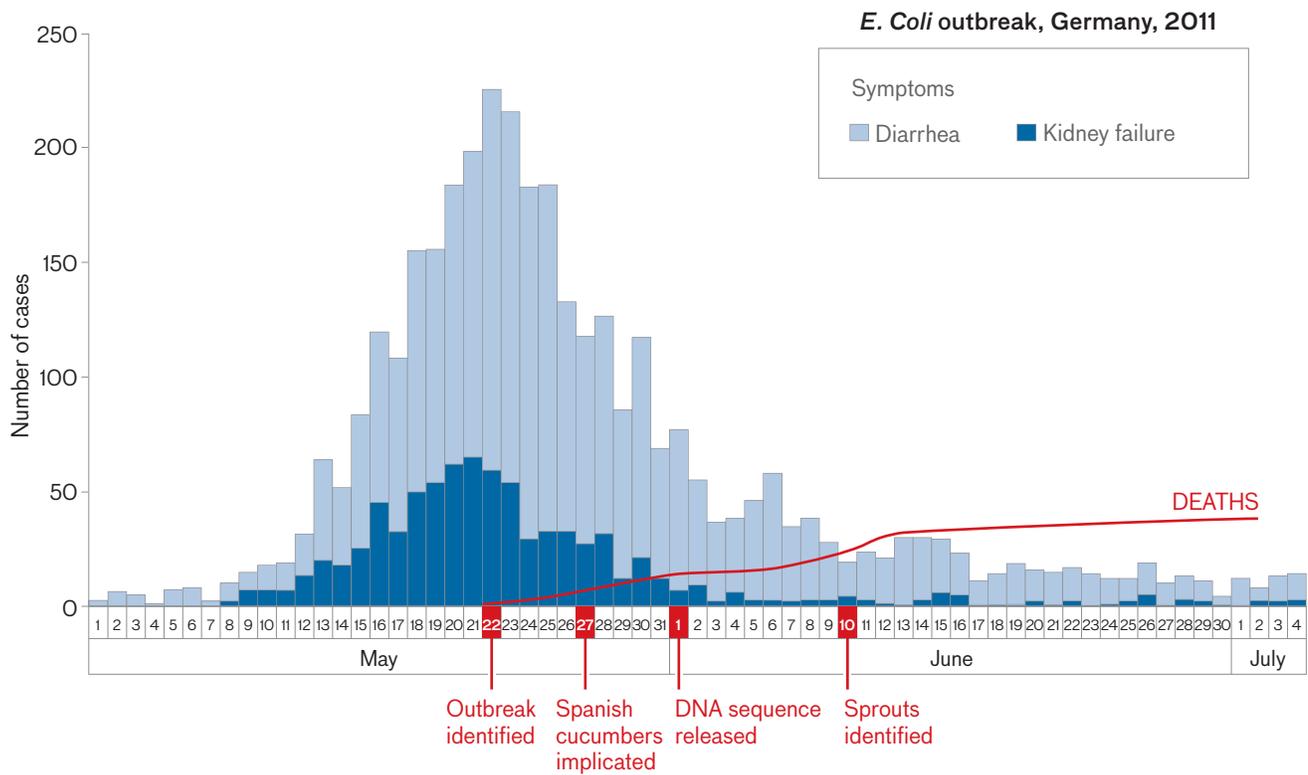
Figure 2b shows that there are a number of places in the "Current" time frames for the sequencing process where improved algorithms and computation could have a significant impact. Ultimately, with the appropriate combination of technologies, it should be possible to shorten the timeline for identification from weeks to less than one day. A shortened timeline could significantly reduce the number of deaths



**FIGURE 1.** Advances in DNA sequencing technologies are rapidly decreasing the costs of DNA sequencing a whole human genome (blue line). As a result, the number of humans being sequenced is increasing significantly (red line). The amount of data being produced is rapidly outpacing computing technology (black line). Together, these drive the need for more efficient techniques for extracting information from DNA sequences.

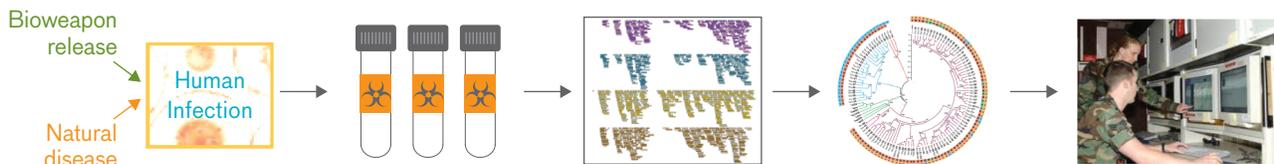
from such an outbreak. Accelerated sequence analysis would also support accelerated bioforensic analysis for attribution of the source (e.g., the specific lab in which the strain originated). Thus, improving DNA sequencing analysis capabilities has multiple significant roles to play in enhancing the full spectrum response to a bioweapon attack. With the rapid progress being made in DNA sequencing, operational use of these technologies is now becoming feasible. Real-world samples (e.g., environmental and clinical samples) contain complex mixtures of DNA from the host or environment, the pathogen(s) of interest, and symbiotic bacteria and viruses, thereby leading to a jumble of genomic data. Complete analysis of these samples requires the application of modern genomics techniques to the study of communities of microbial organisms directly in their natural environments, bypassing the need for isolation and lab cultivation of individual species (i.e., metagenomics) [5].

With rapidly expanding sequencing capacity, the greatest barrier to effective use is efficient and accurate algorithms to (1) sort through the extremely complex DNA sequence information present in real-world environmental and clinical samples, (2) reassemble constituent genomes, and (3) identify pathogens and characterize them for genes of interest for antibiotic resistance and toxins. Accelerating the testing process with high-performance databases [6] and parallel computing software technologies [7, 8] will directly impact the ability to field these systems.



(a)

Processing pipeline and time frames



| 1. Complex background | 2. Sample collection and shipment | 3. Sample preparation and sequencing | 4. Analysis of sample | 5. Time to actionable data |
|-----------------------|-----------------------------------|--------------------------------------|-----------------------|----------------------------|
| Current:              | 2–30 days                         | 1–30 days                            | 7–14 days             | 10–45 days                 |
| Goal:                 | Onsite                            | 3 hours                              | <12 hours             | <1 day                     |

(b)

**FIGURE 2.** Example disease outbreak (a) and processing pipeline (b). In the May to July 2011 virulent *E. coli* outbreak in Germany, the identification of the *E. coli* source was too late to have substantial impact on illnesses [9]. Improved computing and algorithms can play a significant role in reducing the current time of 10–45 days to less than 1 day.

**System Architecture**

Sequence algorithm development is often done in high-level programming environments (e.g., Matlab, Python, and Java) that interface to large databases. Lincoln Laboratory researchers, who developed and extended a parallel supercomputing system called LLGrid [10], have exten-

sive experience with high-level environments and have developed novel technologies that allow an algorithm analyst to write parallel database programs in these environments [6, 7, 11, 12]. Sequence algorithm research often requires many copies of the data to be constructed (in parallel) in many formats. A variety of additional statistics

are also computed (in parallel) on each copy of the data. Together, these formats and statistics allow the analyst to intelligently select the algorithms to explore on the basis of a particular mission goal.

In the testing phase, an analyst is interactively modifying a selected algorithm by processing all the data (in parallel) while exploring a range of parameters. The analyst repeats these tests until an optimal set of parameters is selected for the particular mission goal. Statistics are computed to demonstrate the effectiveness and limits of the selected algorithm. After testing is completed, the algorithm must be tested on a smaller, energy-efficient computing system for deployment.

Both the research and testing phases require rapid data retrieval. The optimal data storage approach depends upon the nature of the data and the analysis. Parallel file systems (e.g., Lustre), distributed file systems (e.g., Hadoop), and distributed databases (e.g., HBase and Accumulo) are all used. Parallel file systems stripe large files across many storage devices and are ideal for delivering high bandwidth to all the processors in a system (i.e., sending the data to the compute nodes). Distributed file systems place files on the local storage of compute nodes and are ideal for processing many different files with the same algorithm (i.e., sending the computation to the data nodes). Distributed databases index the data contents and are ideal for rapid retrieval of specific pieces of data.

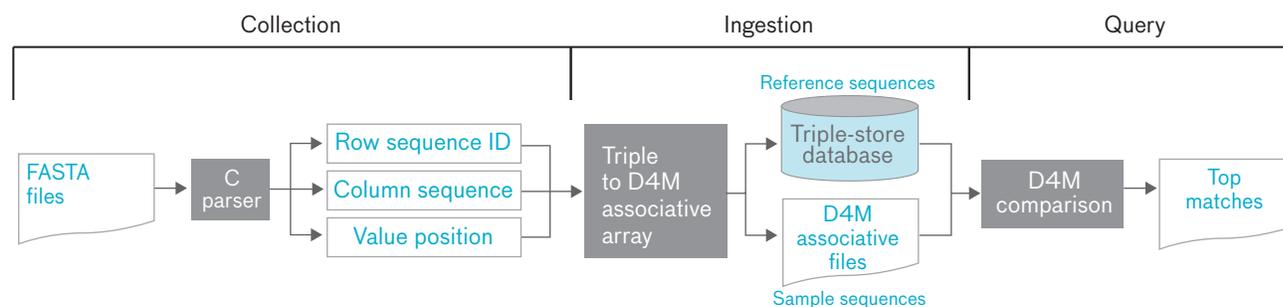
Most sequence algorithms are broken down into a parse collection, database ingest, and query-compare pipeline (Figure 3). Each stage presents different technical challenges and requires different engineering approaches.

### Pipeline

The *collection step* receives data from a sequencer (or other data source) and parses the data into a format suitable for further analysis. This process begins with attaching metadata to the sequence data so that each collected sequence can be uniquely identified. In addition, many analysis techniques rely on constructing hash words (ormers) of the sequences. A typical hash word length can be 10 DNA bases (a 10-mer), 20 DNA bases (a 20-mer), or even 50 DNA bases (a 50-mer).

The *ingestion step* takes data from the collection step and puts them into a database to allow detailed queries based on the specific contents of the data. Recently developed triple-store databases (e.g., Apache HBase and Apache Accumulo) are designed to sit on top of the Hadoop distributed file system. Hadoop is based on the Google Big Table architecture and is the foundation of much of the commercial Big Data ecosystem. These databases have been demonstrated on thousands of nodes. Typical ingest rates for these databases are 100,000 entries per second per node [6].

The *query step* works on a combination of the outputs of the collect and ingest steps. In this step, the data are combined so as to find the sequence matches required by the specific mission. A complete analysis requires one query for each DNA base coming out of the sequencer. Triple stores are designed to support both high ingest and high retrieval rate, and can sustain 100,000 entries per second per node. Using 100 compute nodes to retrieve and analyze the data from the sequencer will allow the proposed system to meet this requirement.



**FIGURE 3.** Processing pipeline using the Dynamic Distributed Dimensional Data Model (D4M) and triple-store database technologies. Sequence data are input as FASTA files (a format commonly used in bioinformatics for representing sequences) and parsed into row, column, and values triples. The reference-data triples are converted to D4M associative arrays and inserted into a triple-store database. The sample data are stored as D4M associative arrays and saved to files. A D4M program then reads each sample file, selects the relevant triples from the database, and computes the top matches.

## Computational Approach

Sequence alignment compares two strings to determine how well they match. Sequence alignment is the core operation in bioinformatics. BLAST (for Basic Local Alignment Search Tool) [13] is the standard software package for sequence alignment and consists of 850,000 lines of mostly C++ code. The runtime of BLAST is a function of the product of the two sequences being compared. For example, the single-core execution time of comparing a 20 Megabase (a length of DNA fragments equal to 1 million nucleotides) sample with a 500 Megabase reference is ~100 minutes. The dominant parallel scheme for BLAST is to break up the sequences and perform separate BLAST calculations on each subsequence. Direct use of BLAST to compare a 600 Gigabase collection with a comparably sized reference set requires months on a 10,000-core system. A key goal for bioinformatics is to develop faster approaches for rapidly identifying sequences from samples. One promising approach is to replace computations with lookups in a database.

Just as important as performance is providing environments that allow algorithm developers to be productive. BLAST is a very large, highly optimized piece of software. A typical algorithm developer or biologist needs additional tools to quickly explore data and test new algorithm concepts. Lincoln Laboratory's system includes the full LLGrid software stack that includes the GridEngine scheduler; Lustre parallel file system; Apache Hadoop, Accumulo, and HBase; Matlab, GNU Octave, MatlabMPI, pMatlab, gridMatlab, LLGridMapReduce, and the D4M graph analysis package. This software stack has proven to be effective at supporting developers on a wide range of applications [10]. Two of these packages are particularly important for bioinformatics: the D4M graph analysis package and the triple-store databases (Apache Accumulo and HBase).

## Software Interfaces

Most modern sequence-alignment algorithms use a hashing scheme in which each sequence is decomposed into short words (mers) to accelerate their computations. Mathematically, this scheme is equivalent to storing each set of sequences in a large sparse matrix where the row is the subsequence identifier and the column is the specific word (Figure 4). Sequence alignments are computed by multiplying the sparse matrices together and selecting

those combinations of sequences with the highest numbers of word matches.

The sparse matrix multiply approach is simple and intuitive to most algorithm developers. However, using sparse matrices directly has been difficult because of a lack of software tools. D4M provides these tools that enable the algorithm developer to implement a sequence-alignment algorithm on par with BLAST in just a few lines of code. D4M also provides a direct interface to high-performance triple-store databases that allows new database sequence-alignment techniques to be explored quickly.

The key concept in D4M is the *associative step* that allows the user to index sparse arrays with strings instead of indices. For example, the associative array entry

$$A(\text{AB000106.1\_1-1343}, \text{ggaatctgcc}) = 2$$

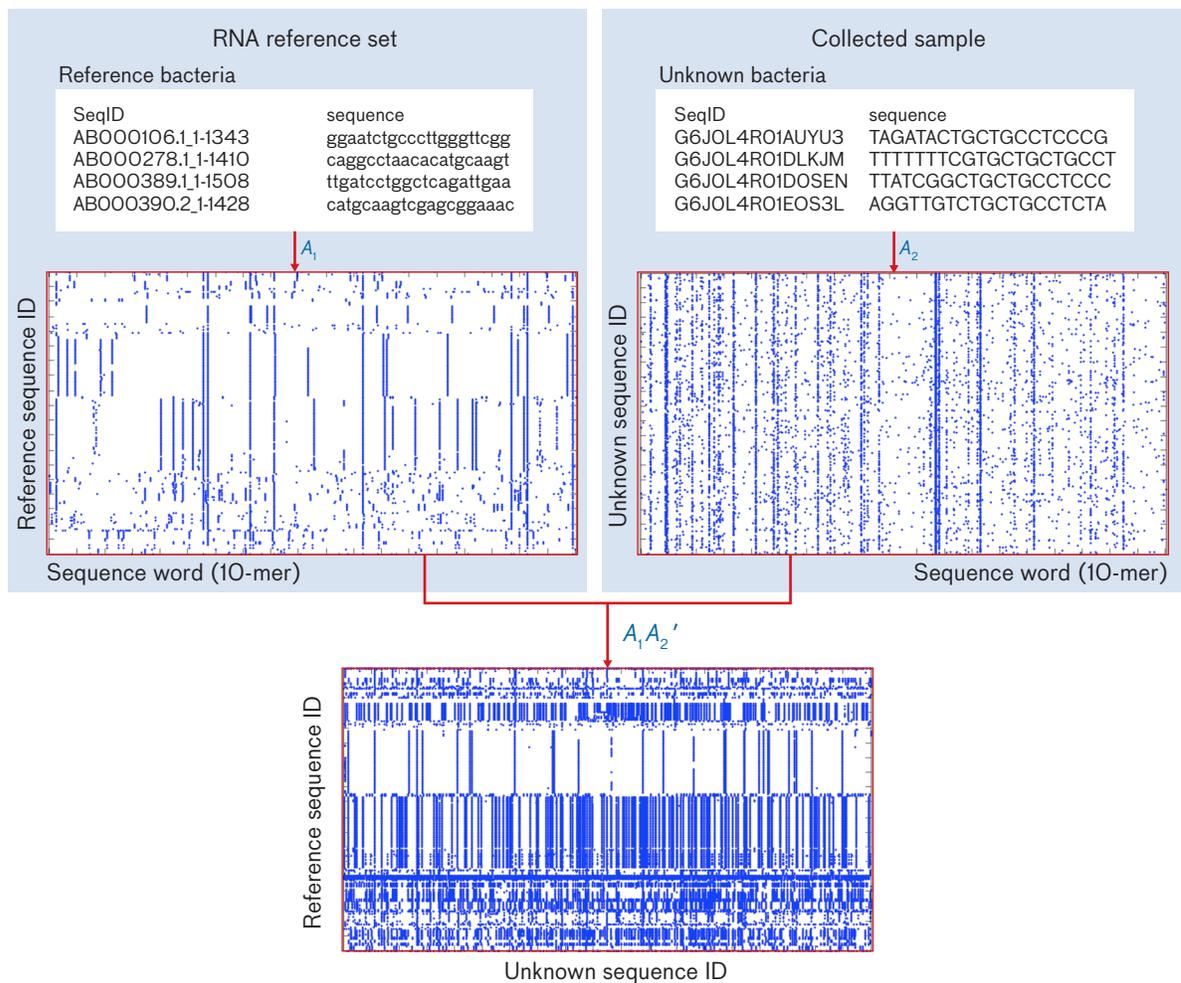
shows that the 10-mer `ggaatctgcc` appears twice in the sequence `AB000106.1\_1-1343`. D4M provides a full associative array implementation of the linear algebraic operations required to write complex algorithms [14, 8]. D4M has also been successfully applied to text analysis and cyber security applications [15, 16].

Associative arrays provide an intuitive mechanism for representing and manipulating triples of data and are a natural way to interface with the new class of high-performance NoSQL triple-store databases (e.g., Google Big Table, Apache Accumulo, Apache HBase, Netflix Cassandra, Amazon Dynamo). By using D4M, complex queries to these databases can be done with simple array indexing operations (Figure 5). For example, to select all sequences in a database table `T` that contain the 10-mer `ggaatctgcc` can be accomplished with the one-line D4M statement:

$$A = T(:, \text{ggaatctgcc})$$

Because the results of all queries and D4M functions are associative arrays, all D4M expressions are composable and can be directly used in linear algebraic calculations. 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:

$$A + B \quad A - B \quad A \& B \quad A|B \quad A*B$$



**FIGURE 4.** Sequence alignment via sparse matrix multiplication. DNA sequences hashed into words (10-mers) can be readily expressed as sparse matrices. The alignment of two sets of sequences can then be computed by multiplying the two matrices together.

Associative array composability can be further grounded in the mathematical closure of semirings (i.e., linear algebraic “like” operations) on multidimensional 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 [17], strong regularity [18], and uniqueness [19].

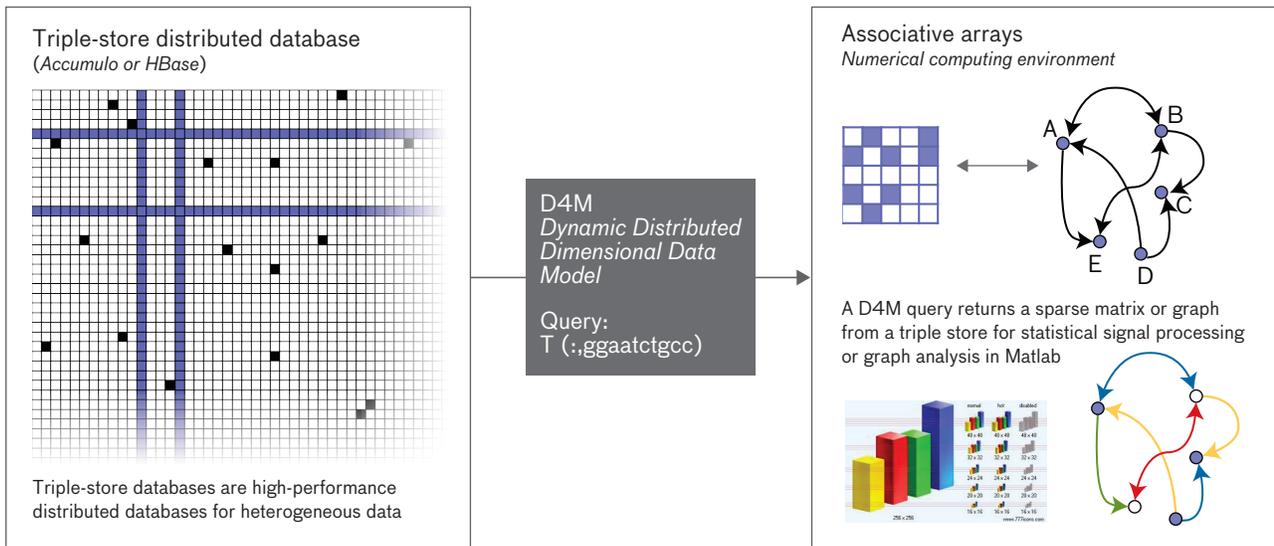
### Software Performance

Triple-store databases are a new paradigm of database designed to store enormous amounts of unstructured data. They play a key role in a number of large Internet companies (e.g., Google Big Table, Amazon Dynamo, and Netflix Cassandra). The open-source Accumulo and HBase databases (both of which use the Hadoop distrib-

uted file system) are both based on the Google Big Table design. Accumulo was developed by the National Security Agency and is widely used in the intelligence community. Accumulo has been tested and shown to scale well on very large systems. The highest published Accumulo performance numbers are from Lincoln Laboratory’s LLGrid team [6]. The LLGrid team demonstrated 650,000 inserts per second using eight dual-core nodes and 4,000,000 inserts per second using eight 24-core nodes (Figure 6).

### Algorithm Performance

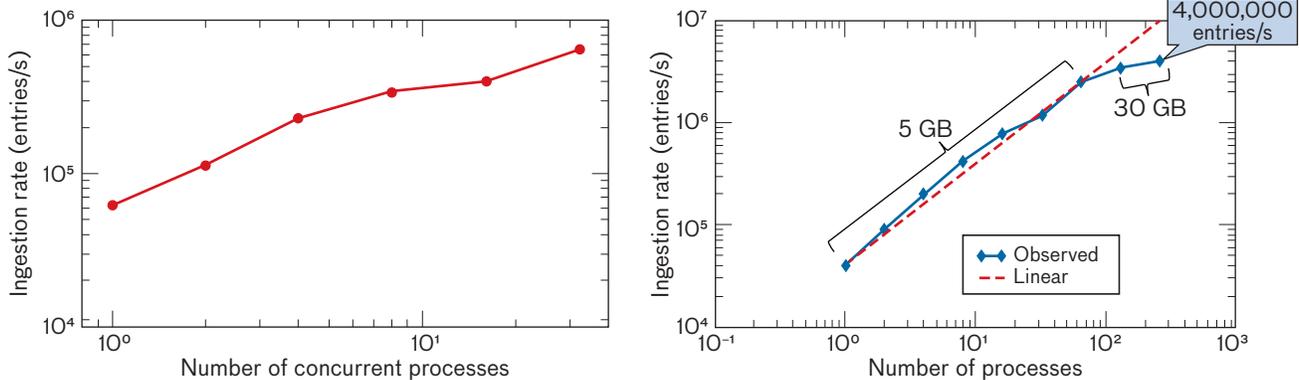
High-performance triple-store databases can accelerate DNA sequence comparison by replacing computations with lookups. The triple-store database stores the



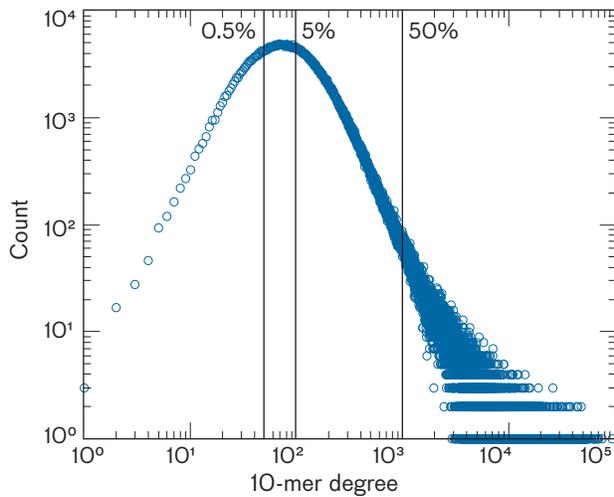
**FIGURE 5.** D4M binding to a triple store. D4M binds associative arrays to a triple store (Accumulo or HBase), enabling rapid prototyping of data-intensive Big Data analytics and visualization. D4M can work with a triple store or directly on files.

associative array representation of the sequence (see Figure 4) by creating a unique column for every possible 10-mer. A row in the database consists of the sequence identification (ID) followed by a series of column and value pairs. The storage is efficient because only the nonempty columns are kept for each row. By using this format, the database can quickly look up any sequence ID in constant time. By also storing the transpose of the associative array, it is possible to look up any 10-mer in constant time. D4M hides these details from the user so that the database appears to be able to quickly look up either rows or columns.

The Accumulo triple store used here can tally data as they are inserted. By creating row and column tallies, it is possible to compute the row and column sums as the sequences are inserted into the database. The sequence data can be viewed as a bipartite graph with edges connecting a set of sequence ID vertices with a set of 10-mer vertices. The row sums represent the number of edges coming out of each sequence ID vertex (i.e., the outdegrees). The column sums represent the number of edges going into each 10-mer vertex (i.e., the indegrees). Figure 7 shows that the vast majority of edges occur in very popular 10-mers. However, these



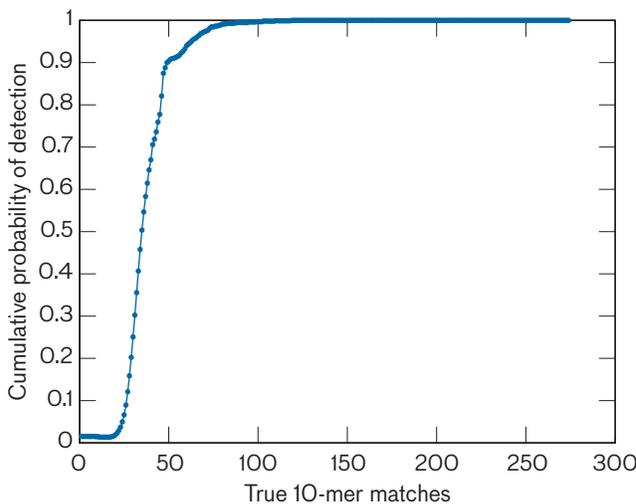
**FIGURE 6.** Triple-store insert performance [6]. Left: insert performance into a triple-store database running on eight dual-core nodes. Right: insert performance into a triple-store database running on eight 24-core nodes. Values in brackets show the total amount of data inserted during the test.



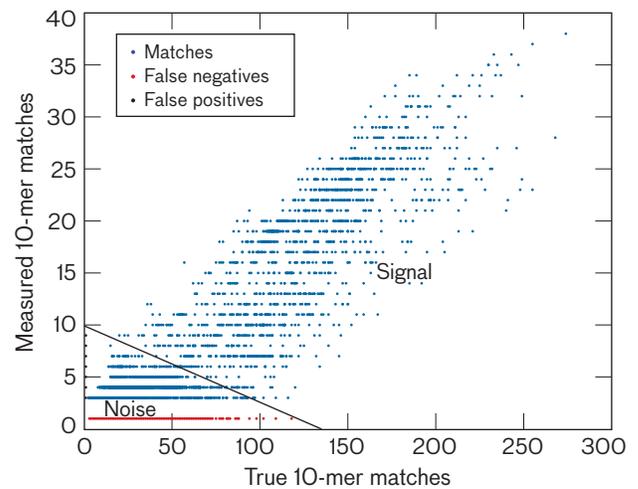
**FIGURE 7.** Reference 10-mer distribution. More than 50% of all edges are in 10-mers that appear in over 1000 sequences. Likewise 0.5% of all edges appear in 10-mers that appear in fewer than 50 sequences.

popular 10-mers have very little power to uniquely identify a sequence match.

The procedure for exploiting the distribution of 10-mers stored in the database is as follows. First, the sample sequence is loaded into a D4M associative array. All the unique 10-mers from the sample are extracted and used to query the column tally table. The least popular 10-mers from the reference are then selected from the full database table and compared with the sample. The

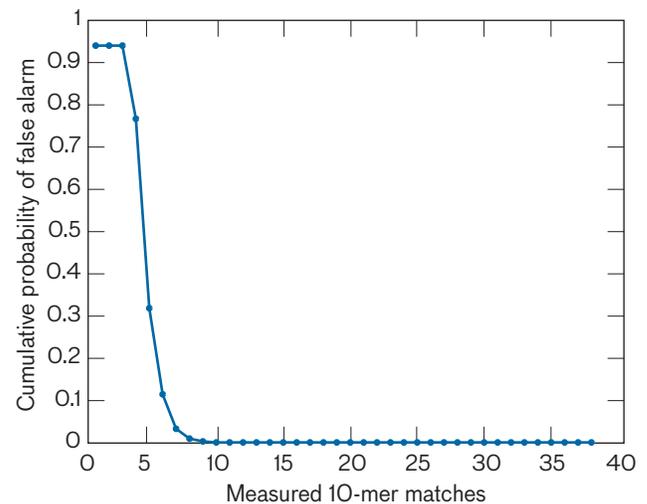


**FIGURE 9.** Cumulative probability of detection. 100% detection of all true matches >100.

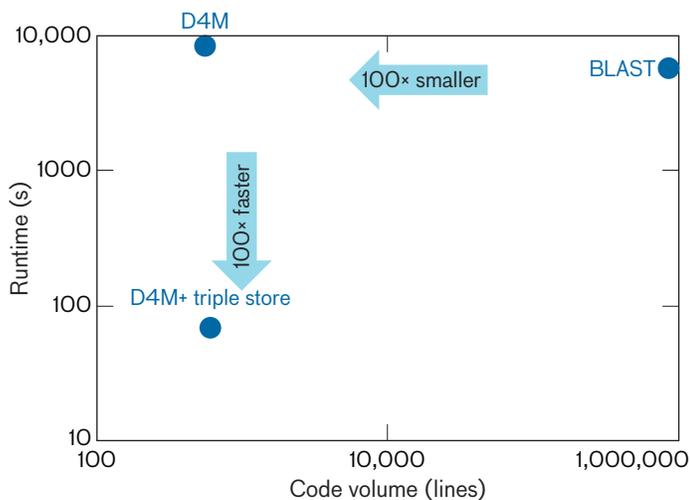


**FIGURE 8.** Selection results for subsampling 0.5% of data. The 20 MB sample data are compared with a 500 MB reference dataset. True matches are shown in blue. False negatives are shown in red and projected just above the x-axis. False positives are shown in black and projected just to the right of the y-axis. All strong matches are detected using 0.5% of data. The black line shows the approximate boundary between signal and noise.

least popular 10-mers are selected because they have the most power to uniquely identify the sequence. By subsampling the data to the least popular 10-mers, the volume of data that needs to be directly compared is significantly reduced. Figures 8, 9, and 10 compare the quality of the results produced by subsampling 0.5% of the data



**FIGURE 10.** Cumulative probability of false alarm. Measured matches >10 are always matches.



**FIGURE 11.** Sequence-alignment implementations. The D4M implementation requires 100× less code than BLAST. D4M+Triple Store reduces run time by 100× compared to BLAST.

with full 100% sampling. Figure 8 shows that all strong matches are detected. Figure 9 shows that all sequences with true matches >100 are detected. Figure 10 shows that a subsampled match of >10 is always a true match.

The benefit of using D4M in this application is that it can significantly reduce programming time and increase performance. Figure 11 shows the relative performance and software size of sequence alignment implemented using BLAST, D4M alone, and D4M with a triple store. In this specific example, D4M with a triple store provided a 100× coding and a 100× performance improvement over BLAST.

### Future Directions

D4M has the potential to significantly accelerate the key bioinformatics operation of DNA sequence comparison. Furthermore, the compactness of D4M allows an algorithm developer to quickly make changes and explore new algorithms. D4M brings linear algebra to string-based datasets and allows the techniques of signal processing to be applied to the bioinformatics domain. Combined, these capabilities should result in new techniques for rapidly identifying DNA sequences for early detection and identification of bioweapons, early detection and identification of epidemics, identification of suspects from genetic samples taken from bomb components, determination of extended family kinship from reference and forensic DNA samples, and prediction of externally visible characteristics from genetic samples.

### Acknowledgments

The authors are indebted to the following individuals for their technical contributions to this work: William Arcand, William Bergeron, David Bestor, Chansup Byun, Matthew Hubbell, Peter Michaleas, David O’Gwynn, Andrew Prout, Albert Reuther, Antonio Rosa, and Charles Yee. ■

### References

1. K. Madduri, “High-Performance Metagenomic Data Clustering and Assembly,” SIAM Annual Meeting, 2012, Minneapolis, Minn., available online at [www.graphanalysis.org/SIAM-AN12/07\\_Madduri.pdf](http://www.graphanalysis.org/SIAM-AN12/07_Madduri.pdf).
2. S Reinhardt, “Graph Analytics for Subject-Matter Experts: Balancing Standards, Simplicity, and Complexity,” SIAM Annual Meeting, 2012, Minneapolis, Minn., available online at <http://www.graphanalysis.org/workshop2012.html>.
3. R.F. Service, “A \$1000 Genome by 2013?” *Science-NOW*, July 2011, available online at [news.sciencemag.org/sciencenow/2011/07/a-1000-genome-by-2013.html](http://news.sciencemag.org/sciencenow/2011/07/a-1000-genome-by-2013.html).
4. K. Wetterstrand, DNA Sequencing Costs, Data from the National Human Genome Research Institute Large-Scale Genome Sequencing Program, available online at [www.genome.gov/sequencingcosts](http://www.genome.gov/sequencingcosts), accessed 8 March 2012.
5. K. Chen and L. Pachter, “Bioinformatics for Whole-Genome Shotgun Sequencing of Microbial Communities,” *PLoS Computational Biology*, vol. 1, no. 2: e24, doi:10.1371/journal.pcbi.0010024, 2005.
6. C. Byun, W. Arcand, D. Bestor, B. Bergeron, M. Hubbell, J. Kepner, A. McCabe, P. Michaleas, J. Mullen, D. O’Gwynn, A. Prout, A. Reuther, A. Rosa, and C. Yee, “Driving Big Data with Big Compute,” *Proceedings of the 2012 IEEE High Performance Extreme Computing Conference*, Waltham, Mass., 2012, available online at <http://iee-hpec.org/2012/agenda.htm>.
7. J. Kepner, *Parallel Matlab for Multicore and Multinode Computers*. SIAM Book Series on Software, Environments and Tools (Jack Dongarra, series editor). Philadelphia: SIAM Press, 2009.
8. J. Kepner, W. Arcand, W. Bergeron, N. Bliss, R. Bond, C. Byun, G. Condon, K. Gregson, M. Hubbell, J. Kurz, A. McCabe, P. Michaleas, A. Prout, A. Reuther, A. Rosa, and C. Yee, “Dynamic Distributed Dimensional Data Model (D4M) Database and Computation System,” *Proceedings of the 2012 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2012, pp. 5349–5352.
9. Robert Koch Institute, *Final Presentation and Evaluation of Epidemiological Findings in the EHEC O104:H4 Outbreak, Germany 2011*. Berlin: Robert Koch Institute, 2011, available online at [www.rki.de](http://www.rki.de).
10. N. Bliss, R. Bond, J. Kepner, H. Kim, and A. Reuther, “Interactive Grid Computing at Lincoln Laboratory,” *Lincoln Laboratory Journal*, vol. 16, no. 1, 2006, pp. 165–216.
11. J. Kepner and J. Gilbert, *Graph Algorithms in the Language*

of *Linear Algebra*. SIAM Book Series on Software, Environments and Tools (Jack Dongarra, series editor). Philadelphia: SIAM Press, 2011.

12. J. Kepner, W. Arcand, W. Bergeron, C. Byun, M. Hubbell, B. Landon, A. McCabe, P. Michaleas, A. Prout, T. Rosa, D. Sherrill, A. Reuther, and C. Yee, "Massive Database Analysis on the Cloud with D4M," *Proceedings of the 2011 High Performance Embedded Computing Workshop*, 2011, available online at <http://www.ll.mit.edu/HPEC/agendas/proc11/agenda.html>.
13. C. Camacho, G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T.L. Madden, "BLAST+: Architecture and Applications," *BMC Bioinformatics*, vol. 10, no. 421, doi:10.1186/1471-2105-10-421, 2009.
14. J. Kepner, "Spreadsheets, Big Tables, and the Algebra of Associative Arrays," Mathematical Association of America & American Mathematical Society Joint Mathematics Meetings, SIAM Minisymposium on Applied, Computational, and Discrete Mathematics at National Laboratories and Federal Research Agencies, 4–7 Jan. 2012, available online at [http://jointmathematicsmeetings.org/meetings/national/jmm2012/2138\\_program\\_friday.html](http://jointmathematicsmeetings.org/meetings/national/jmm2012/2138_program_friday.html).
15. 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," *Proceedings of the 2012 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2012, pp. 5329–5332.
16. N. Arcolano, "Statistical Models and Methods for Anomaly Detection in Large Graphs," SIAM Annual Meeting, Minneapolis, Minn., 2012, available online at [www.graphanalysis.org/SIAM-AN12/03\\_Arcolano.pdf](http://www.graphanalysis.org/SIAM-AN12/03_Arcolano.pdf).
17. J. Plavka, "Linear Independences in Bottleneck Algebra and Their Coherences with Matroids," *Acta Mathematica Universitatis Comenianae*, vol. 64, no. 2, 1995, pp. 265–271.
18. P. Butkovic, "Strong Regularity of Matrices—a Survey of Results," *Discrete Applied Mathematics*, vol. 48, no. 1, 1994, pp. 45–68.
19. M. Gavalec and J. Plavka, "Simple Image Set of Linear Mappings in a Max-Min Algebra," *Discrete Applied Mathematics*, vol. 155, no. 5, 2007, pp. 611–622.

### About the Authors



**Jeremy Kepner** is a senior staff member in the Computing and Analytics Group. He earned a bachelor's degree with distinction in astrophysics from Pomona College. After receiving a Department of Energy Computational Science Graduate Fellowship in 1994, he obtained his doctoral degree from the Department of

Astrophysics at Princeton University in 1998 and then joined MIT. His research is focused on the development of advanced libraries for the application of massively parallel computing to a variety of data-intensive signal processing problems. He has published two books and numerous articles on this research. He is also the co-inventor of parallel Matlab, Parallel Vector Tile Optimizing Library (PVTOL), Dynamic Distributed Dimensional Data Model (D4M), and the Massachusetts Green High Performance Computing Center (MGHPCC).



**Darrell O. Ricke** is staff member in the Bioengineering Systems and Technologies group. His work focuses on bioinformatics, software development, and data analysis for biomedical, forensic, and bio-defense projects. He received bachelor's degrees in computer science and genetics and cell biology, and a master's degree

in computer science from the University of Minnesota. He holds a doctoral degree in molecular biology from Mayo Graduate School.



**Dylan Hutchison** is pursuing a bachelor's degree in computer engineering and a master's degree in computer science at Stevens Institute of Technology. Inspired by the logic of computer science, he rallied for technical exposure through internships from business application design at Brown Brothers Harriman to parallel and

distributed computing research at Lincoln Laboratory. His recent focus is on methods to represent and compensate for uncertainty in inference systems such as Bayesian networks.