

PUSH BASED SYSTEM-APPLICATION FOR MOLECULAR DATA ANALYSIS

Suraj Nakka U20904490
Sudheendra Gembali U28986906
Akshita Tripuraneni U20731838

I. ABSTRACT

Because of the headway of the current PC recreation frameworks and cutting edge instrumentation, numerous logical fields create, and require control of enormous information. Known exploratory information investigation frameworks, and also conventional DBMSs, take after a force based sort of engineering outline, where the executed questions command the information required.

In this paper, we adopted a method to process the data efficiently [MAIN REFERENCE]. We have used SPARK platform to load the input data file frame by frame into the memory for accessing. Each time a frame is accessed, the frame is queried for calculations such as Moment of Inertia, Sum of Masses, Dipole Moment, etc. Each time a frame is queried, an output file is generated containing the outputs of those queries. Subsequently, outputs are written into their respective files (text files).

II. INTRODUCTION

The measure of information produced by experimental applications running on cutting edge PC recreation frameworks is becoming quick. This huge information is forcing a critical weight on the information investigation programming. Despite the fact that the current information investigation frameworks are designed to manage substantial volume of information, they are not upgraded for high throughput information examination. Customarily, the information is accessible just for certain timeframe (e.g., streams) so the breaking down framework ought to respond to it in an extremely productive manner. Another test forced to the information examination frameworks is that the information is typically being gotten to through abnormal state investigative questions which calculation is significantly more intricate than the calculation of straightforward totals. Some of these questions are generally the bottleneck of the examination frameworks, in light of the fact that they take a considerable measure of time to be executed and the

frameworks are not intended to handle different inquiries on the same information stream in the meantime, therefore diminishing the general proficiency of the information investigation.

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]. In addition, these records may have diverse geological starting point, presenting distinctive dialects and structures and as a rule containing spontaneous messages, blunders, malevolent substance, and so on. In this manner, some low level information consistency and cleaning on top of the information access and administration issues ought to be viewed as and perhaps fused during the time spent scientific examination with a specific end goal to accomplish applicable result.

The already expressed tremendous data constrained issues are also as often as possible appeared in various other application fields, e.g., test data examination. In various consistent fields, the crucial computational system for looking at physical/blend properties of basic structures is the atom entertainment. Such reenactments, when performed in the field of assistant and sub-nuclear science, are routinely called Molecular Simulations (MS). MS are PC amusements of complex regular, physical or invention structures. These reenactments, in perspective of some major speculative models, are by and large utilized as a key examination instrument for thinking about the behavior of the typical structures. The crucial units of such structures are normal particles, (for instance, particles, molecules, stars, et cetera.) and they relate among each other for a beyond any doubt time allotment taking after guessed conventional forces. The amount of these essential units is colossal, generally in the extent of numerous thousands to millions. For example, a single reenacted review of a collagen fiber may include 890,000 particles. Besides, the multiplications produce datasets including more than one sneak peak (packaging) of the structure's state at diverse time minutes. Each of these edges contain each one of the particles, together with each one of their estimations, for instance, spatial bearings, mass, charge, rate, forces, et cetera. Conventionally, a noteworthy number of such housings (in the few thousands) are being

conveyed and set away all through an ordinary reenactment. Sums measured in the midst of the generations are bankrupt down 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, by supplying a model description of the biophysical and biochemical processes that are being unfold at a nanoscopic scale.

So as to accomplish target revelation and to clarify the workings of the exploratory frameworks, researcher must dissect the information delivered by the MS. These investigations intermittently contain calculation of exceptionally complex amounts that show measurable properties of the information. Such questions are of extraordinary significance to researcher on the grounds that they are the essential get together pieces for a progression of basic amounts expected to diagram the experimental frameworks [6]. The routines by which the huge information is being gotten to and in addition these inquiries are executed can either expand or diminish the effectiveness of the framework.

We trust a push-based sort framework can cure such issues. In such arrangement the request don't request the data. Perhaps, the data is pushed onto the inquiries normally and is being readied by the dynamic request. This diminishes the I/O development overhead and likewise the cpu/data lethargy procured for each request that would have been displayed by a power based setup. Such a system arrangement is especially suitable for some exploratory applications which share the going with segments. To begin with, the exploratory examination much of the time incorporates executing different logical request frequently took care of on a gigantic part, if not most of the created data. Second, countless fields use the same surely understood, and minimal number of data examination primitives as the crucial squares on which they develop their disclosure. Besides, three, the exploratory data, once set away is never balanced. New data just adds to the current dataset, making the data faultless contender for spilling.

To consolidate, data concentrated applications every now and again require tremendous measure of storage space and genuine planning capacity, moreover need brisk data access and high throughput data examination. In this way, the need of a structure that will be streamlined to get to huge data fast, with high throughput, and also successfully execute the deliberate request with a likelihood of running diverse inquiries on the same data stream is of a mind

boggling criticalness to scientists.

The following modules have been computed:

Sum of Masses	$= \sum m_i$
Sum of Charges	$= \sum q_i$
Moment of Inertia	$= \sum m_i r_i^2$
Moment of Inertia on z-axis	$= \sum m_i r_z^2$
Dipole Moment	$= \sum q_i r_i$
Center of Mass	$= I/M$

III. One-body queries: The vast majorities of the inquiry modules said above are not unpredictable and just contain calculations of straightforward one-body functions. These queries were coded as discrete modules in our framework. Each of these modules takes few qualities as information (e.g., molecule determination, outlines selection (for the autocorrelation capacities), number of particles, and so on.). The framework pushes the information as it gets to be accessible onto these modules. The questions are being executed on the choice and are placed in a "prepared" mode, anticipating the following outline's information. In the first place, the more essential inquiries, similar to aggregate mass, are being processed. The aftereffects of such inquiries are brief put away (in primary memory) and are accessible for use whenever a more perplexing query needs them.

A. Working of the system

The following is the procedure we followed to implement:

- (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)
- (3) Push the data to queries
- (4) A query acts upon the pushed data (first executing the lower level, sub-queries)
- (5) Repeat steps 3-4 for all queries
- (6) Output the results into text files
- (7) Go to step 2 and load the next frame

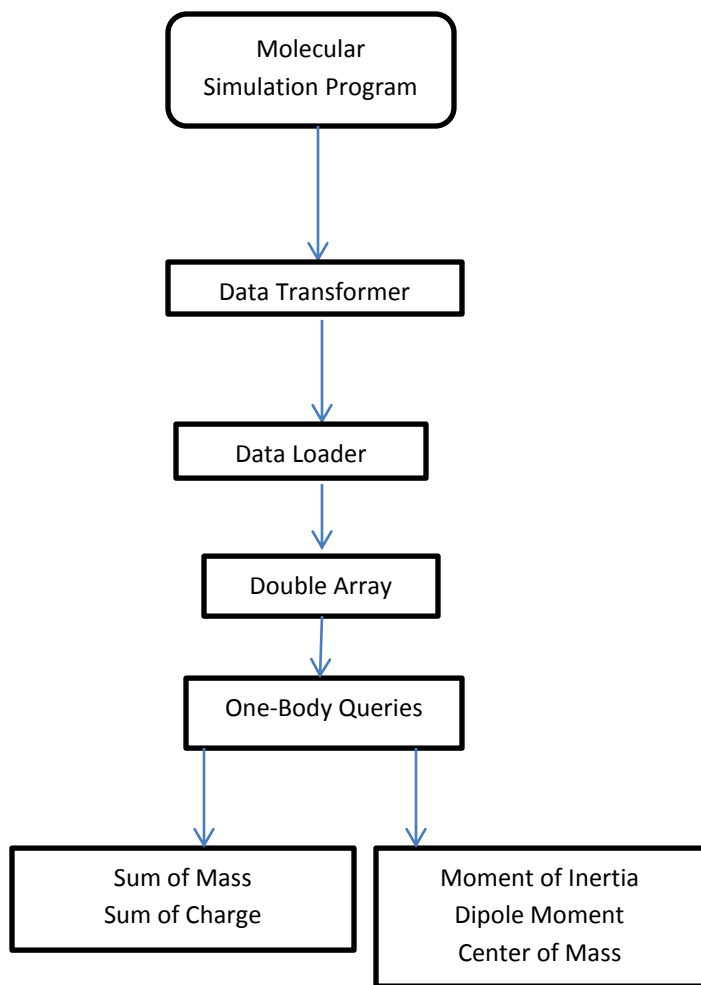


Fig-1

IV. ENVIRONMENT USED

The following environment has been used to build the application:

Platform: **Spark 1.1.0**

Language: **SCALA**

Build Tool: **SBT**

A. Spark Overview

Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala and Python, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including **Spark SQL** for SQL and structured data

processing, **MLlib** for machine learning, **GraphX** for graph processing, and **Spark Streaming**.

Spark 1.1.0 is the second release on the API-good 1.X line. It is Spark's biggest release ever, with commitments from 171 designers. This release brings operational and execution changes in Spark center including another usage of the Spark mix intended for substantial scale workloads. Spark 1.1 adds huge augmentations to the most current Spark modules, MLlib and Spark SQL. Flash SQL presents a JDBC server, byte code era for quick expression assessment, an open sorts API, JSON backing, and different components and enhancements. MLlib presents another insights library alongside a few new calculations and enhancements. Spark 1.1 additionally manufactures Spark's Python support and adds new parts to the Spark Streaming module.

B. Running the Examples and Shell

Spark comes with several sample programs. Scala, Java and Python examples are in the examples/src/main directory. To run one of the Java or Scala sample programs, use `bin/run-example <class> [params]` in the top-level Spark directory. (Behind the scenes, this invokes the more general [spark-submit script](#) for launching applications). For example,

```
./bin/run-example SparkPi 10
```

You can also run Spark interactively through a modified version of the Scala shell. This is a great way to learn the framework.

```
./bin/spark-shell --master local[2]
```

“-- master local[2]” means the process is split among two parallel threads resulting in better performance.

The application uses “--master local[4]” that means the tasks are being carried out on 4 parallel threads.

The --master option specifies the master URL for a distributed cluster, or local to run locally with one thread, or local[N] to run locally with N threads. You should start by using local for testing. For a full list of options, run Spark shell with the --help option.

Spark also provides a Python API. To run Spark interactively in a Python interpreter, use bin/pyspark:

```
./bin/pyspark --master local[2]
```

Example applications are also provided in Python. For example,

```
./bin/spark-submit examples/src/main/python/pi.py 10
```

C. Launching on a Cluster

The Spark cluster mode overview explains the key concepts in running on a cluster. Spark can run both by itself, or over several existing cluster managers. It currently provides several options for deployment:

- Amazon EC2: our EC2 scripts let you launch a cluster in about 5 minutes
- Standalone Deploy Mode: simplest way to deploy Spark on a private cluster
- Apache Mesos
- Hadoop YARN

D. Spark Configuration

Spark provides three locations to configure the system:

- Spark properties control most application parameters and can be set by using a SparkConf object, or through Java system properties.
- Environment variables can be used to set per-machine settings, such as the IP address, through the conf/spark-env.sh script on each node.
- Logging can be configured through log4j.properties.

V. CODE IN SCALA FOR THE ABOVE MODULES

A. For Calculating Moment of Inertia ($\sum(m_i * \sqrt{(x^2 + y^2 + z^2)})$)

```
for (i <- 0 to 100)           /*For all the frames*/
{
    var logRDD =
sc.textFile(logFile+"%s.txt".format(i),2).cache

    var tempX = logRDD.filter(line =>
line.contains("ATOM")).map(_._split("
")(4)).map(_._toFloat).map(x => x*x).collect

    var tempY = logRDD.filter(line =>
line.contains("ATOM")).map(_._split("
")(5)).map(_._toFloat).map(x => x*x).collect

    var tempZ = logRDD.filter(line =>
line.contains("ATOM")).map(_._split("
")(6)).map(_._toFloat).map(x => x*x).collect

    var k = 0
    for(k <- 0 to tempX.length-1) {
        tempXYZRDD(k) =
tempX(k)+tempY(k)+tempZ(k)
    }

    var tempXYZ = tempXYZRDD.map(x => sqrt(x))

    var tempM = logRDD.filter(line =>
line.contains("ATOM")).map(_._split("
")(8)).map(_._toDouble).collect

    var j = 0
    for (j <- 0 to tempXYZ.length-1) {
        tempRDD(j) = tempM(j)*tempXYZ(j)
    }

    moiRDD = tempRDD.reduce(_+_ )
/*Calculating sum of masses for each frame*/

    println("Frame %s -> Moment of Inertia :
%f\n".format(i,moiRDD))

    str += "Frame %s -> Moment of Inertia :
%f\n".format(i,moiRDD)
}
```

Sample Output for the first 15 frames

```

F:\ADB Project-3\proj3_MSDataAnalysis\proj3_MSDataAnalysis\output\MOL.txt - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins
dpm.scala x moi.scala x moiz.scala x som.scala x soq.scala x DP
1 Starting Time Sat Dec 05 05:59:26 EST 2015
2 Frame 0 -> Moment of Inertia : 52951118.879528
3 Frame 1 -> Moment of Inertia : 52883565.567591
4 Frame 2 -> Moment of Inertia : 52777343.130193
5 Frame 3 -> Moment of Inertia : 52636461.466502
6 Frame 4 -> Moment of Inertia : 52523582.716053
7 Frame 5 -> Moment of Inertia : 52468008.380997
8 Frame 6 -> Moment of Inertia : 52399420.228187
9 Frame 7 -> Moment of Inertia : 52379886.683617
10 Frame 8 -> Moment of Inertia : 52376091.908597
11 Frame 9 -> Moment of Inertia : 52414733.563967
12 Frame 10 -> Moment of Inertia : 52396982.412910
13 Frame 11 -> Moment of Inertia : 52458144.328263
14 Frame 12 -> Moment of Inertia : 52474161.155478
15 Frame 13 -> Moment of Inertia : 52470992.291862

```

```

var tempxyz = tempxyzRDD.map(x => sqrt(x))
//Compute sqrt(x2+y2+z2)

var tempq = logRDD.filter(line =>
line.contains("ATOM")).map(_.split("
")(7)).map(_.toDouble).collect //Extract Charge

var j = 0
for (j <- 0 to tempxyz.length-1) {
    tempRDD(j) = tempq(j)*tempxyz(j)
//Compute Charge*sqrt(x2+y2+z2)
}

dpmRDD = tempRDD.reduce(_+_ )
//Calculating Sum of Dipole Moment for each frame

println("Frame %s -> Dipole Moment :
%f\n".format(i,dpmRDD))

str += "Frame %s -> Dipole Moment :
%f\n".format(i,dpmRDD)

}

```

B. For Calculating Dipole Moment ($\sum (q_i \times \sqrt{x^2 + y^2 + z^2})$)

```

for (i <- 0 to 100)          /*For all the frames*/
{
    var logRDD =
sc.textFile(logFile+"%s.txt".format(i),2).cache

    var tempx = logRDD.filter(line =>
line.contains("ATOM")).map(_.split("
")(4)).map(_.toFloat).map(x => x*x).collect //Extract x2

    var tempy = logRDD.filter(line =>
line.contains("ATOM")).map(_.split("
")(5)).map(_.toFloat).map(x => x*x).collect //Extract y2

    var tempz = logRDD.filter(line =>
line.contains("ATOM")).map(_.split("
")(6)).map(_.toFloat).map(x => x*x).collect //Extract z2

    var k = 0
    for(k <- 0 to tempx.length-1) {
        tempxyzRDD(k) =
tempx(k)+tempy(k)+tempz(k) //Compute x2+y2+z2
    }
}

```

Sample Output for the first 15 frames

```

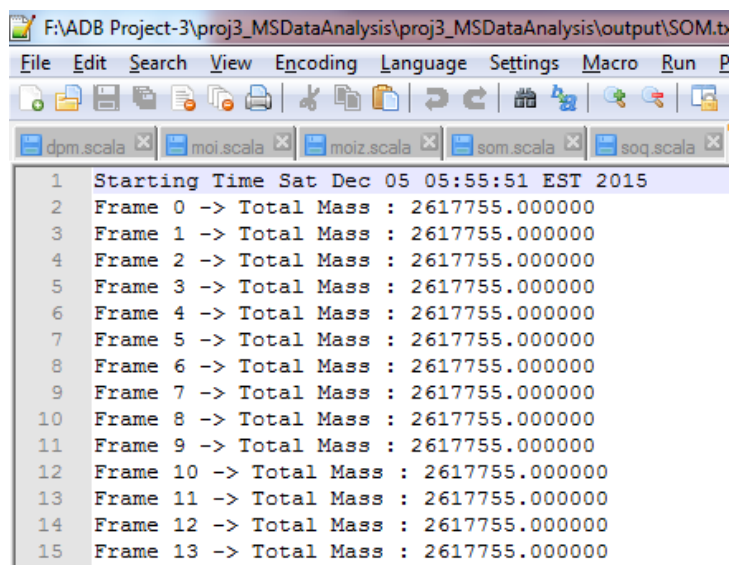
F:\ADB Project-3\proj3_MSDataAnalysis\proj3_MSDataAnalysis\output\DPM.txt - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window
dpm.scala x moi.scala x moiz.scala x som.scala x soq.scala x DPM.txt x
1 Starting Time Sat Dec 05 06:05:47 EST 2015
2 Frame 0 -> Dipole Moment : -210.604019
3 Frame 1 -> Dipole Moment : -200.311102
4 Frame 2 -> Dipole Moment : -215.457183
5 Frame 3 -> Dipole Moment : -209.216526
6 Frame 4 -> Dipole Moment : -214.762106
7 Frame 5 -> Dipole Moment : -207.346250
8 Frame 6 -> Dipole Moment : -209.598458
9 Frame 7 -> Dipole Moment : -208.973359
10 Frame 8 -> Dipole Moment : -215.817458
11 Frame 9 -> Dipole Moment : -213.309355
12 Frame 10 -> Dipole Moment : -204.975949
13 Frame 11 -> Dipole Moment : -207.517303
14 Frame 12 -> Dipole Moment : -206.173670
15 Frame 13 -> Dipole Moment : -201.854923
16 Frame 14 -> Dipole Moment : -203.803992
17 Frame 15 -> Dipole Moment : -202.352515

```

C. For Calculating Sum of Masses ($\sum(m_i)$)

```
var framesize
for (i <- 0 to 100)          /*For all the frames*/
{
    var tempRDD =
massRDD.slice(framesize*i,framesize*(i+1)) /*slicing
each frame into temporary array*/
    somRDD = tempRDD.reduce(_+_ )
/*Calculating sum of masses for each frame*/
    println("Frame %s -> Total Mass :
%f\n".format(i,somRDD))
    str += "Frame %s -> Total Mass :
%f\n".format(i,somRDD)
}
val end_time = Calendar.getInstance().getTime()
str += "Ending Time %s".format(end_time)
printToFile(str)           /*store the output to a
text file */
}
```

Sample Output for the first 15 frames



```
1 Starting Time Sat Dec 05 05:55:51 EST 2015
2 Frame 0 -> Total Mass : 2617755.000000
3 Frame 1 -> Total Mass : 2617755.000000
4 Frame 2 -> Total Mass : 2617755.000000
5 Frame 3 -> Total Mass : 2617755.000000
6 Frame 4 -> Total Mass : 2617755.000000
7 Frame 5 -> Total Mass : 2617755.000000
8 Frame 6 -> Total Mass : 2617755.000000
9 Frame 7 -> Total Mass : 2617755.000000
10 Frame 8 -> Total Mass : 2617755.000000
11 Frame 9 -> Total Mass : 2617755.000000
12 Frame 10 -> Total Mass : 2617755.000000
13 Frame 11 -> Total Mass : 2617755.000000
14 Frame 12 -> Total Mass : 2617755.000000
15 Frame 13 -> Total Mass : 2617755.000000
```

CONCLUSION

A SPARK application has been built using SCALA language that loads the input data frame by frame thereby causing less overhead on the memory. The application provides a GUI giving options to select one of the queries to be implemented on the input data. Upon selection of an option, the data is loaded one frame at a time and the respective computations are done. The application offers high performance for the given input data as the computations are done in parallel on the data retrieved (from input data). A Spark.RDD makes this possible by pushing the data into multiple task threads.

REFERENCES

Main Reference: "Push Based System for Molecular Simulation Data Analysis ", Vladimir Grupcev, Yi-Cheng Tu , Joseph Fogarty, Sagar Pandit

- [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] Daan Frenkel et. al., *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.