Big Data over Cloud: Enabling Drug Design Under Cellular Environment

B. S. Sanjeev(✉) and Dheeraj Chitara

Department of Applied Sciences, Indian Institute of Information Technology - Allahabad, Prayagraj 211012, UP, India
sanjeev@iiita.ac.in

Abstract. Molecular Dynamics (MD) simulations mimic the motion of atoms. While simulations often require weeks to complete, the terabytes of data generated in the process also present a challenge for analysis. Recent studies discovered drug-receptor interactions with the cellular environment. Comprising of hundreds of small and large molecules seen in cells, such complex studies are necessary for improving drug design. Current resources and techniques do not provide, in reasonable time, necessary insights into such systems. In this study, we developed an algorithm to identify molecules interacting with drug-receptor complexes. Using first ever application of big data framework to MD studies, we demonstrate that our approach enables rapid analysis of MD data. Finally, we propose a cloud-based, Spark-enabled, self-tuning, scalable and responsive framework to accomplish optimal MD studies for drug-development within available resources.

Keywords: Big data · Cloud infrastructure · Molecular dynamics simulations · Drug design · Spark

1 Introduction

Molecular Dynamics (MD) simulation is a powerful technique to study molecules in atomistic detail. It models the movement of atoms, ions and molecules subject to forces acting on them. Similar to a video capturing the motion with a series of images, *evolution* of a molecular system too is captured as a sequence of *frames*. A frame holds the necessary details of the system corresponding to the particular time it represents. Analysis of the frames provides insights into the dynamics of the system itself.

Standard software (e.g., AMBER [18]) are available to carry out MD simulations on biomolecules. Such studies have been used to examine a variety of phenomena such as protein folding [21], stability [14] and intermolecular interactions [3]. In particular, they are of great value in drug design and drug discovery, providing vital inputs on ligand docking, virtual screening, dynamics of drug-bound protein/receptor complexes, allosteric modulation and role of water molecules in drug binding [4,17]. However, the *length* of a simulation (in terms of time) needs to be adequate for meaningful interpretation of data. Smaller systems generally

S. N. Srirama et al. (Eds.): BDA 2021, LNCS 13147, pp. 279–295, 2021.
https://doi.org/10.1007/978-3-030-93620-4_20

attain equilibrium faster and larger systems require longer simulations. This places additional hurdles on simulating large systems, and often results in runtimes ranging from days to weeks. Unsurprisingly, longer simulations of larger systems produce data running into terabytes.

Development of a new drug costs about a billion dollars [22]. While designing a drug *in silico* in itself is not an expensive task, the final hurdle that besets the acceptance of any drug involves clinical trials. The environment around the drug-receptor complex inside human body is very different from the simple theoretical models used for the drug design. Large number of detailed studies have consistently highlighted the fact that cellular environment could drastically alter the properties of biomolecules [9,15], and argued for the development of more realistic models in drug design [6,7]. A recent study simulated drug-bound enzyme in presence of hundreds of molecules seen in cells [19]. The analysis of the data required newer approach and is discussed here. Given the size of computations and the volume of data, applying big data framework (through Apache Spark) was a natural choice as it enables simultaneous loading of data from multiple frames along with concurrent computations. However, none of the current tools, including CPPTRAJ [16] and MDTraj [11], use big data frameworks (e.g., Spark or MapReduce) for the analysis of data from MD simulations.

Given the motivation to study complex MD simulations, the primary goal was to provide a scalable framework that could process voluminous data while concurrently performing computationally intensive tasks. The other goal was to design a more capable frame-work for drug design over Cloud infrastructure that could mix scalability with given priorities and resources.

In this paper, we demonstrate the necessity for a new framework in the study of MD simulations. Section 2 introduces the chosen simulated systems and the parameters for study. Section 3 demonstrates that big data approach facilitates rapid analysis of MD data. Based on this advance, we present a Spark-based approach to Cloud that provides a self-tuning, flexible and scalable framework to deliver speedy insights of immense value in drug design. Section 4 concludes the paper.

2 Materials and Methods

Proteins are polymers made of 20 types of units called amino acids. To perform biologically meaningful activities, they take different shapes in 3-D space. Certain calculations on proteins use only C_α atoms (see Fig. 1) to represent the corresponding amino acids. Often two or more polymers of amino acids combine to form a single functional protein.

Two systems based on a protein called Main protease (M^{pro}) of SARS-CoV-2 virus were covered for the current study. The first system (called M^{pro}-free hereafter) consisted of M^{pro} along with 10 different types metabolites present in the cell, potassium ions and water molecules. The second system (called M^{pro}-drug hereafter), had 3 different drugs (viz., Elbasvir, Glecaprevir and Ritonavir [19]) bound to M^{pro} and *crowder* proteins to emulate large molecules seen in

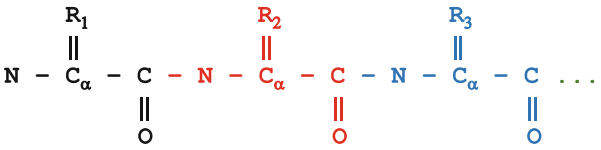

Fig. 1. Protein is a polymer made of 20 types of amino acids. Chemically, they differ at C_α atoms shown as R_is. Hydrogen atoms are not shown.

cellular environment, in addition to the above molecules. For reasons beyond the scope of this work, 5 units of these drugs were *docked* (i.e., bound) to M^{pro}. A schematic diagram of M^{pro}-drug is shown in Fig. 2.

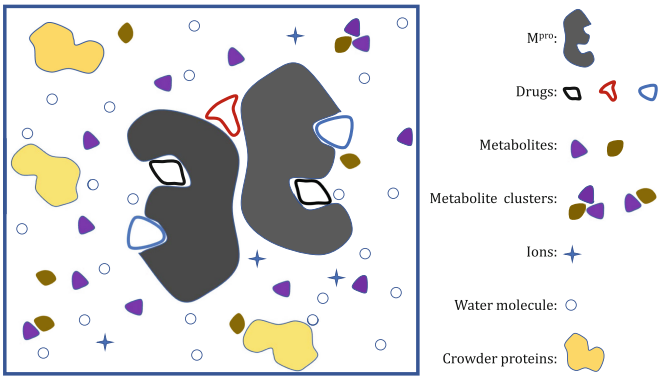

Fig. 2. A schematic model of M^{pro} system with 185,255 atoms. The more realistic model for simulation involved not only system of interest (M^{pro} protein bound to 5 drugs of 3 types) but also the *crowded* environment. This environment consisted of crowder proteins, metabolites seen in cells, ions and water. The second system, M^{pro}-free, did not have crowder proteins and drugs.

While a docking study identifies *where* drugs may bind on a protein, MD simulations help to identify *if* such a complex is viable. Analyzing the dynamics of simulated system involves processing numerous frames of data generated by MD simulation. Details of the system at a given time (such as coordinates and sizes of the simulated box) are captured through a frame associated with it. Sequence of frames thus reproduce evolution of system modelled by simulation. While simulations run over hundreds of nanoseconds (ns), simulated data is typically stored (as frames) at intervals of 1 picosecond (ps). In M^{pro}-drug, 8 protein GB1s (each with 56 amino acids) were used as protein crowders along with metabolites of 10 different types. Details of the two chosen systems is given in Table 1.

Table 1. Composition of MD simulations of M^{pro}-free and M^{pro}-drug systems is given below. Size of M^{pro} is given terms of amino acids (aa.). Crowders comprising of crowder proteins (Cr. proteins), metabolites and ions were incorporated in the simulations.

Item	M^{pro}-free	M^{pro}-drug
Protein	608 a.a.	608 a.a.
Drugs	0	5
Cr. proteins	0	8
Metabolites	95	170
K^+ ion	90	190
Total atoms	100,093	185,255
Frames	1,000,000	200,000
Data size of frames (GB)	1201	445

Apache Spark is a unified analytics engine, developed for big data applications analytics [8,24]. The Spark-based suite of programs, called SparkTraj, was written in Scala language as proofs-of-concept implementation for scalable computation on MD data. Three parameters were computed for the evaluation of performance, viz., (a) radius of gyration (RoG), (b) molecular contacts (MCon), and (c) water networks (WNw). CPPTRAJ [16] is the standard accompanying tool of AMBER package. It was used as benchmark implementation for serial version to compute radius of gyration (RoG) of C_α atoms. MCon is the total pairs of atoms that are within a cutoff distance of 5 Å, with no equivalent serial implementation. Ankush [20] is a program to find water networks. The parameters are defined below and are of interest for various reasons as discussed in Sect. 3.1 under **Parameter selection for benchmarking**.

Radius of Gyration: The compactness of a protein can be measured using radius of gyration, which would increase if the protein begins to unfold. In the current work, this was equivalent to finding the root-mean-squared deviation of C_α atoms. RoG of only the first chain of M^{pro} was computed.

$$\mathrm{RoG} = \sqrt{\frac{1}{N}\sum\nolimits_{i=1}^{N}(r\_\overline{r})^2}$$

Molecular Contacts: Typical MD simulations consider only biomolecule(s) of interest apart water molecules and ions. Cellular environment comprises of many other molecules that influence the dynamics of system of interest. Under such circumstances, it would be necessary to find molecules (such as metabolites) getting in proximity with protein of interest. As shown in Fig. 3, this information was captured using MCon algorithm that finds the number of pairs of atoms within a distance of 5 Å between two molecules.

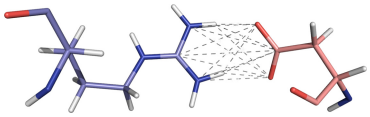

Fig. 3. Due to crowded environment within cells, molecules often bump into one another. The number of pairs of atoms that are within a distance of 5 Å is called as molecular contacts (MCon). MCon reflects proximity of molecules. The figure shows a few such contacts between two molecules represented with dotted lines.

Water Networks: A water network is a set of polar atoms (oxygen/nitrogen atoms of solute shown as P_i in Table 2) that are within 3.5 Å distance with a common water molecule. As shown in Fig. 4, some networks such as $\{P_1, P_2\}$ are formed independently by different water atoms. Larger networks such as $\{P_a, P_b, P_c, P_d\}$ give rise to smaller networks such as $\{P_b, P_d\}$ and $\{P_a, P_c, P_d\}$. Occurrence (occ.) of a water network is defined as the number of frames in which a given network is seen. From all observed water networks (WNw^c in Table 2), we *selected* for benchmarking water networks with a minimum occurrence of 40% (WNw in Table 2), though lesser cutoffs were used for the study. *Trajectory* for each WNw, which shows its presence or the absence in each frame ordered by time, was computed. Using trajectory, Maximum Residence Time (MRT), defined as the highest number of sequential frames in which a particular WNw continues to exist, was also computed.

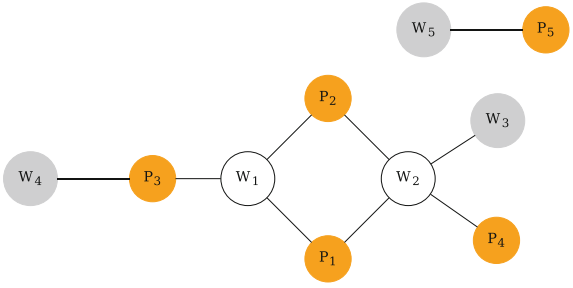

Fig. 4. Two nodes (i.e., atoms) are connected if, (a) their distance is within 3.5 Å, and (b) if least one node is oxygen of water. A water network is a set of 2 or more nitrogen/oxygen atoms of solute connected to a common water molecule. The networks seen here are $\{P_1, P_3\}$, $\{P_2, P_3\}$, $\{P_1, P_2, P_3\}$ through W_1, $\{P_1, P_4\}$, $\{P_2, P_4\}$, $\{P_1, P_2, P_4\}$ through W_2, and $\{P_1, P_2\}$ independently through W_1 and W_2.

It is necessary to not just process but also to read data in parallel to obtain meaningful scaling of computation when size of data is large. To this end we used Apache Spark (referred also as 'Spark') with Scala. Spark is an open-source distributed general purpose framework that is designed for large-scale data analysis and offers implicit data parallelism with fault tolerance [8]. AMBER generates

Table 2. From all candidate water networks (WNw^c), only those networks with minimum occurrence (i.e., 40%) were filtered (WNw). Also see Fig. 4.

Item	M^{pro}-free	M^{pro}-drug
Polar atoms (P_i)	2,250	4,076
Water molecules (W_i)	29,593	54,968
WNw^c	5,566,453	3,261,310
WNw	655	1,317

data in NetCDF format [10]. As Spark does not have native support for NetCDF format, SciSpark was used to read the simulation data. SciSpark is a framework developed by NASA and UCLA for applications in earth and space sciences [13]. The basic approach for computation we used was to read the MD data into SciSpark's Resilient Distributed Datasets (RDDs) [23] first, followed by computation of necessary parameters through Spark Datasets and DataFrames. SciSpark supports HDFS [1] as well as native Linux file systems. In the next section, Spark's application to MD data analysis is discussed in detail.

NVIDIA Tesla V100 based GPU nodes were used for simulations. For benchmarking, a cluster with 20 data nodes was used. GRIDScaler SFA7700X appliance with 15 GB/s of throughput was the storage server for the two systems. GPU nodes and data nodes of the cluster have two Xeon Gold 6148 CPUs with 384 GB of RAM. The big data cluster supported Apache Spark (ver. 2.3.0) and Scala (ver. 2.11.8) for computations.

3 Results and Discussion

3.1 Spark-Based Processing of MD Simulation Data

AMBER18 [18] was used for simulating the chosen systems, which saves output in one or more *mdcrd* files in NetCDF format. Based on the user input, one or more frames can be written in each mdcrd file. NetCDF format allows parallel reading of a single file. To work within the limitations of HDFS block size [2], only a limited number of frames per file were stored. Each mdcrd file of M^{pro}-free and M^{pro}-drug systems contained data 50 frames. AMBER stores the detailed description of a simulated system (such as names of atoms, various molecules, types of bonds, etc.) in a separate file called *parmtop*.

Apache Spark is a framework designed for massive in-memory data processing with lazy evaluation. This framework was used to process the simulation data. To implement efficient solutions, both Spark operations as well as sequence of their application should be carefully chosen. Sub-optimal approach may lead to significant delays due to *data shuffling* (i.e., movement of massive data across nodes due to its redistribution). It may even lead to expensive recomputations of Spark's RDDs. Hence, we developed the solutions to circumvent these significant

barriers. Another potential rate-limiting step avoided was the generation and use of intermediate data on storage.

Parameter Selection for Benchmarking: Three carefully chosen parameters shape the proof concept implementations. Radius of Gyration (RoG) is a trivial calculation akin to finding root-mean-square deviation of less than 0.5% of atoms. This computation allows to estimate the throughput of data available to Spark. Computation of Molecular Contacts (MCons) requires examination of hundreds of millions possible contacts in each frame. This is a straightforward computation can be done using either an efficient User-Defined Function (UDF) or a Dataset method. As each frame needs to be processed only once and results can be collected with relative ease. Given the resilient nature of Spark RDDs and the involvement of huge data across multiple phases, water networks demand careful computations to avoid performance hazards such as data shuffling.

Radius of Gyration: CPPTRAJ [16] was used to compute the radius of gyration for benchmarking of serial variant. It is the standard accompanying software for AMBER package and is used to compute a wide variety of parameters from simulation data.

Algorithm 1: Calculation of Radius of Gyration

read topology from parmtop
create $mdRDD$ from mdcrds
obtain $time, RoG$ through:
  flattening $mdRDD$ into $frRDD$
  creating Dataset using additional parameters
  invoking Dataset's method for RoG
sort RoG w.r.t. $time$

Algorithm 1 shows the approach used for the Spark version. One should first note that if a system has N atoms, it has $3N$ coordinates (i.e., 3-D coordinates x_1, y_1, z_1, ..., x_N, y_N, z_N). Details of the system were read from the *parmtop* file (as discussed in the previous section). Then, the mdcrd files were read parallelly into a Resilient Distributed Dataset (RDD) called mdRDD loaded through SciSpark. mdRDD was then *flattened* into another RDD, called frRDD, so that every element was now a tuple that contained information of a particular frame in the form of ($time_i$, $box_{i,x}$, $box_{i,y}$, $box_{i,z}$, $x_{i,1}$, $x_{i,2}$, ... $x_{i,3N}$), where i is the frame number. Then, a Spark Dataset, with a method to find RoG for a frame, was created using frRDD. Invoking this method, RoGs of all frames were computed, data was collected into a Scala list, and then sorted w.r.t. the time of the frames. Sorting by time was not done using Spark itself in any of the implementations for reasons discussed later in this section. Complexity of the computing RoG within a frame in terms of the number of residues in the given chain (n_{res}) is $O(n_{n_{res}})$. Additional parameters include parmtop and cutoff data.

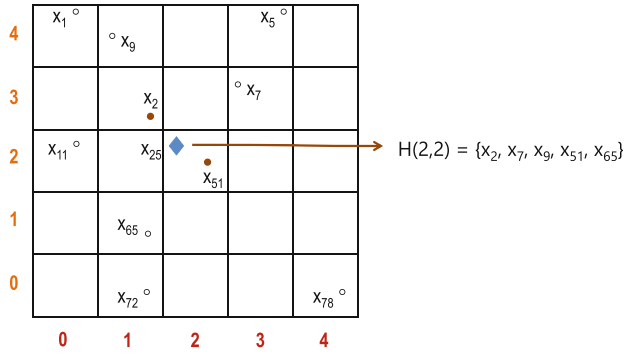

Fig. 5. The diamond shows the atom of interest, circles are other atoms, and only those represented with filled circles have relevant contacts. If we discretize space into squares with size cut-off distance (d) itself, examination of 9 squares is adequate to find potential contacts of an atom. A hash data structure, with tuples made of its discretized coordinates as key, can store all potential atoms that need to be examined for any atom in a given square. The same approach can be extended to 3-D for speedy computation of molecular contacts and water networks.

Molecular Contacts: Unlike the other two parameters, MCon could not be benchmarked with serial version due to absence of equivalent implementation. As shown in Algorithm 2, much of the procedure to compute molecular contacts was similar to that of the radius of gyration, but with two differences. The output from each frame was not a number (like RoG) but instead a string, which, when split, gave triplets (r_i, r_j, cnt) where r_ks were residue numbers and *cnt* was the number of contacts they shared. Also, unlike RoG, MCon requires finding contacts between two large sets of atoms. M^{pro}-drug system, for instance, has 9,796 atoms for the M^{pro}-drug complex and crowder proteins and metabolites constitute 10,364 atoms. This translates to over 101 million pairs of atoms. Clearly, a brute force way of counting is prohibitively expensive. As explained below, only potentially feasible pairs of atoms were scanned.

A 2-D illustration of the idea is shown in Fig. 5. First, we scanned every atom from one set and placed them in all possible (i.e., 9) squares where they may have contacts. Then, given an atom of interest from the other set, say x_{25}, we can immediately examine the list of atoms corresponding to the square. This approach allows us to linearize the computation time to find MCon as physically each square can only have a certain number of atoms at any time. The time taken for calculating MCon was comparable to that of RoG, which is not very surprising considering the overall complexity of finding MCon within a frame

now being $O(L_xL_yL_z)+O(n_{systemAt})+O(n_{crowderAt})+O(n_{conacts}log(n_{conacts}))$ where L_is are lengths of simulation box along x, y and z directions.

Algorithm 2: Calculation of Molecular Contacts

read topology from parmtop
create $mdRDD$ from mdcrds
obtain $time, \{MCon\}$ through:
  flattening $mdRDD$ into $frRDD$
  creating Dataset using additional parameters
  invoking Dataset's method for $MCon$
sort $MCon$ w.r.t. $time$

Water Networks: Water networks were benchmarked against the serial version that works in two stages. In the first stage, it generates 2 files per frame; one that contains the contacts between polar atoms and water molecules, and the other between water molecules themselves. In the second stage, it processes these files to determine water networks. The first stage uses a C++ program and the second stage a Perl script. While this version worked well when it was originally developed, currently it is too inefficient for large systems with hundreds of thousands of frames. Hence, as shown in Algorithm 3, a Spark version was developed to update it.

Algorithm 3: Water network calculations

read topology from parmtop
create $mdRDD$ from mdcrds
create and cache $nwDF$ through:
  flattening $mdRDD$ into $frRDD$
  creating Dataset through additional parameters
  obtaining $time, \{WNw^c\}$ through Dataset's method
create $\{WNw\}$ using $nwDF$ and occ. cutoff
compute $WNwTraj$ using WNw and $nwDF$

Unlike the previous two algorithms, the study of water networks involves multiple phases. Phase 1 involves processing all frames to find all networks seen, called candidate water networks (WNw^c). The number of networks could be very high. For example, M^{pro}-free simulation generated over 5.5 million networks. In phase 2, the presence or absence of every network in each frame is used to compute occurrence of networks, and only networks with a minimum occurrence are selected (WNw). Finally, using the data available from phase 1, trajectory of the selected networks are be computed. RDDs have a tendency to recompute. Given the amount of data involved in every phase, a careful implementation is necessary to avoid data shuffling and expensive recomputations by Spark.

The approach we use is as follows. To prevent recomputations from slowing down the process, we first find and cache all (candidate) water networks seen in each frame $\{\text{WNw}^c\}$. The occurrence of each network is then computed, and only those networks of interest are selected based on occurrence $\{\text{WNw}\}$. The trajectories (WNwTraj), consequently maximum residence times (MRTs), are computed from the cached data. Network trajectories can be stored if desired. However, we found that in general, it was adequate to have the summary featuring occurrence, MRT, and the window in which MRT was observed. As discussed in the case of MCon, the hash-list approach described through Fig. 5 was applied to short-list the interacting polar atoms of each water. First, all polar atoms are scanned and placed in lists corresponding to cuboids where they may have contacts. Then, for each water, the corresponding list of its hash are scanned. Once all connected atoms for any particular water are known, all subsets (i.e., networks) with at least 2 elements can also be created. The complexity of finding water networks within a frame is $O(L_xL_yL_z) + O(n_{polarAt}) + O(n_{waterAt}) + O(n_{wnet}log(n_{wnet}))$.

A vital aspect of calculating distance between atoms involves taking into account the *periodic boundary condition* (PBC) of the simulation box. It means that during simulations a molecule does not face a wall at the edge of the box. Instead, it can move transparently from one side, say from right in Fig. 2, and appear from the left. (The rationale for PBC is beyond the scope of this work.) Hence, the two metabolite clusters seen here are not far away but are much closer. The size of the simulation box itself may change during simulation and hence frame-wise box lengths are stored in mdcrd files.

3.2 Benchmarks and Insights

The two chosen systems for the proof-of-concept implementations (M^{pro}-free and M^{pro}-drug) were benchmarked. Given the necessity for consistency and substantial resource requirements, three variants were tested on the GPFS-based DDN storage solution capable of 15 GBps throughput for both read and write operations. These were, (a) serial version (serial), (b) Spark-version with data on native GPFS filesystem (gpfs), and (c) Spark-version with data on HDFS (hdfs) filesystem. HDFS storage was available from the same GPFS server through a connector. Times of completion for the three parameters are given in Table 3.

Table 3. Time taken for completion of any job in serial mode (serial) is the longest, compared to Spark when input was read from native GPFS filesystem (gpfs) or HDFS filesystems (hdfs). All times are in seconds and speedups are shown in parenthesis when available.

Parameter	M^{pro}-free			M^{pro}-drug		
	serial	gpfs	hdfs	serial	gpfs	hdfs
RoG	8,691	1,889	273	1,931	584	131
MCon	–	1,928	735	–	584	176
WNw	667,000	1,953	619	251,010	614	218

It is clear from Table 3 that performance differs significantly when the input is read from HDFS, instead of the GPFS filesystem. In particular, *speedup*, as defined in Eq. (1), shows that close to 5 times speedier computation can be done if input is read from HDFS instead of GPFS filesystem.

$$speedup = \frac{time_{serial}}{time_{spark}} \tag{1}$$

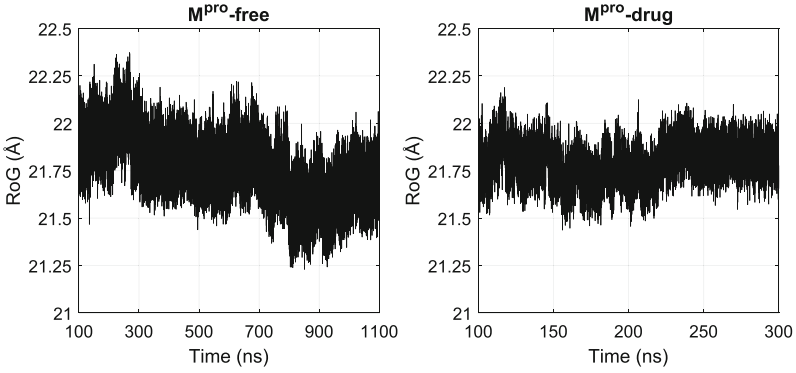

Fig. 6. The radii of gyration for M^{pro}s were stable, indicating that no unfolding of these proteins happened during course of simulations. Shown here is the RoG of one monomer.

The RoGs of the two systems are shown in Fig. 6. The values show that no unfolding of proteins occurred during the simulation. Speedups over 1000 were obtained for water network calculation. As the serial version could not be run over the entire trajectory, after confirming with shorter runs (with up to 20% of data), linear scaling was applied to estimate the time required for computations. MCon, another important parameter that requires only a single pass over each frame and is critical for the analysis of crowder-based simulations, performed faster than WNw. While it could be argued that serial version for water network could be improved, it is evident that the times for completion of MCon and WNw are comparable to that of RoG; all three of them requiring just a few minutes to complete their tasks. While CPPTRAJ required 8,691s to compute *RoG*, Spark (hdfs) for *WNw* was completed within 619s. In other words, compared to RoG, SparkTraj's WNw was 14 times faster over 1.2 TB!

Interesting events were captured using MCon. As Fig. 7 shows, multiple metabolites were seen interacting with drug R1. The most significant interactions were seen with only two drugs, a Ritonavir and a Glecaprevir. With further analysis based on this information, a cluster of 5 metabolites were identified that were seen blocking the movement of this drug, perhaps for the first time in any simulation. Ordinarily, such long-duration associations occur due to hydrogen bonds, salt bridges, π-π interactions, etc. Here, no long-lasting hydrogen bonds

were seen as these metabolites were mobile, though still in the vicinity of the drug during the entire course of simulation. Such interactions need to be computed, and a parameter such as MCon is very useful for the same. Detailed insights and their significance obtained using SparkTraj is beyond the scope of this work and is discussed elsewhere [19].

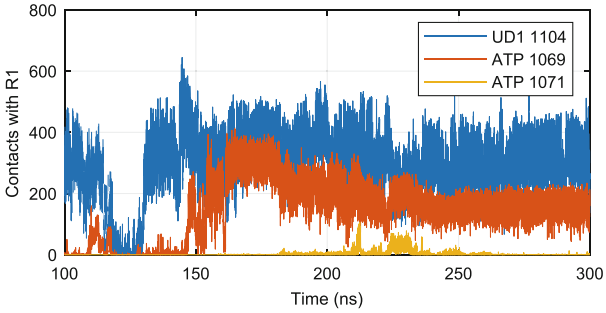

Fig. 7. MCon is useful to identify metabolites that are in proximity. Seen here is a drug (Ritonavir R1) that had consistent interactions with a few metabolites.

As already discussed in Introduction, water networks (WNw), especially those that stabilize drug binding to proteins and other receptors, are of interest. With the current study, we have been able to quickly identify such water networks. As shown in Fig. 8, they were also seen between drugs and M^{pro}.

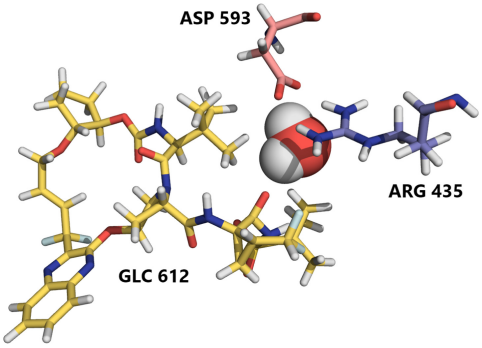

Fig. 8. Water Network (WNw) present between a drug (GLC 612) and two amino acids of M^{pro}.

Another crucial aspect of benchmarking software would be to find how the time for completion *scales* with size of the data. Normally, processing time being same for each frame, increasing number of frames should lead to an increased time for completion by the same factor. For a large system of terabyte size such

as M^{pro}-free, increasing input size by 10-fold matched the expected performance where GPFS filesystem was used (see Table 4). In case of M^{pro}-drug system (20K vs 200K frames), the escalations were smaller. It is very interesting that the escalation of times were close to half in case of HDFS environment w.r.t. GPFS filesystem.

Table 4. With a 10-fold increase in data size, escalation in times for completion was almost by the same factor for the larger M^{pro}-free system when data was read from GPFS filesystem, and by much less in case of HDFS. M^{pro}-drug had lesser increment.

Parameter	M^{pro}-free		M^{pro}-drug	
	gpfs	hdfs	gpfs	hdfs
RoG	8.7	3.5	6.0	2.3
MCon	7.8	7.7	6.5	2.6
WNw	8.5	4.8	5.9	2.9

To improve the performance of Spark implementation, many variations were considered and examined. For instance, sorting (by time) using Spark was less efficient due to data shuffling. So the data was first collected and was sorted directly using Scala. While we contemplated different ways to improve the performance further, it was not pursued to avoid possible over-engineering; reckon that time required to read mdcrds was at least 60% of the total runtime.

To appreciate the immense value and contribution of such studies, it would be pertinent to reckon a few insights from the simulations of M^{pro} complex. Apart from metabolites and metabolite clusters, identified initially through MCon, the simulations in crowded environment yielded other fascinating insights as well. They include, possible preference of certain metabolites to particular sites, movement of a free amino acid in a probable (drug) binding site, and *crawling* of a drug over the surface from one pocket to another in presence of crowders (both proteins and metabolites).

3.3 Framework for Cloud-Based MD Simulation Service

The unique advantage of Cloud infrastructure over typical high-performance computing servers is the ability to scale resources on demand. However, the operating costs of Cloud infrastructure vary based on type, quantity and duration [5,12]. We present a framework to expedite the best MD simulations studies possible within the available resources.

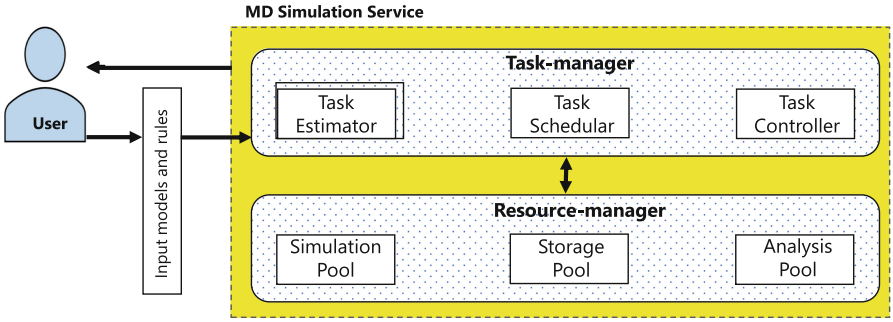

Fig. 9. Block diagram of the proposed framework for MD studies. Task-manager estimates task size, generates an execution plan, and in concert with Resource-manager, dynamically orchestrates tasks and resources. Periodic updates and final results are sent to user for information and feedback.

As shown in Fig. 9, given the models to simulate and the accompanying rules (such as given priorities and available funds), Task Estimator first runs sample jobs to assess the time required on various platforms, determines the number, types of servers, infrastructure, and services that could be acquired within given constraints. Task Schedular then prepares a schedule of jobs to be submitted. In certain cases there could more specific rules. For instance, consider *screening* of drug-candidates. A user may require only those drug-candidates that retained interactions with the target protein throughout the simulation(s). This could be monitored by Task Controller that can preemptively terminate simulations in which drugs lose interactions. Other reasons to pre-emptively discard a model from simulation studies include extreme changes in RoG, insufficient binding energy between drug and protein, etc. This ability to dynamically preempt simulations of poor drug-candidates allows users to channel the resulting savings into examination of additional drug-candidates or to enhance infrastructure for the computations. Task-manager updates Resource-manager on inputs, schedules, and resources to be deployed. Resource-manager then acquires, manages and frees appropriate software, services or (virtual) hardware from Cloud. The necessary software and hardware resources for simulation, analysis, and storage together form corresponding pools. As computations progress, Task-manager updates user on the latest status of simulations/analysis (scheduled, underway, completed, or terminated), available results, deployed resources, and the projected time for completion of the work. If required, the user may opt to intervene and update inputs to alter or improvise the study undertaken, or expand its scope and allocated funding.

This above framework can be implemented by the Cloud provider (First-party API) or could be developed by third parties (Third-party API). Competent clients may develop solutions through their own efforts and deploy them either on public or private Clouds. In principle, it is of practical utility, and with relative ease, the solution can be deployed by providers of Cloud infrastructure

to identify even those drug-receptor candidates that may work well in aqueous conditions but fail in physiological environment. This would definitely help in containing both time and phenomenal costs involved in the development of new drugs.

3.4 Limitations

Installing and using software designed for Apache Spark requires more expertise compared to their serial counterparts. Further, we have studied different systems on the only state-of-the-art commercial storage that was available. It was selected as the server provides access to both local as well as HDFS filesystems on the same hardware, and was connected a dedicated big data server with 20 nodes. The performance gains may wary between different storage solutions. Like for any distributed computing, a few issues are critical for performance. One is the latency and bandwidth available to load the input files. The second is the amount of time required for computation itself. For instance, latency and (data) bandwidth could be significantly improved using more expensive solid-state storage devices such as NVMe SSDs even on less expensive desktop systems. This could improve the performance of parameters that have little computation, such as RoG. However when a frame requires significant amount of computation, then having higher core count, RAM and networking bandwidth would be more useful. Nevertheless, the above proposed cloud framework could be designed to select optimal storage and computational resources based on available funds.

SparkTraj is freely available for the interested researchers.

4 Conclusions

In this paper, using Apache Spark, we demonstrated that big data approach provides substantial speedups in MD studies. This is especially needed in the more cell-like environment involving a plethora of biomolecules. A responsive, scalable, and self-tuning framework that pairs Spark with the flexibility of Cloud infrastructure for the MD studies is presented. This framework enables users to optimally utilize the available resources. Insights obtained using the proposed approach were discussed. Complex MD studies, accompanied by such newer and more versatile tools, would bridge the gap between theoretical modelling and experimental observations. Such studies may soon become an imminent requirement in the capital intensive yet time-constrained pharmaceutical industry. The relevance of such advances, especially in these pandemic times, cannot be overstated.

Acknowledgment. We thank the Central Computing Facility of IIIT Allahabad (IIIT-A) for all the computations carried out. We thank Dr. Muneendra Ojha (IIIT-A) and Dr. Jagpreet Singh (IIIT-A) for proof-reading the manuscript. BSS thanks Mr. Asari Ramprasad (IIIT-A) and Ms. Shivani Maheshwari (Postman Inc.) for useful discussions on SciSpark and Cloud infrastructure. Dr. Arumugam Madhumalar and Mr. Zahoor Ahmad Bhat (JMI, New Delhi) are also gratefully acknowledged for the help

provided regarding M^{pro} simulations. SparkTraj was developed by one of the authors (BSS), and analysis was done jointly by the two.

References

1. Borthakur, D.: The hadoop distributed file system: architecture and design. Hadoop Project Website **11**(2007), 21 (2007)
2. Borthakur, D., et al.: HDFS architecture guide. Hadoop Apache Project **53**(1–13), 2 (2008)
3. Bux, K., Moin, S.T.: Solvation of cholesterol in different solvents: a molecular dynamics simulation study. Phys. Chem. Chem. Phys. **22**(3), 1154–1167 (2020)
4. De Vivo, M., Masetti, M., Bottegoni, G., Cavalli, A.: Role of molecular dynamics and related methods in drug discovery. J. Med. Chem. **59**(9), 4035–4061 (2016)
5. Amazon EC2: Amazon Web Services (2006). http://aws.amazon.com/
6. Feig, M., Yu, I., Wang, P.h., Nawrocki, G., Sugita, Y.: Crowding in cellular environments at an atomistic level from computer simulations. J. Phys. Chem. B **121**(34), 8009–8025 (2017). https://doi.org/10.1021/acs.jpcb.7b03570. PMID: 28666087
7. Gao, M., et al.: Modulation of human IAPP fibrillation: cosolutes, crowders and chaperones. Phys. Chem. Chem. Phys. **17**(13), 8338–8348 (2015)
8. Karau, H., Warren, R.: High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark. O'Reilly Media Inc., Sebastopol (2017)
9. Kuznetsova, I.M., Turoverov, K.K., Uversky, V.N.: What macromolecular crowding can do to a protein. Int. J. Mol. Sci. **15**(12), 23090–23140 (2014)
10. Li, J., et al.: Parallel netCDF: a high-performance scientific I/O interface. In: SC 2003: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, p. 39. IEEE (2003)
11. McGibbon, R.T., et al.: MDTraj: a modern open library for the analysis of molecular dynamics trajectories. Biophys. J . **109**(8), 1528–1532 (2015)
12. Microsoft: Microsoft Azure (2014). https://azure.microsoft.com/
13. Palamuttam, R., et al.: Scispark: applying in-memory distributed computing to weather event detection and tracking. In: 2015 IEEE International Conference on Big Data (Big Data), pp. 2020–2026. IEEE (2015)
14. Pikkemaat, M.G., Linssen, A.B., Berendsen, H.J., Janssen, D.B.: Molecular dynamics simulations as a tool for improving protein stability. Protein Eng. **15**(3), 185–192 (2002)
15. Rincón, V., Bocanegra, R., Rodríguez-Huete, A., Rivas, G., Mateu, M.G.: Effects of macromolecular crowding on the inhibition of virus assembly and virus-cell receptor recognition. Biophys. J . **100**(3), 738–746 (2011)
16. Roe, D.R., Cheatham, T.E., III.: PTRAJ and CPPTRAJ: software for processing and analysis of molecular dynamics trajectory data. J. Chem. Theory Comput. **9**(7), 3084–3095 (2013)
17. Salo-Ahen, O.M., et al.: Molecular dynamics simulations in drug discovery and pharmaceutical development. Processes **9**(1), 71 (2021)
18. Salomon-Ferrer, R., Case, D.A., Walker, R.C.: An overview of the amber biomolecular simulation package. Wiley Interdiscip. Rev. Comput. Mol. Sci. **3**(2), 198–210 (2013)
19. Sanjeev, B.S., Chitara, D., Arumugam, M.: Physiological models to study the effect of molecular crowding on multi-drug bound proteins: insights from SARS-CoV-2 main protease. J. Biomol. Struct. Dyn. (2021). https://doi.org/10.1080/07391102.2021.1993342

20. Sanjeev, B.: Ankush. Indian Institute of Science (2004)
21. Shaw, D.E., et al.: Atomic-level characterization of the structural dynamics of proteins. Science **330**(6002), 341–346 (2010)
22. Wouters, O.J., McKee, M., Luyten, J.: Estimated research and development investment needed to bring a new medicine to market, 2009–2018. JAMA **323**(9), 844–853 (2020)
23. Zaharia, M., et al.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 2012), pp. 15–28 (2012)
24. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I., et al.: Spark: cluster computing with working sets. In: HotCloud 2010, no. 10, p. 95 (2010)