Push-based System for Molecular Simulation Data Analysis

Vladimir Grupcev, Yi-Cheng Tu
Department of Computer Science and Engineering
University of South Florida
4202 E. Fowler Ave., ENB 118
Tampa, FL 33620, U.S.A.
Email: {vgrupcev, tuy}@mail.usf.edu

Joseph Fogarty, and Sagar Pandit
Department of Physics
University of South Florida
4202 E. Fowler Ave., ISA 2019
Tampa, FL 33620, U.S.A.
Email: jcfogart@mail.usf.edu, pandit@usf.edu

***Abstract*—Many scientific fields generate, and require manipulation of big data. Known scientific data analysis systems, as well as traditional DBMSs, follow a pull-based architectural design, where the executed queries mandate the data needed. This design, while suitable for traditional transaction-based workloads where number of queries retrieve small parts of data located at various places of the database, is ill-fitted for applications involving complex analysis on most of the data. Such design involves redundant and random I/O, considerably affecting the data throughput in the system. In this paper, we design and implement a push-based type system that allows high-throughput data analysis in the process of scientific discovery. Our design improves throughput in two ways: i) it uses a sequential scan-based I/O framework that loads the data into the main memory, and then ii) the system pushes the loaded data to a number of pre-programmed queries. By this way the system lowers the unnecessary I/O overhead imposed by the randomized, index-based scan and that of a multiple data reads if each query were to be fed separately. Considering the amount of data and the number of executed queries, we believe our system provides substantial improvement over the current data analyzing systems. The efficiency of the proposed system is backed by the results of extensive experiments using real MS data. The running times of our system are compared to those of the GROMACS system. The comparison shows the advantage and the potential of using such push-based system for data system analysis.**

***Index Terms*—Big data, push-based system, molecular simulation, scientific databases, spatial distance histogram.**

I. Introduction

The amount of data generated by scientific applications running on advanced computer simulation systems is growing rapidly. This big data is imposing a significant stress on the data analysis software. Even though the existing data analysis systems are configured to deal with large volume of data, they are not optimized for high throughput data analysis. Often times, the data is available only for certain period of time (e.g., streams) so the analyzing system should react to it in a very efficient way. Another challenge imposed to the data analysis systems is that the data is usually being accessed through high-level analytical queries which computation is a lot more complex than the computation of simple aggregates. Some of these queries are usually the bottleneck of the analysis systems, because they take a lot of time (often many days) to be executed and the systems are not designed to handle multiple queries on the same data stream at the same time, thus decreasing the overall efficiency of the data analysis.

One such example, where fast data analysis is crucial involves online social media [1]–[3]. The social media is known to be used for real-time feedback collection, which is usually associated to a particular topic or product [4]. In order for a system to be able to perform analytical examination of the data produced in such streaming media, the system should have the capability of fast data access. The reason, the millions of data records produced every second [5]. Moreover, these records may have different geographical origin, introducing different languages and forms and often times containing unsolicited messages, errors, malicious content, etc. Therefore, some low level data uniformity and cleaning on top of the data access and management issues should be considered and possibly incorporated in the process of analytical investigation in order to achieve relevant result.

The aforementioned big data imposed issues are also frequently present in many other application fields, e.g., scientific data analysis. In many scientific fields, the main computational method for analyzing physical/chemical attributes of natural systems is the particle simulation. Such simulations, when performed in the field of structural and molecular biology, are commonly called Molecular Simulations (MS). MS are computer simulations of complex biological, physical or chemical structures. These simulations, based on some basic theoretical models, are widely utilized as a basic research mechanism for studying the behavior of the natural systems. The basic units of such systems are natural particles (such as atoms, molecules, stars, etc.) and they interact among each other for a certain period of time following postulated classical forces. The number of these basic units is big, usually in the range of hundreds of thousands to millions. For instance, a single simulated snapshot of a collagen fiber may consists $890,000$ particles. And oftentimes, the simulations produce datasets consisting of more than one snapshot (frame) of the system's state at various time instants. Each of these frames contain all the particles, together with all their measurements, such as spatial coordinates, mass, charge, velocity, forces, etc. Usually, a big number of such frames (in the tens of thousands) are being produced and stored in the course of a

typical simulation. Quantities measured during the simulations are analyzed to test the theoretical model [6], [7]. In short, the MS is proven and powerful tool for understanding the inner-workings of a biological system.

Scientist must analyze the data produced by the MS for scientific discoveries. Such analysis often comprises of computation of very complex quantities that show statistical properties of the data. Such queries are of great importance to scientist because they are the basic assembly blocks for a series of critical quantities needed to outline the scientific systems [6]. The methods by which the big data is being accessed as well as these queries are executed can either increase or decrease the efficiency of the system. The known data analysis systems (e.g., GROMACS [8]) use pull based type design in which the data is being fed to the queries only by demand. When a query needs the data, it requests it, the data is pulled through the system and delivered to the query. This type of design, we believe, introduces two types of issues: (1) overhead in I/O traffic by not allowing multiple queries to be executed on the same data stream of data, and (2) cpu/data latency incurred when a request for the data is sent to the system.

We believe a push-based type system can remedy such issues. In the such design the queries do not request the data. Rather, the data is pushed onto the queries automatically and is being processed by the active queries. This alleviates the I/O traffic overhead as well as the cpu/data latency incurred for each query that would have been introduced by a pull-based design. Such a system design is especially suitable for many scientific applications which share the following features. First, the scientific analysis often involves executing a number of analytical queries commonly processed on a large portion, if not all of the generated data. Second, many of these fields use the same popular, and small number of data analytics primitives as the basic blocks on which they build their discovery. And three, the scientific data, once stored is never modified. New data only appends to the existing dataset, making the data perfect contender for streaming.

In this paper, we describe the idea, design and implementation of such system. We would like to point out that the work presented in this paper is an application of a push-based strategy that has been previously considered in the database community in the works presented in [9]–[11], as well as the idea of sharing same tuple of data or data scans across many queries previously introduced in [12]–[14].

To summarize, data-intensive applications often require big amount of storage space and intensive processing capability, but also need fast data access and high throughput data analysis. Therefore, the need of a system that will be optimized to access big data fast, with high throughput, as well as efficiently execute the analytical queries with a possibility of running multiple queries on the same data stream is of a great importance to the scientific community.

A. Problem Statement

In order to describe the scientific system, scientists must analyze critical statistical properties of the data produced by the simulation. These properties are usually computed through series of queries that are being executed on the whole, or some selection of the MS data. Most of the MS analysis systems widely used today are part of a software systems that also run the simulation. Such examples include GROMACS [15], VMD [16], MDAnalysis [17], Wordom [18], MD-TRACKS [19], SimulaidOne [20], Charmm [21]. Once the simulation is run, the system produces a flat file with all the measurements of each particle. These systems take each query as a user's input and apply them to the data uploaded from these flat files (called trajectories in GROMACS). The biggest issue that these system impose is the fact that for every new query issued by the user, the system has to load a significant part of the data into main memory before executing the query. Such pull-based design involves random I/Os that considerably affect the data throughput. And while the use of an index-based scan might seem better option than a sequential scan, this is the case only when the query accesses small portion of the data. Considering the volume (gigabytes) of data produced by the MS in a single frame and the large number of frames simulated (tens of thousands), the total volume of data to be analyzed is of huge magnitude. Loading big chunks of data from disk to the main memory every time a query is being executed generates redundant I/O thus greatly degrades system performance. In this paper, we design and implement a push-based system which allows high-throughput data analysis by avoiding random and redundant I/Os.

B. Our Approach

To remedy the aforementioned problems of pull-based systems, our idea is to build a system that would load the MS data from hard disk in the main memory only once and then execute as many queries as needed/wanted on that data, without the need of reloading the same data over and over again. Considering the volume of data in a single MS frame as well as the number of frames produced during a single simulation, this type of system can save a lot of time for MS analysis. The system would contain many, most often used queries that can also be run as user's input. These queries would be pre-programmed as separate modules in our system, but they would be able to take certain attributes, like data selection for instance, as user's input. Once the system is run, all (or a selection) of these queries would be executed against the MS data or certain selection. The system would act as a type of push-based system, essentially pushing the loaded data to all the queries. Once we have the data in the memory, the reading can loop as many times as the queries need it. This is useful for some non-streaming functions/queries that cannot be computed in a single run of the data. Such one query is the SDH. But we have also come up with a way to resolve this in a way, by creating a quad-tree based database for this query. This quad-tree (we called Density Map (DM)) is being populated while the data is being loaded into main memory. After the DM is populated, we can compute the SDH with much faster algorithm than the brute-force method used everywhere else.

Another improvement that our system provides is the following: by thorough observation we have discovered that many of the quantities computed in the analysis of the MS system share some of the basic parts. In order to take advantage of that, our systems precomputes these basic parts and has them handy (stored in memory) whenever the more complex functions need them. So, for instance, when the total mass of the system is needed, our system just pulls the precomputed value out and serves the query needing it. And if more than one query uses the same basic part, that translates into time saving, however minuscule it might be.

C. Contributions and roadmap of the paper

We have designed and implemented a push-based system that can be used as a tool for MS systems analysis. We have tested the system on real MS data. The experimental results show the superiority of the proposed system over one of the most widely used MS analysis system, i.e., GROMACS. The efficiency improvement ranges anywhere from 3 to 100 times improvement (of the overall running time) per set of selected queries run on a different selection of the MS data. If we take in account the number of such queries that the scientist run every day, we believe our system would be of great benefit to MS community. In addition to the higher efficiency that our system achieves, we also believe that our idea will initiate a new breed of database related, push-based systems that could be used to analyze the MS systems in a more efficient way. Having a push-based like system, we were also able to improve the computation of the more complex quantities. Namely, the computation of some such complex quantities involves computation of much less complex functions (sub-parts), and many such sub-parts are being shared among some of the complex quantities. So, by computing these more basic, sub-quantities in advance, we were able to achieve however minuscule time gain when computing the more complex queries.

The major technical contributions presented here are:

- Design the network (tree like) of the most commonly used queries in MS (physics);
- Build the system: design and build the modules, representing the quantities to be computed in an efficient manner, following the already built network;
- Develop a scientific simulation database benchmark that can be used for evaluating similar systems and products.

The remainder of this paper is arranged as follows: In Section II we give a survey of related work; In Section III, we show the design of the query network built from the most widely used quantities in MS system analysis; In Section IV, we describe our push-based system for MS data analysis; In Section V, we present the benchmark designed to test our system and results of comprehensive set of experiments; At the end, we conclude this paper by Section VI.

II. Related Work

The idea of data streaming has been broadly used in many fields. The main usage however is aimed at processing live data generated online. There are many references for data stream management, but we believe the presentation in [22] encapsulates the majority of the ideas, problems and solutions. In the past decade, however, the database community started to follow the data stream idea to process stored data. Processes can take advantage of the streaming data at any time the data is being pushed through the system. This gives rise to push-based design for data management systems. The idea of such push-based design was previously considered in projects such as DataPath [9], Volcano [10] and QPipe [11] among others. These works show ideas in which the data-driven dataflow is compared to the demand-driven dataflow showing the need for the later. They also talk about maximizing the data and work sharing among queries at runtime. Essentially, we incorporate such ideas in our design of the push-based system.

On the other side, the scientific community has steadily progressed from processing massive data files towards employing database systems for the storage, acquisition, and analysis of big scientific data [23], [24]. The widely used and popular relational database systems are conventionally designed and optimized to better manage the data produced by the business type applications. But such database systems (DBMS) are not well equipped to deal with the type and quantity of scientific data produced by molecular simulations. In the recent past, the DBMS community has made some attempts to design database systems optimized for handling scientific data. Such examples include the BDBMS project [25] that deals with annotation and provenance of the sequence data in biosciences, and the PeriScope project [26], designed to efficiently handle declarative queries against bio sequences. On top of the aforementioned examples, there are also ideas for new DBMS frameworks aimed at the management of scientific data [27]–[29]. One of those systems is the SciDB [29], [30] and it is closest to the idea presented in this paper. SciDB is data management and analytics system that is primarily used in application domains involving very big scale array data. This system, like the one presented in this paper is designed around a multi-dimensional array data-model and it uses arrays to store the data. SciDB stores petabytes of data on a number of machines and runs its queries on those machines. It is made for high performance, high-availability, fault tolerance, and scalability. However, to the best of our knowledge, it too follows the pull based design. As mentioned earlier, this type of design can impose I/O overhead and decrease the data throughput. Aside the mentioned issue, the design and build of such DBMS optimized for scientific data management come with a additional challenges. Such challenges as well as their probable resolution are outlined in [31]. Recently, there have been some efforts aimed at designing and building MS data managements systems on top of relational databases. Such efforts are presented through projects like BioSimGrid [32] and SimDB [33] that were developed especially for molecular simulations. However, to the best of our knowledge, such systems still lack the efficiency needed for MS data management as well as efficient query processing strategies.

Generally, the data produced in the process of molecular simulation is being stored in large, plain files with no structure.

Queries are executed onto such files producing the quantities that scientists use to analyze the molecular system. Such simulation and analysis packages include: GROMACS, VMD, MDAnalysis, Wordom, MD-TRACKS, SimulaidOne, Charmm among the others. But to the best of our knowledge, all of these systems work on a similar basis: they take a user defined query and execute it. For the query to be executed, the data has to be loaded into the main memory. Then the result is either produced onto the display or written to a file. When the next user query comes, the system again loads significant part of the data into memory and executes the query. We believe that there is a room for improvement of such systems, given the fact that many of the user defined queries executed during system's analysis are fairly static. In other words, there is a number of queries that a user would always want to execute on a given simulation data. Furthermore, the selections of MS data onto which such queries might be executed, are also fairly constant (i.e., oftentimes the user selects the same group of atoms (e.g., all hydrogen atoms) to calculate given quantity). So, by running such queries automatically once the system has loaded the data into memory, we believe we can save a lot of time that otherwise would have been spent in loading the same data into main memory anytime a query is executed. Similar idea for running queries in advance and reusing results has been proposed in [34], where they preemptively execute queries that are predicted to be useful for the user based on user's history of executed queries. In contrast to that, we build the network of mostly used queries and a benchmark that can serve to test data analysis systems. Our system can also take user's queries as input as well. We believe our system is an improvement over the MS analysis systems that are used today.

III. Network of Queries

MS Queries. To study some important statistical features of an MS system, scientists need to "extract" various statistical quantities out of the data. To achieve this, queries are executed against the data and most of these queries are analytical in nature. Such queries are mathematical functions that translate a selection of atoms (atoms' measurements) to a scalar, vector, a matrix, or a data cube [33]. Once the simulation is done, the analysis carried out will depend on the structure being studied as well as the features of the system that need exploring. Some of the more popular queries, including density (atom counts), first-order statistics (mean), second-order statistics (variance), and histograms among others, can be seen in Table I. The queries shown in this table are the ones that we have also incorporated in our system. We assume that the MS system comprises of n particles and r_i, m_i, c_i and q_i denote coordinates, mass, charge, and number of electrons of a particle i, respectively.

There are two types of functions used to analyze an MS system. The first are one-body functions, which are usually algebraic [23] and only involve quantities(attributes) from a single atom. Each atom is being processed a constant number of times, thus bearing a linear (total) running time. This type of functions is very suitable for push-based system in which the data is being read once and acted upon. In a single run though the data all such queries will produce final results. Except the SDH and the RDF, all other queries shown in Table I fall into this category. Most of these functions are defined on a single frame of the MS data. Only the autocorrelation functions are defined on two distinct frames.

Function Name	Equation/Description
Moment of Inertia	$I = \sum_{i=1}^{n} m_i r_i$
Moment of Inertia on z axis	$I_z = \sum_{i=1}^{n} m_i r_{zi}$
Sum of masses	$M = \sum_{i=1}^{n} m_i$
Center of mass	$CoM = \frac{I}{M}$
Radius of Gyration	$RG = \sqrt{\frac{I_z}{M}}$
Dipole Moment	$D = \sum_{i=1}^{n} q_i r_i$
Dipole Histogram	$D_z = \sum_{i=1}^{n} \frac{D}{z}$
Electron Density	$ED = \frac{\sum_{i=1}^{n} (e_i - q_i)}{dz \cdot x \cdot y}$
Heat Capacity	$HC = \frac{3000 \cdot \sqrt{T} \cdot boltz}{2 \cdot \sqrt{T} - n \cdot df \cdot VarT}$
Isothermal Compressibility	$I = \frac{VarV}{V_{avg} \cdot boltz \cdot T \cdot PresFac}$
Mean Square Displacement	$msd = \langle (r_{t+\Delta_t} - r_t)^2 \rangle$
Diffusion Constant	$D_t = \frac{6 \cdot msd(t)}{t}$
Velocity Autocorrelation	$V_{acor} = \langle (V_{t+\Delta_t} \cdot V_t) \rangle$
Force Autocorrelation	$F_{acor} = \langle (F_{t+\Delta_t} \cdot F_t) \rangle$
Density Function	Histogram of atom counts
SDH	Histogram of all distances
RDF	$rdf(r) = \frac{SDH(r)}{4 \cdot \pi \cdot r^2 \cdot \sigma_r \cdot \rho}$

TABLE I: Popular analytical queries in MS

The second type of functions are multi-body functions that are holistic in nature. Their computation involves more than one atom's attributes and usually cannot produce final result in a single run of the data. Such queries include the Radial Distribution Function (RDF) [6], [35], [36] as well as some quantities associated with chemical shifts [37]. Generally, such functions are computed through histograms. The RDF is obtained from a histogram of all pairwise atom distances (this is the Spatial Distance Histogram or SDH). The straightforward (often brute-force) way of computing these functions is a very time consuming process. Also, these methods cannot produce the final result in a single run of the MS data, making such functions unsuitable for our idea of a push-based system. However, in our previous work, we have designed a data structure together with an algorithm that opens up the possibility for such queries to be executed in a push-based type environment. Further details on this are given in Section IV.

Fig. 1 represents an idea to show how such query processing system can be improved. The idea behind it is that some of the queries share sub-routines. Having all the queries made as separate modules, these sub-routines can be computed once the

data is being pushed through the system and then used when needed. Having in mind the amount of data in a single frame that the queries need to go through, and the huge number of frames in MS, we believe this can be immense improvement in terms of total running time.

IV. Building the system

A. MS Data retrieval and in memory organization

A typical MS system generates and stores the data in a number of trajectory files, including multiple frames(snapshots in time). Such MS generated data oftentimes goes through a simple lossless compression and, depending on the simulation software it may be stored in a binary format. Such trajectory file format is one of most often used MS file format (e.g., GROMACS, PDB). But such format is unrecognizable to our system. So our system has to do three things before it starts executing the queries: 1) Read the MS data from a trajectory file, 2) Translate the MS data to a form recognizable to our system, and 3) Load the data to memory.

Data read - transform. The MS data is stored in multiple files and possibly in different formats as well. One such file holds the global data (identifying the system and the simulation). Another file holds each frame's data which contains general information about the frame, but the main part is a sequential list of each atom's info, including atom's mass, position, charge, number of electrons, velocities, forces, etc. Another file (topology file) holds the molecule info, essentially identifying what atom belongs to which molecule.

To extract the data from these files, we have created an "extractor/transformer" of the attributes needed for the execution of the queries in our system. This transformer is a separate piece of code that does three things: 1) it reads the MS generated data; 2) it translates the data into a format that our system can read (taking only the information our system needs); and 3) it stores the data in a file that has basic structure to it. So, in the end, the data transformer produces a data file that our system takes as input. This code serves as a connection between an MS system (e.g., GROMACS) and our system. With this, our system can essentially be used as an add-on to GROMACS or other simulation systems and help improve the efficiency of the data analysis.

Data organization in main memory.[1] Once the data is in a format our system can read, it is being loaded into the main memory one frame at a time. The organization of the in-memory particle's data is in the form of a simple, two dimensional array where a single row represents an atom in the system with all its attributes. We also keep (in a one dimensional array) crucial system's information for each frame, like temperature, energy, pressure, etc. We have used such structures because they are very suitable for simulating a push-based type of system: a simple sequential read of the array gives that on-line type of data stream. So, as the system reads the array, it pushes the data onto the query-modules. The one-body (algebraic) queries will produce a final result at the end of the first sequential read. However, the two-body (holistic) functions, like SDH and RDF, require multiple scans of the data.[2] Therefore, each frame's data array can be continuously read (in a loop manner) as many times as needed.

B. Query modules

There are two types of queries in MS data analysis: algebraic or one-body, and holistic or two-body queries.

1) One-body queries: Most of the query modules in Table I are not complex and only contain computations of simple one-body functions. These queries were coded as separate modules in our system. Each of these modules take few attributes as input (e.g., atom selection, frames selection(for the autocorrelation functions), number of atoms, etc.). The system pushes the data as it becomes available onto these modules. The queries are being executed on the selection and are put in a "ready" mode, awaiting the next frame's data. First, the more basic queries, like total mass, are being computed. The results of such queries are temporary stored (in main memory) and are available for use anytime a more complex query needs them.

2) Two-body queries: In general, queries involving two-body functions are more complex and cannot provide the final result in a single data read if a straightforward method is used for their computation. However, in the proposed system we have incorporated a data structure and an algorithm for the SDH (also RDF) from our previous work that is suitable for push-based type of system. In this subsection we give a brief description of the data structure and the algorithm designed in [38], [39] and implemented in the system proposed in this paper. For more detailed information, refer to our previously published work on this topic [38], [39]

The data structure. The data space is represented by a conceptual data structure we named Density Map (DM). The density map splits the simulation space into a grid of equal size regions (or cells). The cells are cubes in 3D and squares in 2D (we use 2D data to elaborate and illustrate the proposed ideas in this paper). Resolution of a density map is the reciprocal of the cell size in that density map. In order to generate higher resolution density map, we split each cell of the current resolution's grid into four smaller cells of equal size. This design allows us to use a region quad-tree [40] to organize density maps of the same data but with different resolutions. A node in the quad-tree represents a single cell from the DM. Therefore, a density map of a certain resolution is the set of all nodes of one level of the tree. Each tree node records the cell's location in the density map (coordinates of corner points) as well as the number of particles in each cell. We name the afore describe tree the Density-Map tree (DM-tree).

The algorithm. The main part of the DM-SDH algorithm is a procedure named ResolveTwoCells whose inputs are

[1]This paragraph talks only about data organization for one-body queries. That of two-body queries (e.g., SDH) is discussed later on.

[2]However, we have previously deigned and created a data structure and an algorithm that can take the advantage of a single data read and produce final results for SDH computation [38], [39]. We have incorporate this into our system presented in this paper.

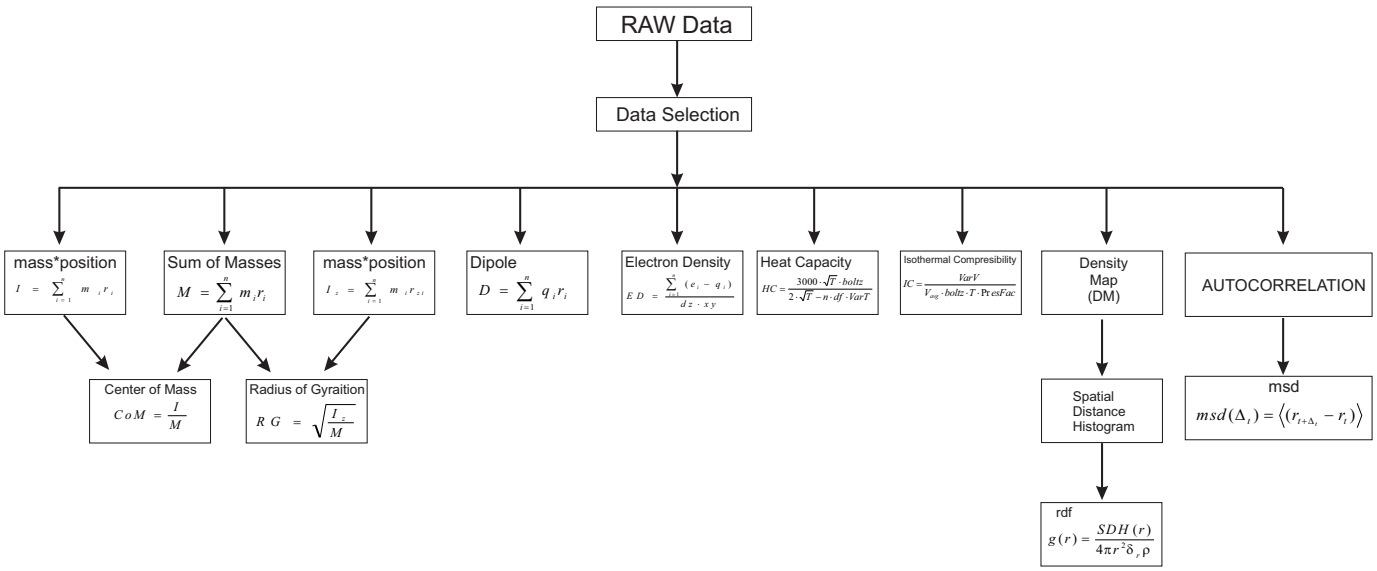

Fig. 1. MS Modules Structure

two cells from the density map. It computes, in constant time, the minimum and maximum distance between the two cells. A pair of cells is *resolvable* if both the min and max distance between them fall into the same SDH bucket i. If so, the distance count of bucket i increases by $n_A n_B$ (n_A and n_B are the number of particles in cell A and B, respectively). Otherwise, the cells are non-resolvable and we either:

(1) Go to the next density map with higher resolution and resolve all children of A with those of B, or
(2) If leaf-level has been reached: compute every distance between particles of A and B and update the histogram accordingly.

In order to generate the complete SDH, the RESOLVETWO-CELLS procedure is executed for all pairs of cells for a given density map DM_k and the algorithm would recursively call the procedure (action (1) above) until leaf-level has been reached (action (2) above).

On top of the DM-SDH algorithm, we have also incorporated two approximate SDH algorithms (ADM-SDH), introduced and described in [39], and [41]. These approximate algorithms are substantially faster than the the brute-force algorithm and also than the DM-SDH algorithm as they take advantage of some heuristic. For more details on the ADM-SDH algorithms please see the aforementioned work.

C. Working of the system

Here we give a brief overview of how the system works at runtime. Note that data transformation is only executed once. Here are the steps (Fig. 2) taken through out the analysis:

(1) Execute the data transformer
  (a) Read the MS data from trajectory files
  (b) Extract the info needed for our system
  (c) Save the read data to a file recognizable to the system
(2) Load the data into main memory (one frame at a time)
  (a) Load data into a double array (for one-body queries)
  (b) Load data into the quad-tree structure (for two-body queries:SDH, RDF)
(3) Push the data to all queries
(4) A query, if available, acts upon the pushed data (first executing the lower level, sub-queries)
(5) Store intermediate results (results of sub-queries)
(6) Repeat steps 3-5 if needed.
(7) Output results
(8) Go to step 2 and load the next frame (if needed).

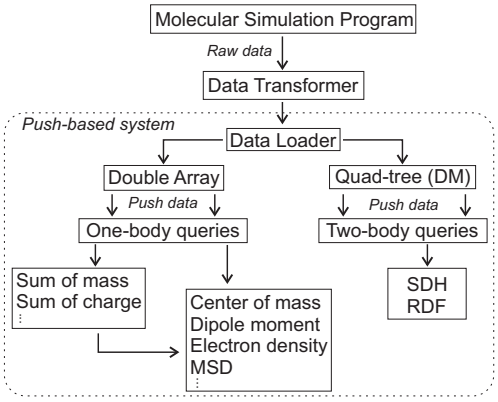

Fig. 2. Push-based system flow

Step 2, loading the data into main memory is different for SDH query compared to the one for the one-body queries. The reason is the different data structure used to store the data in memory. While we use double array to store the data for the one-body queries, we use quad-tree like data structure to store the data needed to compute the SDH. The loading to the double array is straightforward. However, to load the quad-tree structure, we need to use some of the info from the data itself. Namely, the coordinates of the atoms are used to determine in which tree node an atom belongs. That way we

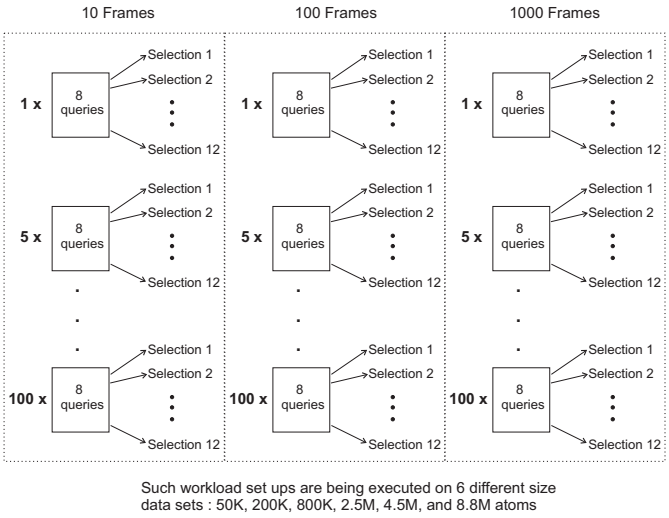

Fig. 3. Workload setup

build the density map. To solve the SDH problem in a push-based manner, we convert the problem into populating a data structure in push-based manner. This data structure will then be used as an input to our DM-SDH algorithm that, although not completely in "on-the-fly" way, is a great improvement over the naive methods used in much of today's MS analysis systems (i.e., GROMACS, PDB, CHARMM, etc.).

V. Experimental results

The system was implemented in C++ programming language and tested on real molecular simulation data. The experiments were carried out on an Apple MacPro machine with 8GB of memory and two Quad-Core Intel Xeon 3GHz processors. The MacPro was running OS X Mavericks 10.9.3 operating system. We have compared the results obtained by our system to those obtained by GROMACS (v. 4.5.7). Both systems were analyzing the same data sets.

Data sets. Six data sets from different simulations were used. All simulations were done on a POPC [3] lipid bilayer, but were all set to produce data of different sizes. We have tested the system on simulations with $52,400$; $209,600$; $838,400$; $2.5M$; $4.4M$; and $8.8M$ atoms. Also, since the simulations were run separately, they produced six different MS systems with distinct characteristics (distinct structure, atom's positioning, etc.). From all of the generated data sets we have randomly selected sets of 10, 100, and 1000 consecutive frames for the purpose of our experiments. This gave us 18 different datasets on which we tested our system.

Query work load. Two types of query workload were used: 1) one involving one-body queries only 2) one including two-body queries (SDH and RDF) as well. The reason for this is that GROMACS, the system we used to compare our system to, only has a naive method of solving the RDF (SDH) problem

[3]POPC is a chemical compound composed of a diacylglycerol and phospholipid. Its full name is 1-palmitoyl-2-oleoyl-sn-glycero-3-phosphocholine and it is one of the most important lipids in bio-physical molecular simulation.

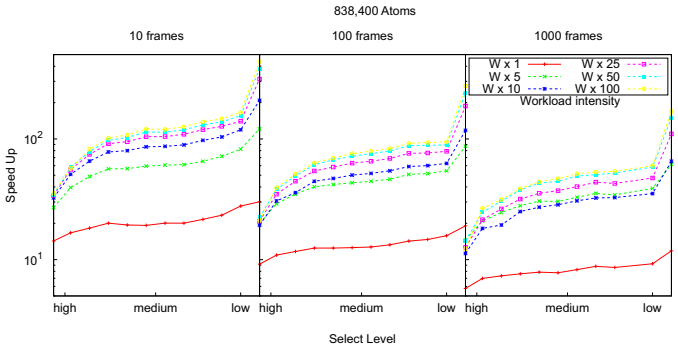

Fig. 4. Speed up over different levels of atom selection for 838K atoms

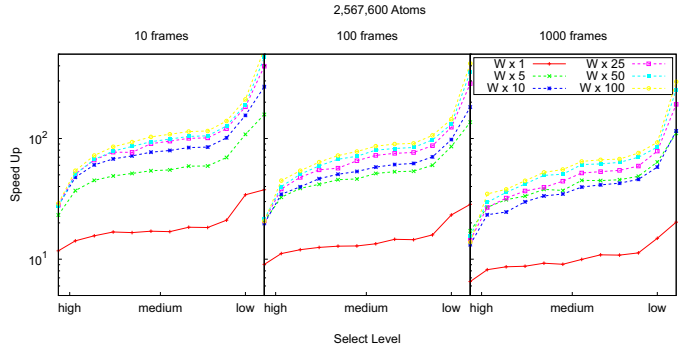

Fig. 5. Speed up over different levels of atom selection for 2.5M atoms

(like almost all MS analysis systems). In our system we have incorporated SDH (RDF) algorithms that are far more superior to the naive method, and comparing the systems like that would not have been fair (we believe).

One-body queries only. The following set of one-body queries were included in the test workload: mean square displacement (msd), radius of gyration, dipole moment, center of mass, velocity autocorrelation, electron density, mass density, and charge density. This set of queries were pointed to us, by a group in the physics field with extensive MS background, as one of the most commonly used in the field of collagen bilayer MS system analysis. A workload group contains all 8 queries executed on one of the 12 selections, making 12 groups. Such groups are executed on six different size data

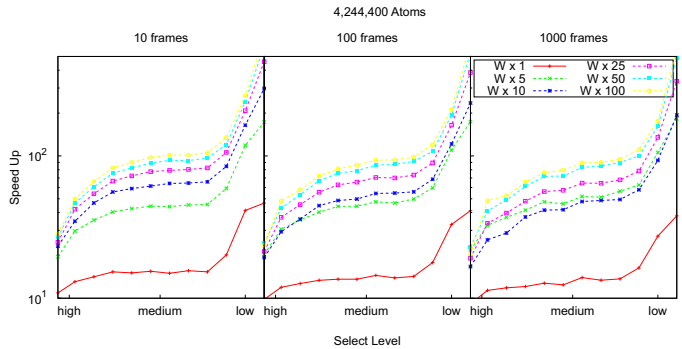

Fig. 6. Speed up over different levels of atom selections for 4.2M atoms

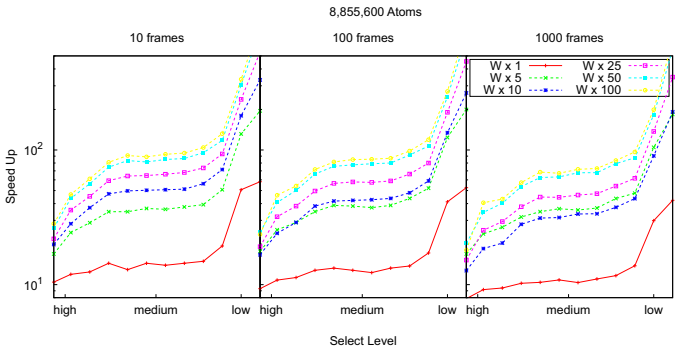

Fig. 7. Speed up over different levels of atom selections for 8.8M atoms

sets, with 10, 100 and 1000 frames. This workload is then repeated 5 more times, by executing each of the queries in the groups 5, 10, 25, 50, and 100 times, essentially just magnifying the workload intensity. In total, we have $12 \times 6 \times 3 \times 6 = 1,296$ different workload setups to test the system on. Fig. 3 shows the organization of the workload setup.

A. Benchmark

Through extensive collaboration with a research group from the Physics department at USF, we have come up with a benchmark that can be used for testing the efficiency of an analysis system for molecular simulations. The benchmark consist of three essential parts: 1. Simulation data produced by an MS, 2. Queries that are to be executed onto that data in order to produce some information of interest, and 3.Benchmark parameters that control the size of the benchmark.

1) Benchmark Data: The data used in the benchmark was real MS data, produced through GROMACS. The initial, pre-simulation data file consisted of 200 POPC and 12000 solvent molecules. This system was used because it is sufficiently diverse, containing enough distinct POPC and solvent molecules (e.g., each POPC molecule includes approximately 52 different atoms) and yet simple enough to be easily transformed into another system of different size. By using the $genconf$ function in GROMACS, we produced pre-simulation files of different sizes (essentially by changing the system's size). Six different sized pre-simulation files were created. A molecular simulation was then run on these 6 files, each producing an MS system of certain size. All of the simulations were set up to produce 1000 frames, all with same number of particles. The produced files contained: $52,400$; $209,600$; $838,400$; $2.5M$; $4.4M$; and $8.8M$ atoms per frame. So, for example, the file with $52,000$ atoms holds $52,000,000$ records in total (1000 frames, each containing $52,000$ records). This simulation data comes mostly in binary formats and in trajectory files having a lot of unneeded overhead. Therefore, it was transformed to a data arrays files containing only crucial information of the particles and the system. The size of the files ranged from $135MB$ for $52,000$ atoms to $24GB$ for 8.8 million atoms (this is for data with 100 frames).

2) Benchmark Queries: The queries selected to be included in this benchmark were derived through a thorough observation of the way an MS system is being analyzed. They were found to be the base of the analysis of many MS systems. Table I shows these queries.

3) Benchmark Parameters: There are several parameters that can be used to control the overall size of the system. We divide the parameters into two groups:
1. Data size parameters including the size of the dataset, number of frames, and the data selection (within the selected dataset) onto which the queries are being executed. 2. Workload size parameters including the number of queries to be executed and the number of times each query is executed.

By changing these parameters, we can produce a versatile testing benchmark for MS analysis systems.

B. Results

We have run extensive experiments over all the different setups of workload. However, in this paper we present only the workload setups of 4 data size sets: $838,400$, $2.5M$, $4.2M$, and $8.8M$ atoms because we believe they convey enough information about the efficiency of our system compared to that of the GROMACS system. The running times of our push-based system were compared to those of the GROMACS system. The first set of figures, namely Figures 4-7, represent the speedup that our system obtains over the GROMACS system with various atoms selection levels. We define the selection levels based on the number of comparisons we have to make in order to extract the needed group(selection) of atoms. For example, if we want to do analysis on all molecules containing oxygen, or hydrogen, or carbon we would go over each molecule and compare its components to the selection list. The bigger the selection list, the higher the select level in our system. For better visualization, we note three different selection levels: high (at least 10 comparisons made), medium (between 1 and 10 comparisons made), and low select level (with one or less comparisons made). As seen in the figures, for high selection level, the speedup is smaller compared to that achieved in low select levels. The reason for this, we believe is in that the amount of time our system spends extracting the atoms group increases with the level of selection. Even though our system still shows considerable speedup over GROMACS in high level selections, we do believe there is room for improvement in our system and that is immediate future work we are planning. These figures also show the relation of the speedup to the workload intensity, i.e., the higher the workload intensity the higher the speedup.

The connection between the workload intensity and the speedup is better represented in the next set of figures, Figures 8-11. They show the speedup our system achieves over the GROMACS system on a varying workload intensity. Each of those figures show the speedup with different dataset sizes (e.g., $838,000$, $2,567,600$ atoms, etc.), including 10, 100, and 1000 data frames. The speedup is calculated simply as a ratio between the running time of our system on a certain set of workload and that of the GROMACS system on the same workload. These figures show that the speedup over varying workload intensity achieved by our system ranges anywhere

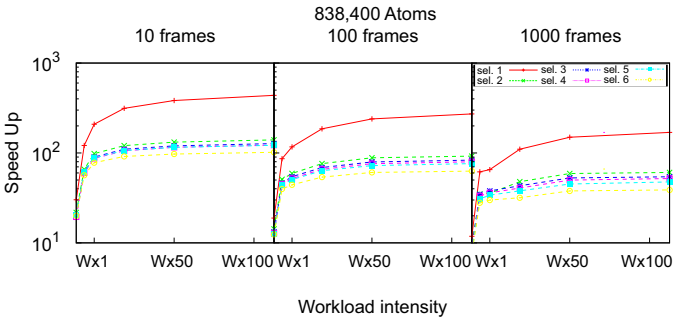

Fig. 8. Showing speedup for different workload intensity for 838K atoms

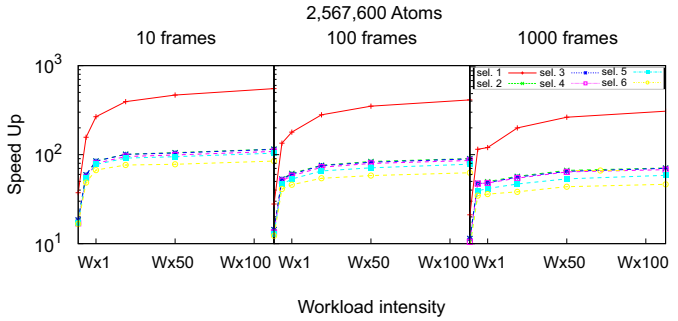

Fig. 9. Showing speedup for different workload intensity for 2.5M atoms

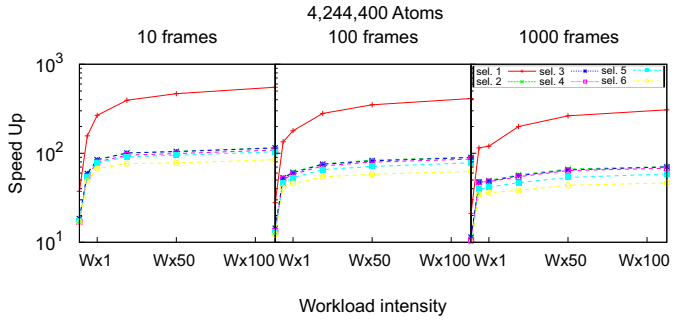

Fig. 10. Showing speedup for different workload intensity for 4.2M atoms

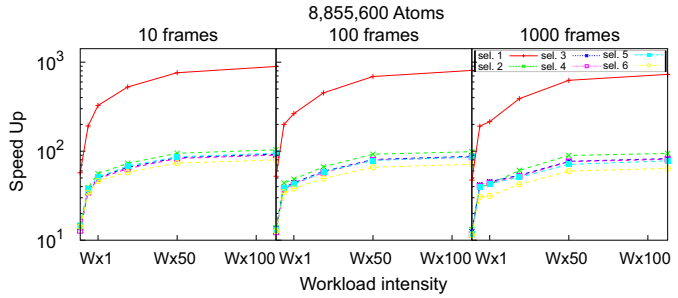

Fig. 11. Showing speedup for different workload intensity for 8.8M atoms

from about 10 to 1000 times, depending on the size of the dataset, number of frames and the selection of the atoms.

Figure 12 shows the speedup our system achieves over all workload intensity (average workload intensity) with varying dataset sizes. It is clear that, our system has better performance than the GROMACS system. The speedup presented in this set of figures ranges from about 15 to 650 times.

The last figure, Figure 13, shows the speedup our system achieves over all workload intensity and all select levels with varying dataset sizes. This figure, in a way, summarizes the previous two sets of figures, bringing together the workload and the different selections through the average. It is clear that, again our system has better performance than the GROMACS system. The speedup ranges anywhere from 50 to 250.

All four sets of figures show that such push-based design has clear advantages over the pull-based type of design incorporated in the GROMACS system.

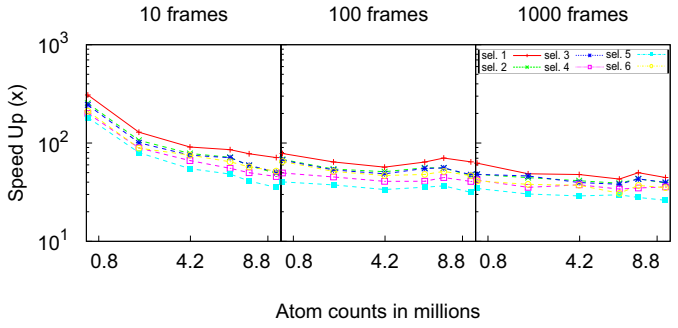

Fig. 12. Showing speedup for different data size (atom count)

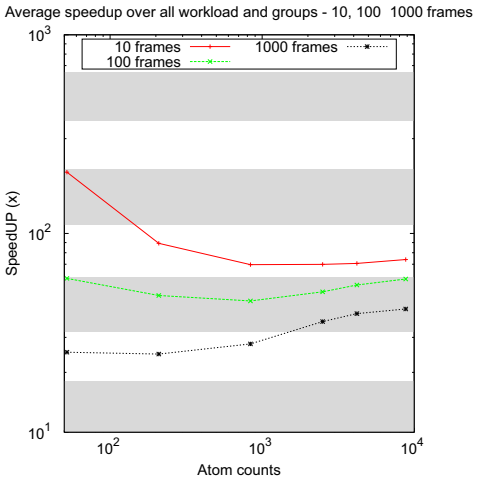

Fig. 13. Showing speedup for different data size (atom count)

VI. Conclusions and Future Work

The objective of our work is to design and implement improved data analysis system that can be used in the field of molecular simulation system's analysis. In this paper, we introduce such a system on a push-based type design, where data from data arrays is being pushed onto available queries in the system. These queries are being executed on the pushed data and produce intermediate / final results that would be used as part of the data analysis. We are able to achieve an improvement over existing, pull-based type designs because of the I/O overhead such designs introduce when dealing with large volumes of scientific data. Also, our queries are

able to be executed on the same stream of data, making it suitable solution for streaming circumstances. We designed a benchmark that can be used to test data analysis systems. We use this benchmark to compare our system to one of the most frequently used MS analysis systems – GROMACS. The efficiency and speedup achieved by our system is supported by extensive experiments. The results show that our push-based design achieves up to about 1000 times speedup in comparison to a pull-based design, i.e., GROMACS.

One direction of our future work will be to further improve our push-based design. Through the extensive experiments we have learned that our design can be improved when the atom selection clause involves many conditions. This improvement may be in the direction of improving the algorithmic design, or data presentation/organization we have used in the system.

Acknowledgements: The project described was supported by an award (R01GM086707) from the National Institute of General Medical Science (NIGMS) of the US National Institutes of Health (NIH). A part of this work was support by an award (CAREER, IIS-1253980) from US National Science Foundation (NSF). The authors would like to thank Dr. Anand Kumar for his time and knowledge contributed towards completion of the work.

REFERENCES

[1] D. H. et al., "Big data: The future of biocuration," *Nature*, vol. 455, pp. 47–50, 2008.

[2] B. Huberman, "Sociology of science: Big data deserve a bigger audience," *Nature*, vol. 482, p. 308, 2012.

[3] D. Centola, "The spread of behavior in an online social network experiment," *Science*, vol. 329, pp. 1194–1197, 2010.

[4] J. Bollen, H. Mao, and X.-J. Zeng, "Twitter mood predicts the stock market," *Journal of Computational Science*, vol. 2, pp. 1–8, 2011.

[5] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, 2014.

[6] D. Frenkel and B. Smit, *Understanding Molecular Simulation: From Algorithms to Applications*, 2nd ed. Academic Press, Inc., 2001, vol. 1.

[7] David Landau *et. al.*, *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press, 2005.

[8] Gromacs group, "GROMACS - Online Reference." [Online]. Available: http://gromacs.org/

[9] Subi Arumugam and Alin Dobra and Christopher Jermaine and Niketan Pansare and Luis Perez, "The datapath system: A data-centric analytical procesing engine for large data warehouses," *SIGMOD*, vol. 1, pp. 519–530, 2010.

[10] Goetz Graefe, "Volcano - an extensible and parallel query evaluation system," *TKDE*, vol. 6, pp. 120–135, 1994.

[11] Stavros Harizopoulos and Vladislav Shkapenyuk and Anastassia Ailamaki, "Qpipe: A simultaneously pipelined relational query engine," *SIGMOD*, pp. 383–394, 2005.

[12] Gorge Candea and Neoklis Polyzotis and Radek Vingralek, "A scalable, predictable join operator fo highly concurrent data warehouses," *VLDB*, pp. 277–288, 2009.

[13] P Unterbrunner and G Giannikis and G Alonso and D Fauser and D Kossmann, "Predictable performance for unpredictable workloads," *VLDB*, vol. 2, pp. 706–717, 2009.

[14] Marcin Zukowski and Sandor Heman and Niels Nes and Peter Boncz, "Cooperative scans: Dynamic bandwidth sharing in dbms," *VLDB*, pp. 723–734, 2007.

[15] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, "GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation," *Journal of Chemical Theory and Computation*, vol. 4, no. 3, pp. 435–447, March 2008.

[16] W. Humphrey, A. Dalke, and K. Schulten, "Vmd: visual molecular dynamics," *Journal of Molecular Graphics*, vol. 14, pp. 33–38, 1996.

[17] N. Michaud-Agrawal, E. J. Denning, T. B. Woolf, and O. Beckstein, "Mdanalysis: A toolkit for the analysis of molecular dynamics simulations," *Journal of Computational Chemistry*, vol. 32, pp. 2319–2327, 2012.

[18] M. Seeber, M. Cecchini, F. Rao, G. Settanni, and A. Caflisch, "Wordom: a program for efficient analysis of molecular dynamics simulations," *Bioinformatics*, vol. 31, pp. 2658–2668, 2010.

[19] T. Verstraelen, M. V. Houteghem, V. V. Speybroeck, and M. Waroquier, "Md-tracks: a productive solution for the advanced analysis of molecular dynamics and monte carlo simulations," *Journal of Chemical Information and Modeling*, vol. 48, pp. 2414–2424, 2008.

[20] M. Mezei, "Simulaid: a simulation facilitator and analysis program," *Journal of Computational Chemistry*, vol. 23, pp. 2625–2627, 2007.

[21] B. R. Brooks *et. al.*, "Charmm: the biomolecular simulation program," *Journal of Computational Chemistry*, vol. 30, pp. 1545–1614, 2009.

[22] L. Golab and T. Ozsu, *Data Stream Management*. Morgan And Claypool, 2010.

[23] A. S. Szalay, J. Gray, A. Thakar, P. Z. Kunszt, T. Malik, J. Raddick, C. Stoughton, and J. vandenBerg, "The SDSS Skyserver: Public Access to the Sloan Digital Sky Server Data," in *Proceedings of International Conference on Management of Data (SIGMOD)*, 2002, pp. 570–581.

[24] M. Arya, W. F. Cody, C. Faloutsos, J. Richardson, and A. Toya, "QBISM: Extending a DBMS to Support 3D Medical Images," in *ICDE*, 1994, pp. 314–325.

[25] M. Y. Eltabakh, M. Ouzzani, and W. G. Aref, "BDBMS - A Database Management System for Biological Data," in *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Resarch (CIDR)*, 2007, pp. 196–206.

[26] J. M. Patel, "The Role of Declarative Querying in Bioinformatics," *OMICS: A Journal of Integrative Biology*, vol. 7, no. 1, pp. 89–91, 2003.

[27] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, "The End of an Architectural Era (It's Time for a Complete Rewrite)," in *VLDB*, 2007, pp. 1150–1160.

[28] B. Howe, D. Maier, and L. Bright, "Smoothing the ROI Curve for Scientific Data Management Applications," in *CIDR*, 2007, pp. 185–195.

[29] P. G. Brown, "Overview of scidb: large scale array storage, processing and analysis," in *SIGMOD Conference*, 2010, pp. 963–968.

[30] P. Cudre-Mauroux *et. al.*, "A demonstration of scidb: A science-oriented dbms," *VLDB*, vol. 2, pp. 1534–1537, 2009.

[31] J. Gray, D. Liu, M. Nieto-Santisteban, A. Szalay, D. DeWitt, and G. Heber, "Scientific Data Management in the Coming Decade," *SIGMOD Record*, vol. 34, no. 4, pp. 34–41, December 2005.

[32] M. H. Ng, S. Johnston, B. Wu, S. E. Murdock, K. Tai, H. Fangohr, S. J. Cox, J. W. Essex, M. S. P. Sansom, and P. Jeffreys, "BioSimGrid: Grid-enabled Biomolecular Simulation Data Storage and Analysis," *Future Generation Computer Systems*, vol. 22, no. 6, pp. 657–664, June 2006.

[33] M. Feig, M. Abdullah, L. Johnsson, and B. M. Pettitt, "Large Scale Distributed Data Repository: Design of a Molecular Dynamics Trajectory Database," *Future Generation Computer Systems*, vol. 16, no. 1, pp. 101–110, January 1999.

[34] Jialin Liu and Yin Lu and Yong Chen, "In-advance data analytics for reducing time to discovery," *Proceedings of The 2014 IEEE International Conference on Big Data*, pp. 329–334, 2014.

[35] M. Bamdad, S. Alavi, B. Najafi, and E. Keshavarzi, "A new expression for radial distribution function and infinite shear modulus of lennard-jones fluids," *Chemical Physics*, vol. 325, no. 2-3, p. 554562, June 2006.

[36] J. L. Stark and F. Murtagh, *Astronomical Image and Data Analysis*. Springer, 2002.

[37] D. S. Wishart and A. M. Nip, "Protein Chemical Shift Analysis: A Practical Guide," *Biochemical and Cell Biology*, vol. 76, pp. 153–163, 1998.

[38] Yicheng Tu *et. al.*, "Computing distance histograms efficiently in scientific databases," in *ICDE*, 2009.

[39] Anand Kumar *et. al.*, "Distance histogram computation based on spatiotemporal uniformity in scientific data." in *EDBT*, March 2012.

[40] J. Orenstein, "Multidimensional tries used for associative searching," *Information Processing Letters*, vol. 14, no. 4, 1982.

[41] Vladimir Grupcev *et. al.*, "Approximate algorithms for computing spatial distance histograms with accuracy guarantees." *TKDE*, 2012.