Bulletin of the Technical Committee on

Data Engineering

December 2013 Vol. 36 No. 4 IEEE Computer Society

Letters

Special Issue on Scientific Data Management

Conference and Journal Notices

The TC on Data Engineering

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TCDE web page is http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

IEEE Computer Society Activities

Those who read these letters know that I have commented before about the shortcomings in the way the Computer Society is organized and run. The organization currently concentrates financial control at the top and makes it difficult for technical committees like our Technical Committee on Data Engineering (TCDE) to undertake initiatives, limiting bottom up entreprenurial opportunities. My thought has long been that changing this would open up new possibilities for the TCDE to better adapt to the current landscape of professional activities and engagements. And those changes would benefit the database technical community.

I have nothing final to report here. However, there are two committees within the Computer Society that are currently studying the fundamental organizational arrangements, one led by the Technical Activities Committee Chair Sven Dietrich, the other by Computer Society President-Elect Dejan Milojicic. For the Computer Society to thrive in the future, my view is that this kind of organizational scrutiny is an essential first step. I hope to have something more definitive to report in my next letter.

The Current Issue

Science has long prided itself on the primacy of data. The big breakthroughs usually occur because of data that is unexplained by the prior paradigm. The exploitation of science by technology depends upon using data to improve processes and techniques. So data is central to the scientific enterprise.

Given that, one would perhaps conclude that databases are similarly central. But that has not been the case. Databases were originally developed to solve the business data processing problem. Think payroll and orders. It was not developed with science in mind. But that is changing.

Two of the key people in re-orienting the database area toward the needs of science have been Jim Gray and Michael Stonebraker. Jim worked early on with astronomers to use database technology to serve as the backbone for their sky survey. Mike's efforts have resulted in the multi-university SciDB effort to provide database technology broadly to many scientists in many disciplines.

The current issue captures the now sizable effort to provide database technology tailored to the needs of scientists. It also demonstrates that this effort is a two way street in which methods from other sciences can influence how computer science might use data. This is both an active and a very important area of database research. Juliana Freire, the editor for the issue, has succeeded in bringing together a set of papers that captures many of the threads in "scientific data management", from leaders in this area. This area is a great opportunity for our field to help the overall scientific enterprise. My thanks to Juliana, who has worked hard to produce an issue that is a great introduction to the area and a valuable snapshot of its state of the art.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editor

The explosion in the volume of digital data and its wide availability is revolutionizing many scientific domains. At the same time, scientists faced with this data deluge must overcome many challenges to manage and explore these data. Complex processes are needed to acquire, process, and analyze the data. Even through there are robust and efficient databases systems, they fail to meet many of the requirements of emerging scientific applications which involve diverse data and require operations that go beyond what is currently supported. These new users and applications present new research problems in data management as well as a great opportunity for our community to have practical impact.

In this issue, we have collected a set of articles that highlight new directions for database research; relate limitations in current data management technology; and provide examples of how database research has been successfully applied to scientific problems in different domains, including neuroscience, astronomy, ecology. Motivated by needs in simulation sciences, Heinis et al. present a series of techniques they have developed to enable the construction and analysis of bigger and more detailed spatial models. They discuss the application of these techniques to real neuroscience datasets and show that they obtain a considerable performance improvement over the state of the art. In the context of a collaboration between astronomers and database experts, Vandeplas et al. are working on a platform for processing next-generation telescope image collections. They aim to process large volumes of sky images and allow questions to be efficiently answered over these data. Their paper describes the architecture of AscotDB, the system they have built, as well as techniques they have developed to address performance and usability issues. Stonebraker et al. presents an overview of the scientific database research at M.I.T, and summarize their work on making SciDB elastic, providing skew-aware join strategies, and producing scalable visualizations of scientific data. Alex Szalay considers the problem of analyzing very large simulation data, which are becoming increasingly harder to access, analyze and visualize. To allow broader usage of these data, he posits that analyses and visualizations must move to where the data resides and discusses the challenges in creating such interactive laboratories. Talbert et al. discuss challenges faced by ecologists in the field of species distribution modeling. Because of the scale of the data and the number of different models that are available, analyses are complex and require computationally-intensive sensitivity analysis accounting for various sources of uncertainty. While there exists technology to support these analyses, they are out of reach for many scientists who do not have a computer science background. To address this problem, Talbert et al. propose solutions that make use of scientific workflow systems. The last paper in this special issue examines the problem of reproducibility in science. While reproducibility is essential in this era of data-intensive science, the practice has not been widely adopted. One reason that is often cited is the fact that creating reproducible experiments is hard and time consuming. Chirigati et al. posit that this due in part to the lack of appropriate tools that support the tasks required for reproducibility. Besides characterizing the key tasks involved in the lifecycle of reproducible experiments, they propose a computational reproducibility benchmark. The benchmark aims to provide a means to categorize existing tools and better understand the features they support and how well they are supported.

These papers underscore importance of cross-domain synergies. They provide concrete examples of how database research has benefitted different scientific domains and how new research questions can be derived based on the needs of other areas. They also give evidence that data management is an essential component of science, and that our community has many challenging and significant problems to tackle.

I would like to thank all of the authors who agreed to share their work and experiences, as well as Dave Lomet who has provided invaluable guidance during the process of putting this issue together.

Juliana Freire
New York University
New York, New York

Enabling Scientific Discovery Via Innovative Spatial Data Management

Thomas Heinis, Farhan Tauheed, Mirjana Pavlovic, Anastasia Ailamaki
Data-Intensive Applications and Systems Laboratory
École Polytechnique Fédérale de Lausanne, Switzerland
{firstname.lastname}@epfl.ch

Abstract

Researchers in several scientific disciplines are struggling to cope with the masses of data resulting from either increasingly precise instruments or from simulation runs on ever more powerful supercomputers. Efficiently managing this deluge of data has become key to understand the phenomena they are studying. Scientists in the simulation sciences, for example, build increasingly big and detailed models, as detailed as the hardware allows, but they lack the efficient technology to update and analyze them.

In this paper we discuss how innovative data management techniques we have developed, enable scientists to build and analyze bigger and more detailed spatial models and how these techniques ultimately accelerate discovery in the simulation sciences. These include spatial join methods (in memory and on disk), techniques for the efficient navigation in detailed meshes, an index for range query execution on complex and detailed spatial data as well as in-memory mesh indexes. The evaluation of these techniques using real neuroscience datasets shows a considerable performance improvement over the state of the art, and that the indexes we proposed scale substantially better for the purpose of the analysis of bigger and denser spatial models.

1 Introduction

The simulation of spatial models has become a standard practice complementing traditional methods for understanding natural phenomena across many scientific disciplines. Examples cover various domains and include the simulation of peptide folding [3], star formation in astronomy [4], earthquakes in geology [6], fluid dynamics as well as the brain simulation in neuroscience [9].

Although the scientific disciplines are vastly different, the process of building and simulating a model is identical. As Figure 1 illustrates, data from the wet lab, from literature and from medical records is used to assemble an initial model. The model is then analyzed, validated and finally simulated. During and also after simulation, the model needs to be analyzed and visualized. The results of the visualization and analysis are finally fed back to the building phase to build more realistic models. The unprecedented amounts of data in every phase throughout the building and simulating process makes the data management in simulations challenging.

It is, however, not only the amount of data that keeps on growing and challenging current indexing methods. To develop a better understanding the scientists continuously increase the size and complexity of the simulations,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

pushing the limits of their computing infrastructure. In neuroscience, for example, scientists build models on the subcellular level (e.g., modelling neurotansmitter), thereby making the model considerably more detailed and dense. The data management challenges pertain to the size as well as complexity or level of detail of the models.

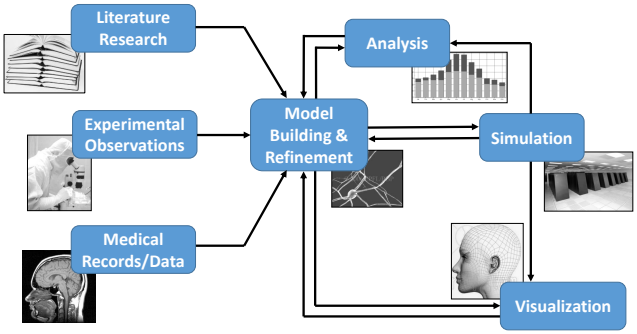


Figure 1: Steps of a simulation workflow.

The increasing size and complexity, i.e., the density (number of spatial elements per volume), of spatial datasets used in simulations renders existing tools and algorithms inadequate. Increasing density (or level of detail) in simulation datasets challenges the standard indexing approaches like the R-Tree (or its improvements [2]). The more elements a spatial dataset packs in the same space, i.e., the denser it is, the more the elements on the leaf level of the R-Tree overlap. More detrimental to query execution time, however, is that more overlap on the leaf level translates into disproportionately more overlap in the tree structure of R-Tree [15] and query execution time of the R-Tree consequently does not scale well with density. Similarly, the update frequency and scale in simulation applications (for example in earthquake or computational fluid dynamics simulations almost all elements change position in every step) render known update mechanisms for spatial indexes inefficient. Novel indexing techniques consequently need to be developed to provide efficient and scalable access to dense and frequently updated spatial datasets.

In this paper we identify the state-of-the-art spatial indexes used in the simulation workflow that do not scale with increasing density/level of detail of the spatial models. We develop new indexes for range query execution, spatial join and prefetching of spatial data based on the idea that while density prevents the state of the art from scaling, the proximity of the elements in dense datasets can be used to our advantage. More particularly, the approaches we develop are based on the key insight that the connectivity (inherent or added) in datasets can be used to recursively explore datasets independent of density (and hence avoiding the overlap problem).

In the remainder of this paper we present the data management techniques based on connectivity to support building and simulating large-scale models. The paper is organized along the simulation workflow: we first give an overview over the methods we have developed for building [11, 14] models, for analyzing [15, 17] spatial models and finally for analyzing running simulations [12, 16]. We subsequently conclude and discuss the impact the connectivity-based indexing techniques have on a neuroscience use case.

2 Model Analysis & Visualization

A crucial type of query in the model building process and in the model analysis phase is the spatial range query. Range queries are repeatedly used to visualize parts of the models or to ensure that the models built satisfy statistical constraints (in case of neuroscience, for example, testing the tissue density, synapse density, etc.). It is also equally important for many of their analysis to execute a series of range queries that follow one of the many structures (e.g., a road in a road network in GIS) in the model to assess the quality or validity of the model.

2.1 Range Queries on Dense Spatial Models

The efficient execution of range queries to build and validate models is pivotal today and will become even more important in the future where the models will be increasingly dense (i.e., biorealistic through modeling phenomena on the subcellular level). While today's spatial indexing approaches [2] execute range queries efficiently on many datasets, they unfortunately do not do so on today's detailed and dense spatial models [15]. To make matters worse, they will only scale poorly to more detailed future models as experiments show [15].

Today's methods are based on the hierarchical organization of spatial data (R-Trees and variants [2]) and therefore suffer from overlap [2] and dead space. Overlap increases considerably with increasing level of detail/density of the dataset: as the number of spatial elements in the same unit of space increases, so does the overlap of tree-based indexes [15]. Despite numerous proposed improvements, e.g., reducing overlap through splitting and replicating elements (R+-Tree [2]), the fundamental problem remains the same and needs to be addressed to enable the simulation scientists to build and analyze more detailed models.

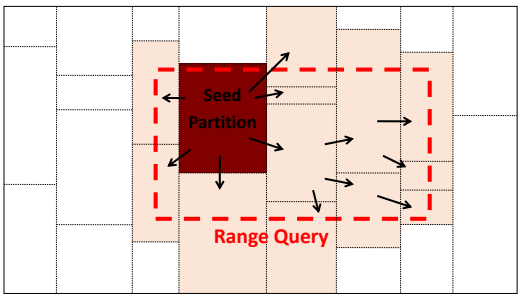


Figure 2: Query execution: given an inital group of elements (dark), FLAT recursively visits all neighbors of the seed partition.

To enable scientists to build and analyze models with an unprecedented level of detail, we developed FLAT [15]. The key insight we use is that while finding all elements in a particular range query in an R-Tree suffers from overlap, finding an arbitrary element in a range query is independent of overlap and thus is a cheap operation. With this insight, we develop a two-phase query execution approach where we (1) find an arbitrary element e in the query range using an R-Tree and (2) recursively retrieve all other elements (the neighbors of e) within the range using neighborhood information (what elements neighbor what other elements) previously added to the dataset. Both phases are independent of overlap and density as the first only depends on the height of the R-tree and the second only depends on the number of elements in the range query. FLAT thus becomes independent of overlap and scales to much denser/detailed models.

When indexing, FLAT needs to build an R-Tree used to find an initial element and it also needs to compute neighborhood information of the elements. To curb the amount of information stored, FLAT only stores neighborhood information on the level of groups of elements instead of on the level of single elements. FLAT groups spatially close elements together (and stores them on the same disk page), indexes the groups with the R-Tree and finally computes the neighborhood information between the groups. The neighborhood information itself is stored in the leaf nodes of the R-Tree. Figure 2 illustrates how queries are executed using the groups of elements: given an initial arbitrary group in the query range, FLAT recursively retrieves all neighbors until all groups in the query range are retrieved.

FLAT works on arbitrary datasets from different disciplines and as measurements show it can speed up query execution by up to one order of magnitude on a dense spatial model from neuroscience [15]. The improvement over the state of the art is bigger, the more densely packed the spatial datasets are. Indeed, similar results are obtained for equally densely packed spatial datasets from astronomy and computer vision [15].

2.2 Executing Guided Spatial Range Query Sequences

An important type of query for the model analysis process is the execution of a series of range queries: following a structure in the model, e.g., a neuron branch, neuroscientists need to interactively execute range queries to analyze the model. On the result of each range query they compute different types of statistics (tissue density, synapse placement, synapse count, etc). Series of range queries are not only crucial for the neuroscientists, but also for other scientists who analyze road networks, arterial trees and others.

Executing a sequence of range queries is an interactive process where a scientist follows a structure, executes a query, performs analyses or computes statistics on the query result, studies the analyses results and then decides on the location of the next query and executes it. The disk is idle during the computation of statistics (between two range queries) because execution of the series is interactive and the next query location is unknown. To speed up the execution of the series data can be prefetched at potential query locations while the disk is otherwise idle.

State-of-the-art approaches to prefetch spatial data only do so with low accuracy because they rely only on limited information, i.e., the position, of previous queries to predict the location of the next query. One particular

approach [13] uses the last query position and prefetches around it. More sophisticated approaches [1] use the last few positions, fit a polynomial and extrapolate the polynomial to predict the next query location. Series of range queries on structures like neurons or arterial trees, however, are not smooth at all but jagged and are therefore impossible to interpolate accurately with a polynomial. A different class of approaches [8] learns from past user behavior by keeping track of all paths visited in the past and by basing predictions on the accumulated history. Given the massive size of today's spatial models, however, it is unlikely that any path will be visited twice, therefore making prefetching strategies based on past paths visited inaccurate.

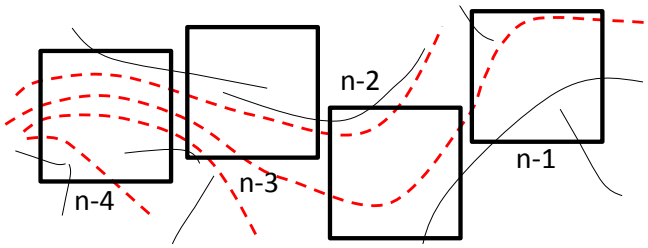


Figure 3: Pruning the irrelevant structures (solid lines) from the candidate set (dashed lines) in subsequent queries (solid squares) of the series.

To prefetch more accurately and to speed up the execution of series of range queries we develop SCOUT [17]. SCOUT departs from previous approaches because it does not consider previous query positions, but instead takes into account the previous query content (the structures in the previous query) knowing that scientists are likely to follow one of the structures in the previous queries.

SCOUT summarizes the structures of the most recent query q, i.e., it identifies the topological skeleton in q and approximates it with a graph. The graph of q represents all the structures the scientist is potentially following and SCOUT therefore prefetches data at all locations where the graph edges and therefore the structures exit query q. Range queries are executed to prefetch data at these locations until the user executes a new query in the series.

Clearly, having to prefetch at several different locations will reduce accuracy as only one of the prefetched locations is correct. SCOUT uses candidate pruning to reduce the number of prefetching locations, exploiting that all previous queries must contain the branch the scientist follows. To prefetch for the n^{th} query, SCOUT thus only needs to consider the set of branches leaving the $(n-2)^{th}$ query and the set of branches entering the $n-1^{th}$ (most recent) query. The branch followed is in the intersection of both sets. As the number of queries in a series increases, the number of branches in the intersection between two consecutive queries decreases continuously and the branch the user follows can be identified. Figure 3 shows a series of range queries ($n-1$ to $n-4$th query) obtained by a neuroscientists interactively executing queries, i.e., through executing $n-4$ and depending on the result executing query $n-3$. The figure also illustrates how through iteratively reducing the set of candidates, SCOUT can reliably identify the structure the scientist follows after only a few queries, ultimately speeding up query series by a factor of up to 15× [17].

3 Model Building

A crucial operation in building spatial models is to find the intersections of spatial elements. In many simulation models, overlap or intersection between elements no longer realistically reflects the system modeled (neurons cannot overlap in reality and neither can celestial objects). A spatial join is consequently needed in the model building phase to detect errors, i.e., intersections.

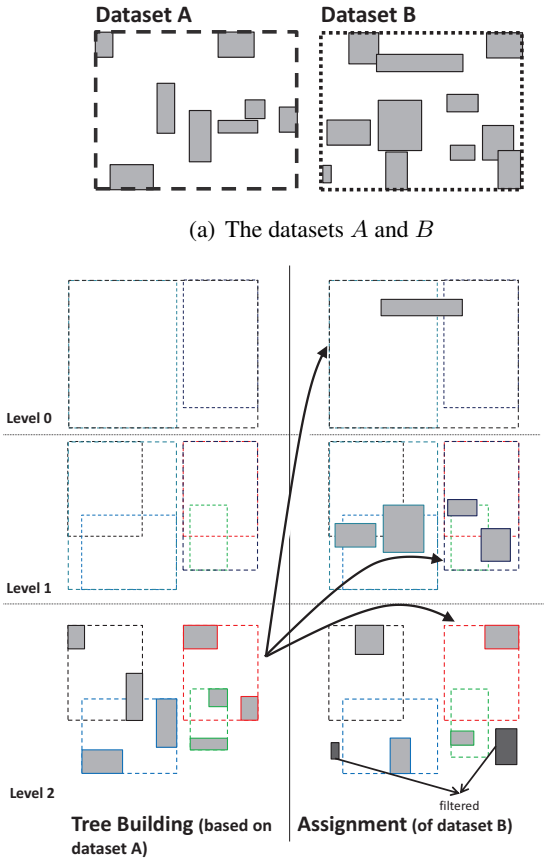
Neuroscientists use the spatial join for yet a different application: to obtain a biorealistic model they need to determine where to place synapses (structures that permit electrical impulses to leap between neurons). Prior research shows [7] that synapses need to be placed where neurons are within a given distance to each other, translating the problem into a spatial or distance join.

3.1 In-Memory Spatial Join

If the spatial model fits into the main memory of a single machine or in the aggregate memory of a supercomputer, the spatial join needs to be performed in memory. Despite decades of research into spatial joins, only two algorithms have been particularly designed to join two datasets in memory: the nested loop join [5] and the sweep line approach [5]. Neither of the approaches scales well: the nested loop join has quadratic complexity while the sweep line approach is inefficient in case too many elements are on the sweep line (excessive comparisons of elements nearby in one dimension but distant in a different dimension).

Existing work developed for disk [5] can of course also be used in memory. Disk-based spatial joins can be categorized into space- or disk-oriented partitioning approaches and while both classes have advantages, they also have clear disadvantages: space-oriented approaches [5] need to replicate elements (elements intersecting with two partitions need to be copied to both) increasing the memory footprint and incurring multiple detections of the same intersections. Data-oriented approaches [5], on the other hand, suffer from the overlap problem shared by all R-Trees. Overlap in data-oriented approaches degrades performance already today, but more importantly, it will increase with denser future datasets [15].

Given the shortcomings of current in-memory spatial join approaches and the challenges of disk-based spatial joins, we develop a novel in-memory spatial join algorithm called TOUCH [11]. TOUCH avoids space-oriented partitioning because space-oriented partitioning requires replication of elements which (a) increases the memory footprint and (b) requires multiple comparisons between copies of elements (as well as making the removal of duplicate results necessary). TOUCH also targets at avoiding the problem of overlap prevalent in approaches based on data-oriented partitioning.



(a) The datasets A and B

(b) Tree building, assignment and joining phases

Figure 4: The three phases of TOUCH: building the tree, assignment and joining.

TOUCH uses data-oriented partitioning to avoid the replication problem and builds an index similar to an R-Tree on the first dataset A (all elements of A are in the leaf nodes). To avoid the issue of overlap, it does not probe the index for each element of the second dataset B. Rather, it assigns each element b of B to the lowest (closest to the leafs) internal node of the index that fully contains b. Only after all elements of B are assigned to nodes of the index, TOUCH performs the actual join: elements of B in each internal node n are tested for intersection with all leaf nodes (containing elements of A) reachable from n. Figure 4 illustrates the process, i.e., how the index is built on dataset A, how the elements of dataset B are assigned to nodes and finally, how internal nodes are joined with leaf nodes.

Our measurements show that TOUCH outperforms known in-memory approaches as well as disk-based approaches used in memory [11]. TOUCH is the fastest to perform the join followed by PBSM [5], a simple space-partitioning approach used in memory. Although PBSM is the fastest competitor, it is still one order of magnitude slower and also uses considerable more memory (a factor of 8 $\times$ more).

3.2 Selective On-Disk Spatial Join

The spatial join, however, is not only crucial in memory. To build large-scale models that do not fit into main memory, efficient out-of-core methods are required to support simulation scientists. A particularly important disk-based spatial join needed by simulation scientists is the joining of datasets of different density, i.e., of similar spatial extent but with a vastly different number of spatial elements. Example applications include adding a few roads or other spatial objects to GIS datasets, adding the branches of one neuron to a spatial model of the neocortex and similar applications. The efficiency of the join is pivotal as it is oftentimes executed repeatedly to join several sparse datasets with one dense dataset.

Many approaches for disk-based spatial joins [5] have been developed in the past and each can be used to join a dataset A_i (with few elements) and B (with a massive number of elements). Existing approaches, however, are not efficient for a join where with a very small A_i, only a small subset of B needs to be retrieved (and tested against A_i). State-of-the-art approaches based on space-oriented partitioning (e.g., PBSM [5]) create coarse-grained partitions and thus the entire dataset B is read for a join, leading to excessive disk access. Approaches based on data-oriented partitioning, on the other hand, require hierarchical trees (e.g., synchronized R-Tree [5]) to access the data and thus suffer from the well documented problems of overlap, also resulting in excessive disk access.

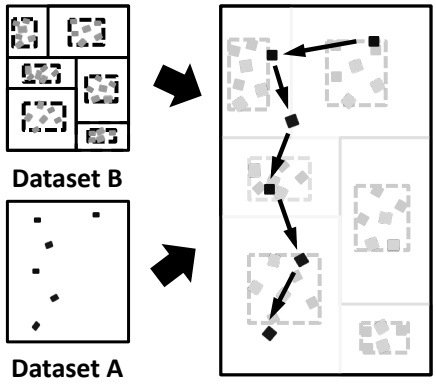


Figure 5: GIPSY uses the sparse dataset to walk/crawl through the dense dataset.

We develop GIPSY, a novel approach that avoids the coarse-grained partitioning of space-oriented approaches and instead uses the fine-grained data-oriented partitioning, thereby enabling the join to read from B only the small subset required. At the same time GIPSY avoids the excessive disk page reads and comparisons due to overlap in the tree structure of data-oriented approaches. Instead of traversing a tree like data-oriented approaches (e.g., the R-Tree), GIPSY traverses the data using a crawling approach [15].

More precisely, GIPSY partitions dataset B in data-oriented fashion and adds neighborhood information to B, i.e., what elements neighbor what other elements. It then takes the elements of the sparse dataset A_i and visits them one after the other by walking between them using the neighborhood information previously added to the dense dataset B. Once GIPSY arrives at the location of a particular element e_i of the sparse dataset, it uses crawling (again using the neighborhood information) to find all elements of the dense dataset around e_i's location and tests them for intersection with e_i. Once no more elements intersecting with e_i can be found, GIPSY walks to the position of the next spatial element e_{i+1} of A_i in the dense dataset B. Figure 5 illustrates how GIPSY uses the sparse dataset to direct walking in the dense dataset.

With its novel combination of crawling and data-oriented partitioning, GIPSY achieves a 2 to 18$\times$ speedup compared to the fastest approaches (indexed nested loop and PBSM, both in [5]) when joining several A_i with B. The evaluation [14] further shows that the improvement over the state of the art grows considerably bigger, the more sparse datasets A_i are joined with B or the bigger the difference in density between A_i and B is.

4 Model Simulation

To analyze, steer and monitor the spatial model while the simulation runs, a number of spatial range queries needs to be executed on the spatial model entirely stored in main memory of the simulation infrastructure at every step of the simulation. Indexing the spatial model speeds up range query execution, however, current indexing methods cannot efficiently support the large-scale updates of simulation applications. During simulation the

spatial models undergo massive changes, resulting in a change of position of all spatial elements in the model at every step of the simulation. Maintaining an index in face of changes on this scale incurs considerable overhead.

Current methods particularly designed to support large-scale updates on spatial indexes cannot cope with simulation applications: not enough queries are executed on the index at every time step to amortize the cost of rebuilding lightweight spatial indexes (MOVIES [10]) at every time step or the cost maintaining indexes designed specifically to reduce update cost (LUR-Tree or QU-Trade [10]). Approaches particularly designed for moving objects that index trajectories (e.g., STRIPES, TPR-Tree [10]) cannot be used either because the unpredictable nature of the movements in simulation applications cannot be interpolated with polynomials.

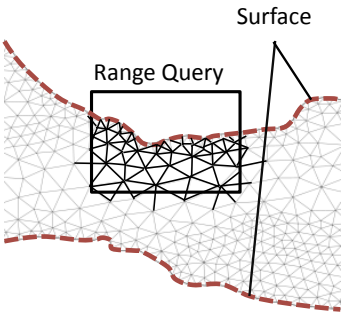


Figure 6: OCTOPUS: Starting from the surface, the edges of the polyhedra will be visited.

To support the efficient execution of range queries on entirely main memory resident meshes undergoing frequent and massive changes we develop OCTOPUS [16]. Even with unpredictable changes affecting the entire mesh dataset OCTOPUS outperforms state-of-the-art methods by working on the mesh datasets directly instead of maintaining complex data structures (like indexes). OCTOPUS uses the connectivity of the mesh to avoid accessing the entire mesh when computing query results: given any vertex inside the query region, OCTOPUS can use the mesh connectivity to recursively retrieve all vertices inside the query. Using the current state of the mesh directly in memory has the advantage that the result can be computed without having to consider the change of the vertex location in the last update.

Solely depending on the connectivity of the mesh, however, bears the risk that the result is incomplete because parts of the mesh in the query region may not be connected (as illustrated in Figure 6). OCTOPUS therefore starts the mesh traversal from the mesh surface enclosed in the query region and recursively retrieves all neighboring elements within the query range (in the absence of a surface vertex in the query range, it starts a directed walk from a random element to find a vertex inside the range). By building on the key insight that every vertex inside a query range is connected to at least one vertex on the surface, we prove [16] that OCTOPUS retrieves the complete result. To find start elements on the mesh surface, OCTOPUS maintains a set of pointers to the surface elements. The set only infrequently needs maintenance because the surface only changes in the rare event where the connectivity of the mesh changes.

Our experiments show that OCTOPUS achieves a speedup between 7.2 and 9.2$\times$ compared to the state of the art. OCTOPUS will scale better with increasingly detailed meshes than other approaches because it only needs to keep track of the mesh surface: when meshes become more detailed, the size of surface grows only quadratic whereas the size of the complete mesh grows cubic. We also develop a general version for the efficient execution of spatial range queries on arbitrary data other than meshes [12].

5 Conclusions

To study in more detail how a natural phenomena works, scientists in different disciplines build and simulate increasingly big, complex and detailed models of the system they study. State-of-the-art methods no longer can be used to efficiently build, analyze and validate models because they have grown too big and too detailed. We have therefore developed new tools and indexes by carefully analyzing strengths and weaknesses of the state of the art. Where it is efficient and scalable, we use elements of known approaches (e.g., data-oriented partitioning in GIPSY, the R-Tree to find a random start element in FLAT etc.) and combine them with novel ideas (e.g., using query content in SCOUT). The resulting methods execute queries/joins faster and scale better to bigger and denser spatial models.

We have tested and deployed the indexes and techniques in the context of the Blue Brain Project (BBP [9]) where neuroscientists attempt to build and simulate the human brain. The impact of the methods we have

developed on the BBP are substantial as today they can build and analyze bigger models faster. While previously limited to building and analyzing models of 100'000 neurons, the new methods have enabled them to grow the models to 10's of millions (with a theoretical maximum of 33 million neurons on the current infrastructure). The new tools have not only accelerated discovery, but have also enabled it: analyses, e.g., assessing if the synapse density is bio-realistic, have not been possible without efficient means to access the spatial models.

Our work also demonstrates that despite decades of research in spatial data management, many challenges remain. Increasing main memory as well as novel storage technology (in the memory hierarchy), for example, means that several spatial indexes need to be redesigned. New types of datasets (e.g., dense, complex spatial datasets) make new indexes necessary and new types of queries (e.g., series of range queries) also call for the development of new indexes. Many interesting opportunities for exciting research therefore remain.

6 Acknowledgments

Part of this work is supported by a grant from the Hasler Foundation (Smart World Project No. 11031), from the Human Brain Project (Grant 604102) and from the Department of the Navy (Grant N62909-12-1-7010).

References

[1] A. Chan, R. W. H. Lau, and A. Si. A motion prediction method for mouse-based navigation. In *International Conference on Computer Graphics*, pages 139–146, 2003.

[2] V. Gaede and O. Guenther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2), 1998.

[3] S. Gnanakaran, H. Nymeyer, J. Portman, K. Y. Sanbonmatsu, and A. E. Garcia. Peptide folding simulations. *Current Opinion in Structural Biology*, 13(2):168–174, 2003.

[4] J. Gray, A. Szalay, A. Thakar, P. Kunszt, C. Stoughton, D. Slutz, and J. Vandenberg. Data Mining the SDSS SkyServer Database. In *Technical Report, MSR-TR-2002-01, Microsoft Research*, 2002.

[5] E. H. Jacox and H. Samet. Spatial join techniques. *ACM TODS*, 32(1):7, 2007.

[6] D. Komatitsch, S. Tsuboi, C. Ji, and J. Tromp. A 14.6 Billion Degrees of Freedom, 5 Teraflops, 2.5 Terabyte Earthquake Simulation on the Earth Simulator. In *Supercomputing*, 2003.

[7] J. Kozloski, K. Sfyrakis, S. Hill, F. Schürmann, and H. Markram. Identifying, Tabulating, and Analyzing Contacts Between Branched Neuron Morphologies. *IBM Journal of Research & Development*, 2008.

[8] D. Lee, J. Kim, S. Kim, K. Kim, K. Yoo-Sung, and J. Park. Adaptation of a Neighbor Selection Markov Chain for Prefetching Tiled Web GIS Data. In *Advances in Information Systems*, 2002.

[9] H. Markram. The Blue Brain Project. *Nature Reviews Neuroscience*, 7(2):153–160, 2006.

[10] M. F. Mokbel, T. M. Ghanem, and W. G. Aref. Spatio-temporal Access Methods. *IEEE Data Engineering Bulletin*, 2003.

[11] S. Nobari, F. Tauheed, T. Heinis, P. Karras, S. Bressan, and A. Ailamaki. TOUCH: In-Memory Spatial Join by Hierarchical Data-Oriented Partitioning. In *SIGMOD*, 2013.

[12] M.-A. Olma, F. Tauheed, T. Heinis, and A. Ailamaki. BLOCK: Efficient Execution of Spatial Range Queries in Main-Memory. Technical report, EPFL, 2013. `https://infoscience.epfl.ch/record/190731`.

[13] D.-J. Park and H.-J. Kim. Prefetch Policies for Large Objects in a Web-enabled GIS Application. *Data & Knowledge Engineering*, 37(1):65–84, 2001.

[14] M. Pavlovic, F. Tauheed, T. Heinis, and A. Ailamaki. GIPSY: Joining Spatial Datasets with Contrasting Density. In *SSDBM*, 2013.

[15] F. Tauheed, L. Biveinis, T. Heinis, F. Schürmann, H. Markram, and A. Ailamaki. Accelerating range queries for brain simulations. In *ICDE*, 2012.

[16] F. Tauheed, T. Heinis, and A. Ailamaki. OCTOPUS: Efficient Query Execution on Dynamic Mesh Dataset. In *ICDE*, 2014.

[17] F. Tauheed, T. Heinis, F. Schürmann, H. Markram, and A. Ailamaki. SCOUT: Prefetching for Latent Structure Following Queries. In *VLDB*, 2012.

Squeezing a Big Orange into Little Boxes: The AscotDB System for Parallel Processing of Data on a Sphere

Jacob Vanderplas[1,2] Emad Soroush[1] Simon Krughoff[2] Magdalena Balazinska[1]
Andrew Connolly[2]

[1] Department of Computer Science and Engineering, University of Washington, Seattle WA
[2] Department of Astronomy, University of Washington, Seattle WA

Abstract

AscotDB is a new, extensible data analysis system developed at the University of Washington for the interactive analysis of data from astronomical surveys. AscotDB is a layered system: It builds on SciDB to provide a shared-nothing, parallel array processing and data management engine. AscotDB wraps SciDB with a Python middleware that enables efficient storage and manipulation of spherical data, such as images from telescopes or satellites. The goal is to support the efficient storage of raw pixel-level data without any prior preprocessing steps. To enable both exploratory and deep analysis of the data, AscotDB's front-end design integrates a python interface with a graphical interface based on the Astronomy Collaborative Toolkit (ASCOT). AscotDB supports seamless switching between these two modes of interaction and captures a precise trace of a user's operations on the data to ensure repeatability. In this paper, we present an overview of the AscotDB system. While based on astronomy as its key application-domain, AscotDB primitives are general enough to be applicable to other scientific fields concerned with data on a sphere.

1 Introduction

Astronomy, like other scientific fields, is currently moving to a new realm of research driven by large datasets. One example of this new approach is the upcoming Large Synoptic Survey Telescope (LSST) [4], which will begin operations toward the end of this decade. The LSST will repeatedly image the entire southern sky over ten years, resulting in unprecedented volumes of uniformly-calibrated astronomical data – up to a hundred Petabytes by the end of the survey. The scientific results from these data will depend on our ability to perform fundamental operations at scale, including the detection of faint objects through the stacking of multiple exposures and the detection of variable objects through the differencing of exposure pairs, all while taking into account the temporally and spatially varying *point-spread function* – that is, the mathematical model of how light from any given object is detected by the camera after refraction by the atmosphere and telescope optics.

The AscotDB project is a collaboration of an inter-disciplinary team comprising astronomy and database experts with the goal of answering one question: What would be the most transformative tool for processing these next-generation telescope image collections? The AscotDB system has emerged in answer to this question.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

In order to image the visible sky, LSST will undertake repeated exposures over ten years with each image partially overlapping with hundreds of others. To enable efficient stacking and comparison of these images, it is vital to store the data in a way that allows efficient indexing of pixel positions both on the sky and in time. While pipelines are being designed by the LSST team to handle this image processing task and create *catalogs* of detected objects, the truly transformative science will come from providing scientists with the ability to directly query the *raw* data, and to enable *interactive and exploratory* computation and visualization of that data.

AscotDB integrates several pieces of technology: the SciDB engine for data storage and processing, Ascot for graphical data exploration, and Python for easy programmatic access. None of these technologies solves the problem on its own. More importantly, their naïve combination is also insufficient.

SciDB. SciDB [14] is an open-source database system with *inherent* support for multi-dimensional arrays. The inherent support means the entire SciDB stack is designed based on the array data model. In prior work [5, 10, 15], we showed that an array-based database system such as SciDB is well-suited to store and manipulate small patches of sky images pre-projected (or *warped*) onto a regular Cartesian grid. Although this SciDB-based solution is compelling, the solution remains a step removed from the raw data: it requires significant overhead to individually warp images to each local tangential projection before loading them into the database for an analysis, and the local nature of the tangential projection means the entire sky cannot be analyzed at once. Because SciDB is built for efficient computing over Cartesian arrays, it is not straightforward to use it for the inherently spherical data gathered by astronomical surveys.

Ascot. While efficient data storage and manipulation is an important piece of the astronomer's task, another vital component is the ability during an analysis to visually interact with and explore data in order to determine the next analysis step. The AStronomy COllaborative Toolkit (ASCOT) [9][1] developed in the Astronomy community, is a collection of Web-based gadgets that facilitate collaboration between astronomers. These gadgets are assembled into a dashboard and communicate using a node.js server. Through the use of a customizable dashboard interface, users can easily visualize, manipulate, and share large data sets from many different sources. Ascot, however, only enables the display of individual images and the analysis of pre-processed *catalog* data. It does not support the large-scale analysis of pixel data.

Python. For both data analysis and visualization, Python is fast becoming a defacto standard in the astronomy research community. The scientific Python ecosystem, built around the core tools of NumPy & SciPy [12], Matplotlib [7], and IPython [13] (including the web-based interface of the *IPython notebook*), provides a complete environment for the analysis and visualization of data at a small to medium scale. Due to its advantages, a wide selection of current and future astronomical surveys are now building their data analysis pipelines using Python. Of particular relevance for this work, the LSST project plans to use Python as their primary data pipeline interface. In this environment, it is increasingly important for data analysis tools to provide Python-hooks for any computing infrastructure. Python alone, however, provides limited data management capabilities and is non-trivial to use on LSST-sized datasets.

AscotDB builds on the combination of these three pieces of technology to provide a compelling and powerful environment for the exploration, analysis, visualization, and sharing of large astronomical datasets. AscotDB further contributes several techniques: It extends SciDB with native support for efficient iterative computations (Section 2.1). It provides more intuitive Python language bindings through a new SciDB-py package (Section 2.2). It further integrates the graphical and programmatic interfaces to support seamless switching between the two modes of interaction (Section 2.3). Finally, it adds a middleware-layer that enables the manipulation of

[1] `http://ascot.github.io/`

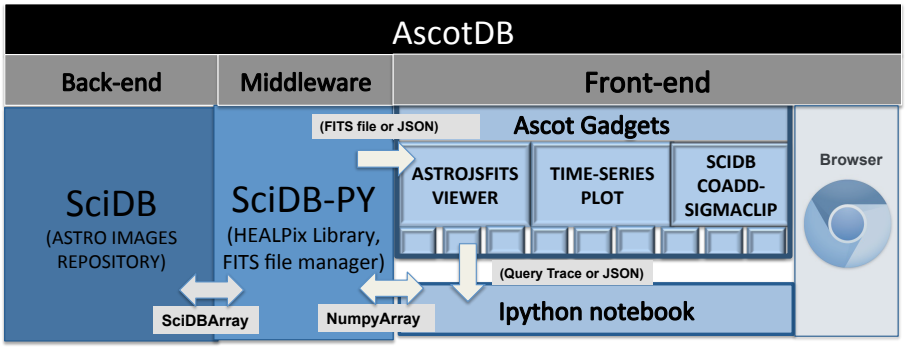


Figure 1: AscotDB architecture: SciDB as back-end, python middleware, Ascot and IPython as front-ends.

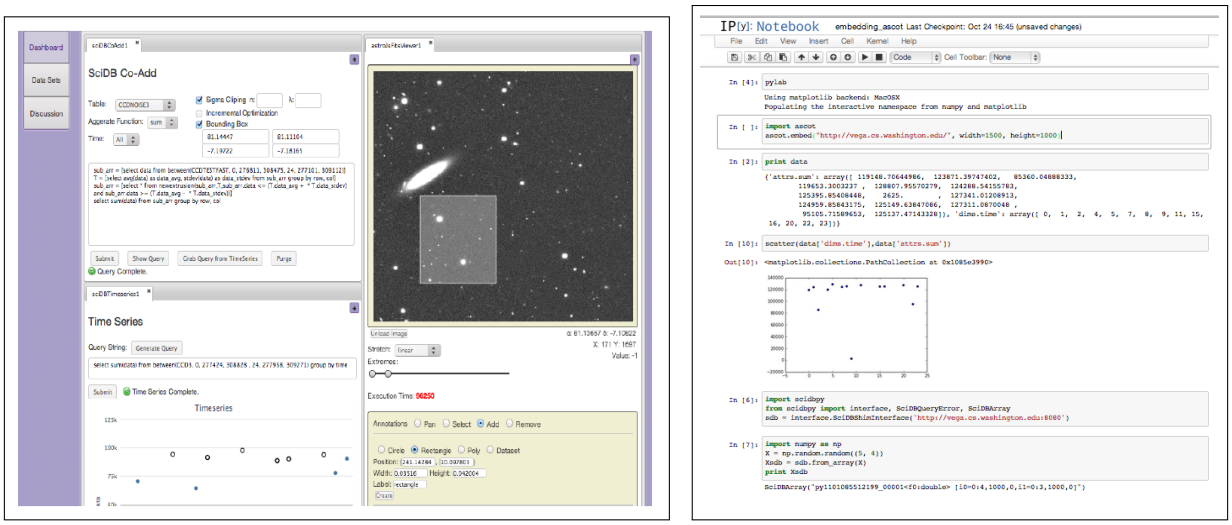
spherical data in SciDB (Section 3). Because it focuses on producing a tool that is transformative and accepted by scientists, AscotDB illustrates how necessary technologies must be integrated and extended.

2 AscotDB

The basic operations astronomers perform on images fall into two categories: detection and measurement. Detection is the process of identifying the location of individual sources in an image, where the sources might be stationary objects such as individual stars and galaxies, or moving/transient objects such as asteroids or supernovae. Measurement involves computing well-calibrated statistics from the light of the individual objects: for example, computing the total optical flux from a star through a detailed model of its response across the CCD pixels. AscotDB provides the user with two modes of interaction with the data: (1) Visual interaction with Ascot gadgets that we describe in Section 2.1 – these visual interactions are an important component of the detection process; and (2) an IPython interface for programmatic interaction for both detection and measurement that we describe in Section 2.2. Importantly, AscotDB integrates the two modes of interaction for a seamless experience (Section 2.3). The overall architecture of AscotDB and the high level interaction between components is diagrammed in Fig. 1.

2.1 Graphical Front-End Support

Figure 2(a) shows a screenshot of AscotDB's graphical interface. AscotDB retains important Ascot features including its extensibility through the addition of new gadgets and its sharing capabilities across users. AscotDB, however, radically transforms Ascot's data analysis capabilities. Originally, Ascot enabled users to view a *single* telescope image at a time and overlay on the image catalog data extracted from a back-end relational DBMS. In contrast, AscotDB enables users to manipulate raw pixel data. For example, users can *stack* images of the sky that fall within a region, *clean them* using an iterative process, re-run a source detection algorithm, annotate interesting sources, and generate visual summaries (e.g., light curves) before initiating a measurement process. To support these novel operations, AscotDB stores the telescope image data inside SciDB and translates operations on the interface into queries over SciDB's arrays. Because scientists are technically savvy users and because graphical interfaces can inaccurately capture a user's intent, AscotDB always shows the queries that it generates and enables a user to modify them before executing them. While the general integration of Ascot with SciDB is mostly an engineering project, ensuring high-performance required important extensions to the SciDB engine. In particular, several operations including source detection and data cleaning are iterative in nature. To support these iterative tasks at interactive speeds, we extended SciDB with native support for array iterations, described in a recent system demonstration [10]. The support includes the execution of such iterations in shared-nothing clusters and various optimizations such as incremental processing. Figure 3 illustrates the performance of SciDB



(a) Graphical front-end of AscotDB.

(b) Programmatical front-end of AscotDB.

Figure 2: Two modes of interaction with AscotDB: visual interface and IPython interface. The visual interface is embedded in IPython, but here is shown separately for purposes of illustration.

with *incremental iterative processing* extensions on an application called "Sigma clipping"[2]. Incremental iterative processing is based on the idea that the output at each iteration differs only partially from the output at the previous iteration. Performance can thus significantly improve if the system computes, at each iteration, only the part of the output that changes rather than re-computing the entire result.

A second important design decision in AscotDB's graphical front-end was to strike a balance between a tool specialized for one analysis task and a general-purpose system such as Tableau [17]. For AscotDB, we opted to develop one gadget per high-level activity (*e.g.*, data cleaning, image stacking, and time-series extraction) since the community typically performs a small set of such activities. AscotDB enables users to assemble these gadgets and thus activities in arbitrary combinations.

2.2 Front-End Python Support

Python has seen significant uptake among astronomers and other domain scientists because it is open-source, cross-platform, and easy for non-programmers to learn. Furthermore, third-party packages such as Numpy for array-oriented computing, Scipy for fast scientific algorithms, Matplotlib for publication-quality visualization, and IPython for interactive computation and data exploration provide a foundation for a host of more specialized packages implementing algorithms used in a wide variety of scientific fields.

For this reason, an intuitive Python interface to SciDB is essential to make it as useful as possible for the astronomical data analysis task. While SciDB does provide a low-level Python API for the execution of Array Query Language (AQL) and Array Functional Language (AFL) queries, the interface is too opaque for it to be useful to most astronomers. For this reason, we have created the SciDB-Py[3] package, which provides a high-level Python interface to SciDB, designed with an API familiar to users of the NumPy array computing library. Rather than interacting with data arrays via a query language like AQL or AFL, SciDB-py allows users to express operations in a high-level scripted manner, and thus makes the computing power of SciDB more accessible to data scientists.

[2] http://db.cs.washington.edu/myria/repository/uw-cat.html

[3] http://jakevdp.github.io/SciDB-Py

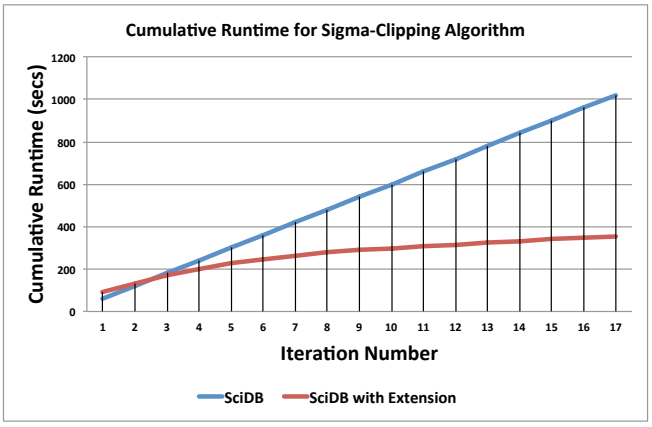


Figure 3: Cumulative runtime of an iterative data cleaning algorithm called *Sigma-Clipping* on SciDB (single machine) with and without *incremental iterative processing* extensions. 3D input array comprises 66 million cells.

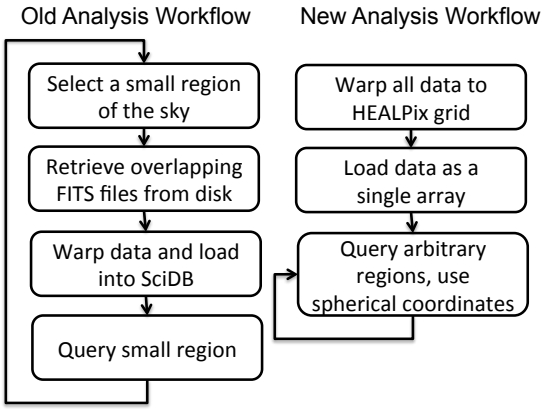


Figure 4: Data analysis workflow with and without our spherical-coordinates extension. In the old workflow, even if one caches the output of warping small regions, each region must be warped multiple times. Additionally, queries can never span large regions.

The following shows a short SciDB-Py session along with the equivalent SciDB AFL query commands:

SciDB-Py Python Code:

```
# Interface to a scidb cluster on the local machine via Shim
from scidbpy import interface
sdb = interface.SciDBShimInterface('http://localhost:8080')

# Create a random array with 1000 rows & 5 columns:
X = sdb.random((1000, 5))

# Compute the mean and standard deviation of each column
xcolmean = X.mean(0)
xcolstd = X.std(0)

# center and normalize each column
XC = (X - xcolmean) / xcolstd
```

Equivalent AFL session

```
/* The following are the AFL queries generated and
   executed automatically by the scidbpy module */

store(build(<f0:double> [i0=0:999,1000,0,i1=0:4,1000,0],
            random()/2147483647.0), X);

store(aggregate(X, avg(f0), i1), Xcolmean);
store(aggregate(X, stdev(f0), i1), Xcolstd);

store(project(apply(cross_join(X, Xcolmean, X.i1, Xcolmean.i1),
                    x, X.f0 - Xcolmean.f0_avg), x), tmp);
store(project(apply(cross_join(tmp, Xcolstd, tmp.i1, Xcolstd.i1),
                    x_0, tmp.x/Xcolstd.f0_stdev), x_0), XC);
```

The operations specified in this script are executed entirely within the SciDB architecture through the automatically generated AFL queries; the actual AFL queries are transparent to the user, and the Python interpreter itself never sees the data. By providing such an intuitive, high-level wrapper around the efficient storage and operations available in SciDB, SciDB-py makes the power of SciDB accessible to the average scientific Python user.

The Python interface to AscotDB is provided via the IPython notebook as illustrated in Figure 2(b). IPython is a set of tools designed to facilitate the entire life-cycle of a scientific project, from data exploration to publication. One component is a browser-based *notebook* with the support for editing and running Python code, as well as rich objects such as embedded plots, html objects, and mathematical expressions. Integrating the interactive, visual scientific computing environment of IPython with the power of the SciDB-Py interface within the AscotDB environment enables seamless sharing and processing of large astronomical datasets.

2.3 Graphical Interface and Python Integration

An important feature of AscotDB is the integration of both the graphical and python modes of interaction such that the user can go seamlessly back and forth between them. The main question to be answered is how to keep the working data in both modes synchronized. One possibility is to keep track of all the actions in the Ascot

front-end in the form of SciDB queries and, after switching the mode to Python, have the system run the same queries in the same order in the background. This approach makes it easier to track the lineage of data and thus facilitates reproducibility. However, a session where a user interacts with a graphical engine can be long, can contain a large number of queries, and can dwarf the remaining Python script. Another approach is to move minimal amounts of data between the two interfaces to keep the working data synchronized. In AscotDB, we chose to synchronize the two interfaces by capturing user queries because of this approach's lineage tracking and reproducibility properties. To address the problem of large query sessions, however, in the AscotDB system, we are experimenting with a variety of techniques to automatically extract minimal query sets, which produce the results from a visual analysis session. Our key aim is to minimize the number of queries but without re-ordering them nor combining them into more complex and difficult-to-understand queries. It is critical for the user to identify in the summary script all the key steps that he or she took during the visual exploration.

3 Spherical Coordinates

While SciDB provides an efficient environment for storing and manipulating very large array-oriented data, the arrays must be Cartesian: that is, data on a simple N-dimensional rectangular grid. In many scientific fields, especially those relying on astrophysical and geophysical data, the data lie on a sphere and this assumption breaks down.

In the case of a full-sky multiply-imaged optical dataset like LSST, this makes the global analysis of imaging data within SciDB very difficult. The data consists of individual images covering ~ 10 square degrees, which are taken at different times and cover partially-overlapping regions of the sky. Software tools exist that can *warp* overlapping images to a uniform tangential projection for the purposes of image stacking and differencing, but the distortion characteristics of a tangential projection are such that a single warping is sufficient over only a small region of sky. The result is that each individual exposure must be partially warped on the order of four separate times in order to analyze a full sky's worth of data. With images stored as binary files on disk, there is significant overhead in data I/O for these multiple warpings and coadditions. Fig. 4 illustrates this workflow in contrast with the new one that our approach enables.

To avoid this overhead both in performance and data management complexity, we add a middleware layer to SciDB that enables direct operations on spherical data, with the ultimate goal of pushing these middleware extensions into SciDB. This task boils down to the classic problem of full-sphere projections. Cartographers over the centuries have invented schemes for representing the earth's surface as a flat map; the storage of spherical data within SciDB requires an analogous flattening. Many such partitions have been proposed and used in the literature, each with distinct advantages and disadvantages.

The *Hierarchical Equal Area isoLatitude Pixelization* (HEALPix) [6] spherical pixelization is commonly used in astronomy, especially for datasets which cover a large portion of the sky. This approach has three key scientific benefits. First, it produces pixels of equal area, which significantly facilitates essential operations based on integration, such as kernel-smoothing, convolution, and regridding. Second, it can be represented in a number of useful ways, including a hierarchical encoding that facilitates spatial queries such as neighbor-searches, a ring encoding which facilitates efficient computations of the fast spherical harmonic transform (SpHT), and a *double-pixellization*, outlined below, which allows us to take advantage of the efficient array processing primitives already implemented within SciDB.

HEALPix is based on a tesselation of twelve equal-area partitions of the sphere, each of which are hierarchically sub-divided in a quad-tree structure to attain arbitrarily precise angular resolution. The twelve fundamental partitions are chosen so as to minimize the distortion within any single sub-pixel, and oriented so as to align sub-pixels along lines of constant latitude (vital for SpHT computations). The resolution of a HEALPix map is governed by the number of levels in the pixel hierarchy, specified via the parameter $N_{\rm side} = 2^\ell$, with $\ell = 0, 1, 2....$ The number of pixels is $N_{\rm pix} = 12N_{\rm side}^2$, and it follows that the angular size of each pixel is

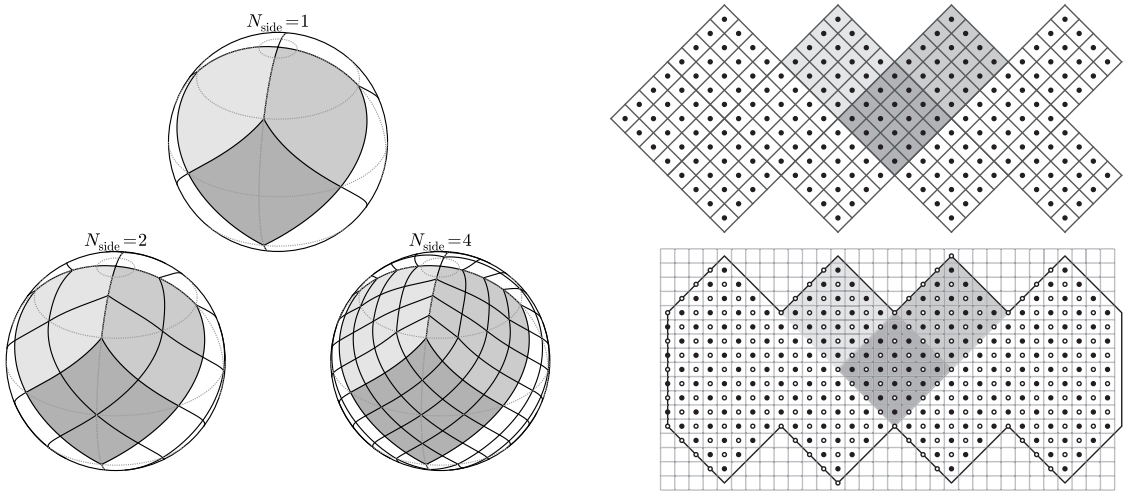


Figure 5: (Left) First three levels of the HEALPix hierarchical pixelization, shown in Orthographic projection. (Right) Schematic showing the construction of the HEALPix double-pixelization scheme for $N_{\rm side} = 4$

$\Omega_{\rm pix} = (\pi/3)N_{\rm side}^{-2}$. A visualization of the first three levels of HEALPix partitioning is shown in Fig. 5(left).

3.1 Representation and Transformations

While the equal-area hierarchical layout discussed above has advantages, one disadvantage is that it cannot be represented easily as a Cartesian grid of pixels. While the subdivisions within each of the 12 fundamental partitions do form a warped 2-dimensional grid, the junction of the three shaded regions in Fig. 5(left), for example, shows that the boundaries between the twelve fundamental partitions do not fit this Cartesian model.

We choose to address this using the *double pixellization* scheme outlined in [3]. As shown in the upper panel of Fig. 5(right), if the twelve fundamental partitions can be unwrapped onto the plane in a modified cylindrical equal-area projection (as specified in [6]), the pixels are arranged in a partially-filled regular grid, rotated 45° with respect to the primary x, y axes. Additionally, in this orientation, each valid (x, y) pair has an invertible mapping to a (latitude, longitude) pair (see details in [3, 6]), easing the ability to convert between the HEALPix representation and the standard world coordinate system.

The key insight for the double pixelization is that by interposing a new pixel between each of the standard HEALPix pixels, we can recover a grid with a standard orientation which preserves nearly all of the desirable properties of the original HEALPix pixelization, as shown in Fig. 5(right). Here, the solid points show the pixel centers from the standard pixelization, while the open points are the pixel centers added to construct the oriented grid. One disadvantage of this re-orientation is the fact that eight of the pixels (those located at the concave edges) now represent only $3/4$ the area of the other pixels: the missing area is accounted for by the addition of a new pixel at each pole. Thus the number of pixels in the double-pixelization scheme is $24N_{\rm side}^2 + 2$, compared to the $12N_{\rm side}^2$ pixels of the standard pixelization. One other disadvantage of the double pixelization is that regridding between HEALPix representations of different resolutions (i.e. changing $N_{\rm side}$) becomes much less straightforward. Note that on this grid, the pixels have a filling-factor of $\sim 75\%$, and at any desired resolution this can be efficiently stored within SciDB as a sparse Cartesian array.

The transformations needed to map a variety of image-plane coordinate representations onto HEALPix coordinates are defined via the WCS standard [2], which is implemented in standard Python tools such as *astropy*[4]. Astronomical image data typically comes in the form of *Flexible Image Transfer System* (FITS) files, whose

[4]"A Community Python Library for Astronomy" `http://astropy.org`

headers contain image projection parameters within the FITS/WCS standard. Because of this, the process for loading images onto the HEALPix grid is generic enough to be used for any modern astronomical data. Note that any individual image (or set of images at a given time) will fill only a small subset of the entire field: in SciDB, this can be represented as a sparse two-dimensional array.

There are two potential approaches warping observed data to the HEALPix grid: it can be performed as a preprocessing step prior to loading the data into the database, or as an operation within the database itself.

The first approach is to pre-warp images in an analog to the typical approach. Images, however, are now pre-projected onto a super-resolution grid of HEALPix pixels rather than a local tangential projection. A suitable super-resolution warp allows the precise pixel overlap to be treated as a second-order effect. While the typical approach warps each image to one of a number of local tangential representations, the new spherical approach requires each image to be warped only once onto a global HEALPix representation, cutting computation and data access costs up to about 75% for the warping calculation. The final dataset is represented in SciDB as a sparse 3-dimensional array, containing HEALPix-gridded two-dimensional images indexed by the time at which they were taken. Fig. 4(right) illustrates the workflow when using this method.

The second approach pushes the warping step into SciDB itself, obviating the need for preprocessing outside SciDB. By choosing an extremely fine resolution for the raw data within SciDB, the flux within each input image pixel can be stored at the location of the pixel center with nearly arbitrary resolution[5], leaving all other pixels empty. The result is an extremely sparse data structure which efficiently indexes the actual pixel-level data, without need for any preprocessing step. The translation of this sparsely-indexed data into the warped dense HEALPix representation is a well-defined operation rooted in 2D spatial convolution. Because SciDB does not currently implement efficient convolutions, we chose the first approach, leaving the second option for future work.

3.2 Operations

Once the data is warped and stored on a HEALPix grid within SciDB, the basic class of desired operations come virtually for free.

- **Selection:** SciDB implements efficient selection along the axes of the stored array. This allows efficient selection of data at a given time, as well as angular selection on the sky. Because there is a straightforward algebraic relationship between typical local sky queries and the HEALPix double-pixel coordinates, desired data selection can be implemented as a native SciDB query.
- **Aggregation:** In the same way, built-in SciDB aggregates can be used to compute statistics of interest, most notably the *co-addition* of sub-images across time.
- **Iterations:** Many essential image processing operations, from trimmed co-addition to PSF matching, require an iterative approach to the data. As discussed in Section 2.1, we extended SciDB to enable efficient iterative operations on large datasets. This work can be used with the spherical projections to allow powerful iterative analysis of full-sky datasets.
- **Convolution:** Many image processing tasks require efficient convolutions, from warping to object detection and tracking to PSF matching for image coaddition. Though SciDB currently implements a rudimentary windowing convolution, it does not contain the full convolution operator required for these tasks. Implementation of efficient convolutions is an important area of future work, as it will allow a wide variety of image processing tasks to be implemented in a scalable way within SciDB.
- **Regridding:** With the HEALPix double-pixelization scheme, regridding can be implemented using a convolution operation, with some extra care at the boundaries.

[5]Technically, due to the int64 index labels, the resolution is limited to $\gtrsim 0.1$ pico-arcsec, which is more than sufficient for any current astronomical observation.

Because the translation between spherical coordinates and SciDB array coordinates is negligible in time, the performance of the above operations is equal to the performance of the underlying SciDB queries.

4 Related Work

Because the AscotDB project is so broad in its aims and the array of technologies it utilizes, there is related work in many areas.

EXTASCID [4] is an extensible parallel system that provides native support both for arrays and key-value relations. The execution strategy in EXTASCID is through the UDA interface with easy reasoning for parallelism, while AscotDB provides parallelism through SciDB with native support for arrays only. On the other hand, AscotDB provides support for iterations and direct query on spherical data with two modes of interaction: visual front-end and Python programmatic front-end.

AstroShelf [11], similar to AscotDB, is a collaborative system that enables astrophysicists to investigate celestial objects using catalog data hosted at different sites. Astroshelf is a stream processing system that matches events (either celestial events or user annotations) to users who are likely to be interested in them. In contrast, AstroDB focuses on interactive, collaborative processing of archived data.

Blaze [1], often billed as the "Next generation NumPy", aims to implement a very general array framework within Python. It will support a wide variety of table and array-like structures capable of handling arbitrary local and distributed memory layouts, type heterogeneity, axis labels, missing or masked values, and other commonly-requested features. SciDB is just one of a wide variety of potential backends for large-scale distributed array storage and computing through Blaze: in this light, SciDB-Py can be considered a precursor to the much more ambitious framework Blaze promises to implement.

For spherical partitioning of datasets, one well-studied method is the Hierarchical Triangular Mesh (HTM) [16], used in the SDSS SkyServer. HTM is efficient, but suffers two disadvantages compared to the current work: First, its triangular representation does not lend itself to the Cartesian representation required to take advantage of operations within SciDB. Second, its non-equal pixel areas add overhead to integration-based tasks common in image manipulation and processing.

5 Conclusion

Efficient data exploration, visualization, and analysis in the context of future large astronomical surveys will require a combination of advances in the areas of large-scale data storage, processing, and visualization, as well as specialized operations suitable for data on the sphere of the sky. We have presented one set of approaches to this problem: AscotDB. We utilize a HEALPix-based spherical projection to unravel spherical data and store it within a three-dimensional SciDB array, indexed both by time and by location on the sphere. We choose the representation carefully so as to utilize the full functionality of the native array-processing power of SciDB. With the current capabilities of SciDB, this approach already decreases the image pre-processing load by around a factor of four; with future work on a native convolution operator within SciDB, the need for an out-of-database preprocessing will be eliminated. The work here, along with future directions of development in SciDB, points to a system where a full-sky worth of time-domain astronomy imaging data can be directly stored and indexed in a way that will enable efficient image analysis tasks to be performed on-demand and at-scale.

Acknowledgments: This work is partially supported by grants from the LSST corporation, NASA grant NNX09AK84G, DOE award DESC0002607, NSF grants ACI-1226371 and IIS-1110370, and the Intel Science and Technology Center for Big Data.

References

[1] Blaze: A python compiler for big data. http://continuum.io/blog/blaze.

[2] M. R. Calabretta and E. W. Greisen. Representations of celestial coordinates in FITS. *A&A*, 395:1077–1122, December 2002.

[3] M. R. Calabretta and B. F. Roukema. Mapping on the HEALPix grid. *MNRAS*, 381:865–872, October 2007.

[4] Yu Cheng and F. Rusu. Astronomical data processing in extascid. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, SSDBM, pages 47:1–47:4, 2013.

[5] Cudre-Mauroux et. al. A demonstration of scidb: a science-oriented dbms. In *Proc. of the 35th VLDB Conf.*, pages 1534–1537, 2009.

[6] K. M. Górski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann. HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere. *ApJ*, 622:759–771, April 2005.

[7] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[8] Large Synoptic Survey Telescope. http://www.lsst.org/.

[9] D. Marcos, A. J. Connolly, K. S. Krughoff, I. Smith, and S. C. Wallace. ASCOT: A Collaborative Platform for the Virtual Observatory. In *Astronomical Data Analysis Software and Systems XXI*, volume 461 of *Astronomical Society of the Pacific Conference Series*, page 901, 2012.

[10] M. Moyers, E. Soroush, S.C. Wallace, S. Krughoff, J. Vanderplas, M. Balazinska, and A Connolly. A demonstration of iterative parallel array processing in support of telescope image analysis. In *Proc. of the 39th Int. Conf. on Very Large DataBases (VLDB)*, 2013.

[11] P. Neophytou, R. Gheorghiu, R. Hachey, T. Luciani, D. Bao, A. Labrinidis, E. G. Marai, and P. K. Chrysanthis. Astroshelf: understanding the universe through scalable navigation of a galaxy of annotations. In *SIGMOD'12: Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 713–716, 2012.

[12] Travis E. Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007.

[13] Fernando Pérez and Brian E. Granger. IPython: a System for Interactive Scientific Computing. *Comput. Sci. Eng.*, 9(3):21–29, May 2007.

[14] J. Rogers et al. Overview of SciDB: Large scale array storage, processing and analysis. In *Proc. of the SIGMOD Conf.*, 2010.

[15] E. Soroush, M. Balazinska, and D. Wang. ArrayStore: A storage manager for complex parallel array processing. In *Proc. of the SIGMOD Conf.*, pages 253–264, June 2011.

[16] A. S. Szalay, J. Gray, G. Fekete, P. Z. Kunszt, P. Kukol, and A. Thakar. Indexing the Sphere with the Hierarchical Triangular Mesh. *eprint arXiv:cs/0701164*, January 2007.

[17] Tableau. http://tableausoftware.com.

SciDB DBMS Research at M.I.T.

Michael Stonebraker[1], Jennie Duggan[1], Leilani Battle[1], Olga Papaemmanouil[2]
[1] Computer Science and Artificial Intelligence Laboratory, MIT
[2] Department of Computer Science, Brandeis University
{stonebraker, jennie, leilani}@csail.mit.edu, olga@cs.brandeis.edu

Abstract

This paper presents a snapshot of some of our scientific DBMS research at M.I.T. as part of the Intel Science and Technology Center on Big Data. We focus our efforts primarily on SciDB, although some of our work can be used for any backend DBMS. We summarize our work on making SciDB elastic, providing skew-aware join strategies, and producing scalable visualizations of scientific data.

1 Introduction

In [19] we presented a description of SciDB, an array-based parallel DBMS oriented toward science applications. In that paper we described the tenets on which the system is constructed, the early use cases where it has found acceptance, and the state of the software at the time of publication. In this paper, we consider a collection of research topics that we are investigating at M.I.T. as part of the Intel Science and Technology Center on Big Data [20]. We begin in Section 2 with the salient characteristics of science data that guide our explorations. We then consider algorithms for making a science DBMS elastic, a topic we cover in Section 3. Then, we turn in Section 4 to query processing algorithms appropriate for science DBMS applications. Lastly, in Section 5 we discuss our work on producing a scalable visualization system for science applications.

2 Characteristics of Science DBMS Applications

In this section we detail some of the characteristics of science applications that guide our explorations, specifically an array data model, variable density of data, skew, and the need for visualization.

Array Data Model Science data often does not fit easily into a relational model of data. For example, Earth Science data [18] often comes from satellite imagery and is fundamentally array-oriented. Astronomy telescopes are effectively large digital cameras producing pixel arrays. Downstream, the data is processed into a 2-D or 3-D coordinate system, which is usually best modeled as an array. Moreover, it is often important to keep track of time, as objects are recorded over days or weeks of observations; time is simply another dimension in an array-based system, but must be glued on to a relational model of data.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Field observations are invariably spatially-oriented and queried in a 3-D space (e.g., latitude, longitude, and time). Often additional dimensions are present (e.g., elevation). Searching in a high dimensional space is natural and fast in a multi-dimensional array data model, but often slow and awkward in a relational data model.

Lastly, much of the analytics that scientists run are specified on arrays, for example k-nearest neighbors, spatial smoothing, fourier transforms, and eigenvectors. These are naturally executed in an array DBMS, but require relational data to be cast into arrays for execution. For these reasons, we believe that array DBMSs are a natural fit for science data, and our research has focused in this area.

Sparse or Dense Array Some arrays have a value for every cell in an array structure. For example, it is common practice to "cook" satellite imagery into a dense array, where each cell represents a tile on the earth surface and the cell value is some interpolation over a time period of the actual collected imagery data. For example, one common interpolation is to select the cell value from the possible ones, which has the least cloud cover. This interpolation produces a very large, dense array. On the other hand, the raw satellite imagery is recorded in a three-dimensional space, for example (latitude, longitude, time), or a four-dimensional space (latitude, longitude, altitude, time). Sometimes spherical coordinates are used. In any case, the recorded data is very sparse. Our experience is that an array DBMS must be prepared to cope with data that ranges from very sparse to very dense.

Skewed Data If we imagine a database consisting of the position of each resident of the United States, then the density of points in Manhattan is 10^5 greater than the density in Montana. This sort of skew, whereby some regions of array space have substantially more data than others, is very common in science applications. In astronomy, there are regions of the sky that are more interesting than others, and the telescope is pointed there more often. In a database of ship positions, the vessels congregate in and around major ports waiting to load or unload. In general, our experience is that moderate to high skew is present in most science applications.

Visualization Focus In business data processing, a form-based interface is exceedingly popular. For example, to look up pending airline reservations, one inputs one's frequent flyer number into a form. To find the balance in one's checking account, one enters the account number, and so forth. Scientists rarely want this sort of form-based user interface. Instead, they usually want a visualization system through which they can browse and inspect substantial amounts of data of interest. For example, one Earth Science group is interested in snow cover in the Sierra Nevada mountains. Hence, they want to "fly over" the study area, browsing satellite imagery and then zoom into areas of interest. Therefore, the front end issuing queries to a science database is often a visualization system.

3 Elasticity

Given that scientists never want to discard data, this leads to a "grow only" data store. Also, the amount of data that they want to process often increases with time, as they collect more data from experiments or sensors. Hence, a science DBMS should support both data elasticity and processing elasticity. Of course, elasticity should be accomplished without extensive down time; in the best of all worlds, it is accomplished in background without incurring any downtime. Often science data is loaded an experiment-at-a-time or a day-at-a-time, i.e., periodically. In between load events, it is queried by scientists.

We have constructed a model of this desired elasticity behavior. It consists of three phases, a loading phase where additional data in ingested, followed by a possible reorganization phase, followed by a query phase whereby users study the data. These phases repeat indefinitely, and the job of an elasticity system is three fold:

- predict when resources will be exhausted
- take corrective action to add another quanta of storage and processing
- reorganize the database onto the extra node(s) to optimize future processing of the query load

Our model specifies a cost function for these three activities that minimizes a combination of provisioned resources and time elapsed in querying, data insertion, and incremental reorganization. We use this model to study the impact of several data partitioners on array database performance in terms of workload latency. Our elastic partitioners are designed to efficiently balance storage load, while minimizing reorganization time during cluster expansion operations.

We have implemented this model for SciDB, and the details can be found in [14]. In addition, we have studied the behavior of the model on two different use cases. First, we have explored a MODIS satellite imagery database [1], with appropriate queries primarily from [18]. This data set is large, sparse and uniform, as the satellite covers each portion of the earth's surface at the same rate. In addition, we have explored a U.S. Coast Guard database of ship positions, AIS [7], which are highly skewed as noted above.

The query phase of our workload is derived from real use cases, and each has a mix of select-project-join (SPJ) and domain-specific science queries, many of which are spatial. The SPJ benchmarks have three components: selection, a distributed sort, and an equi-join, and these queries capture simple relational transformations on the data. Our science benchmarks are customized to each use case. AIS executes a k-nearest neighbor query, studying ship density patterns, a ship collision detector, and a regridding from detailed lat/long space to an aggregated n-dimensional projection. Our MODIS evaluation includes a k-means clustering, used to model rainforest deforestation, a windowed aggregate to generate a smoothed image of the satellite's recordings, and a rolling average of measurements for environmental monitoring. We detail our benchmark queries in [1].

3.1 Elastic Array Partitioning

Well-designed data placement is essential for efficiently managing an elastic, scientific database cluster. A good partitioner balances the storage load evenly among its nodes, while minimizing the cost of redistributing chunks as the cluster expands. In this section, we visit several algorithms to manage the distribution of a growing collection of data on a shared nothing cluster. In each case we assume an array is divided into storage "chunks", specified by a "stride" in a subset of the array dimensions. Moreover, every array is assumed to have a time dimension, where the insertion time of values is recorded. Obviously, this dimension increases monotonically.

Elastic array partitioners are designed to incrementally reorganize an array's storage, moving only the data necessary to rebalance storage load. This is in contrast to *global* approaches, such as hash partitioning. The classic hash partitioner applies a function to each chunk, producing an integer, and this hash value, modulus the number of cluster nodes, assigns chunks to nodes. Using this technique will move most or all of the data at each redistribution, a high cost for regularly expanding databases.

Elastic and global partitioners expose an interesting trade off between locally and globally optimal partitioning plans. Most global partitioners guarantee that an equal number of chunks will be assigned to each node, however they do so with a high reorganization cost, since they shift data among most or all of the cluster nodes. In addition, this class of approaches are not skew-aware; they only reason about logical chunks, rather than physical storage size. Elastic data placement dynamically revises how chunks are assigned to nodes in an expanding cluster, and also makes efficient use of network bandwidth, because data moves between a small subset of nodes in the cluster. Note that skewed data will have significant variance in the stored chunk sizes. Hence, when a reorganization is required, elastic partitioners identify the most heavily loaded nodes and split them, passing on approximately half of their contents to new cluster additions. This rebalancing is skew resistant, as it evaluates where to split the data's partitioning tables based on the storage footprint on each host.

In this work, we evaluate a variety of range and hash partitioners. Range partitioning stores arrays clustered in dimension space, which expedites group-by aggregate queries, and ones that access data contiguously, as is common in linear algebra. Also, many science workloads query data spatially, and benefit greatly from preserving the spatial ordering of their inputs. Hash partitioning is well-suited for fine-grained storage partitioning, because it places chunks one at a time, rather than having to subdivide planes in array space. Hence, equi-joins and most "embarrassingly parallel" operations are best served by hash partitioning.

3.2 Hash Partitioning

In this section, we discuss two elastic hash partitioners. The first, *Extendible Hash*, is optimized for skewed data, and the alternative, *Consistent Hash*, targets arrays with chunks of uniform size. Both algorithms assign chunks to nodes one at a time. Each chunk is numbered 1...k based on its position within the source array, and the engine hashes the chunk number to find its location.

Extendible Hash [15] is designed for distributing skewed data. The algorithm begins with a set of hash buckets, one per node. When the cluster increases in size, the partitioner splits the hash bucket of the most heavily loaded hosts, partially redistributing their contents to the new nodes. For data that is evenly distributed throughout an array, Consistent Hash [16] is a beneficial partitioning strategy. Think of the hash map distributed around the circumference of a circle, where both nodes and chunks are hashed to an integer, which designates their position on the circle's edge. The partitioner finds a chunk's destination node by tracing the circle's edge from the hashed position of the chunk in the clockwise direction, assigning it to the first node that it finds. When a new node is inserted, it accepts chunks from several pre-existing nodes, producing a partitioning layout with an approximately equal number of chunks per node. This algorithm assigns the logical chunks evenly over the cluster, however, it does not, address storage skew, because the chunk-to-node assignments are made independent of individual chunk sizes.

3.3 Range Partitioning

Range partitioning has the best performance for queries that have clustered data access, such as grouping by a dimension or finding the k-nearest neighbors for a cell. In this section, we examine three strategies for clustered data partitioning, n-dimensional (K-d Tree and Uniform Range), and time-based (Append).

A *K-d Tree* [12] is an efficient strategy for range partitioning skewed, multidimensional data. The K-d Tree stores its partitioning table as a binary tree. Each node is represented by a leaf, and all non-leaf nodes are partitioning points in the array's space. To locate a chunk, the algorithm traverses the tree, beginning at the root node. If the root is a non-leaf node, the partitioner compares the chunk's first logical coordinate to the node's split point, progressing to the child node on the chunk's side of the divide. The lookup continues until it reaches a leaf node, completing this operation in logarithmic time.

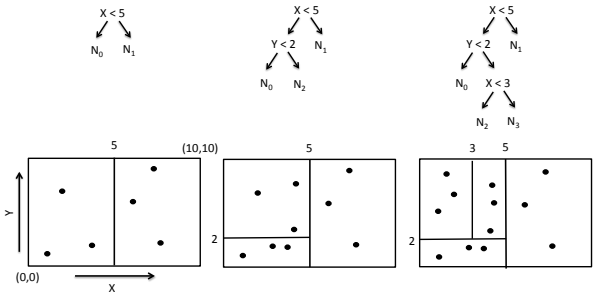


Figure 1: An example of K-d Tree array partitioning.

In this scheme, each host is responsible for an n-dimensional subarray, and partitioning points are defined as planes in array space. When a new cluster node is added, the algorithm first identifies the most heavily loaded host. If this is the first time that the host's range has been subdivided, the partitioner traverses the array's first dimension until it finds the point where there exists an equal number of cells on either side of it, the dimension's median. The splitter cuts the hot host's range at this point, reassigning half of its contents to the new addition. On

subsequent splits, the partitioner cycles through the array's dimensions, such that each is cut an approximately equal number of times.

Figure 1 demonstrates a K-d Tree that begins by partitioning an array over two nodes; it is divided on the x-axis at the dimension's midway point, 5. The left hand side accumulates cells at a faster rate than the right, prompting the partitioner to cut its y-axis for the second split, where this dimension equals 2. Next, the K-d Tree returns to cutting vertically as the third node joins the cluster.

A second variation, *Uniform Range*, optimizes for unskewed arrays. In this approach the array assigns an equal number of chunks to each node, and it executes a complicated global reorganization at every cluster expansion to maintain this balance. This algorithm starts by constructing a tall, balanced binary tree to describe the array's dimension space. If the partitioner has a height of h, then it has $l = 2^h$ leaf nodes, where l is much greater than the anticipated cluster size. Each non-leaf node in the tree specifies a split point, where a dimension is divided in half, and the tree rotates through all dimensions, subdividing each with an equal frequency. For a cluster comprised of n hosts, Uniform Range assigns its l leaf nodes in groups of size $\frac{l}{n}$, where the leaves are sorted by their traversal order in the tree. When the cluster scales out, this tree is rebalanced by calculating a new $\frac{l}{n}$ slice for each host; hence the partitioner maintains multidimensional clustered array storage, without compromising load balancing. This approach has a high reorganization cost compared to K-d Tree, because such operations move most or all of the array at each rebalancing.

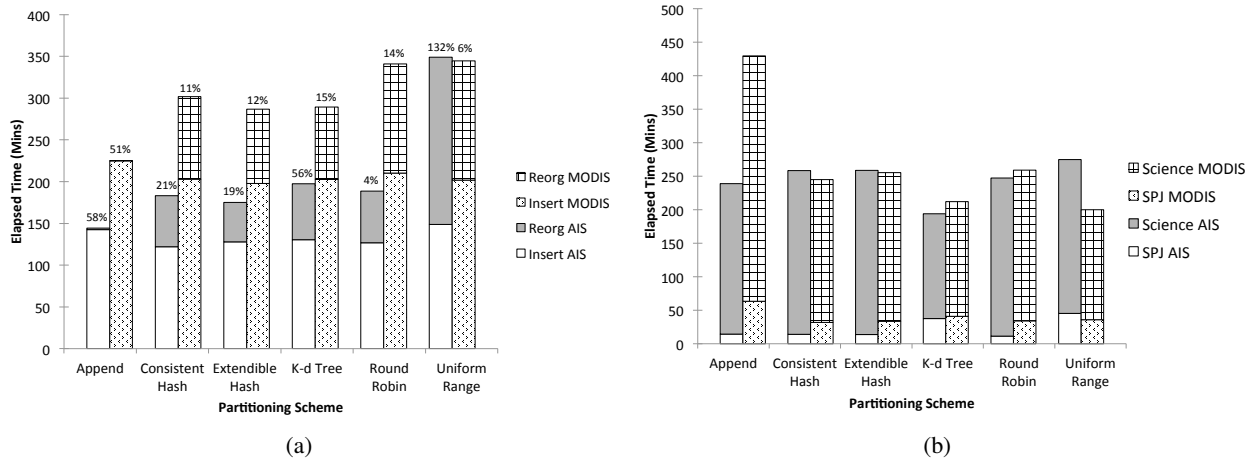
A third variant of range partitioning is an *Append* strategy. Append subdivides the no-overwrite array on its time dimension alone, by sending each newly inserted chunk to the first node that is not at capacity. A coordinator maintains a count of the storage allocated at each node, spilling over to the next one when the current one is full. This partitioner works equally well for skewed and uniform data distributions, as it adjusts its layout based on storage size, rather than logical chunk count. Append partitioning is attractive because it has minimal overhead for data reorganizations. When a node is added, it stores new chunks when its predecessor becomes full, making it an efficient option for a frequently expanding cluster. On the other hand, this partitioner has poor performance if the cluster adds many nodes at once, since it will use only one new node at a time.

To recap, we have studied a collection of elastic partitioning algorithms. All, except Append, are designed for either a skewed or uniform data distribution. The schemes (excluding Uniform Range) are all incremental; hence they move a subset of the chunks during a scale out operation, writing only to new nodes. In the next section we compare these schemes with a baseline strategy of *Round Robin* allocation. Round Robin assigns nodes to chunks circularly based on chunk number. Hence, if chunk n is being assigned to a cluster of k nodes, this approach will send it to node n modulus k. The baseline evenly distributes the logical chunk numbers over all nodes, however when the cluster expands, all hosts usually shift their data to add one or more hosts to the partitioning rotation.

3.4 Elastic Partitioner Results

We studied elastic partitioning on a cluster starting with two nodes, which expands in increments of two nodes. Our MODIS case study consists of adding 630 GB to an empty database over 14 days in 1 day increments. In addition, we experimented with adding 400 GB of AIS ship data spanning 3 years, inserted in 36 batches, each covering one month. In both cases, this is the rate at which we receive data from its source.

Figure 2(a) demonstrates the performance of the various partitioning schemes on our two benchmarks during the ingest and reorganization phases. For both of our use cases, the insert time is nearly constant cost because all of the schemes load the data and then spread it over the cluster according to the partitioning scheme under evaluation. The cost of redistribution during the three expansions is less uniform. Append is a clear winner in this space, as it does not rebalance the data; it only shifts future additions to the newly provisioned nodes. K-d Tree and hash partitioning both perform well, as they incrementally reorganize the data by writing only to the newly provisioned nodes. Round Robin and Uniform Range globally redistribute the data, and hence have a higher time requirement.



(a) (b)

Figure 2: (a) Elastic partitioner performance for data ingest and reorganization. Percentages denote relative standard deviation of storage distribution. (b) Benchmark performance of elastic partitioning schemes.

We assess the evenness of a scheme's storage distribution by the relative standard deviation (RSD) of each host's load, and the percentages are shown in Figure 2(a). After each insert, this metric analyzes the database size on each node, taking the standard deviation and dividing by the mean. We average these measurements over all inserts to indicate the storage variation among nodes as a percent of the average host load, and a lower value indicates a more balanced partitioning.

The randomized allocations, Consistent Hash, Extendible Hash, and Round Robin do best because they subdivide the data at its finest granularity, by its chunks. Append exhibits poor load balancing overall; this scheme only writes to one node at a time, no matter how many are added. Skew strongly influences the performance of our range partitioners. AIS has significant hotspots near major shipping ports, hence it has a very imbalanced storage partitioning for Uniform Range, although this scheme is the best for our uniformly distributed MODIS data. K-d Tree also has difficulty handling skew because it can only subdivide one dimension for each node addition. A more comprehensive approach (i.e., quad-trees) may work better and will be studied as future work.

Figure 2(b) shows the query performance of the two use cases on the benchmark queries in between each load cycle. Our benchmarks demonstrate that for SPJ queries, the partitioners perform proportionally to the evenness of their storage distribution. The science benchmarks show that clustered data access is important for array-centric workloads. K-d Tree has the best performance for both workloads, as it facilitates clustered reads, and is moderately skew resistant. Append performed poorly in the science benchmarks for two reasons. First, it balances query execution poorly because this strategy partitions the data by time; therefore when a pair of new nodes are added, one of the hosts will not be used immediately. Second, new data is "hotter" in our benchmarks, as some of the queries "cook" the new measurements into results, and compare them with prior findings.

In summary, we found that K-d Tree was the most effective partitioner for our array workloads, as it breaks up hotspots, and caters effectively to spatial querying. For data loading and reorganization, the append approach is fastest, but this speed comes at a cost when the database executes queries over imbalanced storage. When we consider end-to-end performance, summing up the findings in Figures 2(a) and 2(b), K-d Tree is the fastest solution for MODIS, whereas Append and the skewed range partitioner are on par for AIS.

4 Query Processing

At first blush, one could simply use a traditional cost-based relational optimizer, and then repurpose it for the operators found in an array DBMS. There are two reasons why this is not likely to succeed. First, the commutative join and filtering operations from relational systems are not prevalent in scientific workloads.

Instead, there are many more non-commutative operations that cannot be pushed up or down the query plan tree. Earth science [17], genomics [21], and radio astronomy [22] all exhibit this recipe of few joins paired with complex analytics.

Hence, the opportunities for rearranging the query execution plan are more limited. Second, it is not obvious that the relational join tactics (hash join, merge sort, iterative substitution) are the best choices for array data.

In the common case where skew is present, we have invented a novel *n-way shuffle join*, which is effective at both balancing the query load across nodes as well as minimizing network traffic to accomplish the join. In short, the algorithm entails locating sparse and dense areas of the arrays, and then sending sparse areas to the corresponding dense ones to perform the join. Hence, it minimizes the amount of network traffic to accomplish a distributed join. We have shown that this algorithm can be added to the merge-sort and iterative substitution join algorithms in SciDB and never results in significantly worse performance than non-shuffle approaches. When dense areas in one array line up with sparse areas of a second array, dramatic performance improvements result. We also propose a second approach, which we call a *load balancing shuffle join*. This approach assigns chunks to nodes such that each node executes the join on the same number of cells. Our implementation uses integer programming to find a chunk allocation that minimizes data movement subject to achieving an even distribution of join work across our cluster nodes.

In Figure 3, we evaluated merge join for a pair of 2D 100 GB synthetic arrays that share dimensions and logical chunk sizes. We varied the degree of skew for our input data; for the uniform case all of our chunks are of the same size. For each other case, the per-node partitioning follows a Zipfian distribution, where the parameter denotes the skewness of the input, and higher values denote greater imbalance in the data's distribution. As a baseline, we use the traditional *move-small* strategy of sending the smaller array to the nodes of the larger one for merge joins. Figure 3 shows the time used for data alignment (DA) and join execution (JE).

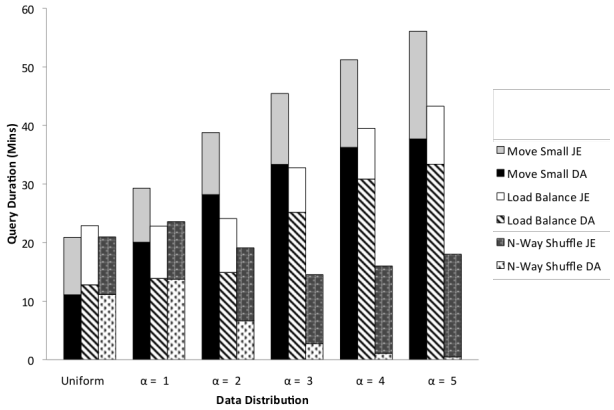
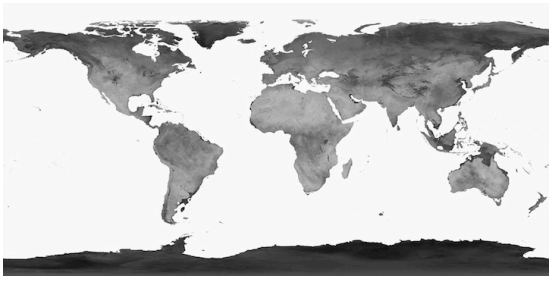


Figure 3: Join duration with varying skew and data shuffling strategies.

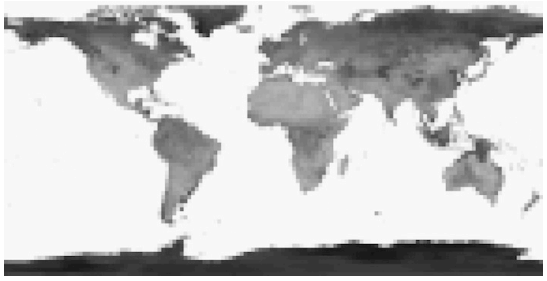
For the uniform case, all of the algorithms perform comparably. When we have slight skew ($\alpha = 1$), the load balancer transfers about the same amount of data as the n-way shuffle, but it has better parallelism in the join execution, producing a slight win. As the skew increases, the n-way shuffle significantly outperforms the other techniques, moving less and less data. For $\alpha \geq 3$, we consistently have a speedup of 3X. We are in the process of finishing a SciDB implementation for our algorithms, obtaining more complete performance numbers, and writing a paper for publication on this work. We then expect to continue this work by developing a complete optimizer that integrates this join processing with the other SciDB operators, and addresses the ordering of cascading joins.

5 Visualization

Modern DBMSs are designed to efficiently store, manage, and perform computations on massive amounts of data. As a result, more analytics systems are relying on databases for the management of big data. For example, many popular data analysis systems, such as Tableau [9], Spotfire [10], R and Matlab, are actively used in conjunction with database management systems. Furthermore, in [23] they show that distributed data management and analysis systems like Hadoop [3] have the potential to power scalable data visualization systems.

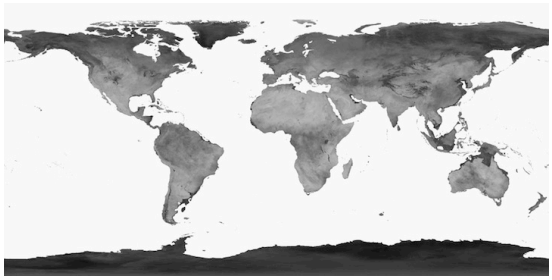


(a) Baseline heatmap visualization of NDSI data



(b) Aggregating NDSI data at 10,000 points resolution

(c) Aggregating NDSI data at 100,000 points resolution



(d) Aggregating NDSI data at 1,000,000 points resolution

Figure 4: Heatmap visualizations produced by ScalaR, using aggregation to reduce the NDSI dataset stored in `ndsi_array`. Dark areas represent high amounts of snow cover

Unfortunately, many information visualization systems do not scale seamlessly from small data sets to massive ones. A given visualization may work well on a small data set with a modest number of points, but will paint the screen black when presented with an order of magnitude more data. Having the individual visualization system deal with this scalability issue has two major flaws. First, code must be included in perhaps many individual modules to accomplish this task, an obvious duplication of effort. Second, visualizations run on the client side of a client-server interface, for which large amounts of data may have to be passed back and forth and computing resources are more limited than on the server side of the boundary. Obviously, a shared server-side system, running close to the DBMS, should prove attractive.

To address these issues, we have developed a flexible, three-tiered scalable interactive visualization system named ScalaR [11] that leverages the computational power of modern DBMSs for back-end analytics and execution. ScalaR decouples the visualization task from the analysis and management of the data by inserting a middle layer of software to mediate between the front-end visualizer and the back-end DBMS. ScalaR has been implemented for SciDB, but is back-end agnostic in design, as its only requirements are that the back-end must support a query API and provide access to metadata in the form of query plans. ScalaR relies on query plan estimates computed by the DBMS to perform resolution reduction, i.e., to summarize massive query result sets on the fly. ScalaR's resolution reduction model can be applied to a wide variety of domains, as the only requirement for using ScalaR is that the data is stored in a DBMS. For example, it has been used with SciDB to visualize NASA MODIS satellite imagery data [1], LSST astronomy data [4], and worldwide earthquake records [8].

We have run several experiments with ScalaR, assessing the runtime performance and resulting visual quality of various reduction queries over two SciDB arrays containing NASA MODIS satellite imagery data. Specifically, we visualized normalized difference snow index (NDSI) calculations over the entire world at various output sizes (data resolutions). The NDSI measures the amount of snow located at a particular latitude-longitude cell on the earth. As a baseline, we also ran the query on the raw data, with no resolution reduction.

Figure 4(a) shows a visualization of the baseline for one of these arrays, which we will refer to as ndsi_array. The baseline query completed in 5.68 seconds. The ndsi_array is a dense SciDB array, containing roughly 6 million data points. To perform aggregation reductions, we used SciDB's `regrid` operation, which divides an array into equal-sized sub-arrays, and returns summaries over these sub-arrays by averaging the array values. Figures 4(b) through 4(d) are the visualized results of these four reduction queries. The smallest reduction to 10 thousand data points was the fastest with 1.35 seconds. The other reductions were comparable to the baseline. Thus we can see that ScalaR produces small aggregate summaries of the NDSI data very quickly, but the resulting image is blurred due to the low number of data points in the result. However, at a resolution of 100,000 data points we produce a visualization that is is a very close approximation of the original with an order of magnitude fewer data points.

Our current focus is twofold. First we are conducting a substantial user study of ScalaR at UCSB on MODIS data and at the University of Washington on astronomy data. In each case scientists have agreed to test the system on real world problems. Besides obtaining feature and ease-of-use feedback, we will obtain traces of user activity. These traces will be used to train our middle tier prefetching system, whereby we plan to use all available server resources to predict (and prefetch) future user data. Our system contains two experts, one is a path expert, which predicts the direction and speed on user browsing and fetches forward along this path. Our second expert looks for patterns in the recent user activity and then looks for similar patterns further afield. Depending on their success, experts are given more or less space in the cache and more or less machine resources to prefetch items. Our prefetching system is nearly operational and we hope to test it in the near future.

We also plan to extend ScalaR's front-end visualizer, which currently requires users to specify all components of the final visualization, including what resolution reduction technique to use, the data limit to impose on the back- end, the x- and y-axes, the scaling factor, and coloring. Lack of experience with visualizing the underlying data can make it difficult for users to make these specific visualization choices in advance, and can result in many iterations of trial and error as users search for a suitable way to visualize the data.

To help users quickly make better visualization choices, we are designing a predictive model for identifying the most relevant visualization types for a given data set. We will use the model to produce sample visualizations in advance, reducing the number of choices a user must make and simplifying his visualization task.

To train our predictive model, we are creating a corpus of visualizations from the web. For each visualization we have access to the data set that was used to create it. We will use this underlying data to learn what features potentially correspond to specific visualization types. The visualizations are collected from a wide variety of web sources, including the Many Eyes website [5], various ggplot2 examples [2], and the D3 image gallery [13].

6 Conclusions

This paper has presented the flavor of on-going array DBMS research. It presented our research directions in the areas of elasticity, query processing and visualization. Other work in this area has been omitted because of space limitations including provenance [24, 25] and on using the DBMS for the MODIS processing pipeline [18].

References

[1] Benchmarks for Elastic Scientific Databases. `http://people.csail.mit.edu/jennie/elasticity_benchmarks.html`.

[2] ggplot2. `ggplot2.org`.

[3] Hadoop. `http://hadoop.apache.org/`.

[4] Large Synoptic Survey Telescope. `http://www.lsst.org/lsst/`.

[5] Many Eyes. http://www-958.ibm.com/software/data/cognos/manyeyes/.

[6] NASA MODIS. http://modis.gsfc.nasa.gov/.

[7] U.S. Coast Guard AIS Ship Tracking Dataset. http://marinecadastre.gov/AIS/default.aspx.

[8] U.S. Geological Survey Earthquake Hazards Program. http://earthquake.usgs.gov/.

[9] Tableau Software. http://www.tableausoftware.com/, 2013.

[10] TIBCO Spotfire. http://spotfire.tibco.com/, 2013.

[11] L. Battle, M. Stonebraker, and R. Chang. Dynamic Reduction of Query Result Sets for Interactive Visualization. In *IEEE BigDataVis Workshop*, 2013.

[12] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[13] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[14] J. Duggan and M. Stonebraker. Elasticity in Array DBMSs. (submitted for publication).

[15] R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong. Extendible hashing–a fast access method for dynamic files. *ACM Trans. Database Syst.*, 4(3):315–344, 1979.

[16] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. STOC, 1997.

[17] G. Planthaber. Modbase: A scidb-powered system for large-scale distributed storage and analysis of modis earth remote sensing data. Master's thesis, MIT, 2012.

[18] G. Planthaber, M. Stonebraker, and J. Frew. EarthDB: scalable analysis of MODIS data using SciDB. In *BigSpatial*, 2012.

[19] M. Stonebraker, P. Brown, D. Zhang, and J. Becla. SciDB: A Database Management System for Applications with Complex Analytics. *Computing in Science Engineering*, 15(3):54–62, 2013.

[20] M. Stonebraker, S. Madden, and P. Dubey. Intel "big data" science and technology center vision and execution plan. *SIGMOD Record.*, 42(1):44–49, 2013.

[21] R. Taft, M. Vartek, N. R. Satish, N. Sundaram, S. Madden, and M. Stonebraker. Genbase: A complex analytics genomics benchmark. Technical report, MIT CSAIL, 2013.

[22] G. van Diepen. SciDB Radio Astronomy Science Case. http://scidb.org/UseCases/radioAstronomy_usec

[23] H. Vo, J. Bronson, B. Summa, J. Comba, J. Freire, B. Howe, V. Pascucci, and C. Silva. Parallel visualization on large clusters using MapReduce. In *Large Data Analysis and Visualization (LDAV)*, 2011.

[24] E. Wu and S. Madden. Scorpion: Explaining Away Outliers in Aggregate Queries. In *VLDB*, 2013.

[25] E. Wu, S. Madden, and M. Stonebraker. SubZero: A fine-grained lineage system for scientific databases. In *ICDE*, 2013.

Data Management Challenges in Species Distribution Modeling

Colin Talbert[1] Marian Talbert[1] Jeff Morisette[1] David Koop[2]
[1] U.S. Geological Survey, Fort Collins, CO
[2] New York University, NY

Abstract

An important component in the fields of ecology and conservation biology is understanding the environmental conditions and geographic areas that are suitable for a given species to inhabit. A common tool in determining such areas is species distribution modeling which uses computer algorithms to determine the spatial distribution of organisms. Most commonly the correlative relationships between the organism and environmental variables are the primary consideration. The data requirements for this type of modeling consist of known presence and possibly absence locations of the species as well as the values of environmental or climatic covariates thought to define the species habitat suitability at these locations. These covariate data are generally extracted from remotely sensed imagery, interpolated/gridded historical climate data, or downscaled climate model output. Traditionally, ecologists and biologists have constructed species distribution models using workflows and data that reside primarily on their local workstations or networks. This workflow is becoming challenging as scientists increasingly try to use these modeling techniques to inform management decisions under different climate change scenarios. This challenge stems from the fact that remote sensing products, gridded historical climate, and downscaled climate models are not only increasing in spatial and temporal resolution but proliferating as well. Any rigorous assessment of uncertainty requires a computationally intensive sensitivity analysis accounting for various sources of uncertainty. The scientists fitting these models generally do not have the background in computer science required to take advantage of recent advances in web-service based data acquisition, remote high-powered data processing, or scientific workflow systems. Ecologists in the field of modeling are in need of a tractable platform that abstracts the inherent computational complexity required to incorporate the burgeoning field of coupled climate and ecological response modeling. In this paper we describe the computational challenges in species distribution modeling and solutions using scientific workflow systems. We focus on the Software for Assisted Species Modeling (SAHM) a package within VisTrails, an open-source scientific workflow system.

1 Introduction

The objective of this paper is to demonstrate the utility of scientific workflow tools in addressing the increasingly complex data management issues associated with species distribution modeling (SDM). We start with a brief overview of SDM and the currently available software options. We then explain two major factors that are contributing to the increasing modeling complexity. First there is the growing array of SDM algorithms

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

available, each having its own parameters, input file formats, and native software. Here the data complexity resides in the metadata associated with the model runs, such as software version, parameter files, and input and output that are not necessarily tracked in the SDM software. Second there is the complexity of data access; which has become a particular challenge with the growing interest of projecting SDM results into the future by using climate projections. We go on to explain how scientific workflow software, in particular VisTrails:SAHM has helped ecological modelers deal with this complexity of the first factor and how we hope to use it to address the second. The organizational capacity and provenance inherent to VisTrails:SAHM serve both to tie previously disparate tools together and maintain detailed records of the input used, output generated, and parameter specifications and allows the modeling to take advantage of machine service architectures to construct models using predictor layers stored on remote servers. We have found that organizing SDMs within the context of scientific workflow simplify data and model complexities, thus allowing analysts more time to concentrate on the ecological implications and useful application of their species distribution modeling activities.

2 Species Distribution Modeling: Overview and Challenges

Species distribution models (SDMs) are used to link species locations to spatially defined environmental variables. There are several algorithms available for fitting species distribution models which vary widely in their origins and complexity from statistical algorithms such as classic generalized linear models (GLMs) [17] and generalized additive models (GAMs) [13] to machine learning algorithms such as random forest [4] and boosted regression trees [11] to network models such as artificial neural networks (ANNs) [19]. The primary focus of the SDM literature and of the SAHM software is on observation-based determination of the correlative relationships between an organism and its environment. Once quantified, these correlative relationships can be used to gain insight into a variety of important ecological and evolutionary questions and model output can be projected onto spatial layers to produce maps of the species' ecological niche. Unfortunately, a large number of sources contribute to uncertainty in the prediction of species distribution especially when projecting to new spatial or temporal extents and when the input layers are often themselves output from other models. Approaches capable of partially quantifying this uncertainty such as ensembles and maps or partitions of quantifiable uncertainty have been recommended in the literature [8]. Proper model assessment often involve a sensitivity analysis on several groups of factors including the models and the environmental data.

It has been argued that climate is often the most basic determinant of a species fundamental niche in that it limits the species' range at the broadest spatial scale [2]. There has been a rapidly growing interest in projecting species distribution models onto future climate scenarios to inform management decisions under a changing climate. In this arena, the quantification or exploration of uncertainty based on the various sources is critical and the most favorable approach involves producing species distribution maps for various combinations of SDM models, emissions scenarios, and coupled Atmosphere-Ocean Generalized Circulate Models (AOGCMs), downscaling methods, as well as any other known sources of uncertainty. This situation creates an explosion in computation time and the data storage requirements for fitting SDMs. One recent paper, for example, generated 8400 projections to describe the uncertainty in fish assemblage projections [5]. The modules we are developing within SAHM are designed to remove the programming and data storage challenges that limit the use of recommended practice for many ecologists.

The scientific data management challenges encountered in species distribution modeling stem from two main factors. First is the number of disparate tools required for a typical SDM workflow. The second stems from the difficulties encountered preparing and using the gridded environmental inputs. While neither of these factors is unique in computational science, certain aspects of SDM make it a good candidate for implementing tools and practices for robust scientific data management. First is the widespread acceptance and use of SDM methods for many ecological research and management questions. The second is lack of experience and training among many practitioners of SDM in the tools and techniques used for scientific data management. Additionally, with

SDM modeling the results can be used to make controversial decisions with respect to endangered or invasive species and proper documentation of the modeling workflow is helpful to review and defend results.

2.1 Integrating SDM Tools

Several tools are currently available for modeling species distribution. We were able to determine the most commonly utilized tools based on a recent survey of SDM software use by ecologists [15] and use this information to review the available tools. Due to the complex and multidisciplinary workflows required for SDMs it is unlikely that the entire workflow could be carried out in any single software package with the exception of possibly R, but this would require significant programming skill. Also, due to the various components, it seems practical that species distribution modeling requires expertise in climate models, remote sensing, GIS, the species being considered, and statistics techniques. Further, if the model output is to be applied to natural resource management questions, expertise in that management domain is also required. All of the software packages with GUI interfaces demonstrated limited flexibility in terms of either modeling for only one type of response, with one modeling algorithm, not providing opportunities to explore the data, or not producing detailed graphics for model evaluation. Most packages with GUI interfaces focused on the model fitting and evaluation components of the workflow but require a separate set of tools and expertise to perform the preprocessing or data layer preparation steps. These issues are only compounded for the ecologist hoping to understand the nuances of how models differ or hoping to map patterns in uncertainty or determine when the spatial projections are extrapolating beyond the environmental calibration space.

It is important to emphasize that the full SDM process with climate inputs will generally need to integrate a variety of tools at different points in the process. For example, a scientist might use a web interface to select and download the required climate inputs such as the USGS Geo Data Portal (GDP) [3]. She might then use a custom script to process them into the format required for their model such as a toolbox in ArcMap. The model is then run using a specific GUI tool such as Maxent. The steps to select the final model and analyze model performance might be done with custom code, perhaps written in R. The modeling is also likely to involve decisions and deliberations by the analyst(s); which may or may not be rigorously documented. The final model output might then be reformatted and visualized in a final GIS program such as ArcMap. This type of workflow often presents challenges with tractability, transportability, documentation of methodology, and repeatability.

2.2 Handling Input Data

A primary data need of species distribution modelers are the geospatially referenced gridded environmental data often referred to as layers which are used as inputs to SDMs. These data can include traditionally mapped GIS layers such as land cover, remotely sensed products such as those from Moderate Resolution Imaging Spectroradiometer (MODIS)[1], and the subset that we will focus for the remainder of this paper on historic and modeled climate products. These climate data can represent interpolations of historical readings such as PRISM (Parameter-elevation Relationships on Independent Slopes Model [12]) or modeled hindcasts or projections of future climate such as those produced under CMIP5 (Coupled Model Intercomparison Project Phase 5 [18]). The native formats and schemas of these data generally fall into a limited number of standards including NetCDF (network common data form) with CF (Climate and Forecast) metadata conventions and HDF5 (Hierarchical Data Format, Version 5). Due to the large data volumes inherent to global and high resolution downscaled regional climate models, the data often come tiled into discrete files either spatially or temporally. The size of individual climate datasets is generally reasonable but cumulatively they can become quite large (100s of GBs to 10s of TBs). Although these data generally adhere to common data schemas, scientists and analysts still often run into novel formats or nuances that require ad hoc tools and workflows. The ability to utilize these non-standard data formats is often beyond the capabilities of researchers without strong technical and programing skills.

These demands have led to the development of technologies aimed at facilitating data use by scientists through distributed storage and remote web service based access. Examples of these technologies include OPeNDAP, (Open-source Project for a Network Data Access Protocol) [22] , THREDDS (Thematic Realtime Environmental Distributed Data Services) [24], and Open Geospatial Consortium (OGC) Web Coverage Service (WCS)[14]. In general these technologies provide web service protocols for remote data access and discovery and provide capabilities for spatial and temporal grid subsetting. These data delivery protocols can be further extended by providing data processing services which leverage remote computational resources, for example, The OGC Web Processing Service (WPS) Interface Standard [14] or the USGS Geo Data Portal [3].

There are currently a number of excellent tools available for researchers who need to consume scientific data from these types of services. These include web-based interfaces, GUI tools, application specific toolkits, and APIs for a variety of programing languages and libraries (Matlab, R, Python, GDAL, etc). While these technologies have gained wide acceptance and use within the earth science community, their use is not currently widespread among ecologists and biologists. This might be partly due to the lack of support for these technologies in many of the current widely-used ecological modeling tools, including those for SDM. Integrating service-based climate data acquisition or processing currently involves either custom code to obtain the data or a multi-tool workflow that can be cumbersome to run and document. By better integrating these technologies with existing tools and workflows, the process could be streamlined and standardized while at the same time realizing better computational efficiencies for species distribution modelers as well as increasing the integration of climate science into ecology and natural resource management.

It is important to note that the temporal format, i.e., discrete daily or monthly time steps, of most climate data must generally be summarized prior to its use in SDMs. This temporal summarization can involve one of several common algorithms, for example, the 19 Bioclimatic (BioClim) variables which capture annual temperature and precipitation patterns needed to model species distributions. Alternately modelers might need custom climate temporal summary algorithms based on the life history of the species being modeled. The flexibility required in generating these custom summaries precludes pre-calculating and caching the intermediate results. But at the same time these summarizations result in data volume reduction by several orders of magnitude.

3 Scientific Workflows and Provenance for Species Distribution Modeling

Scientific workflow systems provide a context in which to specify computational processes which integrate existing applications according to a set of rules [6]. Within this context, scientific workflows allow researchers to model complex analysis processes at various levels of detail and systematically capture the required information necessary for reproducibility and sharing; collectively referred to as workflow ”provenance” [7, 21]. When applied to species distribution modeling, scientific workflow software provides an opportunity to integrate existing SDM modeling routines seamlessly with software to ingest data, preprocess those data, and visualize model results.

3.1 The VisTrails:SAHM System

One current implementation of SDM integrated with scientific workflow systems is the SAHM package which runs within VisTrails [9, 10, 25]. VisTrails provide an approachable and extendable graphical user interface, and also provides sophisticated tools for workflow provenance capture and visualization. VisTrails is written in Python, a programing language which has many powerful and mature packages for scientific data processing including SciPy, NumPy, and matplotlib. Additionally there are efforts underway to enable complex climate data acquisition, processing, and visualization in VisTrails, namely the VisTrails Package UV-CDAT (Ultrascale Visualization - Climate Data Analysis Tools) [20, 25].

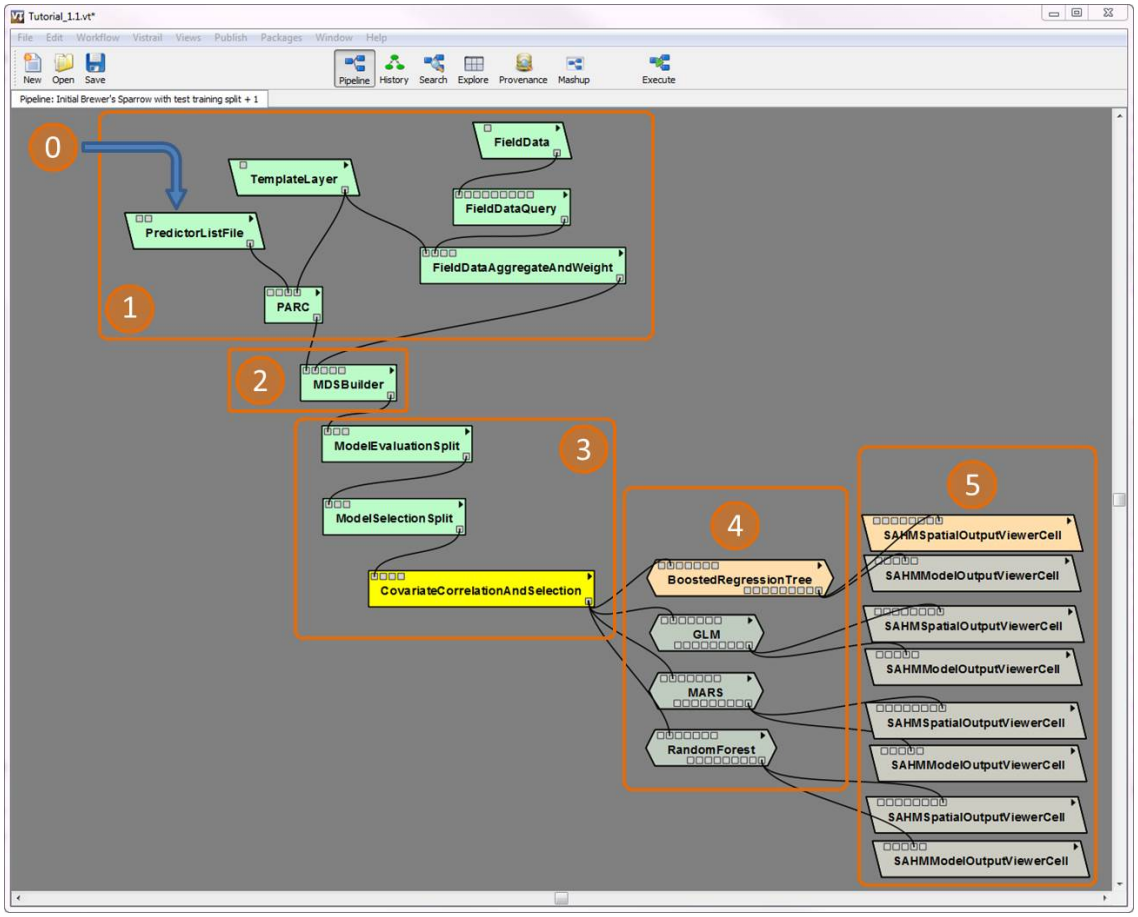


Figure 1: The VisTrails:SAHM User Interface. 1. preprocessing and data clean up, 2. merging input observational data with available covariates, 3. preliminary data exploration, 4. fitting correlative models, 5. model evaluation, comparison, and selection. Item 0 in the figure represents the remote data access and climate data summarization which is not currently implemented in VisTrails:SAHM

SAHM was developed as a VisTrails package to expedited the process and formalize the disparate tools required to build complex SDM workflows. Developing our habitat modeling software within VisTrails offered several advantages over previous tools. We are now able to develop template workflows that ecologist can use as starting points in their own analysis. Modifications to workflows are accomplished by dragging and dropping new modules onto the canvas and connecting these to the existing workflow components rather than by modifying scripts. These modifications can be annotated and tracked through the history view and we can easily compare differences between workflows. Within this framework much of the complexity of data management and multiple tool workflow integration and are hidden from the user. By hiding this complexity we were able to facilitate the use and coordination of fairly disparate technologies currently used in SDM. Fortunately, many of the most commonly used tools for SDM are script-based (Matlab, R, Python), have a scripting API (ArcMap, IDL), or can be run from the command line (Maxent and GDAL), which facilitates their incorporation into scientific workflow software. Figure 1 shows an example SDM workflow in VisTrails:SAHM and will be used here to highlight the complexities of 1) combining multiple analysis tools and 2) ingesting and handling a large array of potential predictor layers. (Additional details on this example workflow and the VisTrails: SAHM package are given in Talbert and Talbert [23]). The tools incorporated in SAHM can be broken loosely into five components (see figure 1): 1. preprocessing and data clean up, 2. merging input observational data with available

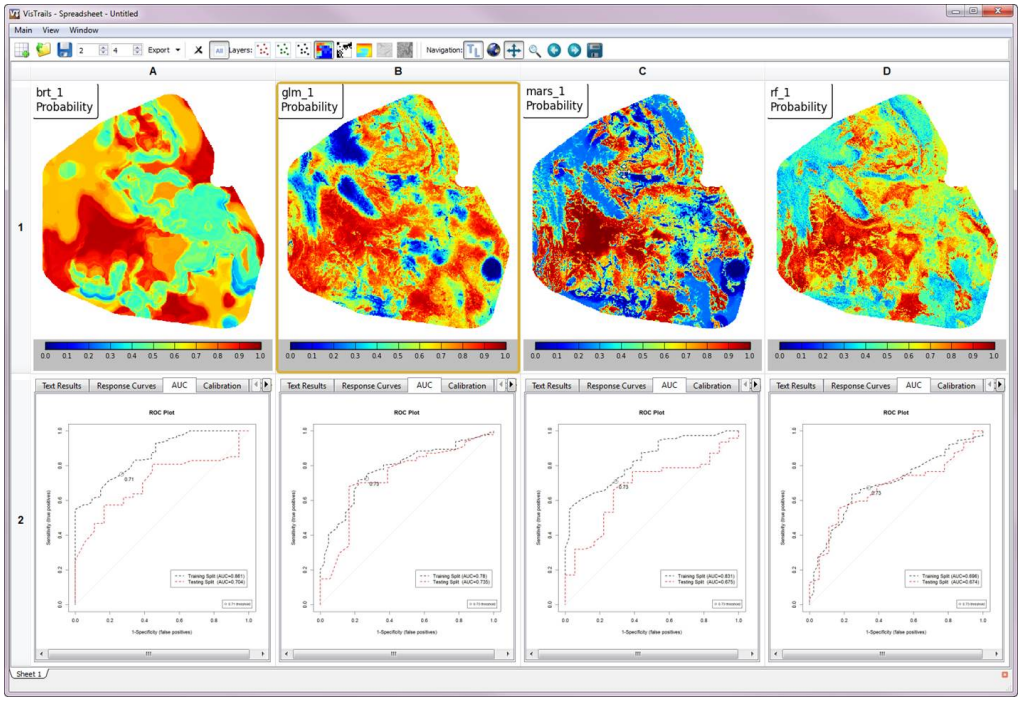


Figure 2: VisTrails:SAHM Output displays and compares continuous and binary maps as well as pertinent evaluation metrics from several models

covariates, 3. preliminary data exploration, 4. fitting correlative models, 5. model evaluation, comparison, and selection. Each module within the SAHM package allows the user to customize the step to meet their need and many modules allow the user to visualize and explore the data allowing subsequent decisions to be informed by previous modules in the workflow.

Another advantage to integrating current SDM tools in the VisTrails platform is the ability to incorporate coordinated visualization of multiple model outputs from multiple model runs. Figure 2 show the VisTrails:SAHM output from four model runs displayed in the built-in VisTrails spreadsheet. Output content is synced across multiple cells to facilitate model comparison. While many other SDM tools provide visualization options the ability to easily organize and traverse numerous outputs is unique to this platform.

4 Future Work

VisTrails:SAHM is a first step towards helping ecologists use increasingly complex SDM workflows and tool sets while also maintaining the crucial provenance and repeatability needed for defensible science. VisTrails also has the potential to provide tractable solutions for the current and emerging input data access and computation demands that SDM scientist encounter. The scope and context of each SDM research question can require specific data management strategies to maximize data acquisition and processing efficiency. These strategies are dependent on several factors including the spatial and temporal extent of the study, the number of different species being modeled, how many iterations will be run, the downloading and processing capacity available locally, and the technical expertise of the researcher. In this section we present four use cases which demonstrate different strategies for handling data acquisition and processing. The first strategy presented, maintaining a local copy of the input data, is the one most often used by species distribution modelers currently. It is presented here as basis for comparison with the other three.

4.1 Input Data Complexity

The traditional model for data management in SDM involves downloading or otherwise obtaining the required gridded spatial inputs and then preprocessing and storing them on a local computer or network (see Figure 3). Once the store of local data has been created there is maximal flexibility in how the data are processed. Given adequate computational resources locally, the processing of large areas, numerous SDMs, and multiple iterations becomes feasible. One drawback of this strategy is that the effort to store and process the data locally can be substantial and the disk space required to maintain local copies of the data can be limiting. Each time a new set of climate models is released, this data management effort will need to be duplicated in full. Another drawback is that the resulting workflow is not transferable to researchers at other locations unless they also obtain an identical copy of the input datasets and directory structure. This can limit repeatability of individual experiments. Using scientific workflow software to manage the initial input data acquisition and preprocessing can solve issues related to transferability of the workflow but does not overcome limitations in data acquisition bandwidth, storage, or processing limitations.

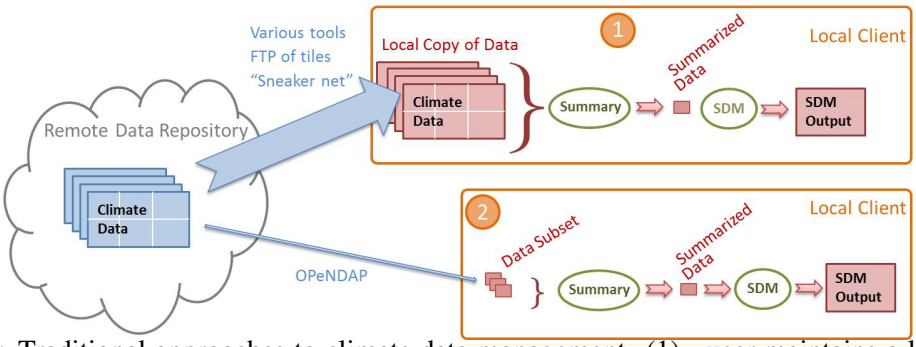


Figure 3: Traditional approaches to climate data management. (1) - user maintains a local copy of input data, (2) - small subsets of data are accessed remotely as needed using OPeNDAP.

A second approach to data acquisition in SDM is to obtain just the spatial and temporal subset of the gridded data needed for a given analysis. This is illustrated in Figure **??**. Many climate and remote sensing data are available in a tiled format which allows a user to download just the individual tiles needed. Identifying and obtaining the appropriate tiles can be cumbersome depending on the distribution method and available tools. One approach to facilitate getting a subset of the data is to use a service-based interface to specify the data needed, for example, WCS or OPeNDAP. While these technologies might be unfamiliar to many ecologists their implementation details can be hidden within user-friendly SDM applications. Implemented correctly, this tool could automatically download and process the appropriate data based on the spatial and temporal extent of the other SDM inputs. This type of solution works especially well for research questions on a local to regional scale. By implementing this strategy in a scientific workflow-based tool, the inherent complexity of the current tools can be hidden from researchers using SDM.

One concern with implementing a tool that automates service-based distribution of subsets of climate data is that it can lead to inefficiencies and duplicated data downloads if caching is not handled appropriately. For example, a single routine to produce a gridded climate summary might start with a call to a service to obtain the required input data. Once the appropriate summary has been generated this routine might delete the downloaded climate inputs to free up disk space. If there is only a single summary to produce, this strategy works fine, but if there are numerous summaries that need to be produced they should not each be downloading the required inputs redundantly. Within a single workflow, this level of optimization is relatively easy to set up nut across workflows and tools this data management can be much harder to manage. Optimally a tool would implement a persistent cache of previously downloaded inputs with a database that tracks multiple downloads of overlapping

but non-coincident spatial extents, total cache size, and differences between the cache contents and the data available from the data server [16].

Since SDMs generally require only a temporal summary grid of individual climate models, a third approach to data acquisition is to use a web service to deliver only this summary grid. Given that climate models can have hundreds to thousands of daily or monthly time steps, this strategy can represent several orders of magnitude reduction in data volume that must be downloaded. Some climate summaries such as decadal Bioclimatic Variables (Bioclim) are sufficiently general that they can be generated ahead of time and delivered using standard web data services such as WCS and OPeNDAP. Unfortunately, many SDM research questions require novel climate summaries which cannot be generated ahead of time. Using a Web Processing Service allows for these summaries to be calculated as needed on a remote server. If sufficient computational resources are available on the server hosting the remote processing, there is also the potential to realize a significant reduction in computation time versus calculating these summaries locally. One example of this type of WPS is currently being developed at the USGS Geo Data Portal (GDP) [3]. This WPS accepts custom parameters for the BioClim algorithms. These parameters are used to generate a server side processing job which produces and returns the desired custom summary grids. The ability of this strategy to scale to non-trivial problems given current infrastructure has yet to be tested. But potentially it could extend the subset of SDM research problems handled by the second option above to those covering a much larger spatial extent. If the processing was happening locally to the data many of the issues of data caching would not be a problem.

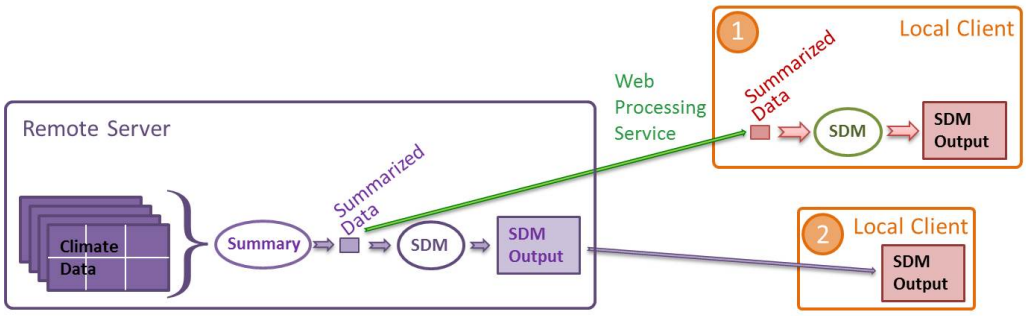


Figure 4: OGC Web Processing Service delivering custom climate summaries (1) or complete SDM model results (2)

A fourth approach to data management in SDM is to transfer even more of the processing workflow to a remote server. Instead of delivering custom climate summaries it is possible to engineer a WPS which runs an entire SDM workflow and returns the final model outputs. In this case, the species occurrence data becomes a parameter that is transfered to the remote server hosting the WPS along with detailed processing instructions. An advantage to having the full workflow details captured by a workflow management software such as VisTrails is the ease by which a detailed complex workflow can be packaged and transported. In this case, all relevant information about the workflow is stored in a compressed XML document. By scaling the data storage and computation resources available to the WPS it is feasible to use this strategy to solve many SDM data access issues. Implementing this solution has significant challenges though, including the trade off between computational flexibility and security on the remote server, managing large data hosted at multiple sites, and various other implementation details, scientific workflow software can be useful in in resolving some of these issues and hiding the unnecessary details from scientists.

5 Conclusions

Within the field of SDM the use of service based data acquisition and processing technologies is currently underutilized. This partly stems from the fact that the tools often require programing skills to be utilized effectively and partly because the need to work with large and unwieldy climate data is still emerging in the field of SDM. As more and more research questions require analysis of climate data the integration of these technologies will necessarily become more widespread. Given that ecologists and biologists often do not have the strong background in scripting required there is a developing need to extend current tools to take advantage of these technologies. In order to gain widespread usage in the community any tools or workflows developed should be user-friendly, transparent, and flexible. By building this next generation of SDM tools in a scientific workflow management system we can achieve this goal while also adding provenance capture, repeatability, and transferability.

Our development efforts have so far focused mainly on porting and optimizing exiting SDM tools for the VisTrails framework. This effort has been well received within the community. We are currently working to extend this framework to take advantage of data provided via a web service as well web processing services. The final phase of our work will extend this further to develop the four data acquisition approaches in which the vast bulk of the data management and processing is hosted remotely on high performance systems.

Acknowledgments: This research is funded by the U.S. Geological Survey and NASA's Applied Sciences Program. Any use of trade, product, or firm names is for descriptive purposes only and does not imply endorsement by the U.S. Government.

References

[1] National Aeronautics and Space Administration. Moderate resolution imaging spectroradiometer data. https://lpdaac.usgs.gov/products/modis_products_table. Accessed: 2013-10-29.

[2] M. B. Araújo and A. T. Peterson. Uses and misuses of bioclimatic envelope modeling. *Ecology*, 93:1527–1539, 2012.

[3] D. Blodgett, N. Booth, T. Kunicki, J. Walker, and J. Lucido. Description of the U.S. Geological Survey Geo Data Portal data integration framework. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 5(6):1687–1691, 2012.

[4] L. Breiman. Random forests. In *Machine Learning*, pages 5–32, 2001.

[5] L. Buisson, W. Thuiller, N. Casajus, S. Lek, and G. Grenouillet. Uncertainty in ensemble forecasting of species distribution. *Global Change Biology*, 16(4):1145–1157, 2010.

[6] A Dedeke. Workflow management: Models, methods, and systems., 2004.

[7] Ewa Deelman, Tevfik Kosar, Carl Kesselman, and Miron Livny. What makes workflows work in an opportunistic environment? *Concurrency and Computation: Practice and Experience*, 18(10):1187–1199, 2006.

[8] J. A. Diniz-Filho, R. D. Loyola L. M. Bini, T. F. Rangel, C. Hof, D. Nogués-Bravo, and M. B. Araújo. Partitioning and mapping uncertainties in ensembles of forecasts of species turnover under climate change. *Ecography*, 32(6):897–906, 2009.

[9] J. Freire, D. Koop, E. Santos, C. Scheidegger, C. Silva, and H. T. Vo. *The Architecture of Open Source Applications*, chapter VisTrails. Lulu.com, 2011.

[10] Juliana Freire, Claudio T Silva, Steven P Callahan, Emanuele Santos, Carlos E Scheidegger, and Huy T Vo. Managing rapidly-evolving scientific workflows. In *Provenance and Annotation of Data*, pages 10–18. Springer, 2006.

[11] J. H. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:337–407, 2000.

[12] PRISM Climate Group. Prism climate data. http://www.prism.oregonstate.edu/. Accessed: 2013-10-29.

[13] T. J. Hastie and R. J. Tibshirani. *Generalized additive models*. London: Chapman & Hall, 1990.

[14] Open Geospatial Consortium Inc. Web processing service. http://www.opengeospatial.org/standards/wps. Accessed: 2013-10-29.

[15] L. N. Joppa, G. McInerny, R. Harper, L. Salido, K. Takeda, K. O'Hara, D. Gavaghan, and S. Emmott. Troubling trends in scientific software use. *Science*, 340(6134):814–815, 2013.

[16] David Koop, Emanuele Santos, Bela Bauer, Matthias Troyer, Juliana Freire, and Cláudio T. Silva. Bridging workflow and data provenance using strong links. In *Proceedings of the 22Nd International Conference on Scientific and Statistical Database Management*, SSDBM'10, pages 397–415, Berlin, Heidelberg, 2010. Springer-Verlag.

[17] P. McCullagh and J. A. Nelder. *Generalized linear models (Second edition)*. London: Chapman & Hall, 1989.

[18] WCRP World Climate Research Programme. Cmip - coupled model intercomparison project - overview. http://cmip-pcmdi.llnl.gov/index.html. Accessed: 2013-10-29.

[19] B. D. Ripley. *Pattern recognition and neural networks*. Cambridge Univ. Press., 1996.

[20] E. Santos, J. Poco, Yaxing Wei, Shishi Liu, B. Cook, D.N. Williams, and C.T. Silva. Uv-cdat: Analyzing climate datasets from a user's perspective. *Computing in Science and Engineering*, 15(1):94–103, 2013.

[21] Carlos Scheidegger, David Koop, Emanuele Santos, Huy Vo, Steven Callahan, Juliana Freire, and Claudio Silva. Tackling the provenance challenge one layer at a time. *Concurrency and Computation: Practice and Experience*, 20(5):473–483, 2008.

[22] Open source Project for a Network Data Access Protocol Inc. Opendap. http://www.opendap.org. Accessed: 2013-10-29.

[23] M. Talbert and C. Talbert. Tuturial for the software for assisted habitat modeling. https://www.sciencebase.gov/catalog/folder/503fbe63e4b09851b69ab463. Accessed: 2013-10-29.

[24] Unidata. Thredds data server. http://www.unidata.ucar.edu/software/thredds/current/tds/TDS.html. Accessed: 2013-10-29.

[25] VisTrails. http://www.vistrails.org.

From Large Simulations to Interactive Numerical Laboratories

Alexander S. Szalay
The Johns Hopkins University, Baltimore, MD 21218, USA
szalay@jhu.edu

Abstract

High Performance Computing is becoming an instrument in its own right. The largest simulations performed on our supercomputers are now approaching petabytes. As the volume of these simulations is growing, it is becoming harder to access, analyze and visualize these data. At the same time for a broad community buy-in we need to provide public access to some of the simulation results. This is becoming another Big Data challenge, where we have to move the analyses and visualizations right where the data is. The paper will discuss the challenges in creating such interactive numerical laboratories.

1 Introduction: Data in HPC

1.1 Motivation

The nature of science is changing, it is increasingly limited by our ability to analyze the large amounts of complex data generated by our instruments and simulations: Jim Gray's "Fourth Paradigm" of science [3, 14]. The largest numerical simulations use tens of millions of hours of CPU time, yet most of the analysis must be performed while the simulations are running, since the output data are too large to be moved or stored for reuse. As a result, it is difficult, if not impossible, to confront model predictions with the exponentially increasing amounts of observational data. Scientists in many disciplines would like to compare the results of their experiments to data emerging from numerical simulations based on first principles. This requires not only that we run sophisticated simulations and models, but that the results of these simulations are available publicly, through an easy-to-use portal. We have to turn the simulations into open numerical laboratories where anyone can perform their own experiments.

Integrating and comparing experiments to simulations is a non-trivial data management challenge. Not every data set from these simulations has the same lifecycle. Some results are just transient and need to be stored for a short period to analyze, while others will become community references, with a useful lifetime of a decade or more. As we have learned over the years, once the data volume reaches a certain size, we have to move the analysis to the data [24, 25] rather than the traditional approach, which moved the data where our computers were. With these large data volumes we have to approach the data in a fully algorithmic fashion — manual exploration in small files is no longer feasible.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

1.2 The emerging simulation data challenge

There is an ongoing effort world-wide to build an exascale computer by approximately 2018. This reflects a scale-up by a factor of 30 compared to the top machines in late 2013. Some of the largest particle-based simulations today already exceed a trillion particles. If we only store their positions and velocities, and a single particle identifier, we have to save 56TB for each snapshot, uncompressed. Needless to say, that this in itself represents a challenge.

At the same time, fewer and fewer codes scale to run well on millions of cores, and as a result, fewer and fewer people will use these ever larger systems. There will be an increasing gap between the wide science community and these top users. It is increasingly important to create usable science products that can be used by a much broader pool of users; otherwise community support will be soon endangered. Thus, we postulate that it is extremely important to identify a mechanism through which data products from the largest simulations in science can be released, publicly shared and used, potentially over extended periods.

1.3 Data lifecycles of simulations

On the Fly Analysis

Most truly large numerical simulations are analyzed on the fly. The analysis tools are integrated with the simulation, and the derived data products are computed while the simulation is running. As these quantities represent only a small fraction of the data, it is easy to save these values often to disk. Full restart checkpoints are thus only generated quite infrequently. The disadvantage is that if a new analysis idea emerges after the run is finished, the whole simulation needs to be redone.

Private reuse

Sometimes a few tens of snapshots are saved to scratch disks, tightly coupled to the supercomputers, and a occasionally a segment of the simulation can be regenerated later from restarting from the nearest snapshot. This is typically done by the same team who ran the original simulation.

Public reuse

There are selected cases when the simulation outputs are made available, usually done through sharing the limited number of snapshot files. These are typically placed at a public file server, and can be downloaded at will. However, this model of data sharing poses practical limits to data downloads at a few terabytes at most. The limiting factor is usually the network bandwidth, although the available storage at the user's end is also a problem.

Public Service portal

Sometimes the simulation outputs are made available through publicly available services, enabling the users to perform either some extractions or computations over the data. This idea of "virtual data" has been around for more than a decade, but it has found limited uses. The Earth Science community has used OpenDAP [19] to expose large data sets and enable a RESTful URL to subset and aggregate the data. In astrophysics, the Millennium Database [16] has been the forerunner of such efforts. In this scenario, the creation of the public service portal and its complex functionalities requires a substantial effort, thus it is only worth doing if the data set will remain public for an extended period, at least a few years.

Archival and long-term curation

There are very few simulations that have reached this stage of their lifecycle. Here an extra challenge is that not every simulation will be equally used by the public, and over the years; some of them will fade into irrelevancy while others emerge as a community reference. It is these latter simulations which need to be kept for a long time, even if just for comparison and reference. For such collections, used by many different refereed publications, reproducibility of these analyses will become another reason to keep the data, even when better and higher resolution alternatives become available. However, as the price of both storage and computation are expected to follow the current trends and become cheaper every day, these "legacy" data sets will comfortably fit in the shadow of the latest and best simulations.

1.4 Petascale numerical laboratories

Even though the largest simulations today are approaching hundreds of billions of particles or grid points, the size of the output generated is typically a few tens of TB, rarely exceeding 100TB and almost never reaching a PB. As already mentioned above, there are many reasons for this. The larger the computer, the more cumbersome checkpointing becomes, and while the biggest machines have a Tbit/sec aggregate sequential bandwidth to their storage, copying 100TB will still take 800 seconds, too long to do often, limiting the number of snapshots. As the interconnect speeds are not going to increase by a factor of 30-100, it is likely that this limitation remains in place, thus even once we have exascale machines, the outputs will still remain in the few PB range.

When the primary consideration is checkpoint restart, it is enough to have a small number of snapshots. On the other hand, if the goal is to be able to reconstruct the fine-grained spatial and temporal history of the simulation, and look at any part in detail, it is important to match the high spatial resolution with an appropriate temporal resolution, i.e. a large number of snapshots. For example, if we simulate a Milky Way-like galaxy, and want to study its dynamics in detail in our laboratory, we need to save outputs more frequently than the characteristic rotational period of 10^8 years. This means that even the simulations need to be designed and run differently if the target is to create a long-lived numerical laboratory used by hundreds of scientists.

2 The Challenges

The premise of this paper is that due to community pressure and demand for repeated posterior analyses some of the best and largest simulations will be turned into public numerical laboratories. This process presents nontrivial challenges for every step of the simulation and reuse process. In particular, we need to figure out

1. Where to store the data for the reuse
2. How to move the data to this location
3. How to look at it, how to render peta/exabytes
4. How to interface to the data
5. What analysis patterns will be used
6. What architectures to use for the posterior analysis

In the following subsections we will look at these in some detail.

2.1 Moving the data

Storage attached to supercomputers has a premium. In order to minimize the overhead due to checkpointing, this storage has to be very fast, also on the fast interconnect. The primary function of this storage is to dump the contents of the RAM into large sequential files at maximum speed. To keep petabytes of data for months to years and running interactive analyses by hundreds of users requires a different architecture.

The most important aspect of this is that the long term data storage should use inexpensive storage technology so that it can keep accumulating important data. This facility (we will call it Numerical Laboratory from now on) has to be interconnected with a high speed network to the primary data site, the HPC machine. Today, many large data sets are moved via "sneakernet" [11], i.e. copying the data onto disk drives and shipping or carrying the drives physically. This works for data that fits onto a small number of hard disks, up to about 10TB, or 3 disks. Beyond this, mounting tens of disks, partitioning, and copying the data, and keeping track of what is where becomes impractical.

Network transfer of larger files is also non-trivial. Over a 10 Gbps link (assuming wire speed) we can move a 100TB data set in about 80,000 sec, or 1 day. Over such a link realistic transfer speeds are more in the 6Gbps/sec range. It is important to note that these transfers must be done from disk-to-disk. Both ends of the transfer must have fast enough I/O to support 1GBps disk reads and writes. Once transfers take more than a few hours, often they get interrupted. Luckily, user friendly technologies, like Globus OnLine [10] are emerging which support automatic recovery from such errors. These are maximizing the network throughput by enabling parallel striping of the transfer.

Many of the national supercomputer sites are in the process of upgrading their connections to 100G. Both ESNet and Internet2 are moving their backbones to 100G. Using these, one can theoretically move a PB in a day. Internet2 provides Layer 3 10G services, but for a while the 100G connectivity will remain Layer2 only, making the routing more cumbersome.

Local networking rules and firewalls can have a dramatic impact on the transfer speeds. Many institutions are setting up demilitarized zones (DMZs) to isolate a segment of their network that can talk to the outside at high speeds. OpenFlow [20] is emerging as a mechanism to have applications configure and negotiate fast virtual circuits across the continent. So, while it is still rather painful today to move data exceeding 100TB, this capability is expected to become increasingly available to a wider set of institutions, while PB-scale data transfers will still be relatively limited.

2.2 Interfacing to the data: files or services?

The simplest thing is to simply provide access to the simulation files. This is the most frequent mechanism used today to disseminate outputs. However, these large files store the data in large contiguous chunks, usually partitioned following the domain decomposition, in a packed (often proprietary) binary format. Access to the data this way is only useful if the simulations are small, or all the data is loaded back into memory. If each snapshot is tens of TB, there are not many user-owned computational facilities that can support such an amount of RAM. Also, the user of the data has to understand the data formats, units, organization, domian decomposition and boundary conditions. Often mastering these requires several days, in not a week, of preparation and coding, prohibitive for a casual user.

Having a service oriented access with a finer granularity has a lot of advantages. Data can be provided "shrink-wrapped", with calibrations applied, in proper physical units, like the archive for the Sloan Digital Sky Survey [12]. Aggregates can be computed and subsets can be extracted on demand. The result is that the user performs the low level data processing right on top of the data, inside the Laboratory, reducing the volume to be transmitted across the wide area network. Of course, this requires enough computing power tightly integrated with the data storage. On the other hand, from a community perspective the individual users can run their petascale analyses form a laptop, rather than forced to build their own facility, so this may be a reasonable value proposition for the community as a whole.

2.3 Access patterns for analysis

There are several distinct access patterns how a posterior analysis will access the simulation data.

Global analytics

These are analyses that require access to the whole simulation volume. Examples of these are computing the Fourier transform of a scalar field, like density, spatial correlation functions, or computing the density for a particle-based simulation using a smoothing kernel. These operations are quite similar to those performed on a supercomputer during the simulation itself (Poisson solvers, etc). As the global data access touches a very large fraction of the data, this pattern is optimally satisfied by loading the sequential dumps of the snapshots. As data sizes grow, running these analyses over a heavily partitioned parallel system is creating an increasing communication overhead, e.g. the transpositions needed for the 3D FFT.

Rendering

Many other analyses are similar to a rendering of a subvolume. These of course include visualizations of large subsets. In cosmology in particular, an interesting challenge emerges: as we create a rendering of a light cone in the Universe, create an image of what a virtual telescope would see, we have to account for the finite speed of light within the simulation. As we look farther and farther along the line of sight, we see simulation at an earlier time. This means that unless we have many snapshots stored, such renderings are impossible to recreate.

The effects of gravitational lensing is currently computed by collapsing the distribution of mass along the line of sight within the simulation onto idealized slabs, where the projected mass density acts as a refractive index, to scatter light gravitationally. This is really a computer graphics problem, rendering the mass onto multiple planes and then doing a parallel ray-tracing.

Another problem resembling rendering is computing the pair annihilation of dark matter particles in a simulation. The collision rate of particles is proportional to the density squared, and the cross section can have an additional dependence on the velocity of the particles. Thus, the resulting maps can be very sensitive to the local environment. However, this is still a rendering problem, where we take a scalar field attached to a particle based support and create a projected map.

Typically these patterns require accessing a substantial fraction, but not all, of the simulation volume. Various spatial data structures, like octrees, space filling curves, can help in optimizing the amount of data to be read into RAM. A spatial index with a flexible geometric search capability is required for efficient data access. GPUs can be extremely useful in these computations.

Localized access

There are computational tasks which require a very localized access. For example in order to compute the fluid velocity at a given location, we need to extract a small volume of a few grid cells in each direction, the size of our kernel, and then compute an interpolated value. Such immersive computations have been used to create a smart laboratory for turbulence [27].

In cosmology simulations often we run the simulation with collisionless dark matter only. Then a friend-of-friends algorithm is used to group of particles into halos and subhalos. Some of these will be called galaxies. Using the merger history of these groups one can heuristically estimate what the physical properties of these galaxies would be. In the Millennium simulation this is precomputed, However, there is a growing need to be able to perform these computations on-demand, changing the heuristic rules leading to a galaxy and understand their impact on the final ensemble of observable galaxies.

These patterns all require a very fine-grained localized access to small parts of the data. Here databases with a high-resolution spatial index can make a huge difference in enabling fast data reads.

2.4 Immersive access, virtual sensors

For a scalable analysis we need to come up with a data access abstraction or metaphor that is inherently scalable. For the user it should not matter whether the data in the laboratory is a terabyte or a petabyte. The casual user should be able to perform very light-weight analyses, without downloading much software or data. Accessing data through the flat files violates this principle: the user cannot do anything until a very large file has been physically transferred.

On the other hand one can create a so-called immersive environment, where the users can insert immersive virtual sensors into the simulation [18]. These sensors can then feed data back to the user. They can provide a one-time measurement, they can be pinned to a physical (Eulerian) location or they can "go with the flow as comoving Lagrangian particles. In this case, assuming that the sensors can access the data server side very fast, the only scaling is related to the number of particles.

By placing the particles in different geometric configurations, users can accommodate a wide variety of spatial and temporal access patterns. The sensors can feed back data on multiple channels, measuring different fields in the simulation. They can have a variety of operators, like computing the Hessian or Laplacian of a field, or applying various filters and clipping thresholds.

This pattern also enables the users to run time backwards, impossible in a direct simulation involving dissipation. Imagine that the snapshots are saved frequently enough that one can interpolate particle velocities smoothly enough. Sensors can back-track their original trajectory and one can see where they came from, all the way back to the initial conditions. We can also imagine cases where one simply retrieves the velocity of the sensor particles, and applies a special equation of motion involving other factors (inertia, friction, stochastic perturbations) and move the particles externally, possibly on a users laptop, anywhere in the world. This simple interface can provide a very flexible, yet powerful way to do science with large data sets from anywhere in the world.

2.5 Data organization, streams and indexes

Much of the efficiency of the numerical laboratory will depend on how the data is organized and laid out on the storage devices. If we talk about several petabytes of data, the only real storage option is using an array of hard disks. These are still mechanical device. They speed of revolution has remained largely constant over the last decade, and the inexpensive commodity drives have settled at around 7200 rpm. As the density of disk platters grows, the disk capacity increases by the inverse square of the magnetic domain size. The read speed increases as the inverse, thus it takes longer and longer to read the ever larger disk platters, disks are becoming sequential devices.

For large data volumes we need first of all a maximally sequential layout. As many numerical simulations cover a 3+1D domain, simply storing a large subvolume as a separate file, ordered in x,y,z is less than optimal for small localized accesses: for each small subvolume we need to perform many separate reads, spaced by a stride apart, for a single extraction. This results in a very poor cache localization, and many random accesses on the external disks.

Space filling curves have been used for a long time [22] as a cache resilient way to access such spatial data. Indeed, using a Peano-Hilbert or a Morton curve enables a mapping of a 3D region to a contiguous sequential location, providing a handle to maximally sequential data access. The key of the space filling curve can then be used as an index into the physical data store, and every subvolume can be represented as a set of range queries over the key. Such operations are highly optimized in relational databases, if this is mapped onto a clustered index (primary key).

At the same time, in particle-based simulations the need often arises to follow trajectories of individual particles. In this case we need an inverted index. Rather than storing the whole date over again, a rather expensive proposition for petabyte data sets, one can create highly compressed indexes, which use the fact that

particles move with a finite velocity [6]. This way we can represent the whole trajectory of the particles in a fraction of the original storage.

2.6 Visualization and Rendering

A visual exploration of the data is a powerful tool of science. While we have immersive caves and large video walls with tens of millions of pixels near our supercomputers, they are not helping much a wider community. For many scientists the most useful aspect of visualization is to help them decide whether an idea or hypothesis is to be discarded or worth further exploration. The faster data can be transformed and looked at, the faster the whole feedback loop from the conception of a new idea to transformation and rendering can be performed, the more useful it becomes. Today much of the user-side visualization is done with a few open source toolkits, executed near the user, typically on a workstation with a mid to high end GPU. This approach, while it can give a fast turnaround, is unfortunately not scalable to large data volumes.

Fast GPUs have changed the landscape for scientific visualization. Their rendering speeds are such, that most visualization projects are limited by the bandwith for data to be loaded into GPU memory. For this reason, just like for Big Data analytics where we have to execute our workflows as close to the data as possible, large scale visualization must also be performed right on top of the data [26].

For petabytes, only remotely driven visualizations will have the bandwidth to access the underlying data sets. This approach is aided by the fact that we have reached a point when user devices can reliably access streaming content at speeds allowing HD video almost anywhere. In this case we only have to move the result of the rendering, not the source, reducing data volumes by many orders of magnitude.

The data access patterns necessary for petascale visualizations have several algorithmic and architectural consequences. The visualizations must be done in parallel, where the distributed components are co-located with a distributed data storage. At the same time, visualizations can have very different locality requirements. In some cases we need the data at a small granularity, where indexes based on space filling curves work very well. In other cases we need a large fraction of the subvolume, organized in (x,y,z) order. Given the ultra high speeds of the GPU on-board RAM, reorganizing data in GPU memory is much faster than regular RAM, thus loading the data in memory in (almost) arbitrary order, and then copying into its final location on the GPU works extremely well in practice [5].

3 Architectures of Numerical Laboratories

One might first think that given the recent proliferation of commercial cloud solutions will bring an immediate solution to the table. There are several interesting and important aspects of these cloud architectures. The most important is that they do not try to be everything for everybody. Due to their commercial nature they have been built with a razor-sharp focus, with several major tradeoffs in their design.

These are extreme scalability, economies of scale and resilience. The clouds built by Amazon, Google and Microsoft are all based on the principle that components will fail, and systems have to survive and recover, and data cannot be lost. This is accomplished by massive data replication and excessive monitoring. The commercial clouds must also be cheap to operate.

The data in these clouds is largely unstructured, from web content to click streams. Also, the connectivity of the data is quite different: mostly it is large graphs describing links or social networks. Much of the software framework for the processing reflects this necessity.

Scientific data is quite different. Much of our data, especially the simulations is highly structured, with well defined schemas and data structures. There are well-defined 2 and 3D domains, with distinct timesteps, and boundaries. With the large data volumes, there are similarities as well, especially how in science we are also exploring phenomenological correlations rather than only searching for strict, causal relationships. Nevertheless,

due to the more structured nature of scientific data, the computations can utilize vectorizable architectures, like GPUs much better.

The economics is also different. Pricing for the external users of the commercial clouds is driven by the perceived business model of a small, but dynamically growing business, which is generating revenue from efficiently using data analytics over small to mid-scale data volumes. The prices for compute cycles are generally very competitive when compared to the costs of running your own.

The same cannot be said about storage, once it reaches Terabytes. The monthly cost of a TB of storage on the commercial clouds is roughly equal to the price of a TB disk drive [21]. Of course, in the cloud one gets redundant storage, service guarantees and the costs of operations are included. But there is also a download charge, and access costs apply. So, for many science projects today putting data in a commercial cloud on a pay as you go basis is simply not affordable. Also, bandwidth between the data and compute is marginal. As a result, very few mid to large scale data sets have been placed into the cloud, other than a few experiments, or making use of occasional free storage space by the providers. It is important to note, that this is entirely a function of the pricing should cloud storage prices drop by order of magnitude, for mid-size data sets in the few tens of TB this could be a very attractive proposition.

For data sets in the 100 TB to PB range, getting data into the cloud is hard. The costs per year are close to 1M, and structured access to the data is not as fast as one can achieve with a dedicated system.

3.1 Amdahl number

It is good to have a simple characterization of the I/O needs of both computational tasks and the underlying computer architectures. Bell, Gray and Szalay [2] has introduced the Amdahl number, the sequential I/O speed in bits/sec divided by the total instructions/sec, following Amdahls seminal paper [1]. This simple metric shows a clear distinction between supercomputers, with Amdahl numbers around 0.001, workstations at around 0.1, and data analytics engines at Amdahl numbers close to 1.

Furthermore, the Amdahl number can also be computed for a computational task, like a numerical simulation, or their posterior analyses, by dividing the output size by the CPU hours, converted to the appropriate units. Typically, our supercomputer simulations have Amdahl numbers much lower than the hardware they are running on. This is almost by definition, as one could only saturate the hardware by continuously maxing out the I/O, at the expense of computations, defeating the purpose of using a supercomputer.

On the other hand, posterior analyses of the data can have very different Amdahl numbers. Streaming analyses of petabytes can easily require Amdahl numbers close to unity. Small-granularity database queries are in this category. As outlined in Section 2.3, there is a whole spectrum of computational tasks whose needs cover a wide range of Amdahl numbers.

One can also define an approximate Amdahl number for a GPU. As an example, an NVIDIA Kepler K20 card has about 1.3Tflops and its I/O is limited by the practical bandwidth of 6GB/sec over a PCIe gen2 backplane. This translates to an Amdahl number of 0.037, definitely higher than a supercomputer.

3.2 Database servers or supercomputers?

The challenge is to define an architecture that is (i) efficient and inexpensive for storing petabytes of data, (ii) provides streaming high bandwidth to the data, (iii) can occasionally perform compute-intensive tasks involving large amounts of data. It is clear, that supercomputers have enormous processing power, but they are too expensive for the posterior processing of large data sets. It is also clear that database servers have been successful in providing fast, indexed access to small granularity data, they do not have the compute power to perform some of the more floating point intensive analyses. But between these two edge cases there is a continuum of problems to be tackled successfully.

3.3 Storage hierarchies for data analysis

Given the wide range of data access patterns, we need a system that is capable of accessing data both sequentially at high speeds, but can also provide high IOPS for small accesses, and is capable of high computational performance. The challenge is that building a homogeneous system that satisfies all these criteria would be next to impossible, and certainly very expensive.

The key is heterogeneity. One can build systems that can be close to optimal along each of these requirements individually. We know how to build fast streaming engines that can move data sequentially at backplane speeds from Hard disks all the way to GPUs, like the JHU Data-Scope. Also we know how to use solid state disks to provide close to a million IOPS per server. And we are learning how to use large memory servers.

Commodity interconnect is becoming very affordable. Infiniband and 40G Ethernet are below 500 USD per port, and latencies are approaching a microsecond. With a million IOPS from an SSD system, the latency of reading a packet of data is similar than over our fastest networks. For memory intensive problems one can combine commodity systems soon with 6TB of memory, tens of TB of SSD and ultra fast interconnects, blurring the traditional domain decomposition issues.

4 Use cases and examples

4.1 Turbulence

Over the last few years we have built several different numerical laboratories that have been used to prototype and explore the ideas outlined in this paper. The first of these deals with turbulence. We took a relatively small 1024^3 box containing a simulation of isotropic turbulence. The simulation consisted of approximately 10,000 timesteps. We created snapshots of the first 100 timesteps, then every 10th, until we reached 1000 snapshots. The outputs were the three velocity components and the pressure. The total data volume is about 30TB.

The data has been reorganized into small, 8^3 cells, which are labeled by a Morton curve or z-index. These cubes are stored as BLOBs in a relational database system (Microsoft SQL Server). The z-index is used as a partitioning key. The database is used as an indexed data access layer. The BLOBs contain a user-defined data type (UDT) of a multidimensional array of a single homogeneous primitive data type. Arrays can be either small (8kB) or large (up to 2GB). The UDT has various methods enabling Matlab-like array manipulations [8].

On top of the indexing is a generalized spatial search library that can be used for data extractions defined by various geometric primitives and their Boolean combinations [17]. All these functionalities heavily use the object-relational interface of SQL Server, enabled through the CLR integration [13].

The data is accessible through various web services. The first service we have implemented was the one of the virtual sensors. Users can use their own laptop to insert an array of sensors, defined by a 3+1 dimensional coordinate. The user can lay out these sensors in various geometric patterns, providing a simple yet flexible interface for accessing data. The service returns the interpolated velocities at the different locations. Other derived quantities, like computing the velocity gradient tensor can also be requested. There are several different language bindings available (C++, Fortran, MATLAB).

The users can then run various statistical analyses over the data returned. They can also implement their own path integrator, and use the velocities to move the particles, but with the additional capability of adding additional components, like a small stochastic force term. However, using the database, one can do analyses that would not be possible in real time when the simulation is running on the supercomputer.

As turbulence is a dissipative phenomenon, one cannot run the simulation backward. As we have stored the full temporal history of the velocity field, at a high enough resolution for path interpolation, we can move the particles backwards in time, and determine where all the particles in a given neighborhood come from! This has led recently to a paper in Nature [9].

This simple interface has been highly successful, by 2011 we have exceeded 100 billion sensor lookups delivered to users all over the world. With time, it became clear, that moving many massless Lagrangian particles along their trajectory was a pattern frequently requested. Thus, we added another service were you can place the sensors which then would move with the flow, reporting their positions and velocities.

For the isotropic turbulence database the data set is large enough that it is non-trivial to stream the data from a remote server to a visualization engine. We have performed several experiments in visualization tight on top of the data. We have built a separate server with a fast motherboard with multiple PCIe x16 slots (gen 2). The system also contained a high-end visualization card (NVIDIA GTX 690) and a high throughput LSI 9211 disk controller, connected to 120GB OCZ Vertex 3 SSD drives, running Windows Server 2008. We have copied 32 snapshots of the turbulence database over to this system. The disks were able to deliver over 1GB/sec sequential throughput.

A rendering engine built in DirectX 10 was used to create various visualizations. The renderer was sending queries to the database, which in turn delivered the velocity field in 8^3 blocks, together with a z-index used for the database organization. The arrival order of the blocks was not guaranteed. The data was immediately copied into the GPU. The real 3D position of each block was calculated from the z-index and the high bandwidth of the GPU memory enabled a very fast copy of the data into its final location, organized in the usual (x,y,z) order for optimal rendering.

The visualization engine had options for calculating various derived quantities, like the velocity gradient tensor, and its invariants, implement various smoothing operators for multiscale visualizations, or to visualize velocity fields by inserting clouds of particles which then moved along the streamlines [5]. During these experiments we have achieved over a GB/sec data transfer speeds, mostly limited by the number of SSDs.

4.2 Cosmological N-body simulations

The Millennium Simulation

We are currently in the process of building several different numerical laboratories out of large-scale cosmological N-body simulations. The first of those was based upon the the Millennium simulation [23]. Its database, built by Gerard Lemson [16] has been in use for over 7 years, and has hundreds of regular users, and has resulted in several hundred refereed publications. The database contains value added data from a simulation originally only containing 10B dark matter particles. These particles form hierarchical clusters, so called sub-halos and halos, which in turn become the sites where galaxies might form. A semi-analytical recipe was used to create mock galaxies in the simulations, and their hierarchical mergers were tracked in a database structure. The merger history was used to assign a plausible star formation rate to each galaxy which in turn can be used to derive observable physical properties.

The database contains several such semi-analytic scenarios. The set-oriented SQL query language makes it remarkably easy to formulate very complex aggregate queries over the temporal history of various subsets of galaxies and create samples that can be compared directly to observations.

Via Lactea II and Silver River

These simulations seek to simulate the formation and evolution of the Milky Way galaxy. Via Lactea II [7, 15] was the first of the two, with about 1B dark matter particles, it was one of the highest resolution such simulation. It led to a bigger simulation, the Silver River, which has over 50B particles.

There are several non-trivial numerical experiments that can be turned into a public laboratory. The first involves a computation of the annihilation of the dark matter particles. The Fermi satellite is in the process of observing high energy radiation from space, and it is not unreasonable to expect that in the near future it might see a sign of excess radiation from the Galactic Center that can be attributed to the annihilation of the elusive dark matter.

In preparation, we have built a numerical experiment, where users can interactively choose a viewpoint relative to the Galaxy, and then compute an image of the high energy radiation coming from the annihilation. Furthermore, the users can choose the physical properties of the cross section for the annihilation. Different physical models for the dark matter can yield dramatically different images. Originally, computing a single annihilation map took over 8 hours on a regular server. Using a rendering algorithm written in the shader language of Open GL, and streaming the data efficiently from the disks, we are now able to compute a single map in 23 seconds, changing the kinds of analyses one can perform [28]. Soon a public service enabling scientists to play with such scenarios will be live.

One can imagine immersive uses of the numerical laboratory, similar to the turbulence case. The Sloan Digital Sky Survey has detected the remnants of several dwarf galaxies which collided with the Milky Way, and their structure was disrupted in the collision, their stars ended up in an elongated "stream [4]. One can easily image a scenario, where a user can take a small dwarf galaxy, consisting of about 1M particles, picks a trajectory and shoots it into the Milky Way. The dwarf galaxy moves in the time-dependent, high resolution potential of the original simulation, and the user can watch interactively how the particles are scattered into the elongated stream, and repeat the experiment until the desired final state is achieved. Our group is currently in the process of building such an interactive application.

The Indra suite of simulations

In astrophysics we have a single universe to observe, and we cannot even change our position, thus we are restricted to a single realization of space surrounding us. The only way to estimate the statistical uncertainty arising from the fact that we can only observe a finite sample of the Universe, the so called "cosmic variance is to use simulations. Most of the large simulations are trying to be the largest of their kind, thus focusing all the computational resources in generating a single realization.

There is also a need to create a statistical ensemble of simulations. Instead of focusing on the largest number of particles, or the highest resolution, we should create an ensemble of simulations, which can be used to estimate statistical uncertainty for our various observables, which can then be used to assess the significance of real observations of the Universe.

We are in the process of creating the Indra suite, eventually consisting of 512 simulations of 1B particles each. While none of the individual simulations are particularly large, their ensemble will have an data volume of more than 1PB [6]. Currently we have already ran the first 100 realizations, and have more than 200TB of data. Using this ensemble one can study covariances on such large scales, where none of the simulations have enough independent subvolumes. As it turns out, studying the density fluctuations on scales of the so called Baryon Acoustic Oscillations (BAO) is one of the most important challenges of cosmology today, and Indra will uniquely be able to support such explorations.

5 Summary

The next generation simulations will have trillions of particles, petabytes of output. Some of the simulations will be multigrid, some will be a combination of fluids and discrete particles. A wide community of users is ready to use the output of these simulations, and compare them to experiments and observations. The science community has used several of these prototype numerical laboratories successfully, even artfully. It is clear that the stage is set to have a new platform, which bridges the current gap between supercomputers and database servers. It is clear that the key to efficient data analysis is a good data layout and efficient data access.

The data access patterns identified in this paper are quite diverse, there is no single platform or architecture that fits all. Most likely we will need a heterogeneous ecosystem of system components, each optimized for different data access patterns and enough internal bandwidth to move parts of the data to the appropriate system

dynamically. This way we can get close to optimal layout for most analyses. In order to do so, we have to understand and characterize how different data access and analysis patterns map onto different system architectures and components.

6 Acknowledgements

The author would like to acknowledge support from the National Science Foundation OIA-1124403, ACI-1244820, OCI-1040114 and the Gordon and Betty Moore Foundation, grant 109285. Many of the ideas emerging here have grown out of conversations, joint projects and many years of collaborative effort with a lot of collaborators, without whom much if this would not have happened. I would like to thank Jim Gray, Jose Blakeley, Tamas Budavari, Kai Buerger, Randal Burns, Laszlo Dobos, Greg Eyink, Ed Givelberg, Kalin Kanov, Gerard Lemson, Piero Madau, Charles Meneveau, Eric Perlman, Joe Silk, Tamas Szalay, Ani Thakar, Ethan Vishniac, Sue Werner, Ruediger Westermann, Simon White and Rosemary Wyse.

References

[1] G. Amdahl. Computer architecture and amdahl's law. *IEEE Solid State Circuits Society News*, 12(3):4–9, 2007.

[2] G. Bell, J. Gray, and A. Szalay. Petascale Computational Systems: Balanced Cyber-Infrastructure in a Data-Centric World. *IEEE Computer*, 39:110–113, 2006.

[3] G. Bell, T. Hey, and A. Szalay. Beyond the data deluge. *Science*, 323(5919):1297–1298, 2009.

[4] V. Belokurov, D. B. Zucker, N. W. Evans, G. Gilmore, S. Vidrih, D. M. Bramich, H. J. Newberg, R. F. G. Wyse, M. J. Irwin, M. Fellhauer, P. C. Hewett, N. A. Walton, M. I. Wilkinson, N. Cole, B. Yanny, C. M. Rockosi, T. C. Beers, E. F. Bell, J. Brinkmann, Ž. Ivezić, and R. Lupton. The Field of Streams: Sagittarius and Its Siblings. *The Astrophysical Journal Letters*, 642:L137–L140, May 2006.

[5] K. Bürger, M. Treib, R. Westermann, S. Werner, C. C. Lalescu, A. S. Szalay, C. Meneveau, and G. L. Eyink. Vortices within vortices: hierarchical nature of vortex tubes in turbulence. `http://arxiv.org/abs/1210.3325`, 2012.

[6] D. Crankshaw, R. C. Burns, B. Falck, T. Budavari, A. S. Szalay, and J. Wang. Inverted indices for particle tracking in petascale cosmological simulations. In *SSDBM*, page 25, 2013.

[7] J. Diemand, M. Kuhlen, and P. Madau. Formation and Evolution of Galaxy Dark Matter Halos and Their Substructure. *The Astrophysical Journal*, 667(2):859–877, Oct. 2007.

[8] L. Dobos, A. S. Szalay, J. A. Blakeley, T. Budavári, I. Csabai, D. Tomic, M. Milovanovic, M. Tintor and A. Jovanovic. Array requirements for scientific applications and an implementation for Microsoft SQL Server. EDBT/ICDT Array Databases Workshop, 13-19, 2011.

[9] G. Eyink, E. Vishniac, C. Lalescu, H. Aluie, K. Kanov, K. Bürger, R. Burns, C. Meneveau, A. S. Szalay. Flux-freezing breakdown in high-conductivity magnetohydrodynamic turbulence. Nature, em 497,466, 2013.

[10] Globus online website. `https://www.globus.org`.

[11] J. Gray, W. Chong, T. Barclay, A. S. Szalay, and J. VandenBerg. Terascale sneakernet: Using inexpensive disks for backup, archiving, and data exchange. *CoRR*, cs.NI/0208011, 2002.

[12] J. Gray, A. S. Szalay, A. R. Thakar, P. Z. Kunszt, C. Stoughton, D. Slutz, and J. vandenBerg. Data Mining the SDSS SkyServer Database. *ArXiv Computer Science e-prints*, Feb. 2002.

[13] J. Gray, A. S. Szalay and G. Fekete. Using Table Valued Functions in SQL Server 2005 To Implement a Spatial Data Library. `http://arxiv.org/abs/cs/0701163`, 2007.

[14] T. Hey, S. Tansley, and K. Tolle. *The Fourth Paradigm Data-Intensive Scientic Discovery*. Microsoft Research, Redmond, WA, 2009.

[15] M. Kuhlen, P. Madau, and J. Silk. Exploring Dark Matter with Milky Way Substructure. *Science*, 325(5943):970–973, Aug. 2009.

[16] G. Lemson and t. Virgo Consortium. Halo and Galaxy Formation Histories from the Millennium Simulation: Public release of a VO-oriented and SQL-queryable database for studying the evolution of galaxies in the LambdaCDM cosmogony. *ArXiv Astrophysics e-prints*, Aug. 2006.

[17] G. Lemson, T. Budavári and A. S. Szalay. Implementing a General Spatial Indexing Library for Relational Databases of Large Numerical Simulations. Proc. SSDBM 11 Conference, 509-526, 2011.

[18] Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, R. Burns, S. Chen, A. Szalay, and G. Eyink. A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence*, 9:31–+, 2008.

[19] Opendap website. `http://docs.opendap.org/index.php/UserGuide`.

[20] OpenFlow Project website. `https://www.opennetworking.org/sdn-resources/onf-specifications/openflow`.

[21] How to build a cheap petabyte? `http://blog.backblaze.com/2009/09/01/petabytes-on-a-budget-how-to-build-cheap-cloud-storage`.

[22] H. Samet. *Applications of spatial data structures - computer graphics, image processing, and GIS*. Addison-Wesley, 1990.

[23] V. Springel, S. D. M. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, J. A. Peacock, S. Cole, P. Thomas, H. Couchman, A. Evrard, J. Colberg, and F. Pearce. Simulations of the formation, evolution and clustering of galaxies and quasars. *Nature*, 435:629–636, June 2005.

[24] A. Szalay and J. Gray. The World-Wide Telescope. *Science*, 293:2037–2040, Sept. 2001.

[25] A. Szalay and J. Gray. 2020 Computing: Science in an exponential world. *Nature*, 440:413–414, Mar. 2006.

[26] M. Treib, K. Bürger, F. Reichl, C. Meneveau, A. S. Szalay, and R. Westermann. Turbulence visualization at the terascale on desktop pcs. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2169–2177, 2012.

[27] Turbulence simulation services website. `http://turbulence.idies.jhu.edu`.

[28] L. Yang and A. Szalay. A GPU-Based Visualization Method for Computing Dark Matter Annihilation Signal. *Astronomical Data Analysis Software and Systems XXII*, 475:73, Oct. 2013.

A Computational Reproducibility Benchmark

Fernando Chirigati[1] Matthias Troyer[2] Dennis Shasha[3] Juliana Freire[1,3]
[1]Department of Computer Science and Engineering, New York University
[2]Institute for Theoretical Physics, ETH Zürich
[3]Courant Institute of Mathematical Sciences, New York University
{fchirigati,juliana}@nyu.edu, troyer@phys.ethz.ch, shasha@courant.nyu.edu

Abstract

Creating and testing reproducible computational experiments is hard. Researchers must derive a compendium that encapsulates all the components needed to reproduce a result. Reviewers must unpack the encapsulated components, run them in an environment that could be different from the source environment, and verify the results. Although many tools support some aspect of reproducibility, there is no common benchmark against which single or multiple tools can be tested. This paper describes a benchmark that can be used to categorize and better understand existing systems. The benchmark will also serve as the basis for a competition whereby tool builders will demonstrate if and how their systems support end-to-end reproducibility.

1 Motivation

Ever since Francis Bacon, a hallmark of the scientific method has been that experiments should be described in enough detail so that they can be repeated and perhaps generalized. When Newton said that he could see farther because he stood on the shoulders of giants, he depended on the truth of his predecessors' observations and the correctness of their calculations. In computational terms, this implies the possibility of (i) repeating (or *replicating*) results on nominally equal configurations, and (ii) generalizing the results by replaying them on new data sets, verifying how they vary with different parameters, and re-using and extending the experiment.

In principle, reproducibility should be easier for computational experiments than for natural science experiments, because not only can computational processes be automated, but also computational systems do not suffer from the "biological variation" problem that plagues the life sciences. Unfortunately, the state of the art belies this apparent ease. Most computational experiments are specified only informally in papers, where experimental results are briefly described in figure captions; the code that produced the results is seldom available and may be tied to specific configurations; and configuration parameters change results in unforeseen ways.

The lack of reproducibility has serious implications and has led to a credibility crisis in computational science [5]. In the absence of reproducibility, it has become difficult and sometimes impossible to verify scientific results, sometimes leading to major mistakes that are corrected only long after they are published, if ever. Furthermore, scientific discoveries do not happen in isolation. Important advances are often the result of sequences of smaller steps. If results are not fully documented, reproducible, and generalizable, it becomes hard to re-use and extend them.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Recently, there has been great interest on reproducibility and in the publication of reproducible results [2, 6, 7, 11, 14, 15, 16, 19, 23]. A number of conferences and journals instituted a reproducibility review process [1, 13, 21, 24], and while some have started to encourage authors to submit the experiments together with their papers, others have this as a requirement [17, 20]. However, these efforts have had limited success. A major roadblock to the more widespread adoption of this practice is the fact that it is hard to derive a compendium that encapsulates all the components (e.g., data, code, parameter settings) needed to reproduce the results, publish and verify them. Indeed, many scientists do not make their data and experiment reproducible [22], because authors complain that the process is too laborious [3].

While there are tools that support reproducibility, these have often been developed in isolation and target specific communities. There is also confusion about what the end-to-end process to attain reproducibility entails. As a result, tools lack important features and cannot be easily integrated with other systems that support these features. Researchers in search of a reproducibility solution are thus left in a quandary, since it is both hard to identify the tools they need and understand the functionality they support.

As a starting point to address this problem, this paper makes two contributions. First, we characterize the key tasks involved in the lifecycle of reproducible results, as well as different modes for attaining reproducibility. We then propose a benchmark that exercises these tasks in one mode that we call *spontaneous*. We have selected scenarios that are common in a variety of scientific domains. We also describe set of criteria to evaluate the benchmark results. Besides serving as a tool to categorize and better understand reproducibility systems, this benchmark will be used in an upcoming competition, where tool builders will implement end-to-end solutions for reproducibility and these will be tested by judges. We hope that through this benchmark and competition, insights will be obtained as to how to build comprehensive and general solutions.

2 Creating and Reviewing Reproducible Papers

In reproducible papers, the results reported, including data, plots and visualizations are linked to the experiments and inputs. Having access to these, reviewers and readers can examine the results, then repeat or modify an execution. Figure 1 shows one reproducible paper. In this paper, all plots have *deep captions* consisting of the workflow used to derive the plot, the underlying libraries invoked by the workflow, and the input data. This information allows the plots to be reproduced. In what follows, we describe tasks required to create reproducible experiments which can be published and shared.

Reproducibility Tasks. The first task is to (i) *create a description of the experiment*. A reproducible experiment must contain the description of the data used, the specification of the experiment – the steps followed, the underlying code needed to execute the specification, and the description of the environment where the experiment was executed. With this description, it should be possible for the author of the experiment to reproduce the experiment at a later time. However, to publish or share the experiment, the author must also (ii) *package all the components of the experiment* so that it can be executed by others in different computational environments. Finally, when creating a reproducible paper, the author needs to (iii) *connect the published results to the experiments*. Once a reproducible paper is submitted (or published), a reviewer should be able to unpack and run the experiments so that she can (iv) *reproduce and validate the results*.

Axes of Reproducibility. We have identified three distinct criteria to characterize experiments with respect to the level of reproducibility [9]: transparency, portability, and coverage. The *transparency* indicates whether partial or complete data and code are available. There are many possibilities for each step of an experiment pipeline, including: (a) partial data, e.g., a set of figures presented in a manuscript; (b) all data; (c) all data data plus the executable scripts; (d) the software system as a white box (source, configuration files, build environment) or black box (executable) on which the pipeline step is performed.

A second criterion is *portability*. An experiment can potentially be reproduced (a) on the original environment (basically, the author of the experiment can replay it on his or her machine); (b) on a similar environment (i.e., same OS but different machine), or (c) on a different environment (i.e., on a different OS and machine).

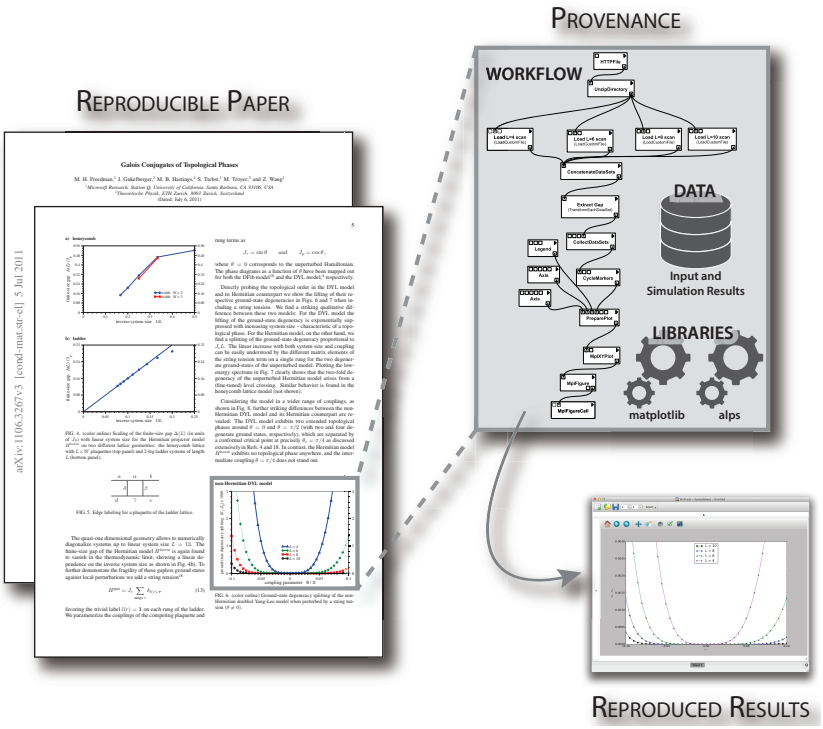


Figure 1: A reproducible paper. This paper by Freedman et al. [8] contains provenance-rich figures that have been created using the VisTrails system. Clicking on a figure downloads the workflow instance and associated provenance needed to derive the figure. This information can be examined and executed in VisTrails, reproducing the plot shown in the figure.

Finally, *coverage* indicates how much of the experiment pipeline is provided: (a) partial pipeline, i.e., only a subset of the experimental pipeline from raw data to final figures can be reproduced, or (b) full pipeline, i.e., the entire pipeline, from raw data all the way up to document, can be reproduced. As an example, experiments that rely on data derived by third-party Web services, special hardware, or lab experiments may not be fully reproduced, because repeating all the steps may be impossible. At that point, coverage would be reduced to intermediate data alone. Note that transparency indicates whether for a step of the pipeline data and/or code are available, whereas coverage has to do with how many steps of the pipeline are available.

Reproducibility Methodology. There are two main modes of creating reproducible experiments: (i) one can *plan for reproducibility* while doing research, or (ii) one can make the experiments reproducible *after the fact*, i.e., after they have been developed on a source machine. For the former, there are both best practices (e.g., the use of version control systems) as well as specialized tools that systematically capture provenance as the experiments are carried out. One advantage of this mode is that not only it ensures that the derived results are reproducible, but also this provenance captures the different trails followed, and can serve as a guide to researchers who want to understand the different choices that were examined [10], e.g., data input and parameter value combinations, or different algorithms used. However, when appropriate tools are not available, or they are simply not adopted, it should still be possible to make experiments reproducible, which is the goal of the second mode, called *unplanned*, or spontaneous. For a given set of results, this entails the creation of an executable description of how the results were derived.

3 The Reproducibility Benchmark

Desiderata. We had a number of goals while designing this benchmark. Besides verifying the ability of tools to perform the different reproducibility tasks, we would also like to understand how well these are supported and under which conditions. Thus, to serve as the basis of the benchmark, it is critical to use an experiment that is representative and covers many of the steps that are common in scientific exploration. Furthermore, the experiment must be expandable, in the sense that it should be possible to tweak it to match different reproducibility methodologies, scenarios (e.g., single and multiple machines), and levels of reproducibility. Last, but not least, to be inclusive, the experiment should also run on different multiple OS platforms.

The Experiment. We selected a computational experiment that includes both the execution of a simulation and the analysis of its results: Monte Carlo simulation of the Ising model, described in detail in [18]. The experiment pipeline consists of three main steps:

1. *Simulation Phase.* First, large-scale simulations are prepared and executed, resulting in the raw simulation output. This phase is commonly time-consuming and not easily reproducible by a reader of the paper. Thus, the output is often archived, as well as all steps to reproduce this data.
2. *Evaluation Phase.* Next, the data is analyzed and evaluated. As an example, the data is plotted in figures that allows readers to judge the quality of the derived results.
3. *Publishing Phase.* Finally, both the figures and other results are included in a manuscript.

Such workflows are common in many scientific domains and exercise important requirements for the different reproducibility tasks. The experiment can run on Windows, Linux and OS X, and it can also be tuned to use different computational environments (e.g., single machine and cluster).

Reproducibility Modes and Evaluation Criteria. To consider the two reproducibility methodologies and the tools that handle them, the benchmark has two categories: *unplanned*, which tests solutions for making an existing experiment reproducible, even though the experiment was never designed with reproducibility in mind; and *planned*, which simulates the creation of an explicitly reproducible experiment from scratch. In both cases, the goal is to evaluate the completeness of the tool with respect to the reproducibility tasks, as well as the transparency of the derived experiment. Additional features that are tested include usability, the ability to run on multiple platforms and on a remote cluster, and the inclusion and automatic update of results in the paper. For the planned mode, the benchmark includes binary and source code, and a textual description of the experiment. Users will utilize this information to simulate the creation of the experiment from scratch. The spontaneous benchmark provides binary code which runs on a particular linux environment. Tools will need to wrap these binaries after one execution of this binary code in the source environment. The wrapped package must then be able to run in a target environment and create an executable paper. Here, we focus on the spontaneous mode.

There will be two kinds of users who deal with this benchmark. *Authors* will create a reproducible experiment using tools, and *reviewers/judges* will unpack the experiment and look at results for different parameter combinations and input files.

1. Author role. The author will make the experiment reproducible by *wrapping* to run on various platforms, after running it on a source platform (or a virtual machine or set of virtual machines representing the source platform). Multiple variations of the experiment are provided. One of them requires execution on a cluster of computers. The experiment can also be tuned to generate different numbers of results, allowing the scalability of the systems to be assessed.

2. Reviewer role. Judges/reviewers will determine usability: the tool-produced package should be easy to unpack, run the experiments, and explore parameter variations. In addition, since reviewers may have to copy the experiments, it is important to consider (and measure) the size of the package. For example, while virtual

machines provide great portability, they can be very large – much larger than packages derived from tools that include only the required dependencies [4, 12]. More important perhaps, virtual machines might capture information the code authors do not wish to reveal.

4 The Spontaneous Reproducibility Challenge

For the spontaneous challenge, there will be a training and a real experiment (binaries and data). The training set will be used by tool developers while preparing their tools for the challenge. The real set will be sent well in advance but encrypted. The decryption key will be sent the day the competition begins. Also, for each challenge, there will be both a tool-building portion and a blinded judgement portion. To create reproducible documents, there will have to be some integration with typesetting packages such as LaTeX or text editors such as Microsoft Word. Since there is no general tool that supports all the reproducibility tasks, we envision that tool builders will form alliances in order to win the benchmark competition.

Besides the evaluation criteria outlined in Section 3, another criterion for the tool-builder part is a timing test: how long does it take for the team to wrap the code and data for each target system? Scoring works as follows assuming N participants: the fastest team for a given target system receives N points, the second fastest receives N-1, ... Any team that does not succeed in wrapping the code for a particular target system receives zero points for that target.

The wrapped code will be uploaded to a judge site where it will be blinded by the chair of the competition. Each wrapped code will then be judged by three judges on the following criteria:

1. Given a configuration file in text format (using the same format for training and test), run through every combination of parameters and calculate the mean and the variance of some parameter (e.g., critical temperature in the case of the Ising model). In the configuration file the possible values of each parameters will be provided. If this works, then the team receives N/2 points. Otherwise zero points.

2. The size of the package that runs on the target machine. The smallest package of those that run receives N points, the second smallest N-1, etc. If the software does not run, then the team receives zero points.

3. How long does it take the judges to run the tool in wall clock time with new parameter settings, generate all the graphs and numerical values, and embed these into a new version of the paper? The fastest team receives N points, the second fastest receives N-1, ... If it doesn't work, then 0 points.

4. Can the judges interact with the paper and change parameters and/or input data and have the figures update without further special action? Any team that can do this receives N points. Otherwise 0.

5. The judges subjective score on a scale of easy (3) to difficult (0). The three judges' scores are summed and then the team with the highest score receives N points, the second highest N-1, ...

The team with the highest score will receive a monetary prize. If more than one team ties for the most points, the prize will be divided equally. All participants will have the option to write an article about their effort in the journal Information Systems.

Acknowledgments. This work was partially supported by the Sloan Foundation, and the National Science Foundation under grants CNS-1229185, IIS-1139832, IIS-1142013 DBI-0445666, DBI-0421604, and N2010 IOB-0519985. We thank M. Dolfi, J. Gukelberger, A. Hehn, J. Imriska, K. Pakrouski, T. F. Rnnow, M. Troyer, and I. Zintchenko for designing and implementing the experiment used in our benchmark.

References

[1] Biostatistics Journal – Information for Authors, 2013. http://www.oxfordjournals.org/our_journals/biosts/for_authors/msprep_submission.html.

[2] P. Bonnet, S. Manegold, M. Bjørling, W. Cao, J. Gonzalez, J. Granados, N. Hall, S. Idreos, M. Ivanova, R. Johnson, D. Koop, T. Kraska, R. Müller, D. Olteanu, P. Papotti, C. Reilly, D. Tsirogiannis, C. Yu, J. Freire, and D. Shasha. Repeatability and Workability Evaluation of SIGMOD 2011. *SIGMOD Rec.*, 40(2):45–48, Sept. 2011.

[3] P. Bonnet, S. Manegold, M. Bjørling, W. Cao, J. Gonzalez, J. Granados, N. Hall, S. Idreos, M. Ivanova, R. Johnson, D. Koop, T. Kraska, R. Müller, D. Olteanu, P. Papotti, C. Reilly, D. Tsirogiannis, C. Yu, J. Freire, and D. Shasha. Repeatability and Workability Evaluation of SIGMOD 2011. *SIGMOD Rec.*, 40(2):45–48, Sept. 2011.

[4] F. S. Chirigati, D. Shasha, and J. Freire. Packing experiments for sharing and publication. In *SIGMOD Conference*, pages 977–980, 2013.

[5] D. Donoho, A. Maleki, I. Rahman, M. Shahram, and V. Stodden. Reproducible Research in Computational Harmonic Analysis. *Computing in Science & Engineering*, 11(1):8–18, Jan.-Feb. 2009.

[6] Elsevier's Executable Paper Grand Challenge – 2011. http://www.executablepapers.com/.

[7] S. Fomel and J. F. Claerbout. Guest Editors' Introduction: Reproducible Research. *Computing in Science and Engineering*, 11(1):5–7, 2009.

[8] M. H. Freedman, J. Gukelberger, M. B. Hastings, S. Trebst, M. Troyer, and Z. Wang. Galois Conjugates of Topological Phases. *Phys. Rev. B*, 85:045414, Jan 2012.

[9] J. Freire, P. Bonnet, and D. Shasha. Computational Reproducibility: State-of-the-Art, Challenges, and Database Research Opportunities. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 593–596, New York, NY, USA, 2012. ACM.

[10] J. Freire, D. Koop, E. Santos, and C. T. Silva. Provenance for computational tasks: A survey. *Computing in Science and Engineering*, 10(3):11–21, 2008.

[11] J. Freire and C. Silva. Towards Enabling Social Analysis of Scientific Data. In *ACM CHI Social Data Analysis Workshop*, 2008.

[12] P. Guo. CDE: A Tool for Creating Portable Experimental Software Packages. *Computing in Science and Engineering*, 14(4):32–35, 2012.

[13] IEEE Transactions on Signal Processing – Reproducible Research, 2013. http://www.signalprocessingsociety.org/publications/periodicals/tsp/.

[14] S. Manegold, I. Manolescu, L. Afanasiev, J. Feng, G. Gou, M. Hadjieleftheriou, S. Harizopoulos, P. Kalnis, K. Karanasos, D. Laurent, M. Lupu, N. Onose, C. Ré, V. Sans, P. Senellart, T. Wu, and D. Shasha. Repeatability & Workability Evaluation of SIGMOD 2009. *SIGMOD Rec.*, 38(3):40–43, Dec. 2010.

[15] I. Manolescu, L. Afanasiev, A. Arion, J. Dittrich, S. Manegold, N. Polyzotis, K. Schnaitter, P. Senellart, S. Zoupanos, and D. Shasha. The Repeatability Experiment of SIGMOD 2008. *SIGMOD Rec.*, 37(1):39–45, Mar. 2008.

[16] J. P. Mesirov. Accessible Reproducible Research. *Science*, 327(5964):415–416, 2010.

[17] Nature Magazine Policy – Reporting Checklist for Life Sciences Articles, 2013. http://www.nature.com/authors/policies/checklist.pdf.

[18] The Reproducibility Challenge. http://www.reproduciblescience.org/index.php/Reproducibility_Challenge.

[19] E. Santos, J. Freire, and C. Silva. Information Sharing in Science 2.0: Challenges and Opportunities. In *CHI Workshop on The Changing Face of Digital Science: New Practices in Scientific Collaborations*, 2009.

[20] Science Magazine – Submission of Supplementary Materials, 2013. http://www.sciencemag.org/site/feature/contribinfo/prep/prep_online.xhtml.

[21] SIGMOD Experimental Repeatability, 2012. http://www.sigmod.org/2012/reproducibility.shtml.

[22] C. Tenopir, S. Allard, K. Douglass, A. U. Aydinoglu, L. Wu, E. Read, M. Manoff, and M. Frame. Data Sharing by Scientists: Practices and Perceptions. *PLoS ONE*, 6, 06/2011 2011.

[23] J. Tohline and E. Santos. Visualizing a Journal that Serves the Computational Sciences Community. *Computing in Science Engineering*, 12(3):78–81, 2010.

[24] VLDB Experimental Reproducibility, 2013. http://www.vldb.org/2013/experimental_reproducibility.html.

Data Engineering

It's FREE to join!

TCDE

tab.computer.org/tcde/

The Technical Committee on Data Engineering (TCDE) of the IEEE Computer Society is concerned with the role of data in the design, development, management and utilization of information systems.

- Data Management Systems and Modern Hardware/Software Platforms
- Data Models, Data Integration, Semantics and Data Quality
- Spatial, Temporal, Graph, Scientific, Statistical and Multimedia Databases
- Data Mining, Data Warehousing, and OLAP
- Big Data, Streams and Clouds
- Information Management, Distribution, Mobility, and the WWW
- Data Security, Privacy and Trust
- Performance, Experiments, and Analysis of Data Systems

The TCDE sponsors the International Conference on Data Engineering (ICDE). It publishes a quarterly newsletter, the Data Engineering Bulletin. If you are a member of the IEEE Computer Society, you may join the TCDE and receive copies of the Data Engineering Bulletin without cost. There are approximately 1000 members of the TCDE.

Join TCDE via Online or Fax

ONLINE: Follow the instructions on this page:

www.computer.org/portal/web/tandc/joinatc

FAX: Complete your details and fax this form to **+61-7-3365 3248**

Name _____

IEEE Member # _____

Mailing Address _____

_____

Country _____

Email _____

Phone _____

TCDE Mailing List

TCDE will occasionally email announcements, and other opportunities available for members. This mailing list will be used only for this purpose.

Membership Questions?

Xiaofang Zhou
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, QLD 4072, Australia
zxf@uq.edu.au

TCDE Chair

Kyu-Young Whang
KAIST
371-1 Koo-Sung Dong, Yoo-Sung Ku
Daejeon 305-701, Korea
kywhang@cs.kaist.ac.kr

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398