Leveraging the Power of Big Data Tools for Large Scale Molecular Dynamics Analysis

Omar A. Mures, Emilio J. Padrón, Bruno Raffin

Leveraging the Power of Big Data Tools for Large Scale Molecular Dynamics Analysis

Omar A. Mures[1], Emilio J. Padrón[2] and Bruno Raffin[3]

Abstract— **Parallel Molecular Dynamics simulations are generating atom trajectories of growing sizes and complexity. Analyzing these trajectories is expensive computationally and time consuming. One reason is the lack of tools that enable the computational biologist to easily implement the analysis while ensuring reduced processing times exploiting the benefits of parallel architectures. In this paper, we present a comparison between two parallel analytics frameworks based on the Map/Reduce paradigm: HiMach, a dedicated framework for trajectory analysis based on MPI, and Flink, a Big Data analytics framework. Both frameworks enable to hide the complexity of parallel code creation to the programmer, providing significant performance gains compared to a sequential execution.**

Keywords— **Molecular Dynamics, Big Data, MapReduce, Flink, High Performance Data Analysis, Parallelization.**

I. Introduction

RECENTLY in the HPC vendor and science community a new tendency is on the rise, since both fields face similar issues, these two worlds need to come together in order to solve bigger and more complex science problems. As a consequence, base technologies themselves are becoming more closely related. Nowadays scientists are taking advantage more than ever of the evolving computing resources available to them. From their local workstations to high performance clusters in large data centers, they rely on these tools to discover new phenomena in their respective fields.

To accomplish the aforementioned tasks, there is an inherent need to simplify the creation of analysis code that can exploit these heterogeneous resources. Big Data analytics tools based on the MapReduce paradigm [1], such as Hadoop [2], enable to take advantage of commodity compute clusters for analysing very large data sets generated from the web or business applications. But when trying to analyse complex scientific datasets they can fall short from a programmers perspective and performance wise.

Molecular Dynamics (MD) is a computer simulation technique for studying physical interactions of atoms and molecules [3]. It is a type of N-body simulation [4], which simulates a dynamic system of particles, typically under the influence of physical forces, such as gravity. Atoms and molecules interact during a preset period of time, allowing the scientist to observe the dynamical evolution of the system. The most common approach determines the trajectories of atoms and molecules by solving Newton's equations of motion for a system of interacting particles. The forces and energies of said particles are computed using inter-atomic potentials or molecular mechanics force fields. This method is applied today in chemical physics, material science and the modeling of biomolecules.

Although plenty of progress has been made in the creation of efficient parallel code for Molecular Dynamics simulations, code for analysis of the results that they produce in parallel is scarce. As a consequence, scientist have to rely on serial tools to analyze the aforementioned data. In this paper we focus on the analysis of atom trajectories generated from Molecular Dynamics simulations. These datasets can be very large and time consuming to analyze, sometimes reaching thousands of gigabytes in size. The size of trajectories is expected to keep on growing further as exascale computers [5] become available, aggravating this need for parallel analysis code. Current serial approaches force the scientist to filter trajectories sometimes randomly in order to find the phenomena of interest, this is far from a perfect approach, and if not resolved, could hurt the scientists' opportunities to perform new discoveries and rendering the purpose of creating more precise simulations useless.

Existing analysis libraries such as MDAnalysis [6] or VMD [7] rely on traditional parallelization approaches based on MPI and/or OpenMP that are often complex to deploy at large scale and use efficiently for computational biologists. In this paper, we investigate how the Map/Reduce paradigm could be leveraged to enable the analysis of large trajectories. We have performed a comparison between two parallel analytics frameworks based on the Map/Reduce paradigm: HiMach, a framework for trajectory analysis based on MPI and VMD, and Flink, a new generation Big Data analytics engine. Both frameworks hide the complexity of parallel code creation to the user, providing significant performance gains compared to sequential executions.

A brief background overview of the subject is given in section II, the proposed solution is explained in section III and the results obtained are presented in section IV.

II. Background

MapReduce is a programming model for processing and generating large data sets in parallel, on commodity clusters. It relies on a simple programming model that uses two main functions, Map and

[1]Computer Architecture Group, Universidade da Coruña, e-mail: omar.alvarez@udc.es

[2]Computer Architecture Group, Universidade da Coruña, e-mail: emilioj@udc.es

[3]University Grenoble Alpes, INRIA, e-mail: bruno.raffin@inria.fr

Reduce. They are inspired by the map and reduce functions that were commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms.

These two functions (see Figure 1) have to be defined by the programmer in order to obtain parallelism for data-intensive tasks. The Map function consumes key-value pairs and produces one or more intermediate pairs:

$$Map(k1, v1) \rightarrow list(k2, v2)$$

The Reduce function, processes a certain key and a list of values associated to it, usually outputting a list with less values than the input list:

$$Reduce(k2, list(v2)) \rightarrow list(v3)$$

For example, a typical MapReduce workload that counts words in a text, will Map (extract) the words in all lines of the text emitting $(k2, v2)$, where $k2$ is the word and $v2$ is 1. This will allow to Reduce (summarize) the results, counting the number of occurrences of each word. In order to do this, each reducer uses the input $(k2, list(v2))$; where $k2$ is the word and $list(v2)$ is the list of values associated to $k2$ (all ones), to sum up the values, obtaining the aforementioned count.

The MapReduce system performs the processing by marshaling the distributed servers, running the required tasks in parallel, managing all communications and data movement between the various parts of the system, while providing redundancy and fault tolerance. As such, a in a single-threaded environment, MapReduce will usually not be faster than a traditional implementation; any performance gains are usually only seen in multi-threaded scenarios.

From a performance standpoint, MapReduce has several limitations. First, as we have seen in Figure 1, after each Map phase, results are written to disk as Map Output Files (MOFs) even if they can fit in main memory, and will be later read by the Reduce phase in order to compute the final results. This is good from a fault tolerance point of view, but when trying to achieve high performance it presents a clear problem, as avoiding disk writes and reads could increase performance substantially. Furthermore, until every map task has finished, the reduce phase cannot start; even if the required data has already been obtained. Second, since the paradigm is quite simple, it presents almost no opportunities for the framework to perform automatic optimizations. While a experimented MapReduce programmer can use these two abstractions to model complicated algorithms, it is clear that this paradigm can fall short in some complex scenarios where more flexibility is required, apart from the fact that there is not enough semantic information in this two constructs in order to infer properties about the user defined functions.

Several efforts have been made to mitigate these problems, for instance the parallel trajectory analysis framework HiMach [8], is a MapReduce like system

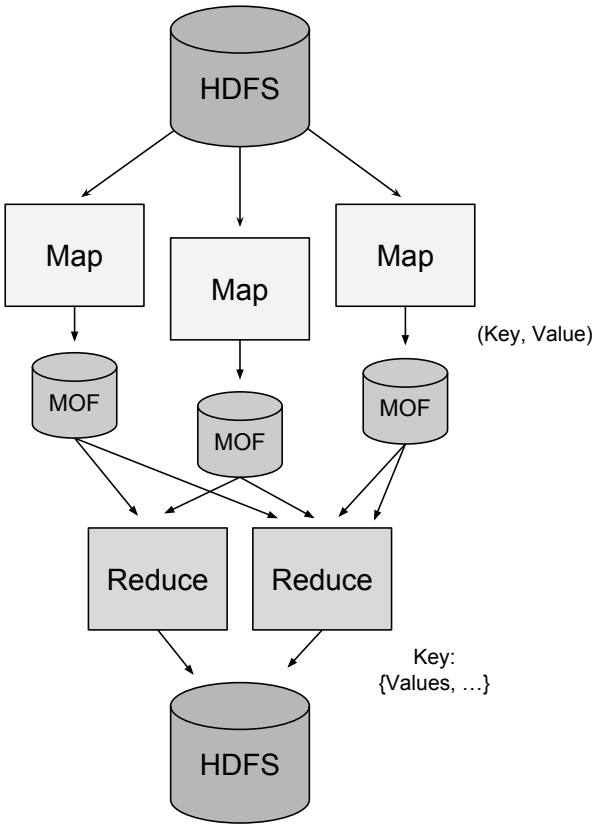

Fig. 1
Typical MapReduce job workflow.

where the user writes sequential trajectory analysis code and the system carries out the parallel analysis of the trajectory automatically. This solution avoids the use of MOFs, as well as, using high performance MPI code that can take advantage of InfiniBand interconnects, improving performance substantially. Although this framework solves some of the issues that have been presented earlier, the parallelization paradigm offered still suffers from the same issues as MapReduce.

Since for complex analysis the MapReduce method is not optimal, a more advanced paradigm that relies on Directed Acyclic Graphs (DAGs) [9] has been chosen in order to allow the user to express more complex analytics tasks. This new approach uses Parallelization Contracts (PACTs) [10], which can be thought of as a generalization of MapReduce in order to ease the expression of many operations, which in turn leads to better expressiveness and enables optimizations to be inferred automatically. The chosen framework is the result of the Stratosphere project [11], an open source project from the Apache Foundation called Flink [12]. To our Knowledge Stratosphere/Flink has never been used for analyzing trajectories.

The usage of such framework will allow computational biologists to more efficiently analyze massive trajectories, that until now had to be simplified, which is not ideal. With the presented framework, a scientist will be able to leverage in a simple way

the power of new generation MapReduce tools, letting them worry about what analysis they want to perform instead of how to parallelize or deal with simplifying the data so it is usable.

III. MDReduce Framework

Since our main objective is easing the creation of parallel code by the scientist, initially we have chosen the Python [13] language for its ease of use and plethora of external libraries. Python has been amply utilized by the scientific community, with libraries for efficient numerical analysis like Numpy [14]. This library is a key component of MDAnalysis, one of the tools that allows scientists to efficiently create parallel Molecular Dynamics analysis code using our system. This are the key technologies that were used in order to implement the MDReduce framework in conjunction with Flink and HiMach.

A. Architecture

Once the reader has been acquainted with the general concepts that have been used in the implementation of MDReduce, it is interesting to take a look at Figure 2, that presents a more accurate picture of the system. In the current MDReduce implementation, the **Core** module contains all the classes related to the system internals. First, the **Configuration** class takes care of parsing either a configuration file (it uses the YAML syntax to store several settings) or the arguments that the user passes to the class constructor. It will take care of initializing the requested back-end (Flink or HiMach), and other trivial setup tasks. Directly related to this class are the available **Backends**. These classes are in charge of dealing with Flink and HiMach internals, these are helper classes for initialing these frameworks and running the requested jobs on them.

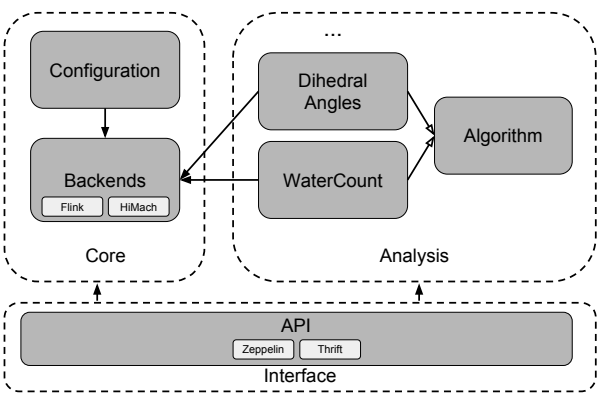

Fig. 2
MDReduce system architecture.

The main supported operations that the user can employ to analyze trajectories in parallel are the following:

In addition, the **Analysis** module contains all the classes related to the Molecular Dynamics analysis methods included in the MDReduce framework.

TABLE I
Supported operations. *Note: The HiMach backend only supports the Map and Reduce operations.*

Operation	Description
Map	Takes one element and produces zero, one, or more elements.
Reduce	Combines a group of elements into a single element.
Filter	Evaluates a function for each element and retains those that fulfill it.
Aggregate	Aggregates a group of values into a single value. Minimum, maximum, and sums.
Join	Joins two data sets by creating all pairs of elements that are equal on their keys.
Union	Produces the union of two data sets.
Sort	Sorts dataset depending on a certain field.

First, an **Algorithm** base class has been created, so that developing new analysis scripts is more intuitive for programmers. Second, several analysis classes have been included to perform the analyses that will be detailed in subsection III-B. These two classes, **WaterCount** and **DihedralAngles** offer an insight for programming analysis codes using the MDReduce platform.

Finally, through the MDReduce API, external programs that use the existing algorithms or perform new types of analyses that can be created by scientists and run on the platform. In order to ease the integration of the framework with external programs, output can be consumed by and external client, or stored in data files. These files can then be analyzed in local workstations, since the amount data should have been substantially reduced, and should now be usable on lesser machines. As can be seen in Figure 2, Zeppelin [15] a new framework that enables interactive data analytics has also been integrated with the system. This allows scientists to submit jobs interactively, and dynamically explore the resulting data. This is a convenient tool to ease the creation and representation of different analyses, since it allows the user to interactively modify code and visualize analysis results.

B. Case Study

To test the feasibility of the proposed system for Molecular Dynamics analyses, we choose two typical tasks for a computational biologist. First, how to obtain a water count in a GLIC channel [16], [17] using MDReduce. Second, another common analysis performed in trajectories consists in computing the dihedral angles [18] in the protein backbone.

B.1 Water Count in a GLIC Channel

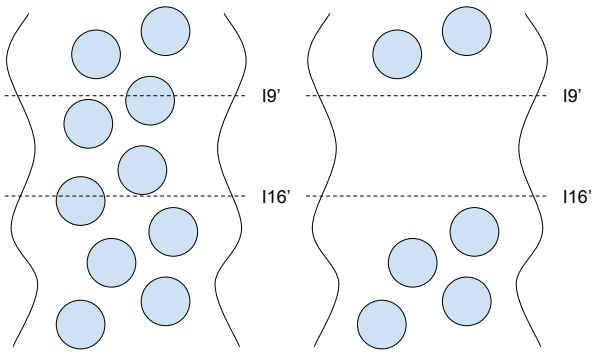

Fig. 3
Graphical representation of the water counting operation in an ion channel.

```
class MapCount(MapFunction):
  def map(self, value):
    u.trajectory[int(value[0])]
    alignto(u, uRef, select="protein and
    backbone and resid 222:233")
    selectWater = u.select_atoms(water)
    if (selectWater.n_atoms > 0) :
      cylindre = selectWater[numpy.where(
    selectWater.coordinates()[:,0]**2  +
    selectWater.coordinates()[:,1]**2 < 100)
    [0]]
      nbWater = cylindre.n_atoms
    else:
      nbWater = 0
    return (value[0],nbWater)
```

Fig. 4
Water count code example in Flink.

As seen in Figure 3, the objective of this analysis is obtaining the amount of water ions that are comprised between certain residues in the ion channel. The final objective is obtaining a water ion count for every frame in the trajectory, since the user will vary the simulation parameters in order to isolate the causes of the closing and opening of the GLIC ion channel. This structure has both open and closed states, the transition mechanism between different states is unknown for now. Understanding this process is crucial, as empowering molecules such as anesthetics and alcohols are believed to affect transition barriers or the relative free energy of states.

To investigate these phenomena, extensive simulations of the GLIC channel with different starting conditions and parameters have been performed. This allows computational biologists to draw conclusions about the pore hydration, organization and the pore lining helix. To characterize the structure of this helix, the first part is performing this analysis. The number of water molecules in the pore zone between the two hydrophobic residues I9 and I16 are counted. After this the scientist can infer which region has been more affected by pore dewetting.

This type of analysis can be easily characterized in MDReduce using either the Flink or HiMach back-ends. Since this is an operation that needs to be computed for every frame, this translates easily to the MapReduce paradigm. One just needs to write a mapper that will perform the water count operation for every frame. The Figure 4 represents how this analysis can be performed using the Flink back-end.

The user has to override Flink's Map function, with the operations that will be applied to the trajectory. First, the frame number corresponding to the instance of mapper is selected, then it is processed with MDAnalysis in order to align it to a reference frame and obtain the water count in the ion channel using atom selections. As can be seen, this code is completely straightforward requiring slight modifications to be run in parallel.

B.2 Dihedral Angles

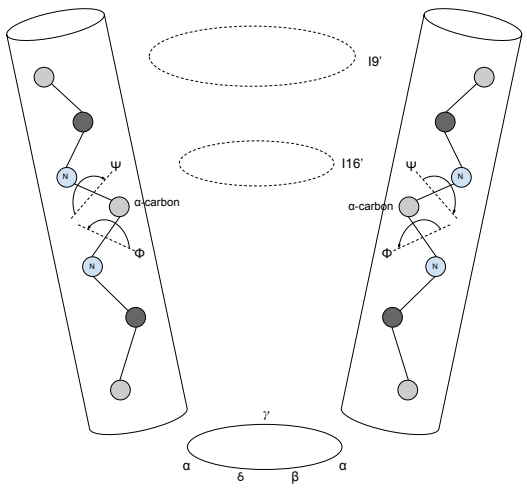

Fig. 5
Graphical representation of the dihedral angle calculation operation in the protein.

As seen in Figure 5, the objective of this analysis is obtaining a way to visualize energetically allowed regions for the backbone dihedral angles of amino acid residues in the protein structure. As mentioned earlier, understanding the reasons why the transition between the open and closed states of the ion channel is crucial for several topics. This second analysis code is the missing link to study this phenomena that depends on the pore organization and lining of the M2 helix. TO help in the description of the state of the pore, the tilt angles are computed and split in two angles: the radial and lateral tilt angles of the M2 helix around the center of mass in respect to the channel axis.

The second analysis, can be performed in MDReduce using either both available back-ends. Since this is an operation that again, needs to be computed for every frame, it is easily adapted to the MapReduce paradigm. The user just needs to write a mapper that will perform the required operations for every frame. The Figure 6 represents how this

```
class MapDihedral(Mapper):
    def __init__(self, u, uRef):
        self.u = u
        self.uRef = uRef

    def map(self, step):
        cursor = step[0]
        frameno, framebuf = cursor
        assert framebuf == None

        self.u.trajectory[frameno]

        alignto(self.u, self.uRef, select="
        protein and backbone and resid 222:233")

        phi_A, psi_A = angle_between_princ_axes(
        TM2_A, alpha_carbones)
        phi_B, psi_B = angle_between_princ_axes(
        TM2_B, alpha_carbones)
        phi_C, psi_C = angle_between_princ_axes(
        TM2_C, alpha_carbones)
        phi_D, psi_D = angle_between_princ_axes(
        TM2_D, alpha_carbones)
        phi_E, psi_E = angle_between_princ_axes(
        TM2_E, alpha_carbones)

        yield(frameno, [phi_A, psi_A, phi_B,
        psi_B, phi_C, psi_C, phi_D, psi_D, phi_E
        , psi_E])

        return
```

Fig. 6

Dihedral angle code example in HiMach.

analysis can be performed using the HiMach back-end.

IV. Results

To test the proposed system we have used the Pluton cluster, a supercomputing facility situated in the Universidade da Coruña. In the Table II, the hardware that the cluster possesses can be seen.

TABLE II

Hardware used for testing.

Hardware	compute-0-16
CPU Model	2 x Intel Xeon E5-2660
Cores/Threads	16/32
CPU Speed/Turbo	2.20 GHz/3 GHz
Memory	64 GB DDR3 1600 Mhz
Disk	1 x HDD 1 TB SATA3 7.2K
Networks	InfiniBand FDR

Add performance number for sequential execution (pure MDAnalyis and/or VMD code) The analysis algorithms explained in subsection III-B, have been used to run the performance tests carried out in this section. These tests have tested several different strategies, that access data with multiple I/O patterns, in order to discern the best I/O access pattern. The tested patterns are:

- **Broadcast**: This strategy utilizes a single file that contains the trajectory, that is then distributed to every map task and accessed in parallel.
- **Split**: The trajectory in this strategy is split in one file per frame, which is then accessed independently by every map task in parallel.

All strategies were then executed using both back-ends available in MDReduce to see which of them fares better for each type of analysis.

Figure 7 shows the job execution times of the water count analysis.

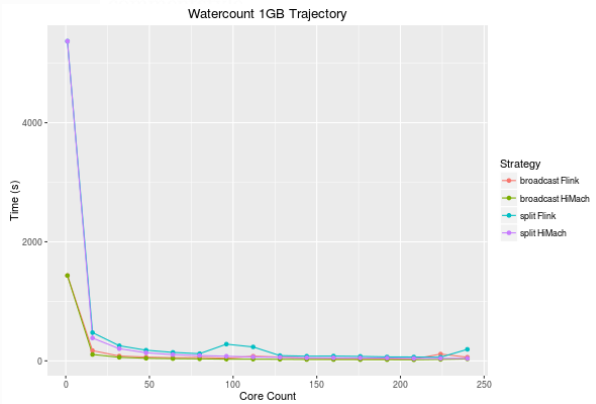

Fig. 7

Run time with each back-end of the water count job.

Legend should be move elsewhere (inside the graph or above it) to have bigger graphs

It is kind of difficult to see which approach fares better, since with every approach the execution time as we use more than one processor descends greatly. The fastest time for this workload, was achieved by the HiMach back-end with the broadcast strategy. This is due to the fact that the HiMach runtime uses MPI and the Infiniband capabilities of our cluster in an optimal way.

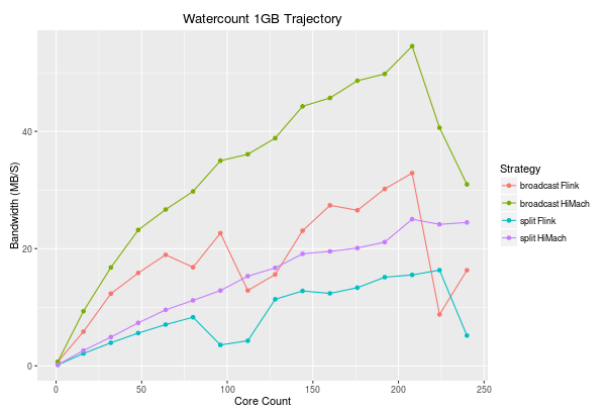

Fig. 8

Bandwidth with each back-end of the water count job.

To further see what is happening in this analysis code in Figure 8, we can see the bandwidth achieved by all the strategies. Here the reader can see better which back-end is the fastest. This results mimic the ones obtained before, but paint a clearer picture

about the performance that each strategy achieves. As was mentioned earlier, the best performance is achieved by HiMach followed closely by Flink. The first thing that one can deduce examining this graph, is that clearly the split access pattern is less efficient than the broadcast pattern. This was predicted, since reading through small files generally causes lots of seeks and stresses the underlying file system, which results in an inefficient data access pattern.

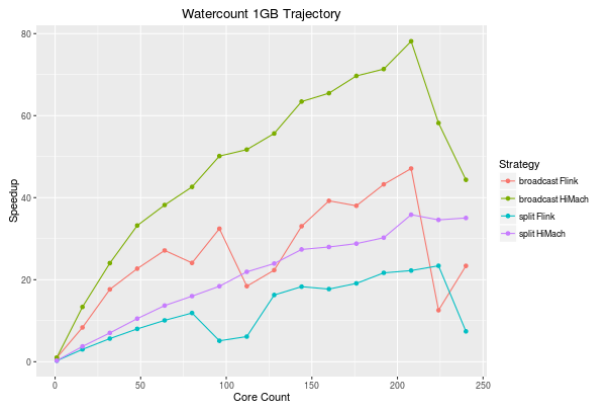

Fig. 9

Speedup with each back-end of the water count job.

In addition, as to ascertain the efficiency of the tested frameworks, in Figure 9 we can see the speedups of all the different code versions. Here we can observe other interesting behavior. Both frameworks scale well but as the number of processors is increased a couple of discrepancies can be observed, and the scaling weakens. In addition, we can see that with the split strategy both back-ends have close speedups. The performance difference is more significant for the broadcast access pattern, probably due to a higher impact of the network performance that HiMachbetter use than Flink.

V. Future Work

Since several inefficiencies in the current implementation are believed to be related to the usage of the Python interpreter from the Java Flink processes, it could be interesting to test the ZipPy [19] python implementation in our system. This Python 3 implementation uses Truffle [20] to generate optimized JVM bytecode from Python scripts. This could certainly improve the efficiency of the analysis performed using the MDReduce framework.

In addition, it could also be interesting to extend our study to include a Spark [21] back-end, today probably the most popular Big Dta analytics framework.

Moreover, to solve one of the main issues with the split data access pattern, other storage solutions like HBase [22] could be explored. This column oriented data store has the potential to avoid the bottleneck that the plethora of small files in a long trajectory could present to the MDReduce framework.

VI. Conclusions

Using MDReduce has proven to improve not only processing times, since it allows users to write almost sequential code that can then be exploited for parallel analysis, but also in terms of programming efficiency. It is clear that using the introduced system, is minimally more complex than writing sequential analysis code for straightforward tasks.

The usage of Flink, can yield performance advantages due to incorporated job optimizer, but also makes programming more complex analysis algorithms possible. It also allow the user to take advantage of the Hadoop ecosystem, for instance allowing the users to visualize data dynamically using Zeppelin.

Our initial performance tests have shown that HiMach, due to its simplicity, is the best option for running straightforward analysis tasks. However the analysis we have tested are simple map/reduce schemes. They did not allow us to probe the capabilities of Flink designed to support more complexe parallelization schemes.

References

[1] Jeffrey Dean and Sanjay Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] "Hadoop," `http://hadoop.apache.org/`, Accessed: 2016-05-01.

[3] Berni J Alder and TE Wainwright, "Studies in molecular dynamics. i. general method," *The Journal of Chemical Physics*, vol. 31, no. 2, pp. 459–466, 1959.

[4] John H Reif and Stephen R Tate, "The complexity of n-body simulation," in *Automata, Languages and Programming*, pp. 162–176. Springer, 1993.

[5] Timothy Germann, "Exascale computing and what it means for shock physics," in *APS Shock Compression of Condensed Matter Meeting Abstracts*, 2015, vol. 1, p. 1002.

[6] Naveen Michaud-Agrawal, Elizabeth J. Denning, Thomas B. Woolf, and Oliver Beckstein, "Mdanalysis: A toolkit for the analysis of molecular dynamics simulations," *Journal of Computational Chemistry*, vol. 32, no. 10, pp. 2319–2327, 2011.

[7] William Humphrey, Andrew Dalke, and Klaus Schulten, "Vmd: visual molecular dynamics," *Journal of molecular graphics*, vol. 14, no. 1, pp. 33–38, 1996.

[8] Tiankai Tu, Charles A Rendleman, David W Borhani, Ron O Dror, Justin Gullingsrud, Morten Ø Jensen, John L Klepeis, Paul Maragakis, Patrick Miller, Kate A Stafford, et al., "A scalable parallel framework for analyzing terascale molecular dynamics simulation trajectories," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*. IEEE, 2008, pp. 1–12.

[9] K. Thulasiraman and M.N.S. Swamy, *Graphs: Theory and Algorithms*, Wiley, 2011.

[10] Alexander Alexandrov, Stephan Ewen, Max Heimel, Fabian Hueske, Odej Kao, Volker Markl, Erik Nijkamp, and Daniel Warneke, "Mapreduce and pact-comparing data parallel programming models.," in *BTW*, 2011, pp. 25–44.

[11] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, et al., "The stratosphere platform for big data analytics," *The VLDB Journal*, vol. 23, no. 6, pp. 939–964, 2014.

[12] "Flink," `https://flink.apache.org/`, Accessed: 2016-05-01.

[13] "Python," `https://www.python.org/`, Accessed: 2016-05-01.

[14] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux, "The numpy array: a structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[15] "Zeppelin," `https://zeppelin.incubator.apache.org/`, Accessed: 2016-05-01.

[16] Prafulla Aryal, Mark SP Sansom, and Stephen J Tucker, "Hydrophobic gating in ion channels," *Journal of molecular biology*, vol. 427, no. 1, pp. 121–130, 2015.

[17] Pierre-Jean Corringer, Marc Baaden, Nicolas Bocquet, Marc Delarue, Virginie Dufresne, Hugues Nury, Marie Prevost, and Catherine Van Renterghem, "Atomic structure and dynamics of pentameric ligand-gated ion channels: new insight from bacterial homologues," *The Journal of physiology*, vol. 588, no. 4, pp. 565–572, 2010.

[18] Ricarda JC Hilf and Raimund Dutzler, "A prokaryotic perspective on pentameric ligand-gated ion channel structure," *Current opinion in structural biology*, vol. 19, no. 4, pp. 418–424, 2009.

[19] Wei Zhang, Per Larsen, Stefan Brunthaler, and Michael Franz, "Accelerating iterators in optimizing ast interpreters," *SIGPLAN Not.*, vol. 49, no. 10, pp. 727–743, Oct. 2014.

[20] Thomas Würthinger, Christian Wimmer, Andreas Wöß, Lukas Stadler, Gilles Duboscq, Christian Humer, Gregor Richards, Doug Simon, and Mario Wolczko, "One vm to rule them all," in *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*. ACM, 2013, pp. 187–204.

[21] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy Mccauley, M Franklin, Scott Shenker, and Ion Stoica, "Fast and interactive analytics over hadoop data with spark," *USENIX; login*, vol. 37, no. 4, pp. 45–51, 2012.

[22] Mehul Nalin Vora, "Hadoop-hbase for large-scale data," in *Computer science and network technology (ICCSNT), 2011 international conference on*. IEEE, 2011, vol. 1, pp. 601–605.