3-16-2015

DCMS: A Data Analytics and Management System for Molecular Simulation

Meryem Berrada
University of South Florida, mberrada@mail.usf.edu

DCMS: A Data Analytics and Management

System for Molecular Simulation

by

Meryem Berrada

A thesis submitted in partial fulfillment of
the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Yicheng Tu, Ph.D.
Sagar Pandit, Ph.D.
Feng Cheng, Ph.D.

Date of Approval:
March 16, 2015

Keywords: Scientific database , Molecular Dynamics, Big Data,
Quadtree, SP-GIST

DEDICATION

I dedicate my thesis to my sweet and loving family, along with my teachers who have constantly been a great source of knowledge and inspiration.

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my advisor Dr.Yicheng Tu for his support and guidance in my research but also for his valuable career advices. He always encouraged me to set my career goals very high, and I will always be grateful for that. I would like to also thank Dr.Pandit Sagar for his assistance in this thesis, and Dr. Feng Cheng for agreeing to serve on the committee. Finally, I would like to thank my department, all of my professors, and everybody who made this experience a very pleasant one.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Despite the fact that Molecular Simulation systems represent a major research tool in multiple scientific and engineering fields, there is still a lack of systems for effective data management and fast data retrieval and processing. This is mainly due to the nature of MS which generate a very large amount of data – a system usually encompass millions of data information, and one query usually runs for tens of thousands of time frames. For this purpose, we designed and developed a new application, DCMS (A data Analytics and Management System for molecular Simulation), that intends to speed up the process of new discovery in the medical/physics fields.

DCMS stores simulation data in a database; and provides users with a user-friendly interface to upload, retrieve, query, and analyze MS data without having to deal with any raw data. In addition, we also created a new indexing scheme, the Time-Parameterized Spatial (TPS) tree, to accelerate query processing through indexes that take advantage of the locality relationships between atoms. The tree was implemented directly inside the PostgreSQL kernel, on top of the SP-GiST platform. Along with this new tree, two new data types were also defined, as well as new algorithms for five data points' retrieval queries.

CHAPTER 1: INTRODUCTION

As technology evolves over time, systems are required to be more efficient and power-saving. Hence, much research efforts are spent on improving current systems and making them faster. However, there is a big challenge that faces these tentative, which is the large amount of data that today's applications have to support. There are two main problems for applications that use a lot of data, which are the storage of data and the manipulation of data. Fortunately, storing data is not a big challenge anymore especially with the cloud technology. However, processing large amount of data remains a problem because it uses intensive computing power and can take a very long time, even hours if the system is really slow, and the users will probably quickly give up on the system if it is that slow. Therefore most systems today, including social networks, bioinformatics and many others use very complex and efficient algorithms to reduce this delay.

This paper presents a solution to improve the efficiency of Molecular simulations systems, which are used for various scientific investigations and new discoveries. Molecular simulation is a computational technique used in many scientific fields especially physics, medicine, biology and chemistry for calculating, analyzing and predicting various physical or chemical properties of natural systems. This type of system stores and processes a large amount of data which usually encompass information about numerous particles (e.g., molecules, atoms) that interact among themselves under some physics/chemistry rules. To provide an efficient tool for this kind of systems, we designed and developed a Database-Centric molecular simulation (DCMS)

framework which is, unlike previous solutions, built on top of a relational database management system (RDBMS) and allows users to share, retrieve and analyze data using a user-friendly interface without having to mess with any raw data. The user interface can be accessed from the URL: http://msdb.cse.usf.edu/msdb/. We have implemented five core functions that give different analyses and computations on the data: (1) center of mass, (2) density, (3) radius of gyration, (4) spatial distance histogram (SDH), (5) radial distribution function (RDF). These functions were implemented differently depending on the query complexity. The center of mass and the density functions are the simplest functions, and therefore they are implemented as stored procedures using PL/pgSQL- which is a built-in procedural language provided by PostgreSQL. The radius of gyration which is a little bit more complex was programmed as C functions and integrated into the PostgreSQL Finally, the most time consuming queries, such as RDF and SDH, were implemented in C++, and then compiled into a loadable object, which is called directly from the server-side code. The infrastructure of the DCMS system is shown in figure 1. The user starts interacting with our system using the web interface which directly talks to the core of the system which is written in php, which then transfers the query of the user to its corresponding function in either PL/pgSQL, the C database integrated function, or the C++ executable. In addition to the DCMS application, we implemented an index tree, TPS, along with to new data types (i.e., 3D point and 3D box) to improve the access and processing of queries that involve 3D points data.

Chapter 2 explores the existing solutions that provide tools for MS data storage, access, processing and analysis. Chapter 3 shows the web interface of DCMS along with its different features. Chapter 4 then explains the database structure, and the different tables schemas. Chapter 5 describes the different data processing functions implemented, along with their input and output parameters. Chapter 6 then investigates the different possible caching solutions for the DCMS

system. Chapter 7 cover the infrastructure and implementation of the TPS tree. Experiments results analyzing the performance of this new indexing tree are also presented in chapter 7. Finally, chapter 8 discusses the conclusion drawn and the possible extensions that could be made to improve the system.

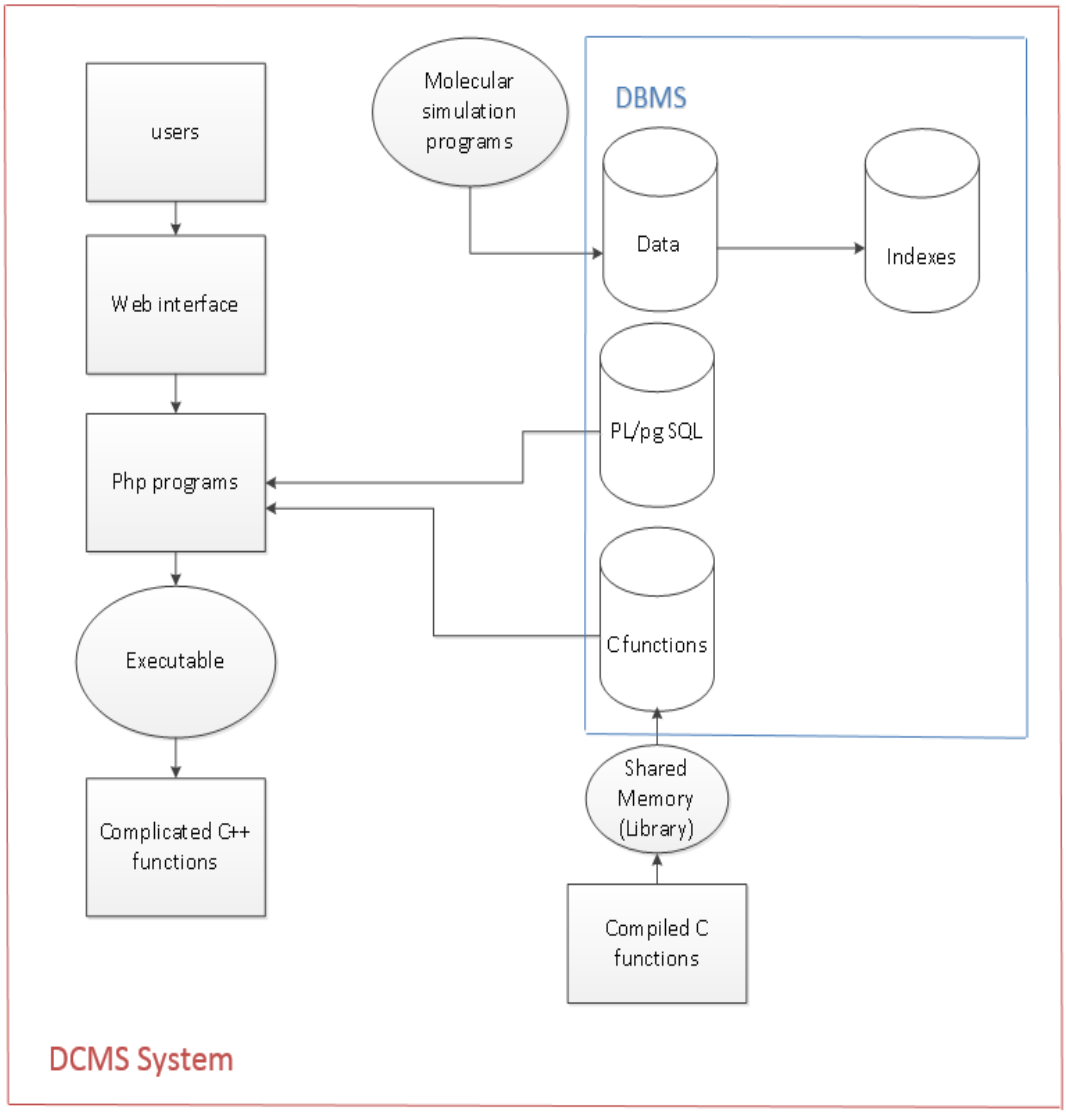

Figure 1. DCMS Architecture

CHAPTER 2: RELATED WORK

There are already many existing systems that strive to give physicians, doctors or any interested users an interface to manipulate molecular simulations - large amounts of data and information about some kind of particles, but they all have different gaps that the DCMS solution tries to solve. One of the major differences of these systems versus MSDV is that some systems store molecular simulation data in files instead of a relational database [5-7]. BioSimGrid and SimDB [1-2], for example, store all of their raw data in files organized in different directories, and use databases only to store metadata information- which comprise a detailed descriptions about files contents. This metadata can then be queried and used to locate different files, and then once a file is found, it is opened and scanned to find the exact row or data of interest. This is obviously very inefficient, because we will have to sequentially scan through all relevant files, possibly organized in different directories, to find the desired data. Also, it is more challenging to ensure the security of data when it is stored in files, especially if different files require different authorization rules. Another drawback is that it is not very convenient because it requires the user to be an experienced programmer and code specific programs for different types of data and queries. So although BioSimGrid and SimDB have an excellent computational performance, they fall short on the storage of large output data.

Another major functionality shortage that current MS systems suffer from is the lack of efficient and convenient data sharing. In fact, they share MS data by shipping the data packed in files along with the format information and analysis tools.

BioSimGrid and SimDB [1-2] allow users to remotely send queries among each other and get back results. This method is based on the reduction of data to smaller volume using methods such as Principal Component analysis, and that only a small representation of data is transferred instead of all of the raw data. In the Database of Simulated Molecular Motions (DSMM) [8], a similar approach is used where the database stores digital movies that are generated from visualizations of MS data.

The Molecular Dynamics database MDDB [3] and Dynameomics [4] are two other projects that are relatively closest to the DCMS. MDDB gives a good interface for domain experts to focus on the scientifically aspect of the system and does not require any prior programming knowledge. It is also mainly focused on the exploration and analysis of simulations instead the post-simulation data management. Dynameomics [4] uses a similar approach as DCMS, but it only concentrates in protein simulations –developed a database that contains protein data from 11,000 simulations; while DCMS can be used for different particles of any type. Furthermore, the optimization techniques in Dynameomics are mostly developed on the application level; while DCMS makes optimizations on both the application layer and the Kernel space of an open-source.

While all the existing solutions are good and present numerous advantages, we propose DCMS which will improve the efficiency of the storage, querying and sharing of data significantly, and resolve all of the issues mentioned above.

Figure 2. Application Interface

CHAPTER 3: DCMS WEB INTERFACE

3.1 Interface Requirements

For a system to be practical, it usually requires a user-friendly interface that hides most of the technical details, and can hence be used by different types of users. Although Molecular Simulations are sometimes used by engineers or computer scientists, it is more commonly used by physicians, doctors or scientists in general, who do not necessary have a strong programming or SQL background. Therefore our system strives to hide all of the implementation details, and permit scientists to use and manipulate data without having to directly deal with raw data or writing programs.

This system is intended to be used by scientists to upload, share, query data, and compute various attributes (e.g. center of mass, density) to analyze it and draw conclusions about the type or some other properties of the data. Therefore DCMS provides numerous user interfaces for data input as well as query processing.

3.2 Interface Infrastructure

The web interface was designed with cutting edge technologies that provide a secure, robust and efficient system. The application is hosted on a MAC OS X operating system, and it was developed using the server-side languages PHP, and the client side language JavaScript. PHP is one of the most popular scripting languages for web development because it is very quick and powerful and has different libraries and extensions available to it. JavaScript is also the most

commonly used client side language; it is vital for the responsiveness of an application and provides the programmer with many advanced tools to improve the user experience.

The web platform consists of seven pages. The first page you enter is the home page which gives a brief overview of Molecular dynamics data and their importance and explains how the system gives efficient methods to access and process such data. On the left of the homepage, there is a menu that allows you to access the other six page as can be seen in figure 2. The second page "Simulation information" allows a user to see all the different simulations that were uploaded to the server along with their attributes, as can be seen from figure 3. The next tab is "Query Interface", and that will give the interface to calculate different functions on any simulation data. This section will be discussed in more details in section 2.3 and chapter 4. "Data upload" can be accessed; in this page, we can upload the API along with a documentation. Finally, the last two tabs are "Publications" and "Contacts us" which are two static pages that include information about previous publications and contact information respectively.

Another important functionality of the system is the security of data; which means that every molecular simulation data set can only be accessed by it owner. Every page is in fact publicly accessible, except the query interface which requires the user to login first. Along with a login, any user can also create an account on our system by providing a few basic information such as name, affliation etc.

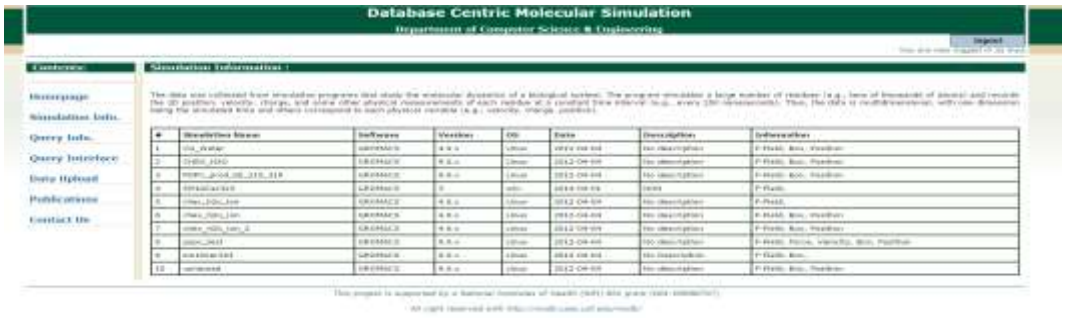

Figure 3. Simulation Info Page

3.3 Interface Functionality

Although the interface consists of several pages, the main utility of the application is the computations of several relatively complex functions on extremely large data sets, and this resides in the "Query Interface" page. We have implemented five core functions so far, and more functions can be added later on. One thing worth noting here is that the system was carefully built to allow functions. In fact, functions were implemented independently to permit this flexibility in extensions. We will go into more details of these five functions in chapter 4.

The query interface, as we said, contains the core functionality of this application. And since the data is tremendously big, we have to do a lot of filtering to allow the user to only query on a specific data set. Instead of laying down all of the filters at first, we give a smaller set of filters first and then depending on the user choices, we give the next options and so on. This is a better way to select data step by step instead of querying from the whole database; this method is also considerated to help users make their choices without getting confused.

The first thing we filter is the simulation; hence, the first filters correspond to simulations attributes (e.g Simulation name, OS, Software used). After a user make his or her choice, a dropdown with the different simulations that correspond to the user selections appear, from which the user can select the exact simulation he or she would like to operate on. After this, the used is prompted to select the function he or she would like to compute. After the user makes this selection, parameters related to that specific function appear, and the user can make his or her selection and press select to start the computation of the function. We have to note that different functions require different parameters because for example the density function requires a density type, number of bins and an axis while the radius of gyration function requires a vector input. Of course there are some certain other variables which are common to all functions such

as frame number, atom types, and molecular names and so on. The application interface can be seen in figure 4.

After the function is calculated on the selected data, the data output is presented in a table with a column for each result type. A back button allows the user to easily go back and change the latest functions selecting without going through all of the selections again. Some functions may also take longer time than others, therefore we notify the user with the availability of the results of his or her query once the computation of the function is over by sending him or her an email with a link to the query results.

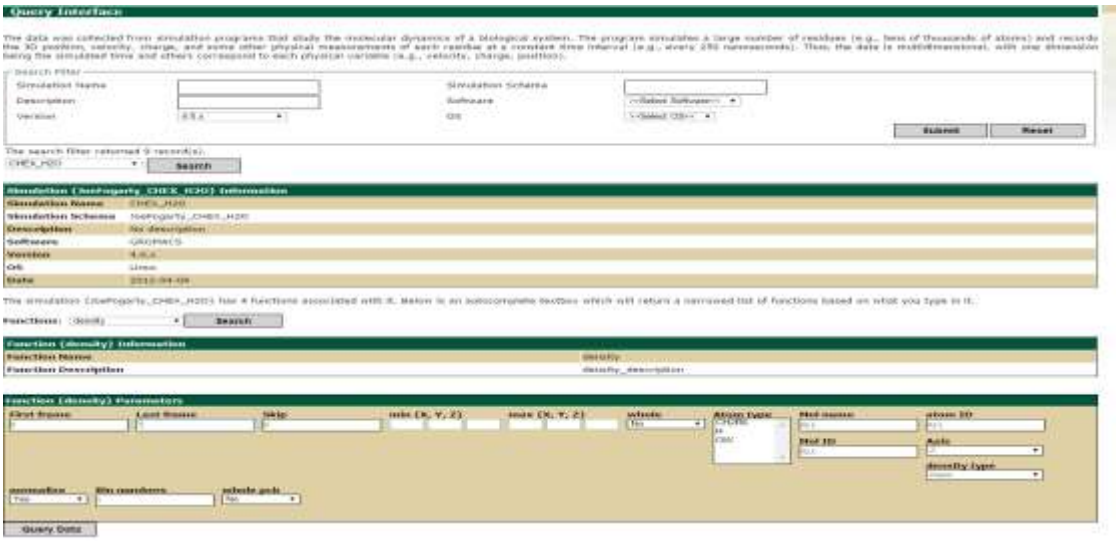

Figure 4. Query Interface

CHAPTER 4: DCMS DATABASE

4.1 Database Structure

An optimized database design requires a good understanding of the data it will store. A typical MS data contains multiple trajectory files that contain a number of snapshots (named frames) of the simulation. For each frame, several physical features of a large number of atoms (or molecules) including forces, charge, mass, velocity, and 3D locations are captured. These entries may also contain some additional information such as atom identifiers to place certain atoms in some molecules. Hence, the main body of a Molecular Simulation database consists of a collection of data items, where each item contains information about an atom or a molecule at a specific frame.

This system is aimed to allow many users to upload several simulations, each of which will hold atoms and molecules information. But instead of allowing all the simulation data to reside in one schema and having similar parameters stored in the same table, we create a new schema for every new simulation. This decision has been made relying on the assumption that the number of simulations is much smaller than the number of data each simulation holds. So for instance, if a simulation holds position information of 10,000 atoms at 10 different frames, then the position table will contain 100,000 rows from one simulation alone. If we add another simulation's data with the same number of atoms and frames, then the position table will have 200,000 rows and so on. Therefore, the tables will be extremely long and will take a very long time to query. Furthermore, the different simulations data will never be used together and are not

correlated in any way. For all of these reasons, we decided to create a new schema for every new simulation uploaded; where all schemas will hold the exact same tables' structures with different data.

There are also several tables created in the public schema that help us situate which simulation and function are being used. A simulation table is created to hold the various simulations along with their corresponding attributes; this later is used to filter the simulation that the user would like to operate on. A user table is also created to hold the different users information and allow access to the database.

We also have another table in the public schema, called function, which comprises all of the functions supported along with their different parameters. Once a function is added to this table, it automatically appears in the choice list of functions; this is how, as described in chapter 2, additions of new functions is automatic and does not require understanding of the existing PHP code. On the other hand to extend the system with new functions, the programmer will still need to call the implemented function and format its output; this is done in a very specific file and is independent of the rest of the server code. This part was not made automatic because different functions are called differently depending on the platform they were built in, and their output format also varies depending on the number of output attributes returned by the function. The database structure is shown in Figure 5; this table is taken from the paper that our lab team and I have published in the journal of big data. Note that the function table is missing because it is only useful for the server code (PHP) and does not encompass any MS data.

4.2 Tables Structure

MS data is usually not captured at every step of the simulation, hence different attributes such as location, force, and charge are outputted at different intervals (steps). So frames may

correspond to a different times for each attribute (e.g frame 10 might correspond to step time 100 for position while frame 10 might correspond to step time 20); this depends on the start time and the frequency at which different parameters are captured. Therefore, we create two linked tables for each attribute, one table stores the attribute information with its corresponding step time (e.g. atomposition), and the other table is used to match step times with frames numbers (e.g. atompositionframe). We assume that when data for two attributes (residing in two different tables) is queried at a certain frame time, that frame corresponds to the atom position frame table. So for instance, to get the position and charge of an individual atom at frame 2, we select that atom's position and charge from the tables atomposition and atomcharge where the frame number in the atompositionframe table is equal to 2 and the step number of the charge table entry is equal to the step number of the atomposition entry (so the frame number in the charge table is not necessarily 2).

Since the atom charge, position, force, velocity, energy, pressure and temperature are all captures at different instance, they reside in separate tables. On the other hand, the atom name, mass, type, atomic number, and number of electrons are usually static and captured together; hence they are incorporated into the same table atominfo. AtomMolecule is one of the most important tables and describes the relationships between atoms and molecules – it specifies which atoms are within which molecules. These relationships are also time-variant and therefore are also mapped from step numbers to frame numbers. Finally, the table Molecules contains molecules static attributes.

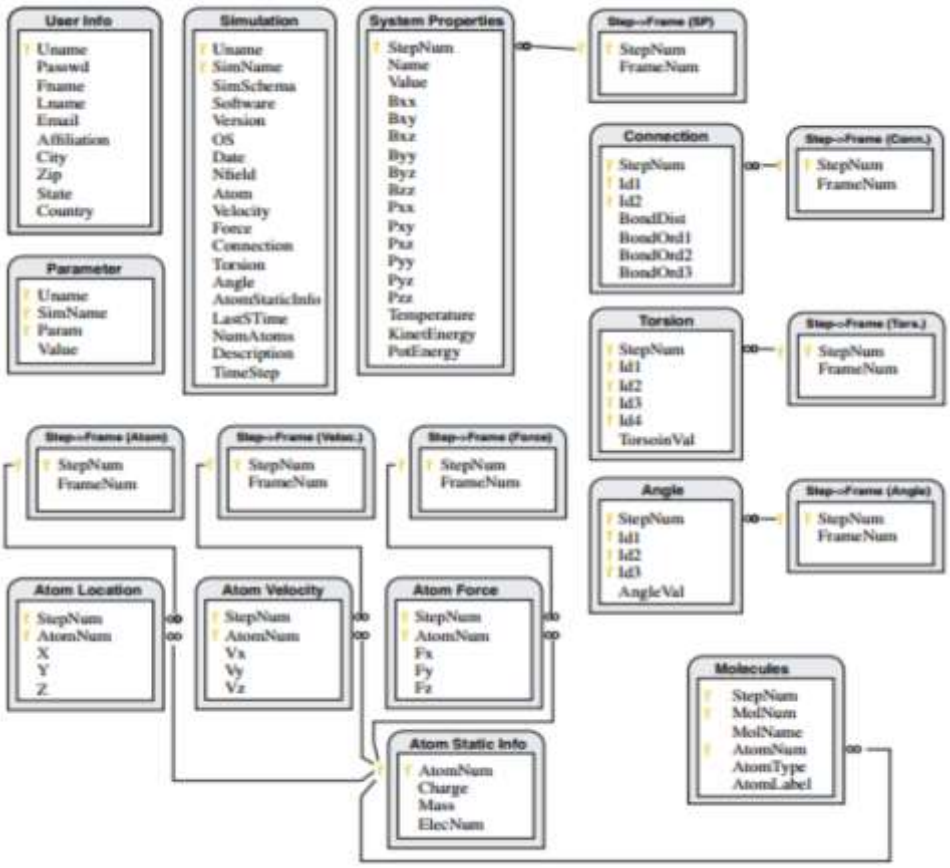

Figure 5. Database Structure

CHAPTER 5: SUPPORTED FUNCTIONS

5.1 Functions

The heart of the DCMS application is the computation of analytical queries on MS data. An analytical function is a mathematical functions which maps the reading of a group of atoms to some specific entity (e.g. scalar, matrix, data cube) [1]. We distinguish between two types of functions, One-body and Multi-body functions. One-body functions require the reading of each atom a fixed number of time, so they operate in O(n), and these include moment of inertia, center of mass, electron density functions. On the other hand, Multi-body functions involve interactions of atoms among each other, so it is at least O(n^2) if they are handled in pairs; and these include more complex queries such as the SDH histogram and the Radial distribution function. [9-11].

One-body functions can be efficiently computed with moderate algorithms; while multi-body queries are much more expensive and require more advanced algorithms. So one major challenge that DCMS solves is the design and implementation of efficient technique to reduce the overhead of both types of queries.

The processing of MS queries is achieved through two steps, the first one is to retrieve the group of desired atoms, and then to calculation the mathematical function. Since MS data tend to be very large, we use indexes to reduce the access time. Two main types of indexes were used in DCMS, the B+ tree which is a bitmap-based index and is automatically chosen by the optimizer, and a new indexing scheme, that we created to better manipulate 3D data, named Time-Parameterized Spatial (TPS) tree. We will go into more details of this tree in chapter 6.

It is worth noting here that the overhead of maintaining (or updating) indexes is negligible, since the database is mostly read-only.

Indexing techniques enhance the rapidity of data access, but do not speed up the computation process; therefore we use many techniques such as an already developed SDH algorithm and caching output results to simplify the time complexity. We also implement the functions in different platforms depending on each function's need.

5.1.1 Center of Mass

The center of mass is the most basic function implemented, and it does not require any additional parameter other than the default selections. We will go through these default parameters here but they are used for all other functions. The first three parameters are firstFrame, lastFrame, and skip and they specify the frames to be used in the calculations; so we read data starting from firstFrame and all the way to lastFrame but skipping a certain number of frames if skip is a non-zero value. The next parameters are min and max, which are 3D vectors that represents the boundary of the region to use – so only read the atoms which are located in this region. Up next is the atom type dropdown, where all the existing types are listed and from which the user can select one or more atom types. Two other options are atomID and molID, which allow the user to specify the range of Atom IDs or Molecule IDs to be used respectively. Atoms can also be filtered by molecules using the parameter MolName - if this is selected, only atoms contained in this particular molecule will be processed. Finally, two very special parameters are implemented, whole and wholePcb. Whole indicates whether we should select all atoms that are in a molecule (specified by moleculeName or moleculeIDs) and disregard the other atom parameters (atomType and atomID). In other words, if whole is true and one of the molecule parameters are chosen, then we disregard the remaining atomic criterions. On the other

hand if set to true, WholePcb positive value imposes that the molecule be in one side of the box (region containing the concerned data). Once the data is selected according to all of the criterion mentioned above, a simple mathematical section computes the center of mass, which consists of the sum of the multiplication of the atom's mass with its position over each axis and divide by the total mass.

This function is implemented in PL/pgSQL, which is the most commonly used stored procedure language for PostgreSQL. Stored procedures are used to delegate the computation task to the database. This encapsulation of the computational logic in the database provides the advantage of eliminating the cost of calculations from the webserver level and shifts it down to the database level. Another advantage of PL/pgSQL is that its execution is optimized by the database query optimizer. Hence, we chose to use PL/pgSQL, which is very straightforward, strong, and is stored in the database level. This method is especially suitable for simple functions such as the center of mass. The output results of center of mass consists of four columns, the frame number, and the center of mass along X, Y, and Z axis.

5.1.2 Density

As mentioned before, the density function, just like every other function, contains the same default parameters as the center of mass query. In addition to that, there are four other parameters specific to this function: densityType, axis, binNumbers, and normalize. Since there exits more than one density function, we decided to merge four density functions (electron, mass, number, and charge density functions) into one density and allow the user to specify the type of density query desired. This this later decision allows for a simpler and clearer interface, and more compact code. The second parameter is the axis, and it specifies along which axis the density will be calculated. The criterion binNumbers represent the number of bins in which the

density will be distributed, and finally normalize indicates whether we should divide the results by the box coordinates.

This is implemented in a similar fashion as the center of mass function, where the atom's mass, charge or atomic numbers are first extracted from the database. Then, a simple mathematical function computes the density by summing up the weights of each atom (where the weight represents one of the four density types). Note that the density function was also implemented in PL/pgSQL since it is a basic function with low time-complexity.

5.1.3 Radius of Gyration

The radius of gyration computes the distribution of the atoms around an input axis (or line). The main use of this latter is to compare how numerous structural shapes behave when projected along an axis. So in other words, it can be used to predict buckling in a compression member or beam [12].

Aside from the default parameters, the radius of gyration function has one additional vector parameter which represents the line to which points are projected. For each frame, we first compute the distance of atoms to the axis and then we multiple that distance squared by the mass, and add the result to Im. After we go through all the atoms in an individual frame, the radius of gyration of that frame is equal to the square root value of Im divided by the total mass. The output results consist of two values for each frame: the frame number and the radius of gyration value corresponding to that frame.

While writing PL/pgSQL functions was good enough for the center of mass and density functions, we hope to achieve better performance for more complex queries. Hence, we chose to implement the radius of gyration function as a C function which we compiled into a dynamically loaded object (shared library), and loaded into the PostgreSQL server; thus, it can be called in an

identical manner as PL/pgSQL functions. There are two main advantages of implementing C function over PL/pgSQL functions. The first benefit is that C functions give simpler access to functionality that is not present in the PL/pgSQL language. A second significant advantage of C functions is that they are also much faster, since PL/pgSQL are themselves implemented in C using serial peripheral interface buses.

5.1.4 Spatial Distance Histogram (SDH)

The Spatial Distance histogram is one of the most commonly used queries to find distance correlations in a data set. In fact, it is a 2-body function which computes the distances between all pairs of points. Note that the only additional parameter for this latter function is the bin width. SDH is calculated by first retrieving the location of all atoms, computing all pairwise distance, and then plotting the distances in a histogram. Since it computes the distances between all points, its time complexity is O(N^2). If the data is very large, which we expect it to be, the processing time of the query with a brute force algorithm (that computes all pairwise distances) will be unacceptably high. To avoid this long delay, we decided to use the SDH density-based algorithm, which reduces the running time to O($N^{5/3}$) [19].

The density-based algorithm's main approach is to work on clusters of atoms instead of individual atoms in deriving the function results. Although this algorithm is almost accurate, it is only approximate because it takes advantage of the fact that atoms are usually uniformly spread out due to the existence of chemical bonds and inter-particle forces, and does not directly compute all pairwise distance [19]. Instead, it uses the spatial locality of atoms to approximate the distance. While his error rate appears to always be less than 1%, this algorithm was proven to be about ten times faster than a simple brute force algorithm. Hence, we consider our choice of

the density-based SDH algorithm safe and effective; especially that 1% error rate is often considered to be below the granularity level of such systems.

The optimized SDH function was already implemented in C++. There were two possible options to integrate this function into our system. We could compile the function into a dynamic loadable object and load it into PostgreSQL, in a similar fashion as we did for the radius of gyration function. However three important observations were made when implementing the radius of gyration: the C language is not exactly the same as regular C, there were many memory restrictions, not all standard data types are supported, and the number of input parameters is limited. For all of these reasons, we decided to choose the alternative approach where we compile and load the C++ function into an executable, and then call the function from PHP in a similar fashion as the remaining functions. The output of Sdh is the bin number and their corresponding distance density (how many distances fall in each bin).

5.1.5 Radial Distribution Function (RDF)

The spatial distance histogram SDH is itself very useful, it is also the basis for calculating other queries such as the radial distribution function (RDF). In fact, RDF is sometimes called the normalized version of SDH. RDF is indeed another example of a pair correlation function that describes the structure of a system – or how atoms are distributed around each other, and is useful especially for liquids [13]. This latter can be used to find various location characteristics of the data – i.e. determine (or predict) the number of particles that exist within a distance d from another particle[14].

RDF can be calculated using the SDH value and dividing by the r-distance corresponding to that bin times the bin width times the density times a constant which is equal to 4π – density equals the number of atoms divided by the volume of the containing box. To keep a concise

code, we uses the same function as was used for SDH, and added one input parameter that specifies whether RDF or SDH is selected. So when the server calls a function, it sends one more parameter isRDF, and depending on this input, we will either normalize the result or simple output the SDH results. The output of RDF consists of the bin numbers along with their distance densities.

CHAPTER 6: CACHING

6.1 A File-based Caching Technique

Since the DCMS system offers multiple functions that run on extremely large data, the processing time is relatively long. Therefore, it is indispensable to use a caching technique to reduce the waiting time on previously ran queries. The most straightforward caching technique is to store the queries result data into files organized into different directories (i.e. one separate directory per simulation). But executions of the same function can use different parameters value; hence, the parameters values need to also be stored along the results. One possible way to store the input values is to store them in the same file as the result – at the top of the file. This method is very simple but lacks efficiency when the number of executions start to rise. In fact, to find the desired results, we will have to check all the files in the appropriate folder and check if the inputs match. Note that there at least eight parameters for each function, and most of them can take several values; so if each value can take 3 different values, then we have $\boldsymbol{8^3} = 512$ possible combination. In reality, most parameters can take a very large variety of values, and hence we may end up with many files caching results of the same query but have 1 or more different parameter values. To solve this issue, we could concatenate the parameters values to the file name, and hence we can directly open the right file containing the results, if it exists. This is a good solution but it makes the assumption that all the parameters values will be small- not take up much space. Although this is true for most parameters, there are some parameters such as atom type that are actual arrays – and unfortunately, we cannot limit their numbers.

In fact, storing the parameters in the file name is not very practical and can also lead to errors with any small variation. For example, if we change the order in which the parameters were selected, we will end up with different file names, and therefore the caching system will not recognize that the same query with the exact same parameters was run before – but with different parameter values order. Thus, storing the query results in file is probably not the most ideal caching technique for DCMS.

6.2 A Database-based Caching Technique

After concluding that a file-based cache won't work properly, we decided to store the function output results in a database. We created tables in the public schema for each individual function. The columns of the tables include the simulation name, all the input parameters for the function (separated, so one column for each attribute), and the output results. Note that if an output results consist of more than one column (or one attribute), we still store the whole raw in one attribute of type string. The caching table structures for the center of mass, the radius of gyration, and the radial distribution functions are shown as examples in figure 6. This decision was made based on the fact that we never use the output attributes to filter a row from the table. So with this technique, if the center of mass function returns five rows corresponding to five different frames – and each row comprises a value for X,Y, and Z; then we will store all the inputs of this query and store "{X,Y,Z} together in the output column. Of course, we will store each frame result in a separate row. This database-based caching strategy eliminates all the drawbacks that the file-based cache suffered from, and further takes advantage of indexing to avoid a sequential scan – as is usually done in file systems. This latter feature is especially important when a new query is a subset of an already ran query or vice versa. So for example when query Q1 runs on frames 1 to 10, and query Q2 runs on frames 5 to 10. Then, we have to

skip the first five frames to get to our desired results. There are some other cases where the need for skip is even more important than here, this usually occurs when the skip value is non-zero. For instance, if query Q1 runs on frames 1 , 3, 5, 7, 9 (so basically the skip input = 1), and we would like to run query Q2 with the same parameters except that skip = 0 and lastFrame = 5, then we can read the results for frames 1, 3, and 5 from the cache, and only compute the function for frames 2 and 4. One thing important to note here is that with both implementations of the file-based cache, we won't be able to detect that the query was already calculated for frames 1,3, and 5 since two parameters (skip and lastFrame) are different from Q1. For all of these reasons, we have concluded that the database caching approach is the best fit for DCMS.

Center Of Mass

Column name	Definition
id	serial NOT NULL
frame_num	integer NOT NULL
atomidmin	integer
atomidmax	integer
molidmin	integer
molidmax	integer
molname	character varying(30)
whole	boolean
whole_pcb	boolean
type	character varying(500)
minx	integer
miny	integer
minz	integer
maxx	integer
maxy	integer
maxz	integer
centermass_x	double precision
centermass_y	double precision
centermass_z	double precision

Radius of Gyration

Column name	Definition
id	serial NOT NULL
frame_num	integer NOT NULL
atomidmin	integer
atomidmax	integer
molidmin	integer
molidmax	integer
molname	character varying(30)
whole	boolean
whole_pcb	boolean
type	character varying(500)
minx	integer
miny	integer
minz	integer
maxx	integer
maxy	integer
maxz	integer
radiusofgyration	double precision
vectorp1x	double precision
vectorp1y	double precision
vectorp1z	double precision

Radial Distribution Function

Column name	Definition
id	serial NOT NULL
first_frame	integer
last_frame	integer
skip_frame	integer
atomidmin	integer
atomidmax	integer
molidmin	integer
molidmax	integer
molname	character varying(30)
whole	boolean
whole_pcb	boolean
type	character varying(500)
minx	integer
miny	integer
minz	integer
maxx	integer
maxy	integer
maxz	integer
rdf	character varying(5000)
r	integer
binwidth	integer

Figure 6. Caching Table Structures

CHAPTER 7: TPS TREE

7.1 Infrastructure

As discussed before, the molecular simulation data is very large, and therefore optimization of the queries run times is necessary. Since the data is usually processed in terms of its location, we most likely need to use an index that can accelerate the retrieval of one (or a range) of data point locations. There are three well-known indexing frameworks in PostgreSQL, B-tree, GIN, and GIST. The first one is B-tree, which is very commonly used for most data types but it is limited to a few comparison operators [18]. GIN stands for Generalized Inverted Index and is also an indexing framework but is generally used for composite data types. The next types is GiST - Generalized Search Tree, and it is a balanced and tree-structure access method in PostgreSQL [17]. While GiST can be used to implement multiple trees such as B trees and R trees, it suffers from the fact that it always needs to be balanced. Therefore, the fourth indexing framework SP-Gist (Spatial Partitioning trees) emerged, which allows for the creation of space partitioning trees, and a variety of non-balanced structures [16]. SP-GiST divides the search space into non-necessarily equal partitions (partitions of equal size), and hence improves the access time on searches that are matched to the partitioning rules [15]. Three types of trees were implemented on top of SP-GiST along with the implementation of SP-GiST: suffix trees (or tries), k-d trees, and quad-trees. Usually for a 2d mapping of data points, the quad-tree index built on top SP-Gist is the most reasonable choice. The quad-tree divides the search space into 4 quadrants, where nearby points are stored together in the same quadrant. It keeps dividing the

sub-quadrants into four equal parts until there is only one point in each quadrant. Note that it only divides the quadrants that contain more than one point; and hence is a non-balanced tree. The quad-tree index can be used to optimize multiple queries since the data points that are closer to each other reside in nearby quadrants. So for example for range queries (to find all points in a certain range), the quad-tree is traversed by only choosing one quadrant at each, that encompass the desired range, until the points contained in that range are retrieved. In fact, the creators of SP-GiST and the quad-tree tested the quad-tree indexing scheme on the geonames data set - which is comprised of 2045446 points, and concluded that the running time is six times faster when using the quad-tree index versus using another index type.

In the Molecular Simulation systems, data is most commonly queried by their locations, and a good spatial index could improve the performance of the system significantly. We decided to use the SP-GiST quad-tree because of the following three reasons: (1) the quad-tree performance on handling spatial queries is usually superior to that of any other existing tree schemes (including the R-tree); (2) the worst-case performance of a quad-tree is when the tree is very unbalanced, but this is very unlikely in MS systems, since MS data points are usually spread out, (3) quad-trees can be easily augmented to build other data structures need for higher-level query processing.

But unfortunately, the quad-tree implemented on top of SP-Gist only supports two-dimensional data, while MS data are projected on a three-dimensional space. Since we believe that a quad-tree can tremendously boost the performance of our system, we decided to build a Time-Parameterized Spatial (TPS) tree that is also built on top of SP-GiST and behaves similarly to the quad-tree, with the only exception that it supports 3D data points.

7.2 Implementation

The SP-GiST package provides internal methods to implement new classes of space partitioning trees, and provides APIs for implementing specific features of data types [16]. We used PostgreSQL (version 9.2) as the database engine, and we extended the current PostgreSQL codebase to include a new tree: TPS. To create the TPS tree, we used the APIs provided by SP-GiST to create two new datatypes 3D point and 3D box (that will be used to store 3D data). In addition, we used the internal methods to create the new indexing tree TPS. The TPS tree was implemented in a very similar manner as the point quad-tree; the main difference is that the quad-tree supports points in a 2D space, while TPS supports points in a 3D space. So we divide the search space into 8 equivalent quadrants instead of 4; and the algorithmic logic of most functions (such as deciding how to create a new inner tuple over a set of leaf tuples) becomes a little bit more complex.

Note that the name of the new version of the quad-tree was named TPS because it will be used to build spatial index for each time frame in the data set. To build the TPS tree, we create a spatial tree for each time frame using bulk loading; and then merge nodes in neighboring trees iteratively. After we implemented the TPS tree, we accordingly modified the query optimizer of the database to generate query execution plans that take advantage of this new tree.

When a new tree is created on top of SP-Gist, some data access queries are also implemented along. We implemented five queries that we find relatively most relevance in our DCMS system. The first and most straightforward ones are point queries which are equivalent to accessing a single point in a 3D space (e.g. find the atom name or some other physical measurements of an atom at a specific location). This query is done by issuing queries with randomly generated atom IDs. The next queries implemented are trajectory queries which

retrieve all data points by fixing the value in one dimension (e.g. find all atoms whose x coordinate is in the range from x = 0 to x =10). This is useful for queries such as the radius of gyration function, which projects all data points into one axis. Another type of queries implemented are range queries, which are generalized trajectory queries with range predicates (e.g. find all atoms in a specific region of the simulated space). Range queries are the main building blocks of many analytical queries and are extremely useful for many visualization tools (e.g. the visualization tool that queries for the data points that are located within a specific region that reflects the underlying distribution. Nearest neighbor (NN) queries find for the point(s) that are closest to a given point in a three dimensional space (e.g. retrieve the 20 closest atoms to a given iron atom). This type of query helps to locate unique structural features (e.g. certain part of the protein that a metal ion is bound to).

7.3 Results

We tested the performance of the queries mentioned above using the TPS tree index, and compared the results to the performance of the same queries without using an index. For this experiment, we used a single MS dataset with 286,000 atoms and 100,000 frames, the total data size of which is about 250 GB – note that this represents the approximate data size of a single simulation in MS applications. This dataset was generated from a previous work done in our lab to simulate a hydrated DPPC system in NaCl and KCl solutions [20]. For further comparison, we ran the same queries against the data analysis toolkit GROMACS – a mainstream file-based system for MS simulation and data analysis [5]. We compared the results again GROMACS because according to the paper A Toolkit for the Analysis of Molecular Dynamics Simulations [21], GROMACS has the best performance over the other popular MS systems, and therefore is considered to be the state-of-the-art in MS data analysis.

We tested the five implemented queries, which are considered to form the basis for the majority of high-level data processing tasks in Molecular Simulations. Note that upon loading the data, PostgreSQL automatically builds a B+-tree index on the primary keys - the combination of step number and atom ID. Then we built the TPS tree on the location of the atoms, which reside on the atomPosition table.

In GROMACS, the queries were ran against the same dataset which was organized in 400 files, each holding 250 time frames- with 286 atoms in each frame. To achieve fair comparisons, we used a grid-based spatial index for the range and nearest neighbor queries in GROMACS [5]. The results of the following types of queries: (1) random point access (RDM); (2) single-atom trajectory retrieval (TRJ); (3) frame retrieval (FRM); (4) Range query (RNG); and (5) Nearest neighbor (NN) queries are shown in figure 7; this table is taken from our previously published work in the journal of Big Data. Note that each result value is the average of five experiments of the same query with different randomly generated input parameters.

The query performance of DCMS achieves a speedup of 1-5 orders of magnitude compared to GROMACS; and hence it is clearly much better, especially when the TPS tree was used. Note that no caching system is used for these experiments. These results also proves the combined benefits of record-based (rather than file-based) I/O and indexes. In fact using indexes, we can directly visit the pages that hold the relevant data records instead of having to search through large files - like in the file-based solution.

An interesting observation in the experiment results is that although all queries ran faster in DCMS (compared to GROMACS0, the trajectory query processing time remains large even in DCMS. However when the TPS tree, the processing time reduces significantly. This, in fact, confirms that the TPS is indeed much more suitable than any other existing index in PostgreSQL.

Another important thing to note is that the processing time for the range queries usually improve when using the TPS tree index however when the range is very large (contains most of the data points), then the TPS tree might take a longer time to process the query. This conclusion makes sense because in general, it is better to sequentially scan the data when we want to query most of the data in a table versus using an index. The final observation we made is that although indexes were used in GROMACS for the range and nearest neighbor queries, the spatial index in GROMACS does not improve the processing time by much; there is still a performance boost of approximately 2 to 3 orders of magnitude in our DCMS system.

System	Queries				
	RDM	TRJ	FRM	RNG	NN
DCMS + TPS	0.0008	13.4	2.56	0.073*	0.029
DCMS	0.069	6239	2.48	0.122*	0.198
GROMACS	45.0	16410	52.5	8.49	16.8

*time depends on the query range

Figure 7. Query Processing Time (in seconds)

CHAPTER 8: SUMMARY AND FUTURE WORK

This paper presents a new solution, DCMS that allows physicians, doctors and scientists to upload, process, retrieve, query and analyze data from a user-friendly interface and in a timely manner. Five of the most predominant queries were implemented: (1) center of mass, (2) density (3) radius of gyration, (4) spatial distance histogram (SDH), (5) radius distribution function. We also wrote an extension of the current PostgreSQL codebase, which encompasses new data types (3D points and 3D boxes), and a new indexing tree (TPS) that significantly improves the query processing time for multiple functions. The results above also confirmed our statements; and therefore, we can conclude that there is significant improvement in data access performance when using DCMS for MS data processing (compared to GROMACS). This speedup is clearly even further boosted with the use of the TPS tree.

There are many extensions that could be done to improve both the performance of the system as well as the user experience. The performance can be enhanced if the TPS tree indexing scheme on all atomPosition tables in our database. Today, the TPS index tree is implemented on the PostgreSQL version 9.2 while the database is hosted on the PostgreSQL version 8.2.6. The TPS extension can be utilized by upgrading the database server to the extended PostgreSQL version, and then copying all the data to the upgraded server. To add functionality, new functions can also be added to the system. Although only five functions were implemented so far, these

latter are some of the most common queries used for MS data, and also represent a basis for many new potential functions – that use these basic functions results to draw some new different properties or conclusions.

REFERENCES

[1] Feig M, Abdullah M, Johnsson L, Pettitt BM (1999) Large scale distributed data repository: design of a molecular dynamics trajectory database. Future Generation Comput Syst 16(1):101–110

[2] Ng MH, Johnston S, Wu B, Murdock SE, Tai K, Fangohr H, Cox SJ, Essex JW, Sansom MSP, Jeffreys P (2006) BioSimGrid: grid-enabled biomolecular simulation data storage and analysis. Future Generation Comput Systs 22(6):657–664

[3] Nutanong S, Carey N, Ahmad Y, Szalay AS, Woolf TB (2013) Adaptive exploration for large-scale protein analysis in the molecular dynamics database. In: Proceedings of 25th Intl. Conf. Scientific and Statistical Database Management. SSDBM. ACM, New York, NY, USA. pp 45–1454

[4] Van der Kamp M, Schaeffer R, Jonsson A, Scouras A, Simms A, Toofanny R, Benson N, Anderson P, Merkley E, Rysavy S, Bromley D, Beck D, Daggett V (2010) Dynameomics: a comprehensive database of protein dynamics. Structure 18(4):423–435

[5] Hess B, Kutzner C, van der Spoel D, Lindahl E (2008) GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. J Chem Theory Comput 4(3):435–447

[6] Brooks BR, Bruccoleri RE, Olafson BD, States DJ, Karplus M (1985) CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. J Comput Chem 4:187–217

[7] Plimpton SJ (1995) Fast parallel algorithms for short range molecular dynamics. J Computing Physics 117:1–19

[8] Finocchiaro G, Wang T, Hoffmann R, Gonzalez A, Wade R (2003) DSMM: a database of simulated molecular motions. Nucleic Acids Res 31(1):456–457

[9] Bamdad M, Alavi S, Najafi B, Keshavarzi E (2006) A new expression for radial distribution function and infinite shear modulus. Chem Phys 325:554–56

[10] Stark JL, Murtagh F (2006) Astronomical image and data analysis. Springer, Berlin, Heidelberg

[11] Frenkel D, Smit B (2002) Understanding molecular simulation: from algorithm to applications. Comput Sci Ser 1. Academic Press

[12] Grosberg AY and Khokhlov AR. (1994) Statistical Physics of Macromolecules (translated by Atanov YA), AIP Press.

[13] Gao, C.; Kulkarni, S. D.; Morris, J. F.; Gilchrist, J. F. (2010). "Direct investigation of anisotropic suspension structure in pressure-driven flow". Physical Review

[14] Wochner, P.; Gutt, C.; Autenrieth, T.; Demmer, T.; Bugaev, V.; Ortiz, A. D.; Duri, A.; Zontone, F.; Grubel, G.; Dosch, H. (2009). "X-ray cross correlation analysis uncovers hidden local symmetries in disordered matter". Proceedings of the National Academy of Sciences

[15] Walid G. Aref, Ihab F. Ilyas, SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees,Journal of Intelligent Information Systems (JIIS), (2/3), 215 -240, 2001.

[16] Mohamed Y. Eltabakh, Ramy Eltarras, Walid G. Aref,Space-partitioning Trees inside PostgreSQL: Realizing and Performance, The 22nd International Conference of Data Engineering, (ICDE), 100 -111, 2006.

[17] Thanaa M. Ghanem, Rahul Shah, Mohamed F. Mokbel, Walid G. Aref, Jeffry S. Vitter, Bulk Operations for Space-Partitioning Trees, The 20th International Conference of Data Engineering, (ICDE), 29 -41, 2004.

[18] Walid G. Aref, Ihab F. Ilyas, An Extensible Index for Spatial Databases, The 13th International Conference on Scientific and Statistical Database Management (SSDBM), 49 -58, 2001.

[19] A. Kumar, V. Grupcev, M. Berrada, J. Fogarty, Y. Tu, X. Zhu, S. Pandit, and Y. Xia. "DCMS: A data analytics and management system for molecular simulation." Journal of Big Data 2:9, November 2014.

[20] Chen S, Tu Y-C, Xia Y (2011) Performance analysis of a dual-tree algorithm for computing spatial distance histograms. VLDB Journal 20(4):471–494

[21] Michaud-Agrawal N, Denning E, Woolf T, Beckstein O (2011) MDAnalysis: A Toolkit for the Analysis of Molecular Dynamics Simulations. J Comput Chem 32(10):2319–2327

[22] A.N. Niazi, B. Rabideau and A.E. Ismail. Effects of Water Concentration on the Structural and Diffusion Properties of Imidazolium-Based Ionic Liquid/Water Mixtures. J. Phys. Chem. B. DOI: 10.1021/ (2013).