Data Stream Management : Processing Molecular Simulation Data using Spark Streaming

Kodanda Ravi Kiran Putta – U31271622
Sainikhil Reddy Basupally - U98314497
Avanish Yemula - U46426155

1. Abstract:-

Lots of information is created in almost every area, and it's important for making progress in those areas. To study this data closely, we need a lot of computation power. Molecular Simulation is a strong method for figuring out how natural systems work. During the simulation, a ton of data is produced as we observe things like where they are and how they change over time. The tricky part is dealing with the tough questions that need a lot of computing effort to answer.

Even though there are various tools available to address this challenge, this project takes a different approach. We employed Apache Spark Streaming, a contemporary big data platform, to parallelize the computation of analytical queries in the context of streaming data. MsSpark comprises three layers: the Apache Spark Streaming layer, MS DStream layer, and MS Query Processing layer. The MS DStream layer supports data specific to Molecular Simulation in a streaming context. The MS Query Processing layer carries out the execution of analytical queries. Caching is utilized to enhance performance, and the system can be expanded to accommodate additional analytical queries in the streaming scenario.

2. Introduction: -

A lot of data is being created in many areas, like physics and genomics. As this data grows, the current ways we store and process it have some limitations. The challenge is to create both the hardware and software that can get useful information from these big sets of data.

Storing the data is not a big problem now because memory costs less. But when it comes to processing the data, map reduce is a popular way to handle large datasets by using parallel and distributed algorithms on a group of computers. It's a quick and effective solution for working with big datasets.

In this Project, we implemented the pipeline using Spark Streaming framework which consumes the data from kafka topic, perform the near real time computation and printing the results to console. To optimize the pipeline, performed the parameter tuning, and batch duration tuning. Partitioned the data in Kafka topic using Frame number of atom as key.

3. Apache Spark:-

Apache Spark is unified analytics engine, which is used for large scale data processing. It provides high-level APIs in Java, Scala, Python and R, 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, pandas API on Spark for pandas workloads, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for incremental computation and stream processing.

For this project Spark Structured Streaming is used which is a seperate framework in Spark. Spark Streaming provides the DStream API, which is internally built on RDD's. DStream provides the data divided into Chunks as RDDs received from the source of streaming to be processed. Spark streaming works on something which we call a micro batch. The stream pipeline is registered with some operations and the Spark polls the source after every batch duration (defined in the application) and then a batch is created of the received data. i.e. each incoming record belongs to a batch of DStream. Each batch represents an RDD.

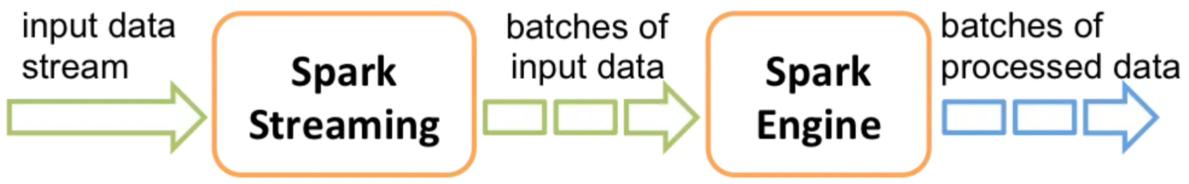
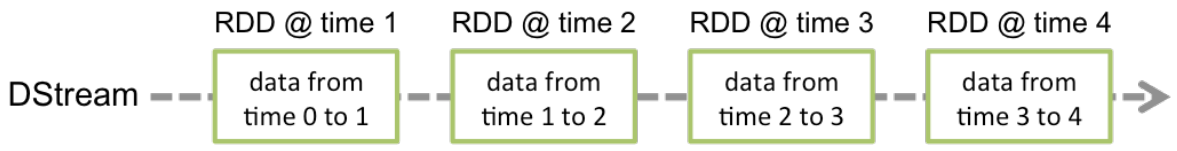


Figure 1.1:- Data Processing model in Spark Streaming

To provide fault-tolerance and stateful event transformation Spark Streaming uses the checkpoint. In this project, as the metrics calculated in current batch is dependent on the metrics calculated so far for that key, stateful transformations is used with updateStateByKey operation. To initialize a Spark Streaming program, a **StreamingContext** object has to be created which is the main entry point of all Spark Streaming functionality.

Discretized Stream or **DStream** is the basic abstraction provided by Spark Streaming. It represents a continuous stream of data, either the input data stream received from source, or the processed data stream generated by transforming the input stream. Internally, a DStream is represented by a continuous series of RDDs, which is Spark's abstraction of an immutable, distributed dataset. Each RDD in a DStream contains data from a certain interval, as shown in the following figure.



Any operation applied on a DStream translates to operations on the underlying RDDs.

4. Apache Kafka Integration

Apache Kafka is a distributed streaming platform that plays a crucial role in our Spark Streaming pipeline. It acts as a highly scalable and fault-tolerant message broker, facilitating the ingestion of molecular data in real-time. This section outlines the key aspects of our Kafka integration.

4.1 Kafka Configuration

To establish a seamless connection between our Spark Streaming application and Kafka, we have configured Kafka parameters as follows:

- **Bootstrap Servers:** The list of Kafka broker addresses.
- **Key Deserializer:** Deserializer for Kafka message keys.
- **Value Deserializer:** Deserializer for Kafka message values.
- **Group ID:** A unique identifier for the consumer group.
- **Auto Offset Reset:** Specifies what to do when there is no initial offset or if the current offset does not exist.

These configurations are essential for Kafka's reliable and efficient message processing within the Spark Streaming context.

4.2 Kafka Topic Selection

In our streaming application, the choice of Kafka topics is pivotal. The Kafka topic serves as the conduit for molecular data from producers to consumers. We have employed a dynamic approach, allowing users to specify the Kafka topic dynamically during runtime. This flexibility enhances the adaptability of our system to diverse molecular data sources. For this project purpose I created the kafka topic with 8 partitions and replication factor of 4.

4.3 Streaming from Kafka

Utilizing the KafkaUtils API, our Spark Streaming application creates a direct stream from Kafka. This direct integration ensures that molecular data is consumed in real-time, providing timely insights into dynamic molecular structures.

```
val stream = KafkaUtils.createDirectStream[String, String]( ssc,
LocationStrategies.PreferConsistent, ConsumerStrategies.Subscribe[String,
String](Set(broadcastVar1.value), kafkaParams) )
```

The above code snippet demonstrates the creation of a direct stream, leveraging the specified Kafka topic and configuration parameters.

4.4 Fault Tolerance and Scalability

One of Kafka's notable features is its inherent fault tolerance and scalability. By distributing data across multiple partitions and broker nodes, Kafka ensures high availability and fault tolerance. This architecture aligns with our goal of building a robust and scalable streaming pipeline for molecular data analysis.

4.5 Real-time Data Ingestion

Kafka's ability to handle high-throughput data streams in real-time aligns seamlessly with our Spark Streaming application's requirements. The combination of Kafka and Spark Streaming allows us to process molecular data as it arrives, enabling timely and dynamic analysis.

6. Algorithm for Spark Streaming Pipeline on Molecular Data

1. **Initialize Spark Streaming Context:**
    - Create a SparkConf with master as "local[*]" and set the application name as "KafkaStreamProcessing".
    - Create a StreamingContext with the configured SparkConf and a batch interval of 3 seconds.
    - Set a checkpoint location for stateful streaming operations.
2. **Configure Kafka Parameters:**
    - Build Kafka parameters with the bootstrap servers, key and value deserializers, group id, and auto offset reset properties.
3. **Broadcast Variables:**
    - Broadcast the Kafka topic name and algorithm index to all executors.
4. **Create DStream from Kafka:**
    - Create a direct stream from Kafka using the specified topic and Kafka parameters.
    - Split each row at spaces, filter out the header, and map the data into a key-value pair.
5. **Cache the DStream:**
    - Cache the DStream as it will be used repeatedly in the subsequent operations.
6. **Run Selected Algorithm:**
    - Based on the user-provided algorithm index:
        - If index is 1, update state by calculating the sum of masses of atoms.
        - If index is 2, update state by calculating the moment of inertia of atoms.
        - If index is 3, update state by calculating the dipole moment of atoms.
        - If index is 4, update state by calculating the center of mass of atoms.
        - If index is 5, update state by calculating the average charge of atoms.
        - If index is 6, update state by calculating the radius of gyration along the X-axis.
        - If index is 7, update state by calculating the radius of gyration along the Y-axis.

        - If index is 8 update state by calculating the radius of gyration along the Z-axis.
7. **Print Results to Console:**
    - Print the results of the selected algorithm to the console.
8. **Start Streaming Context:**
    - Start the Spark Streaming context.
9. **Await Termination:**
    - Wait for the execution to stop.

7. Algorithmic Processing:-

Algorithm for Sum of Masses:

Input:

- **newValues**: List of tuples representing (X-position, Y-position, Z-position, charge, mass, count) of atoms for a given frame.
- **currentState**: Current sum of masses for the given frame if available in the state.

Algorithm Steps:

1. Initialize **currStateV** with the current state value or 0.0 if not available.
2. Calculate the sum of masses (**totalMass**) from the mass component of each atom in **newValues**.
3. Update the state with the sum of masses and the current state value.
4. Return the updated state.

Algorithm for Moment of Inertia:

Input:

- **newValues**: List of tuples representing (X-position, Y-position, Z-position, charge, mass, count) of atoms for a given frame.
- **currentState**: Current moment of inertia for the given frame if available in the state.

Algorithm Steps:

1. Initialize **currState** with the current state values (0.0, 0.0, 0.0) if not available.
2. For each atom in **newValues**, accumulate the products of mass and position along X, Y, and Z axes.
3. Update the state with the accumulated values and the current state values.
4. Return the updated state.

Algorithm for Center of Mass:

Input:

- **newValues**: List of tuples representing (X-position, Y-position, Z-position, charge, mass, count) of atoms for a given frame.
- **currentState**: Current center of mass for the given frame if available in the state.

Algorithm Steps:

1. Initialize **currStateV** with the current state values (0.0, 0.0, 0.0, 0.0) if not available.
2. For each atom in **newValues**, accumulate the products of mass and position along X, Y, Z axes, and total mass.
3. Calculate the total count of atoms.
4. Update the state with the weighted average of positions and total mass.
5. Return the updated state.

Algorithm for Dipole Moment:

Input:

- **newValues**: List of tuples representing (X-position, Y-position, Z-position, charge, mass, count) of atoms for a given frame.
- **currentState**: Current dipole moment for the given frame if available in the state.

Algorithm Steps:

1. Initialize **currState** with the current state values (0.0, 0.0, 0.0) if not available.
2. For each atom in **newValues**, accumulate the products of charge and position along X, Y, and Z axes.
3. Update the state with the accumulated values and the current state values.
4. Return the updated state.

Algorithm for Average Charge:

Input:

- **newValues**: List of tuples representing (X-position, Y-position, Z-position, charge, mass, count) of atoms for a given frame.
- **currentState**: Current average charge and total atom count for the given frame if available in the state.

Algorithm Steps:

1. Initialize **currState** with the current state values (0.0, 0.0) if not available.
2. Calculate the sum of charges and the count of atoms from each atom in **newValues**.
3. Update the state with the weighted average of charges and the total count of atoms.
4. Return the updated state.

Algorithm for Radius of Gyration (X-axis):

Input:

- **newValues**: List of tuples representing (X-position, Y-position, Z-position, charge, mass, count) of atoms for a given frame.
- **currentState**: Current radius of gyration along the X-axis and total mass for the given frame if available in the state.

Algorithm Steps:

1. Initialize **currState** with the current state values (0.0, 0.0) if not available.

2. For each atom in **newValues**, accumulate the products of X-position and mass.
3. Calculate the total mass and moment of inertia along the X-axis.
4. Update the state with the calculated radius of gyration.
5. Return the updated state.

Algorithm for Radius of Gyration (Y-axis):

Input:

- **newValues**: List of tuples representing (X-position, Y-position, Z-position, charge, mass, count) of atoms for a given frame.
- **currentState**: Current radius of gyration along the Y-axis and total mass for the given frame if available in the state.

Algorithm Steps:

1. Initialize **currState** with the current state values (0.0, 0.0) if not available.
2. For each atom in **newValues**, accumulate the products of Y-position and mass.
3. Calculate the total mass and moment of inertia along the Y-axis.
4. Update the state with the calculated radius of gyration.
5. Return the updated state.

Algorithm for Radius of Gyration (Z-axis):

Input:

- **newValues**: List of tuples representing (X-position, Y-position, Z-position, charge, mass, count) of atoms for a given frame.
- **currentState**: Current radius of gyration along the Z-axis and total mass for the given frame if available in the state.

Algorithm Steps:

1. Initialize **currState** with the current state values (0.0, 0.0) if not available.
2. For each atom in **newValues**, accumulate the products of Z-position and mass.
3. Calculate the total mass and moment of inertia along the Z-axis.
4. Update the state with the calculated radius of gyration.
5. Return the updated state.

8. Results:-

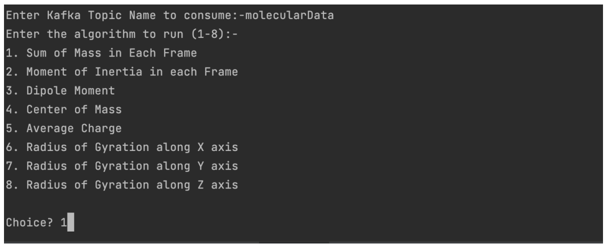
Streaming Pipeline ran with the following config:-

- ➔ Batch Duration Interval :- 3 sec
- ➔ Master:- local[*]
- ➔ Bootstrapserver:- localhost:9092
- ➔ auto.offset.reset:- earliest
- ➔ key Deserializer:- String Deserializer
- ➔ value Deserializer:- String Deserializer

Note:- Before testing each pipeline to simulate the streaming scenario, we have written the producer program which ingests the data from text file to kafka topic.

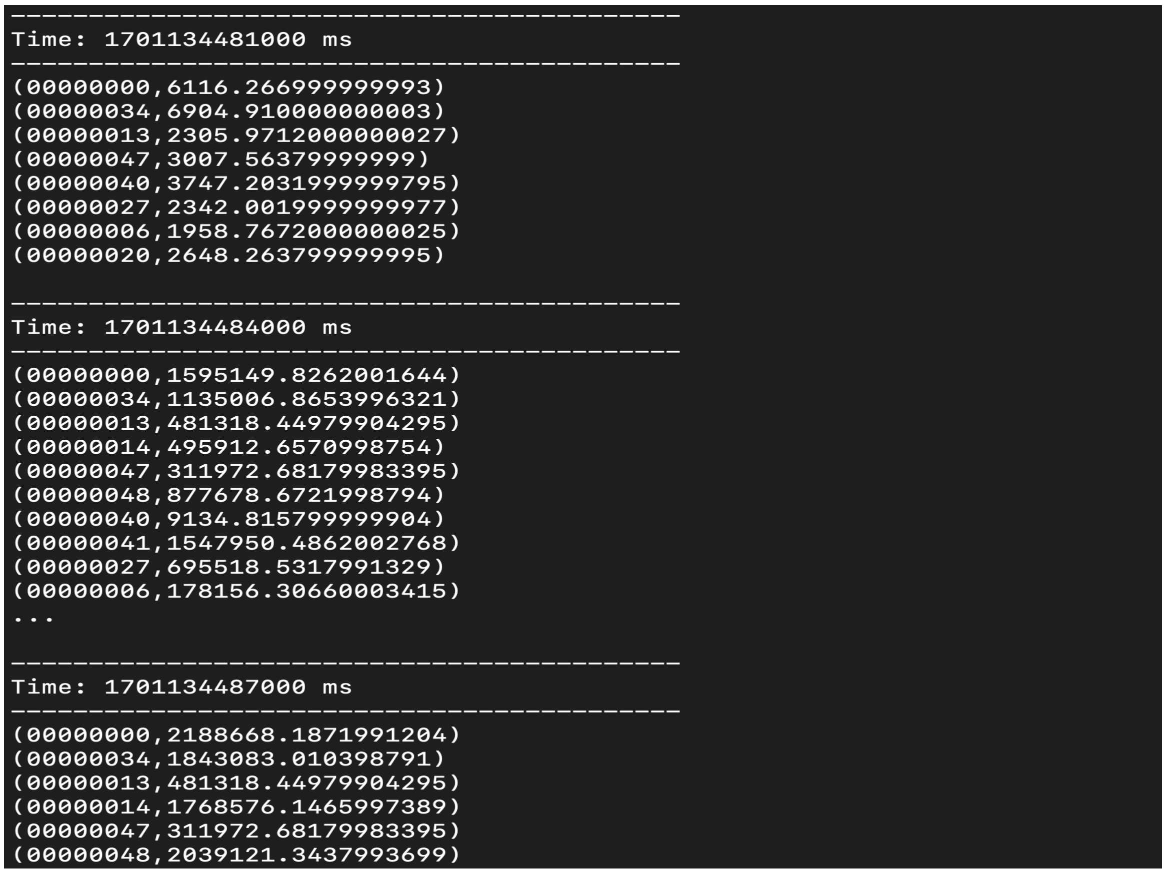
Hence before running each algorithm make sure to delete data in topic and then start the streaming pipeline, after that run the producer program which publishes the data to kafka topic.

8.1) Running Sum of mass in Each Frame Algorithm:-



```
Enter Kafka Topic Name to consume:-molecularData
Enter the algorithm to run (1-8):-
1. Sum of Mass in Each Frame
2. Moment of Inertia in each Frame
3. Dipole Moment
4. Center of Mass
5. Average Charge
6. Radius of Gyration along X axis
7. Radius of Gyration along Y axis
8. Radius of Gyration along Z axis

Choice? 1
```

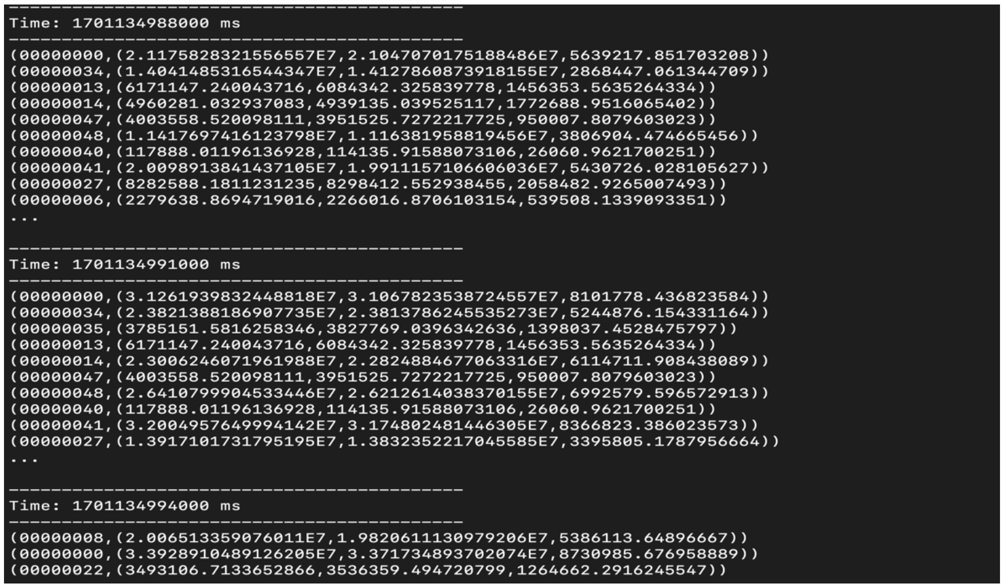


```
-------------------------------------------
Time: 1701134481000 ms
-------------------------------------------
(00000000,6116.266999999993)
(00000034,6904.910000000003)
(00000013,2305.9712000000027)
(00000047,3007.5637999999999)
(00000040,3747.2031999999795)
(00000027,2342.0019999999977)
(00000006,1958.7672000000025)
(00000020,2648.263799999995)

-------------------------------------------
Time: 1701134484000 ms
-------------------------------------------
(00000000,1595149.8262001644)
(00000034,1135006.8653996321)
(00000013,481318.44979904295)
(00000014,495912.6570998754)
(00000047,311972.68179983395)
(00000048,877678.6721998794)
(00000040,9134.815799999904)
(00000041,1547950.4862002768)
(00000027,695518.5317991329)
(00000006,178156.30660003415)
...

-------------------------------------------
Time: 1701134487000 ms
-------------------------------------------
(00000000,2188668.1871991204)
(00000034,1843083.010398791)
(00000013,481318.44979904295)
(00000014,1768576.1465997389)
(00000047,311972.68179983395)
(00000048,2039121.3437993699)
```

8.2) Result of MOI:-

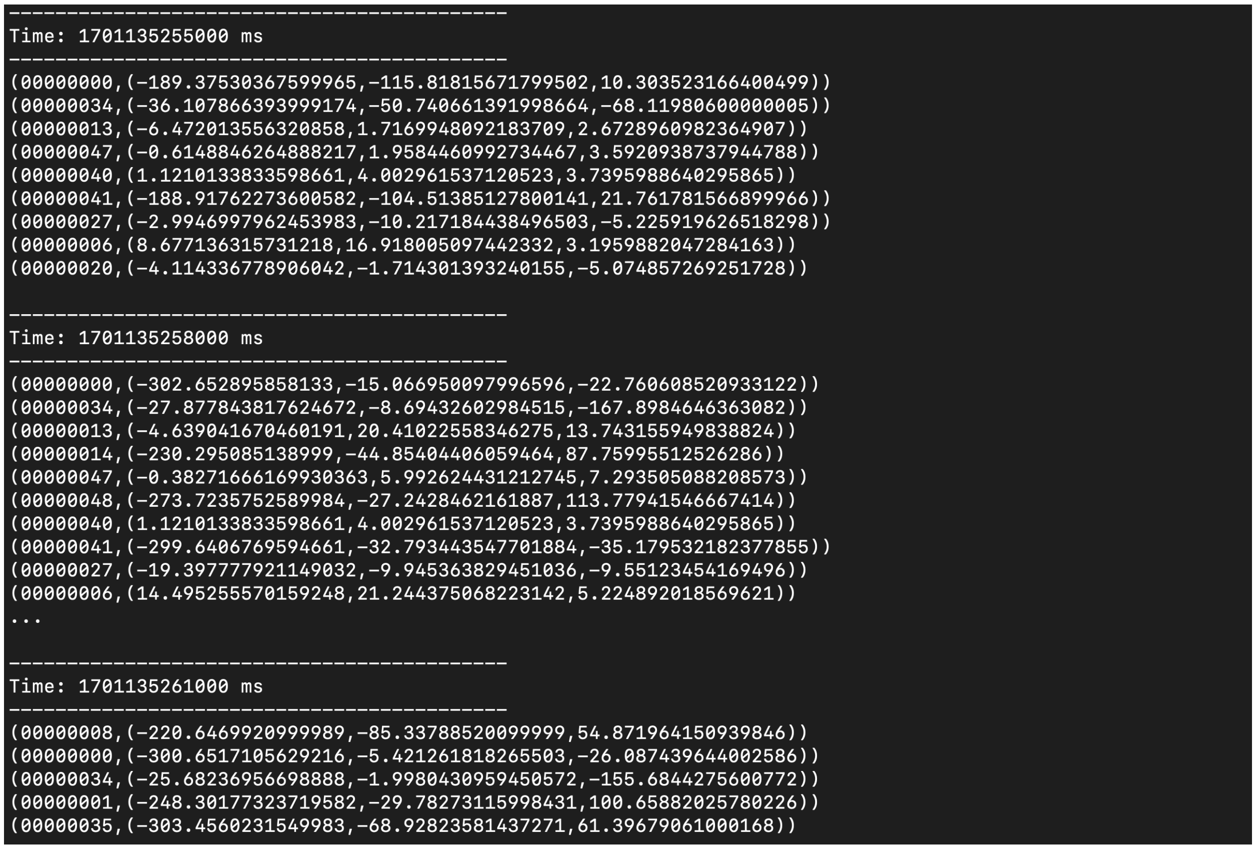


```
-------------------------------------------
Time: 1701134988000 ms
-------------------------------------------
(00000000,(2.1175828321556557E7,2.1047070175188486E7,5639217.851703208))
(00000034,(1.4041485316544347E7,1.4127860873918155E7,2868447.061344709))
(00000013,(6171147.240043716,6084342.325839778,1456353.5635264334))
(00000014,(4960281.032937083,4939135.039525117,1772688.9516065402))
(00000047,(4003558.520098111,3951525.7272217725,950007.8079603023))
(00000048,(1.1417697416123798E7,1.116381958819456E7,3806904.474665456))
(00000040,(117888.01196136928,114135.91588073106,26060.9621700251))
(00000041,(2.0098913841437105E7,1.9911157106606036E7,5430726.028105627))
(00000027,(8282588.1811231235,8298412.552938455,2058482.9265007493))
(00000006,(2279638.8694719016,2266016.8706103154,539508.1339093351))
...

-------------------------------------------
Time: 1701134991000 ms
-------------------------------------------
(00000000,(3.1261939832448818E7,3.1067823538724557E7,8101778.436823584))
(00000034,(2.3821388186907735E7,2.3813786245535273E7,5244876.154331164))
(00000035,(3785151.5816258346,3827769.0396342636,1398037.4528475797))
(00000013,(6171147.240043716,6084342.325839778,1456353.5635264334))
(00000014,(2.3006246071961988E7,2.2824884677063316E7,6114711.908438089))
(00000047,(4003558.520098111,3951525.7272217725,950007.8079603023))
(00000048,(2.6410799904533446E7,2.6212614038370155E7,6992579.596572913))
(00000040,(117888.01196136928,114135.91588073106,26060.9621700251))
(00000041,(3.2004957649994142E7,3.174802481446305E7,8366823.386023573))
(00000027,(1.3917101731795195E7,1.3832352217045585E7,3395805.1787956664))
...

-------------------------------------------
Time: 1701134994000 ms
-------------------------------------------
(00000008,(2.006513359076011E7,1.9820611130979206E7,5386113.64896667))
(00000000,(3.3928910489126205E7,3.371734893702074E7,8730985.676958889))
(00000022,(3493106.7133652866,3536359.494720799,1264662.2916245547))
```

8.3) Results of Dipole Moment:-

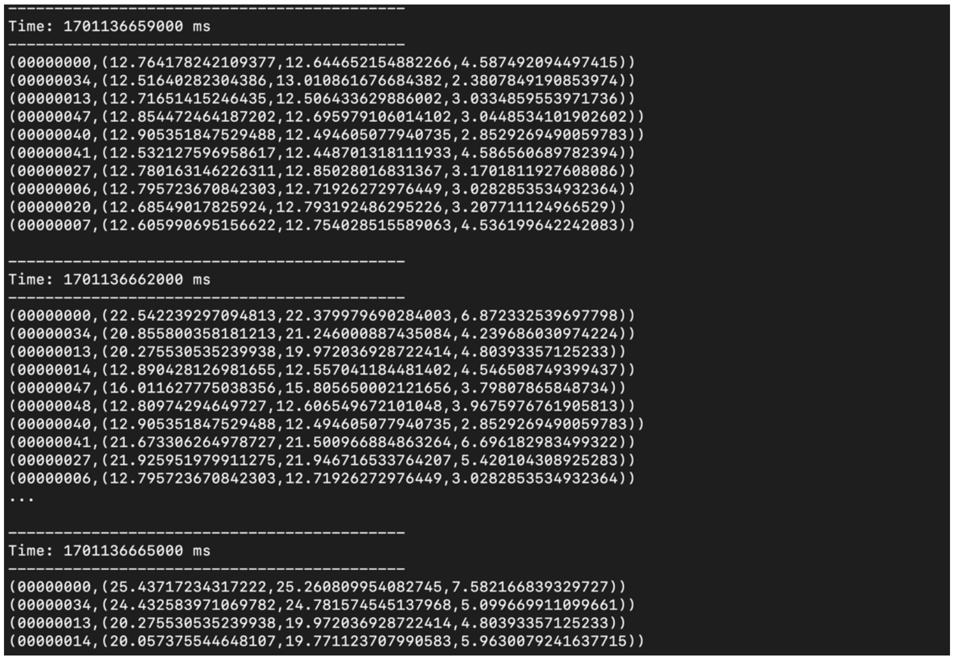


```
-------------------------------------------
Time: 1701135255000 ms
-------------------------------------------
(00000000,(-189.37530367599965,-115.81815671799502,10.303523166400499))
(00000034,(-36.107866393999174,-50.740661391998664,-68.11980600000005))
(00000013,(-6.472013556320858,1.7169948092183709,2.6728960982364907))
(00000047,(-0.6148846264888217,1.9584460992734467,3.5920938737944788))
(00000040,(1.1210133833598661,4.002961537120523,3.7395988640295865))
(00000041,(-188.91762273600582,-104.51385127800141,21.761781566899966))
(00000027,(-2.9946997962453983,-10.217184438496503,-5.225919626518298))
(00000006,(8.677136315731218,16.918005097442332,3.1959882047284163))
(00000020,(-4.114336778906042,-1.714301393240155,-5.074857269251728))

-------------------------------------------
Time: 1701135258000 ms
-------------------------------------------
(00000000,(-302.652895858133,-15.066950097996596,-22.760608520933122))
(00000034,(-27.877843817624672,-8.69432602984515,-167.8984646363082))
(00000013,(-4.639041670460191,20.41022558346275,13.743155949838824))
(00000014,(-230.295085138999,-44.85404406059464,87.75995512526286))
(00000047,(-0.38271666169930363,5.992624431212745,7.293505088208573))
(00000048,(-273.7235752589984,-27.2428462161887,113.77941546667414))
(00000040,(1.1210133833598661,4.002961537120523,3.7395988640295865))
(00000041,(-299.6406769594661,-32.793443547701884,-35.179532182377855))
(00000027,(-19.397777921149032,-9.945363829451036,-9.55123454169496))
(00000006,(14.495255570159248,21.244375068223142,5.224892018569621))
...

-------------------------------------------
Time: 1701135261000 ms
-------------------------------------------
(00000008,(-220.6469920999989,-85.33788520099999,54.871964150939846))
(00000000,(-300.6517105629216,-5.421261818265503,-26.087439644002586))
(00000034,(-25.68236956698888,-1.9980430959450572,-155.6844275600772))
(00000001,(-248.30177323719582,-29.78273115998431,100.65882025780226))
(00000035,(-303.4560231549983,-68.92823581437271,61.39679061000168))
```

8.4) Results of Center of Mass:-

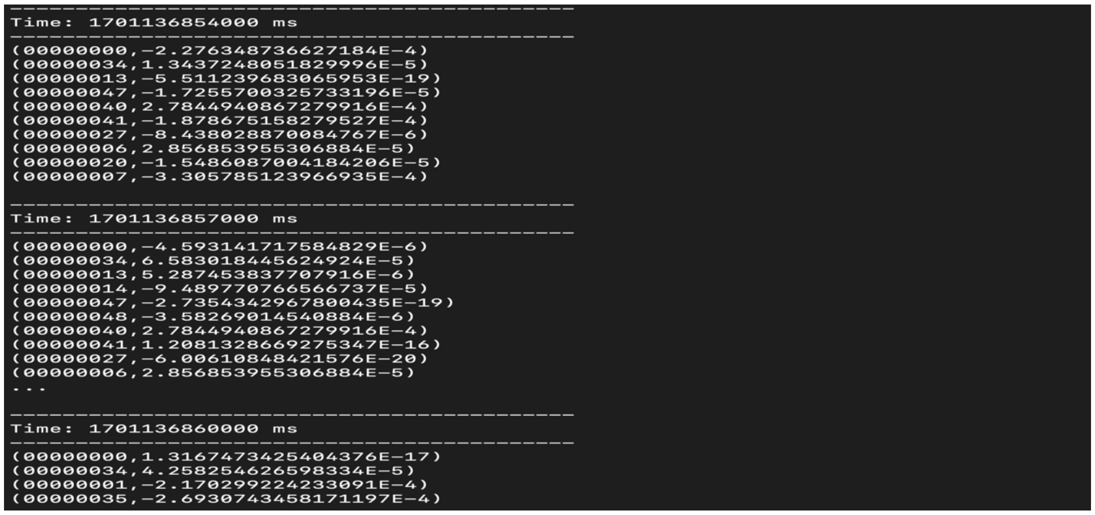


```
-----------------------------------------
Time: 1701136659000 ms
-----------------------------------------
(00000000,(12.764178242109377,12.644652154882266,4.587492094497415))
(00000034,(12.5164028230438 6,13.010861676684382,2.3807849190853974))
(00000013,(12.7165141524643 5,12.506433629886002,3.0334859553971736))
(00000047,(12.854472464187202,12.695979106014102,3.0448534101902602))
(00000040,(12.905351847529488,12.494605077940735,2.8529269490059783))
(00000041,(12.532127596958617,12.448701318111933,4.586560689782394))
(00000027,(12.780163146226311,12.85028016831367,3.1701811927608086))
(00000006,(12.795723670842303,12.7192627297644 9,3.0282853534932364))
(00000020,(12.68549017825924,12.793192486295226,3.207711124966529))
(00000007,(12.605990695156622,12.754028515589063,4.536199642242083))

-----------------------------------------
Time: 1701136662000 ms
-----------------------------------------
(00000000,(22.542239297094813,22.379979690284003,6.872332539697798))
(00000034,(20.855800358181213,21.246000887435084,4.239686030974224))
(00000013,(20.275530535239938,19.972036928722414,4.80393357125233))
(00000014,(12.890428126981655,12.557041184481402,4.546508749399437))
(00000047,(16.011627775038356,15.805650002121656,3.79807865848734))
(00000048,(12.8097429464972 7,12.606549672101048,3.9675976761905813))
(00000040,(12.905351847529488,12.494605077940735,2.8529269490059783))
(00000041,(21.673306264978727,21.500966884863264,6.696182983499322))
(00000027,(21.925951979911275,21.946716533764207,5.420104308925283))
(00000006,(12.795723670842303,12.7192627297644 9,3.0282853534932364))
...

-----------------------------------------
Time: 1701136665000 ms
-----------------------------------------
(00000000,(25.43717234317222,25.260809954082745,7.582166839329727))
(00000034,(24.432583971069782,24.781574545137968,5.099669911099661))
(00000013,(20.275530535239938,19.972036928722414,4.80393357125233))
(00000014,(20.057375544648107,19.771123707990583,5.963007924163771 5))
```

8.5) Results of Average Charge:-

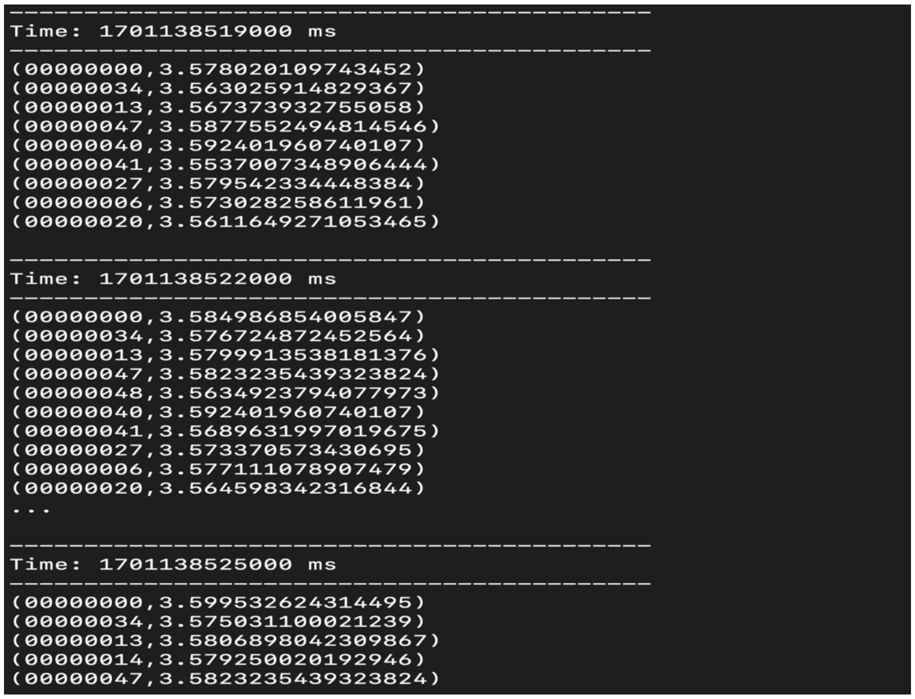


```
--------------------------------------------------
Time: 1701136854000 ms
--------------------------------------------------
(00000000,-2.276348736627184E-4)
(00000034,1.3437248051829996E-5)
(00000013,-5.511239683065953E-19)
(00000047,-1.7255700325733196E-5)
(00000040,2.7844940867279916E-4)
(00000041,-1.878675158279527E-4)
(00000027,-8.438028870084767E-6)
(00000006,2.856853955306884E-5)
(00000020,-1.5486087004184206E-5)
(00000007,-3.305785123966935E-4)

--------------------------------------------------
Time: 1701136857000 ms
--------------------------------------------------
(00000000,-4.593141717584829E-6)
(00000034,6.583018445624924E-5)
(00000013,5.287453837707916E-6)
(00000014,-9.489770766566737E-5)
(00000047,-2.7354342967800435E-19)
(00000048,-3.58269014540884E-6)
(00000040,2.7844940867279916E-4)
(00000041,1.2081328669275347E-16)
(00000027,-6.00610848421576E-20)
(00000006,2.856853955306884E-5)
...

--------------------------------------------------
Time: 1701136860000 ms
--------------------------------------------------
(00000000,1.3167473425404376E-17)
(00000034,4.258254626598334E-5)
(00000001,-2.170299224233091E-4)
(00000035,-2.6930743458171197E-4)
```

8.6) Results of Radius of Gyration along X-axis:-

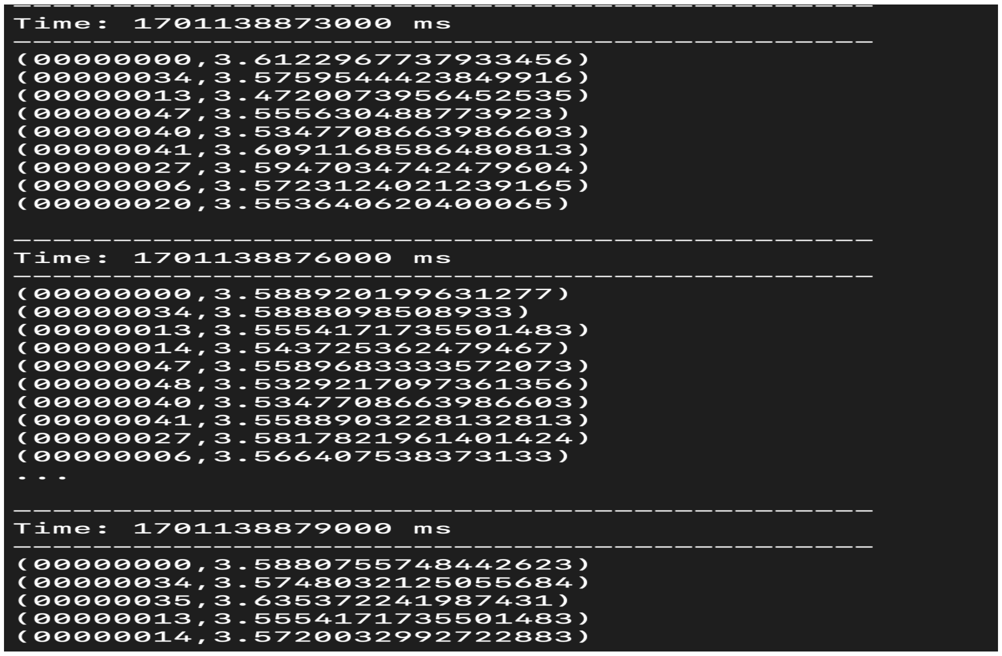


```
-------------------------------------------
Time: 1701138519000 ms
-------------------------------------------
(00000000,3.578020109743452)
(00000034,3.563025914829367)
(00000013,3.567373932755058)
(00000047,3.5877552494814546)
(00000040,3.592401960740107)
(00000041,3.5537007348906444)
(00000027,3.579542334448384)
(00000006,3.573028258611961)
(00000020,3.5611649271053465)

-------------------------------------------
Time: 1701138522000 ms
-------------------------------------------
(00000000,3.584986854005847)
(00000034,3.576724872452564)
(00000013,3.5799913538181376)
(00000047,3.5823235439323824)
(00000048,3.5634923794077973)
(00000040,3.592401960740107)
(00000041,3.5689631997019675)
(00000027,3.573370573430695)
(00000006,3.577111078907479)
(00000020,3.564598342316844)
...

-------------------------------------------
Time: 1701138525000 ms
-------------------------------------------
(00000000,3.599532624314495)
(00000034,3.575031100021239)
(00000013,3.5806898042309867)
(00000014,3.579250020192946)
(00000047,3.5823235439323824)
```

8.7) Results of Radius of Gyration along Y-axis:-

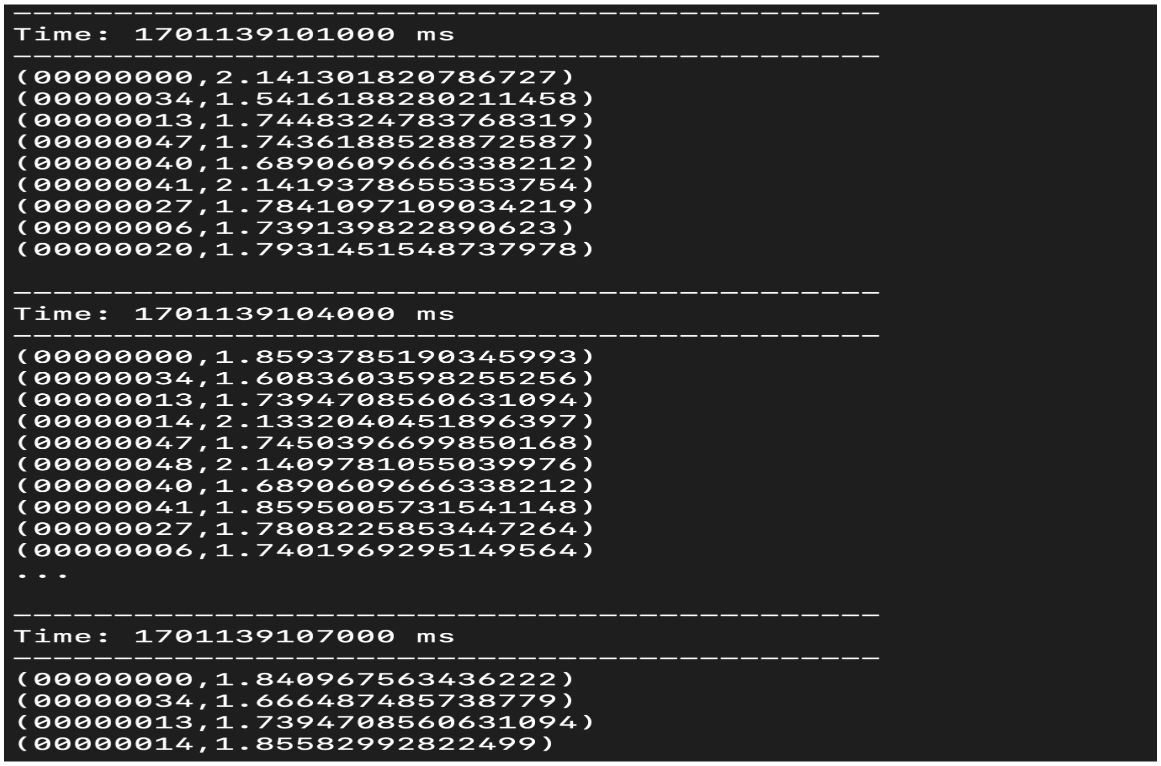


```
-------------------------------------------
Time: 1701138873000 ms
-------------------------------------------
(00000000,3.6122967737933456)
(00000034,3.5759544423849916)
(00000013,3.4720073956452535)
(00000047,3.555630488773923)
(00000040,3.5347708663986603)
(00000041,3.6091168586480813)
(00000027,3.5947034742479604)
(00000006,3.5723124021239165)
(00000020,3.553640620400065)

-------------------------------------------
Time: 1701138876000 ms
-------------------------------------------
(00000000,3.588920199631277)
(00000034,3.5888098508933)
(00000013,3.5554171735501483)
(00000014,3.543725362479467)
(00000047,3.5589683333572073)
(00000048,3.5329217097361356)
(00000040,3.5347708663986603)
(00000041,3.5588903228132813)
(00000027,3.5817821961401424)
(00000006,3.566407538373133)
...

-------------------------------------------
Time: 1701138879000 ms
-------------------------------------------
(00000000,3.5880755748442623)
(00000034,3.5748032125055684)
(00000035,3.635372241987431)
(00000013,3.5554171735501483)
(00000014,3.5720032992722883)
```

8.8) Radius of Gyration along Z-axis:-



```
-------------------------------------------
Time: 1701139101000 ms
-------------------------------------------
(00000000,2.141301820786727)
(00000034,1.5416188280211458)
(00000013,1.7448324783768319)
(00000047,1.7436188528872587)
(00000040,1.6890609666338212)
(00000041,2.1419378655353754)
(00000027,1.7841097109034219)
(00000006,1.739139822890623)
(00000020,1.7931451548737978)

-------------------------------------------
Time: 1701139104000 ms
-------------------------------------------
(00000000,1.8593785190345993)
(00000034,1.6083603598255256)
(00000013,1.7394708560631094)
(00000014,2.1332040451896397)
(00000047,1.7450396699850168)
(00000048,2.1409781055039976)
(00000040,1.6890609666338212)
(00000041,1.8595005731541148)
(00000027,1.7808225853447264)
(00000006,1.7401969295149564)
...

-------------------------------------------
Time: 1701139107000 ms
-------------------------------------------
(00000000,1.840967563436222)
(00000034,1.666487485738779)
(00000013,1.7394708560631094)
(00000014,1.85582992822499)
```

9. Conclusion:

The Spark Streaming pipeline, coupled with Apache Kafka, offers a powerful solution for real-time molecular data analysis. The integration of Kafka ensures efficient data ingestion, fault tolerance, and scalability, while Spark Streaming enables dynamic analyses through various scientific algorithms.

10. Future Work:

Moving forward, there is more scope for enhancing our Spark Streaming pipeline for molecular data analysis. First and foremost, optimizing the implementation of existing scientific algorithms could lead to even faster and more accurate results. Additionally, exploring new scientific algorithms that capture different aspects of molecular behavior could broaden the analytical capabilities of our system. This expansion would open doors to get best insights and patterns in molecular data, providing a more comprehensive understanding of complex structures.

Furthermore, there is room for extending support to additional data sources beyond Kafka, allowing our pipeline to accommodate diverse data streams. This adaptability would make our system more versatile and applicable to a broader range of molecular research scenarios.

Additionally, efforts can be directed towards refining the user interface and interaction mechanisms.

11. References: -

https://digitalcommons.usf.edu/cgi/viewcontent.cgi?article=7468&context=etd

https://medium.com/towardsdataanalytics/spark-streaming-vs-structured-streaming-ef6863d5b60

https://spark.apache.org/docs/latest/streaming-programming-guide.html