Intel "Big Data" Science and Technology Center Vision and Execution Plan

Michael Stonebraker, Sam Madden, Pradeep Dubey
stonebraker@csail.mit.edu, madden@csail.mit.edu, pradeep.dubey@intel.com
http://istc-bigdata.org

Abstract

Intel has moved to a collaboration model with universities consisting of "Science and Technology Centers" (ISTCs). These are located at a "hub" university with participation from other universities, contain embedded Intel personnel, and are focused on some research theme. Intel held a national competition for a 5th Science and Technology center in 2012 and selected a proposal from M.I.T. with a theme of "Big Data". This paper presents the big data vision of this technology center and the execution plan for the first few years.

1. Introduction

Others have categorized "big data" in three ways:

Big volume. The size of the database is too large to manage with current tools.

Big velocity. The data is arriving too fast for software to cope. This has been labeled as "drinking from a fire hose".

Big variety. The data is coming from too many disparate sources, and there is a massive data integration problem.

The members of the ISTC big data team believe that the "big volume" category must be subdivided into two components. Some users want to run conventional SQL analytics on massive data sets. In our opinion, this market is well served by the commercial data warehouse vendors, who are adept at managing peta-scale databases on multi-hundred node server farms, with on-line replication and failover and full transaction support. In fact, we know of a dozen or so installations at this scale from multiple vendors. Hence, we do not see the necessity of a research initiative in this area.

In contrast, we see an emerging need for complex analytics (machine learning, data clustering, predictive modeling, data categorization, etc.) on massive data sets. This market is not well served by the data warehouse crowd. In fact, most of the underlying algorithms are expressed as sequences of linear algebra operations on matrices. Hence, the relational model of data is a poor fit to this class of problems. Obviously, there is a need to mix complex analysis operations with data management (filtering, joins, etc.). As such, statistical packages provide only part of the needed functionality. Therefore, we are performing a major initiative in this area, as will be discussed in Section 2.

In the "big velocity" space, we require a corresponding subdivision. Some applications, for example electronic trading, must consume a "fire hose", looking for complex patterns in the stream of data. For example, if the trading engine thinks stocks A, B and C are correlated, then it would look for movement in any two of the three, and then trade the third based on the expected correlation. Complex event processing (CEP) engines are focused on this use case. The DBMS research community focused on this application area during the last decade, (see for example [8] and [9]), and it is not clear that a major new initiative is warranted in this area.

The second "big velocity" problem is processing the fire hose as before, but dealing with updating a persistent state, rather than searching for patterns. Maintaining the state of massively multiplayer Internet games is an example of this second use case. Here, applications look more like very high performance On-Line Transaction Processing (OLTP) problems. We know of many areas (e.g., maintaining leader boards, ad placement on web pages, maintaining real time risk exposure in trading engines) where there is considerable pain because of high velocity input. In Section 3, we indicate our initiative in this area.

In both areas, we propose end-to-end research programs, which span visualization technology, DBMS technology, and algorithm development, as will be explained in Sections 2 and 3 of this paper. Obviously, Intel is interested in the implications of big data on CPU and storage architectures; hence, we have research tasks in both areas dealing with hardware implications. These will be covered in Section 4.

We believe the "big variety" problem to be exceedingly important. In fact, some of us are active in this area [10]. In the future this ISTC will likely expand its scope into this facet of big data. Also, there are several aspects of "big data" that we are not addressing, largely because other Intel ISTCs are addressing them. These issues include security and privacy, being addressed by the ISTC at Berkeley, and cloud-oriented aspects of big data being tackled by the ISTC at CMU.

2. Complex Analytics – Big Volume

2.1 Motivation

We are motivated by four real world problems, which we briefly describe in this section.

Problem 1: Earth Science and satellite imagery. On the ISTC team are two researchers (Jim Frew and Bill Howe) who deal with Earth Science research using satel-

lite imagery. Frew uses such imagery to analyze snow depth in the Sierra Nevada Mountains to help manage water in the state of California, while Howe works with an oceanography group looking at water quality issues in Puget Sound. Both groups want to do transformations on massive amounts of imagery (MODIS in their case) and then browse the resulting data sets. The requirement is a scalable visualization system connected to a scalable database holding many terabytes of MODIS data.

Problem 2: Medical records and ICU data. Another ISTC member (Peter Szolovits) is interested in predictive modeling of medical data. We have access to 1600 patient days of intensive care unit (ICU) monitoring data, along with corresponding patient records. The goal is predictive modeling of medical events (for example code blues), so early intervention can be taken. This task requires complex prediction on a sea of data. Additionally, we have teamed with the Massachusetts General Hospital (MGH) and have access to their cancer patient database, which contains treatments and test results for all MGH cancer patients for the last 20 years. We are exploring a range of questions from modeling the relationship between cost and outcomes to the effectiveness of early screening and preventative medicine programs like mammograms and flu shots.

Problem 3: Large industrial machine maintenance. We are still looking for a partner in this area, but can describe the task as follows. Consider a complex piece of machinery (jet engine, helicopter, chemical plant, agricultural combine, etc.) Appropriate companies have the entire maintenance history on each piece of equipment and often real time monitoring data. The goal is to predict unscheduled maintenance problems, so they can be dealt with during a previous scheduled maintenance event. Problem 3 is the same kind of predictive modeling as Problem 2, but in a different domain.

Problem 4: Graph data. In the semantic web, Twitter, Facebook, and many science communities, there is a preponderance of graph data. What is needed is complex analytics on very large graphs. As an example, consider finding the average distance between any two humans in the Facebook graph. Other operations include minimum cut sets, reachability, etc.

We have (or expect to have) large amounts of data in each area on a server at MIT. In aggregate, we hope to have about 500 terabytes (0.5 petabytes) of data under management. Although others might suggest storing 0.5 petabytes in a file system, we believe that is the wrong approach to massive data sets. Database Management Systems (DBMSs) offer a range of useful services not addressed by file systems, including a schema (to control data semantics), a query language (to access subsets of data), sophisticated access control (on data granules), data consistency services (integrity control and transactions), compression (to reduce storage space), and indexing (to speed query performance). In our opinion, everybody with a big data problem should use DBMS technology. Hence, the file system is merely the storage layer used by the DBMS.

All of the sample problems in Section 2.1 are array or graph data and are ill-suited to the relational model. As a result we are studying two other options. First, we are exploring array databases, such as SciDB [11][1], as an obvious representation for much of the above data. In addition, we are exploring how to manage graph data. Our options include building a native graph DBMS, simulating graph data as sparse arrays in an array DBMS and a commercial graph DBMSs, such as Neo4J. We are also exploring visualization of large data sets and efficient algorithms for complex analysis.

2.2. Array Databases

We are investigating a variety of issues surrounding array DBMSs, including the following.

Array query languages. The success of the relational model has been helped immensely by a standard notation for queries and updates (SQL). In fact, other technologies, for example object-oriented databases and the entire "NoSQL" movement, have been hampered by a lack of standards. Some of us (Stan Zdonik and David Maier) are working on a standard query language for array data. Our approach is to define a standard abstract algebra of the semantics of operations (e.g. join, restriction, etc.). Then we plan to solicit agreement from the popular array-based DBMSs, including SciDB, Rasdaman and SciQL. Once we have agreement on the meaning of basic operations, we can move on to formalizing an SQL-like notation for arrays. A necessity for such an array query language (AQL) is to operate on arrays with integer dimensions (the standard ones in programming languages) as well as ones with dimensions of other data types (say latitude and longitude). As such the data model must allow arbitrary dimensions of user-defined data types, along with cell values that can be arbitrary vectors. It is also possible that we will extend our work to cover arrays with cell values that are complex data types. Our initial efforts are detailed at http://www.xldb.org/arrayql/.

Another standardization effort builds on the universality of the R statistical environment. R includes computation and visualization, as well as statistics. Hence, it is widely used as a programming and execution model for scientific computation. One of the pet peeves of many R users is the absence of scalability and data management functionality. Hence, we have built an extension of R that allows it to perform scalable execution by passing commands to an array database backend. This system is described in [12].

[1] One goal of our work in the ISTC is to release all code under an MIT or BSD open-source license. Because SciDB is GPL, our implementations do not make use of any SciDB code, and are designed to be able to operate independently from it.

Physical layout of array data in storage. There has been considerable research on the best ways of allocating relational data to storage blocks. However, most commercial RDBMSs allow a table to be sequenced, and that makes it a one-dimensional array. General arrays, on the other hand, can have multiple dimensions, and this allow more opportunities for storage optimization than do tables or sequenced tables. Hence, the best way to "chunk" multi-dimensional arrays onto storage blocks is a question we are working on. This problem gets more interesting if arrays are sparse and have a multitude of holes (nulls). Even harder is the case where arrays are sparse and the empty cells are skewed. For example imagine a two dimensional array with a cell value for every person in the United States. Obviously the density of people in Manhattan is 5 orders of magnitude higher than that in Montana.

We are investigating fixed size chunking systems, which would define a "stride" in one or more dimensions as the size of a "chunk". Such a layout makes query processing straight-forward, but will have problems with sparse and skewed data. On the other hand, we are also considering hierarchically decomposable systems based on splitting chunks, for example using quad trees. This will support skewed data using a regular, but variable size chunking system. Finally, we could also use a hierarchical chunk splitting scheme, for example based on R-trees, whereby all chunks become variable in size and irregular. The more flexible schemes deal with sparseness and skew more effectively, but make processing of joins more difficult. Lastly, we are investigating a scheme to group fixed-size chunks into "super chunks". Our initial results are presented in [13].

No overwrite and versioning of data. In many scientific applications it is important to be able to go back to earlier versions and compare the results of computations. Our approach is to add an extra dimension onto all arrays, which records wall clock time. Then, updates to array data merely add cells in this extra dimension. Hence, arrays have a dimension that grows without bound. This, of course, makes chunking strategies even more challenging, and our initial work in this direction is presented in [1][14].

Seamless on-line reprovisioning. A goal of all DBMSs is to support dynamic reprovisioning. In other words, if a data base is currently allotted X nodes, and more horsepower is needed, then the software should be able to add another Y nodes of storage and processing and then seamlessly move to utilizing all X + Y nodes for storage and processing. There have been extendible techniques developed for record data (e.g., Chord) as well as ones that make no attempt to organize the data for fast access (e.g., Hadoop). We are starting an effort to do the same thing for the chunk-oriented data we see in array-based systems.

Query optimizers. Optimization of SQL commands for relational data has been investigated for years, and appears to be well understood. There are well known strategies for performing joins of tables spread across multiple nodes in a computer system. However, array DBMSs present additional challenges. For example, if two arrays are joined using equality on all of the dimensions, then a straightforward chunk-to-chunk join can be performed. This generalizes the standard merge-sort used in relational systems. In addition, an array system must also perform joins where the join predicate entails matching cell values as well as ones that have a mix of cell values and dimension values. Just as with storage optimization, array systems present a more complex challenge than the simpler relational systems that have preceded array DBMSs.

Provenance. In most of our applications in Section 2.1, there is the possibility of incorrect data. Hence, whenever a result is calculated, a user should be able to trace the derivation of data, if he believes the result to be suspect. We have built an elaborate system that does exactly that, exploiting the semantics of relational and array operators to be able to efficiently work backwards, using a notion of *fine-grained provenance* [7]. We are also currently investigating visualization and other tools to help users understand data quality [20].

2.3. Matrix Calculations

Many big data analytic applications will need to combine data management with linear algebra in the same query, for example, finding the covariance between the historical times series of all pairs of stocks that have a market capitalization over $1B. This is a filter operation (at most an operation that is linear in the matrix size) followed by a covariance computation (cubic in the array size). Obviously, the "high pole in the tent" is the matrix calculation underneath covariance. There is a 10^5 difference (or more) in performance between coding such an operation in Python and in carefully optimized C++. Performance differences can be even greater when considering parallel implementations of such operations, and building efficient implementations can require many man-years of labor. Since there exist carefully optimized implementations of array-parallel operations (e.g., ScaLAPACK for dense arrays and ARPack for sparse arrays), we believe it makes sense for a DBMS to reuse to these libraries (as user-defined functions) whenever possible. Using optimized matrix code should move composite queries to be less dramatically CPU bound.

However, resource management is a problem in this hybrid world, because both the database system and ScaLAPACK are trying to be elastic and take advantage of otherwise idle resources. However, each system uses resources as though it has full control of the system, and does not access memory or disk in a way that is "friendly" to the other system. Hence, we are working on a meta-resource manager to mediate the resource demands of each system.

2.4. Graph Data

It is also clear that RDBMSs are a poor fit for graph data, although Facebook has continued to make them work for their problem. We are working on a number of different tasks in this area.

First, Carlos Guestrin has written a graph processing system (GraphLab) supported by a custom processing engine. This engine, which started as a single node main-memory system, has been extended to support distributed main memory and independently to support a single node disk environment. To truly scale, it must be further extended to distributed rotating storage. This will be the performance baseline, against which any other engine can be compared. One thrust is to implement graphs as sparse matrices. One of our team members (Jeremy Kepner) has a graph-processing engine supported by sparse arrays [24], which we plan to test against the baseline on a GraphLab benchmark.

To complement this activity, we are also working on a graph-specific storage system. We plan to compare these systems, to see if the above sparse matrix simulation of graphs is competitive with a native graph engine. We will also bake off GraphLab on top of Hadoop (and perhaps Pregel). We are skeptical that any Hadoop-based scheme will be competitive.

2.5. Visualization

The traditional form-based user interface (UI) technology is mostly useless in the problem domains of Section 2.1. Instead one needs a visualization system. Our focus is on scalability issues, not on the pixel representation on the screen. For example, MODIS users want an array browser to look through the gridded data that results from domain-specific transformations. Pointed at California, such a browser would overwhelm a conventional screen with data cells (in other words, the screen would be painted black). Instead, middleware software should perform resolution reduction to deliver to the visualization system an understandable amount of data. Our initial system that leverages query optimizer prediction of result sizes is described in [21]. We are working on a much more elaborate system, and are also working on predictive middleware to do intelligent prefetching and caching [22]. In parallel we are also investigating client side caching and how two optimization systems can work together [25].

2.6. Scalable Algorithms

We are working on several new, scalable algorithms, including a new, parallel streaming implementation of the widely used Locality Sensitive Hashing [2] and a new, scalable language for scientific computation called Julia.

PLSH: The goal of the PLSH (Parallel Locality Sensitive Hashing) project is to extend the widely used idea of Locality Sensitive Hashing to run in parallel on Intel multicore chips, distributable across several machines, and to support streaming updates as new data arrives and is hashed. We are planning to deploy it this spring on a collection of 1 billion tweets, looking at applications ranging from finding pairs of users who tweet about similar things to hash tags that a given user should follow.

Julia: Julia is a high-level, high-performance dynamic programming language for technical computing, with syntax that is familiar to users of other technical computing environments. It provides a sophisticated compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library. In addition, the Julia developer community is contributing a number of external packages through Julia's built-in package manager at a rapid pace. Julia programs are organized around multiple dispatch; by defining functions and overloading them for different combinations of argument types, which can also be user-defined. We plan to integrate Julia with SciDB, so that SciDB applications can be written in Julia.

3. "Big State - Little Pattern" High Velocity Problems

As noted above, the second major thrust of our research is in high throughput processing of operations over large amount of state.

3.1. Motivation

High velocity data means drinking from a fire hose, using online transaction processing (OLTP). Obviously, the only way to do this is with a parallel OLTP engine with very high node performance. Our thinking in this area is motivated by our work in [3], which showed that traditional RDBMSs suffered from high overhead, specifically in the implementation of dynamic locking, write-ahead logging, buffer pool management and multi-threading. Only perhaps 10% of the cycles contributed to useful work; the rest goes into the overhead associated with the above four issues. Clearly, one must remove all four of the above sources of overhead to perform dramatically better than traditional systems. Based on these criteria, we designed the H-Store OLTP-oriented DBMS a few years ago (see http://hstore.cs.brown.edu). It solved the four problems by eliminating the buffer pool, executing transactions in timestamp order, implementing command logging rather than data logging, and dividing main memory among the various cores, so there is no multi-threading. H-Store has been shown to be about two orders of magnitude faster than traditional RDBMSs on TPC-C [3]. However, there are substantial issues remaining, as we discuss below.

3.2. "Anti-caching"

We are working on relaxing the requirement that all H-Store data fit in the collective main memory of the allocated nodes. H-Store does not work well in this situation, as the only option is to allow the virtual memory manager on the underlying OS to page data to disk, which is extremely slow. Instead, we are investigating "anti-caching".

When memory is nearly exhausted, we package up the least-used ("coldest") tuples and write them to disk, together with a map of their location. As a result, the most used ("hottest") data resides in main memory and the cold data is on disk (but in main memory format). H-Store has been modified to make a "pre-pass" for any command to ensure needed tuples are in main memory. If not, they are

fetched, and the transaction is delayed until all needed data is main memory resident. Then, the command is executed normally. We have worked out eviction policies, fetch policies, and disk rearrangement policies for this model and have benchmarked it against a traditional RDBMS (MySQL). Additionally, we have benchmarked our system against MySQL with a Memcached main memory cache. Our system is dramatically faster than either system on almost all workloads, and H-Store degrades very slowly as the database becomes larger and larger. A paper on this effort has been submitted [16].

3.3. Concurrency Control

With the advent of interest in high performance main memory transactional data bases and the realization that traditional record-level locking is too slow to be used, there have been numerous ideas for high performance concurrency control, including deterministic time stamp ordering with speculative execution (H-Store), deterministic scheduling via pre-resolving conflicts [17], and multi-version concurrency control (NuoDB, Hekatron). We plan to study these algorithms to see if there are workloads on which one or another is preferred. Such studies were popular in the 1980's for disk-based DBMSs [18].

3.4. Integration of OLTP and Stream Processing

In the past, some of us have worked on complex event processing (CEP) engines. We have built the StreamSQL engine [9] as well as high performance pattern matching systems [4][5][6]. In effect, these are query processing engines that maintain a main memory state (the current partial satisfaction of temporal matches). There is much commonality between a CEP engine and an OLTP engine like H-Store. Each has a set of metadata catalogs, an execution engine, and the need for services such as high availability and crash recovery. As such, it would make perfect sense to combine a CEP engine with an OLTP engine. The composite would have broader applicability as well as allowing the sharing of quite a bit of functionality. Hence, we plan to start a project in this area.

4. Implications of "Big Data" on Computer Architecture

Both "big volume" and "big velocity" have implications for computer architecture as we explain below.

4.1. Big Volume Issues

At the heart of complex analytics lie algorithms with high computational complexity. Additionally data access patterns are often highly irregular, as in simple breadth first search of very large social network graphs. Addressing the architectural needs of a big-data compute platform is therefore quite challenging. Our immediate goal is to assess the new Intel® Xeon Phi™ chips for their capabilities in an end-to-end system, composed of both Intel® Xeon Phi™ and Intel® Xeon® chips.

Our first cut is to run the data management code on the Intel® Xeon® chips and ScaLAPACK on the Intel® Xeon Phi™ chips. The result should be a dramatic speedup in the matrix calculations. Opinions abound as to what the "high pole" will be in this configuration. Clearly, the matrix calculations will be improved significantly, which may result in an I/O bound or network bound composite system.

To support this work, we require a standard benchmark, which can be run on various hardware configurations and on other DBMSs and stat packages. We have developed a genomics benchmark [19] and are in the process of running it on hardware configurations, ranging from low end server clusters to the Stampede supercomputer at the University of Texas, which has thousands of nodes, each composed of Intel® Xeon Phi™ and Intel® Xeon® chips. In addition, we plan to test a variety of solutions capable of executing combined DBMS and statistics workloads.

This work could have substantial ramifications for the design of future high performance computers. Many existing supercomputers have a compute cluster, which is distinct from a companion file system cluster. Instead, we are proposing a much tighter integration of computation and storage management. Also, one can vary the computational resources of nodes by varying the ratio of Intel® Xeon Phi™ boards to Intel® Xeon® boards.

The genomics benchmark noted above has two instantiations, one is a dense array of genome values, while the other is an array of popular genomic sequences (SNPs). Since humans possess only some of these sequences, the array is quite sparse. Our benchmark requires covariance, biclustering and linear regression on such arrays. Optimizing dense array calculations for Intel® Xeon Phi™ is being done by Jack Dongarra, while others in the Intel Lab in Santa Clara are optimizing these operations for sparse matrices.

Additionally, we are working on using GPUs (including the Intel® Xeon Phi™) to efficiently render visualizations of massive scale data, using techniques such as transparency, heat maps, and other techniques to aggregate together many data points and present them most effectively. As a part of this effort, we are looking at pushing some kinds of common data filtering and processing operators into these types of co-processors.

Lastly, fixed function hardware can deliver orders of magnitude improvement in energy efficiency with respect to its programmable counterpart. We foresee opportunities for such acceleration for repeatedly used primitives like, data compression and basic operations on repeatedly used core data structures like a binary tree. This task is aimed at proper identification and abstraction of these functions such that hardware cost is minimized and the ease/portability of software development is not compromised.

4.2. Big Velocity Issues

There are three projects we are investigating in the big velocity realm. The first concerns thread movement, the second deals with flash memory as a replacement for disk, while the third concerns making main memory persistent.

We believe that there is an opportunity for hardware and/or operating system support for moving threads among the CPUs in a cluster. We are working on the details of doing this ultra-efficiently, possibly using new hardware we are developing [23]. If thread movement can be made efficient enough, then we need to revisit the standard DBMS scheme of "move the query to the data". Specifically, current H-Store execution decomposes a command into a tree of operations divided into phases. During each phase, a sub-command is executed at each of perhaps several modes and then reshuffling of data is performed. This strategy is reasonable when the cost of thread migration is expensive.

Cheap thread migration allows us to rethink query execution. In particular, one could have a collection of threads that move from node to node, exchanging synchronization and data messages when necessary. Moreover, a different execution scheme might allow other concurrency control schemes or in "tilting the playing field" toward one or another of the known schemes.

Many enterprises are currently investigating flash memory as a persistence mechanism to replace slower rotating magnetic storage. The primary reason is to improve the performance of secondary storage. Our second project is to explore the use of flash in H-Store. This can be performed in two different ways. It is a "drop on" to replace the disk in our anti-caching system with flash memory that is block addressable. However, our anti-caching system would rather have byte addressable flash system so finer granularity objects could be moved back and forth. We could even try putting the whole data base on flash; thereby using flash as a main memory replacement. We plan to address the performance of all of these configurations on a standard benchmark.

Looking further into the future, our third task is to explore the potential for emerging non-volatile (persistent), byte-addressable memory technologies, such as *phase change memory* (PCM). This technology offers DRAM-like access speed while being non-volatile, without the huge energy overhead and performance degradation of disks. We expect this technology to be better suited for main memory replacement than flash, which would eliminate the need for elaborate recovery schemes when power to DRAM is lost. Intel will provide us with a PCM emulator through which we can test the implications of this technology, both in a conventional H-Store setting as well as in an anti-caching setting. A paper on the performance of these memory systems is in preparation.

5. Summary

This paper has described the newest Intel ISTC focused on big data. As we explained, we are working on both big volume and big velocity issues, leaving big variety as a future topic. Our approach is to develop and leverage DBMS technology, as opposed to file systems. In all cases there are significant implications to the design of future computer systems. For more information about the Intel Science and Technology Center in Big Data, visit our website at http://istc-bigdata.org.

References

[1] A. Seering, P. Cudre-Mauroux, S. Madden, and M. Stonebraker. Efficient Versioning for Scientific Array Databases. In *ICDE* 2012.

[2] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In *VLDB* 1999.

[3] S. Harizopoulos, D. Abadi, S. Madden, and M. Stonebraker. OLTP Through the Looking Glass, And What We Found There. In *SIGMOD* 2008.

[4] Y. Mei, and S. Madden. ZStream: A Cost-based Query Processor for Adaptively Detecting Composite Events. In *SIGMOD* 2009.

[5] R. Newton, L. Girod, M. Craig, S. Madden, and G. Morrisett. Design and Evaluation of a Compiler for Embedded Stream Programs. In *LCTES* 2008.

[6] W. Thies, M. Karczmarek, and S. Amarasinghe. StreamIt: A Language for Streaming Applications. In *ICCC* 2002.

[7] E. Wu, S. Madden, and M. Stonebraker. SubZero: a Fine-Grained Lineage System for Scientific Databases. In *ICDE* 2013.

[8] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query Processing Resource Management and Approximation in a Data Stream Management System. In *CIDR* 2003.

[9] D. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The Design of the Borealis Stream Processing Engine. In *CIDR* 2005.

[10] M. Stonebraker, D. Bruckner, I. Ilyas, G. Beskales, M. Cherniack, S. Zdonik, A. Pagan, and S. Xu. Data Curation at Scale: The Data Tamer System. In *CIDR* 2013.

[11] M. Stonebraker. The Architecture of SciDB. In *SSDBM* 2011.

[12] P. Leyshock. Agrios: A Hybrid Approach to Scalable Data Analysis Systems. In *XLDB* 2012.

[13] E. Soroush, M. Balazinska, and D. Wang. ArrayStore: A Storage Manager for Complex Parallel Array Processing. In SIGMOD 2011.

[14] E. Soroush, and M. Balazinska. Time Travel in Scientific Array Databases. In *ICDE* 2013.

[15] N. Malviya, S. Madden, and M. Stonebraker. Rethinking Main Memory OLTP Recovery. (*submitted for publication*)

[16] J. DeBrabant, A. Pavlo, M. Stonebraker, S. Tu, and S. Zdonik. The Traditional Wisdom is all Wrong. (*submitted for publication*)

[17] A. Thomson, T. Diamond, S. Weng, K. Ren, P. Shao, and D. Abadi. Calvin: Fast Distributed Transactions for Partitioned Database Systems. In *SIGMOD* 2012.

[18] M. Carey, and M. Stonebraker. The Performance of Concurrency Control Algorithms for Database Management Systems. In *VLDB* 1984.

[19] M. Vartek, and R. Taft. A DBMS Benchmark for Complex Analytics. (*in preparation*)

[20] E. Wu, S. Madden, and M. Stonebraker. A Demonstration of DBWipes: Clean as You Query. In *VLDB* 2012.

[21] L. Battle. Resolution Reduction to Augment Visualizations. (*submitted for publication*).

[22] J. DeBrabant, L. Battle, U. Çetintemel, M. Stonebraker, and S. Zdonik. Caching and Prefetching to Support Massive Data Visualization. (*in preparation*).

[23] M. Lis, K. Shim, M. Cho, O. Khan, and S. Devadas. Directoryless Shared Memory Coherence Using Execution Migration. In *ICPDC* 2011.

[24] 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. In *ICASSP* 2012.

[25] Z. Liu, B. Jiang, and J. Heer. ImMens: Real-Time Visual Querying of Big Data. (*submitted for publication*)