Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Manuscript Category

OXFORD

Subject Section

The application of Hadoop in Structural Bioinformatics

Jamie Alnasir [1,*], Hugh P. Shanahan [2]

[1] Institute of Cancer Research, 123 Old Brompton Road, London, SW7 3RP, United Kingdom and
[2] Department of Computer Science, Royal Holloway, University of London, Egham, Surrey, TW20 0EX, United Kingdom.

*To whom correspondence should be addressed.

Associate Editor: Martin Bishop

This is a pre-copyedited, author-produced version of an article accepted for publication in Briefings in Bioinformatics following peer review. The version of record, The application of Hadoop in structural bioinformatics. Alnasir, Jamie; Shanahan, Hugh. Briefings in Bioinformatics, 20.11.2018, p. 1-10, is available online at: https://academic.oup.com/bib/advance-article-pdf/doi/10.1093/bib/bby106/26649330/bby106.pdf

Abstract

The paper reviews the use of the Hadoop platform in Structural Bioinformatics applications. For Structural Bioinformatics, Hadoop provides a new framework to analyse large fractions of the Protein Data Bank that is key for high throughput studies of (for example) protein-ligand docking, clustering of protein-ligand complexes and structural alignment. Specifically we review in the literature a number of implementations using Hadoop of high-throughput analyses and their scalability. We find that these deployments for the most part use known executables called from MapReduce rather than rewriting the algorithms. The scalability exhibits a variable behaviour in comparison with other batch schedulers, particularly as direct comparisons on the same platform are generally not available. Direct comparisons of Hadoop with batch schedulers are absent in the literature but we note there is some evidence that MPI implementations scale better than Hadoop. A significant barrier to the use of the Hadoop ecosystem is the difficulty of the interface and configuration of a resource to use Hadoop. This will improve over time as interfaces to Hadoop e.g. Spark improve, usage of cloud platforms (e.g. Azure and AWS) increases and standardised approaches such as Workflow Languages (i.e. WDL, CWL, Nextflow) are taken up.
Keywords: Structural Bioinformatics, Hadoop, Cloud Computing.

1 Introduction

The Apache Hadoop project [1] is a software ecosystem i.e. a collection of interrelated, interacting projects forming a common technological platform [2] for analysing large data sets.

Hadoop presents three potential advantages for the analysis of large Biological data sets. In the first instance, it is designed for the analysis of large semi-structured data sets; secondly it is designed to be fault tolerant (in essence by ensuring a very large amount of overall redundancy) which becomes almost inevitable for sufficiently large numbers of processors; finally the MapReduce formalism for describing the data sets allows for the easy construction of work flows.

Given the continued importance of computationally analysing protein structures in fields such as Membrane proteins [3], protein-protein interactions and their impact in Systems Biology [4, 5] and protein engineering [6] a platform for easily analysing semi-structured data sets such as those found in the Protein Data Bank [7] on a large would be an important step forward.

On the other hand, Hadoop also presents barriers to its adoption within the community for Bioinformatics and the analysis of structural data. In the first instance Hadoop runs as a series of Java libraries and hence there is a learning curve for any Bioinformatician or Structural Biologist who hasn't used Java before, though we note that more recent additions to the Hadoop ecosystem such as Spark [8] have a wider range of languages (e.g. Python, Scala and R). Correspondingly, unlike data parallel languages such as High Performance Fortran [9], Hadoop cannot be easily retro-fitted into a stable code base even if the original code is written in Java though it is possible to use Hadoop to deploy instances of executables in the same way that batch schedulers do. Finally, implementing Hadoop (and therefore Spark) on a local cluster is not trivial and requires a significant level of expertise

from the relevant systems administrator. As we note, this latter difficulty is obviated on cloud platforms such as Azure and AWS [10].

This paper reviews the range of work that has been done on Hadoop in Bioinformatics and in Structural Bioinformatics in particular. Specifically this paper will determine how stable these platforms are in comparison to other approaches such as batch schedulers and MPI (discussed in sections 1.1.3 and 1.1.4 respectively). The rest of this paper is organised as follows. In the first instance a brief overview of the Hadoop and Spark systems as well as a description of batch schedulers and MPI. It then describes the Hadoop formalism. A brief review of the application of Hadoop in Bioinformatics is provided followed by an in-depth review of the application of Hadoop in Structural Bioinformatics. Specifically we examine the range of cases where Hadoop has been used, the scaling behaviour of such systems in comparison with other platforms, where available and its dependence on the internal configuration of the version of Hadoop used. Finally this paper will draw some conclusions on the scalability of Hadoop and its application in Structural Bioinformatics.

1.1 Distributed computing architectures

1.1.1 Hadoop and MapReduce

Apache Hadoop is a software framework for distributed processing and storage which is typically installed on a Linux compute cluster to facilitate large scale distributed data analysis, though it can be run on a standalone single computer node usually for the purposes of prototyping. Hadoop clusters may be built using commodity hardware, for instance off the shelf equipment such as used in computer farms, and key features are **fault-tolerance** and **data-locality**. In the former case scaling up a cluster to add more machines and disks increases the probability of a failure occurring and hence systems must have a built-in redundancy to compensate for it. In the latter case data-locality provides the ability of the framework to execute code on the same node or at least the same rack of the cluster as where the input data resides. This reduces the amount of network traffic during processing thereby avoiding network bottlenecks [11]. The fault-tolerance and data-locality of Hadoop are made possible by its distributed file system (HDFS) [12] and by Hadoop's resource scheduler YARN (Yet Another Resource Negotiator) [13] which is responsible for cluster management, in particular resource allocation and job scheduling.

The distributed data on HDFS is accessed programmatically using the MapReduce formalism, originally implemented in Java. In this formalism the distributed data accessed from HDFS is a set of *tuples* i.e. pairs of keys and values $< k_i, v_i >,\ 1 \leq i \leq N$ where N are the total number of data entries. For example, the entries in the Protein Data Bank (PDB) would be a set of pairs where k_i would be a specific PDB id and v_i could be a single string with the corresponding PDB data in XML format. All operations are based on these tuples, either creating new tuples (i.e. through a *Map* step) or summarising the data stored in the tuples (i.e. through a *Reduce* step). Extending the above example, via a *Map* step, a specific executable (e.g. a docking program run with a specific small molecule) could be run on each PDB entry to create a log file for each PDB entry. In the MapReduce formalism this means creating a new set if tuples $< k_i, l_i >$ where k_i is again the PDB entry and l_i is a single string with the log file. A *Reduce* step could then be applied on this second set of tuples to create a single tuple which carries some specific set of summary data (e.g. how many structures had a docking score greater than some threshold in the previous *Map* step).

The rise of the use of Hadoop has mirrored the increasing use of cloud platforms. MapReduce is offered as a Platform as a Service (PaaS) by all of the major cloud-service providers (Amazon AWS, Google Cloud and Microsoft Azure) [14].

1.1.2 Apache Spark

Spark is a cluster computing framework that can be used standalone or can utilise Hadoop's distributed file system (HDFS) and a resource scheduler (often Apache YARN), providing an application programming interface (API) for distributed computation. Spark is designed to overcome some of the constraints of Apache Hadoop offering significant performance improvements but keeping the fault-tolerance and scalability features of Hadoop by utilising HDFS [15].

Although Spark supports MapReduce programs, the main constraint Spark overcomes is Hadoop's acyclic data flow model by allowing the re-usability of intermediate data in the form of a data structure that is central to Spark, the Resilient Distributed Dataset (RDD). The RDD serves as an abstraction for distributed memory that allows in-memory computations on large clusters in a fault-tolerant manner, and because an RDD is partitioned across multiple compute nodes, can be rebuilt if for some reason a partition is lost [16]. The RDD is an immutable (read-only) data structure that can encapsulate objects from a number of different programming languages and typically on Spark this is Python, Java, or Scala and In contrast Hadoop development, notwithstanding that Hadoop MapReduce programs can be written in any language supporting the UNIX POSIX standard I/O streams, is predominantly Java based. This is significant from a development perspective because Java requires programmers to possess more specialist object-orientated programming (OOP) knowledge than for example Python, and Java programs tend to have more dependencies on runtime libraries.

An RDD is usually created in two ways, by referencing an external data source, for instance a dataset or file on HDFS, or by parallelising an existing Spark data structure. Parallelising a data structure, for instance an array in Spark, allows it to be operated on in a parallel fashion whereby Spark handles the caching of the RDD in memory across nodes of the cluster. As a result of this, Spark MapReduce operations gain significant performance enhancements over their Hadoop counterparts. This can be achieved because Spark creates an execution plan, in the form of a directed acyclic graph (DAG), of Spark and MapReduce operations to be performed on RDDs. The execution plan models dependencies and allows Spark to optimise execution of a jobs' components in a way that is not constrained to linearity of execution like MapReduce on Hadoop. Spark categorises some functions/procedures in Spark programs as *action* events, for instance a *reduce* step, and others as *transformations*, for instance a *map* step. This allows Spark to process the execution plan DAG using a method known as *lazy evaluation*, that is only *action* events cause data to be loaded into memory, whereas *transformations* in the execution plan are only executed when an *action* with dependency on that transformation is executed [17]. This method improves on cluster utilisation over Hadoop because it allows cluster resources to be acquired and released on an *ad-hoc* basis throughout a complex job. This avoids the rather less desirable but typical scenario in Hadoop in which resources (for instance executor processes - which are finite and dictated by cluster configuration) are reserved at the outset then released back to the cluster (hence making them available to other jobs) only when the running job is finished.

Apache Spark supports Python (through the PySpark API) allowing rapid development with easily installable modules, and offers substantial performance benefits over Hadoop. It is noteworthy that other popular languages in the field of bioinformatics, such as R, also have interfaces to Spark - for example SparkR [18].

1.1.3 Batch schedulers

A batch-scheduler (also referred to as a job-scheduler or workload management software), is a central software component of a compute cluster that manages the workload of computational jobs on a cluster and allocates cluster resources to them [19]. We will refer to the technique as

batch-scheduling and the central software component as the job-scheduler. Generally a computational job on a batch-scheduled system is a normal user program that runs on a single compute node, but can also be a more specialist distributed program that comprise of components written to run on multiple nodes which communicate by passing messages, for example using message passing interface (MPI) (discussed in the following section). Computational jobs are submitted in a batch to the job scheduler which adds them to a queue. The job scheduler decides on how to prioritise competing jobs, what resources to allocate to the jobs. The jobs are then submitted by the job scheduler to compute nodes of the cluster using a scheduling algorithm [20].

Batch-scheduled cluster systems couple job flow control with the ability to reserve (and limit) the allocation of cluster resources such as, for example, CPU cores, physical RAM, virtual memory and execution time. However batch-scheduled clusters offer only "course granularity" control of concurrency at the job-level (unlike MPI systems) and does not render the same level of fault-tolerance and data-locality through lack of a distributed file system such as HDFS that Hadoop provides (discussed above) unless an external distributed file system such as, for example, GPFS or Lustre is employed. Batch-scheduled cluster systems use the job scheduler to deploy "whole" executable programs to compute nodes of the cluster which may or may not run in a parallel fashion - for instance a single program when submitted will run on only one node, while multiple submitted jobs may run either on a single node or be distributed across multiple nodes depending on the load on each compute node and the scheduling rules set.

1.1.4 MPI

High performance computing (HPC) systems are typically reliant on a high degree of inter-process communication. The Message Passing Interface (MPI) is a standard for the development of parallel programming software and libraries for parallel computing architectures that standardises syntax [21]. MPI facilitates concurrent programming by specifically dictating the standard syntax to be used for the messages passed between communicating processes. MPI supports a wide variety of architectures such as multiple computers with distributed memory, shared memory multiple processors and heterogenous combinations of these.

MPI offers an extremely fine granularity of control over the processes involved in the execution of parallel, concurrent programs running across networked computers in clusters. Although MPI is extensively used in high performance computing, it can be used on clusters of standard machines or workstations. MPI requires the programmer to explicitly handle parallel functionality at a lower level than for instance Hadoop which automates parallelism of users programs through the MapReduce formalism. Whilst MapReduce has parallels to MPI programming, especially in relation to MPI functions *scatter* and *reduce*, it offers automatic parallelism, as well as data-locality and fault-tolerance (discussed previously) [22].

2 Applications of Hadoop in Bioinformatics

The emergence of platforms that utilise Hadoop infrastructure that we have discussed, namely Hadoop and Spark, have not been overlooked by researchers in different areas of bioinformatics. A number of projects within the Apache Hadoop ecosystem find useful application in bioinformatics [1]. These include the data-warehousing framework Hive [23] which has an SQL type query language, the high level data-flow language Pig [24] which compiles scripts into sequences of MapReduce steps for execution on Hadoop, the machine-learning and clustering facilities offered by Mahout [25], and HBase a distributed scalable database [26]. All of these projects utilise Hadoop's cluster infrastructure and distributed file system and therefore gain from the scalability and fault-tolerance inherent in their design, as discussed earlier.

2.1 MapReduce

In terms of software applications MapReduce has been employed for a variety of problems in processing biological and sequencing datasets. Some notable projects in the area of sequence alignment are, Cloudburst [27] and CloudAligner [28], which are both based on the RMAP alignment algorithm [29], and CloudBlast [30] which is based on the BLAST algorithm [31]. It is noteworthy that MapReduce can be especially suited for, for example the construction of a de Bruijn graph for *de novo* genome assembly. For example, Contrail is able to build adjacency lists for all the *k-mers* in the genomic sequence reads and then uses distributed aggregation functions such as *reduce* to compress simple chains of length N in $O(log(N))$ rounds using a randomized parallel list ranking algorithm [32].

There are also tools implemented in MapReduce for the analysis of assembled sequencing data, for instance Crossbow [33] is designed for SNP (Single Nucleotide Polymorphism) detection. It uses the Bowtie [34] and the SNP caller SOAPsnp [35]. Differential expression (using RNA-Seq) can be measured using the Myrna software pipeline [36] - pipelines are data-flows comprising of sequential steps in which bioinformatics software are applied to the data [37].

Additionally, a number of programming libraries that facilitate the manipulation and processing of sequencing data file formats such as SAM Sequence Alignment Map and BAM (Binary Alignment Map) have arisen such as the Java based libraries Genome Analysis Toolkit (GATK) [38] developed by the Broad Institute and Hadoop-BAM [39] as well as the Scala based SparkSeq [40] (discussed below). The GATK provides functions for data management in the form of data access patterns, namely the low level implementation is separated from higher level functions, and also provides functions for analysis calculations. The Broad Institute have also developed a Workflow Definition Language (WDL) for use in data analysis pipelines (discussed in the next section). It is a high-level language that is designed to be human readable and writable. It allows researchers to describe analysis tasks, daisy-chain tasks into workflows, and utilise advanced features such as parallelization [41]. WDL was developed out of the necessity for standardisation amongst a number of different pipeline solutions, thereby providing a universal standard. In order to execute analysis pipelines written in WDL, an execution engine is necessary. Cromwell is such an engine, also designed by the developers of WDL, to run on many platforms (Locally, HPC or Google - support for other platforms such as Microsoft Azure and AWS is forthcoming) and can scale elastically to workflow needs [42]. Furthermore, other notable workflow languages that aim to facilitate reproducible workflows are coming to the fore and have started to gain traction, such as CWL (Common Workflow Language) [43] and Nextflow [44]. CWL, targeted at Bioinformatics, medical imaging, Chemistry, Physics and Astronomy is supported by Cromwell, Galaxy and Taverna. Nextflow, which is based on the Groovy language, allows for abstraction of the underlying platform by virtue of an execution engine which can run and parallelise workflow jobs on platforms such as LSF, SLURM and AWS [44].

The provision of pipeline development specifically for the Hadoop platform is also available. For instance, SparkSeq is a library for building genomic analysis pipelines using Scala on Apache Spark. Whilst Scala is supported on the Spark platform it lacks the same user base in bioinformatics as it enjoys amongst the data analytics and machine learning communities. Given the vast amounts of sequencing data being produced [45, 46], the purpose of these tools is to exploit the scalability which the Hadoop and Spark platforms offer, and this offsets any difficulty in

developing or re-writing such applications. However, the development of universal standards, such as WDL offers researchers a means of utilising tools developed for such platforms in a more user-friendly way.

2.2 Spark

A number of applications have been developed specifically for the Apache Spark platform. Guo *et al.* have carried out an extensive review of Bioinformatics applications that use Spark [47]. Nothaft *et al.* [48] have developed an example genomics pipeline in Spark on HDFS. They have demonstrated performance gains on the platform running on Amazon EC2 in comparison to the equivalent pipeline run using existing tools. Their pipeline comprises alignment of reads, pre-processing of reads for QC, variant calling and filtration of positive variant calls. ADAM pipeline steps were evaluated against other well-known tools used for these steps such as GATK, SAMtools and Sambamba. They achieved superlinear speedup when increasing from 8x to 16x nodes, and near linear speedup when scaling from 32x to 128x nodes. This corresponded to a 63% reduction in execution cost on Amazon EC2.

VariantSpark, developed by [49], is a machine learning analysis framework for genomic data implemented using Apache Spark. It operates on large feature vectors and large numbers of samples for genetic classification and can perform real-time analysis. Currently VariantSpark utilises k-means clustering, the next version, however, will implement a random forest algorithm).

The Genome Analysis Toolkit (GATK), used extensively in Variant discovery, provides some of its tools implemented in Spark - previous versions provided Java libraries for Hadoop. Whilst the Spark implemented GATK tools can be run on a single Spark node, performance gains are made when the Spark platform is a cluster of compute nodes which are horizontally scaled up.

While there are a variety of different Genomics applications that use Spark, there are at present very few applications in Structural Bioinformatics. MTTF [50] is a new compact binary data format for PDB data and a single Hadoop sequence file of the data set is available for analysis by Hadoop and Spark. Other papers, such as [51] and [52] point to future plans to transition to Spark, but at present there are no other examples of Spark being applied in Structural Bioinfomatics.

3 Applications in Structural Bioinformatics

The Protein Data Bank (PDB) is an archive of data describing the 3D shapes of proteins, nucleic acids, and molecular-complex assemblies derived from x-ray crystallographic, Nuclear Magnetic Resonance spectroscopy (NMR) and electron microscopy techniques [53, 7]. It also serves as a portal for structural genomics [54].

There are also a number of applications for Structural Bioinformatics implemented using MapReduce on Hadoop, specifically to carry out high-throughput analyses of such data sets which will be discussed. Schematically these are shown in figure 1. Whilst the focus of this section will be on systems developed for the Hadoop platform, for purposes of comparison, it will also refer to similar systems implemented on other platforms. We have provided a table summarising all the applications discussed below in the Supplementary Information.

3.1 Molecular docking

Molecular docking typically involves simulating the electrostatic interactions between a ligand (often a potential drug molecule) and a target protein [58, 59]. It is used to score ligands on their affinity to the target, usually for the purposes of drug development - a process that is complex, time-consuming and expensive [60, 61].

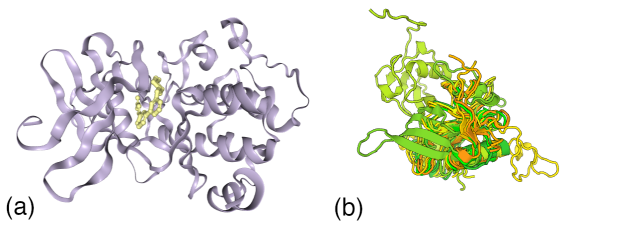

Fig. 1. A summary of Structural Bioinformatics applications discussed here. Image (a) represents docking of a ligand (in yellow) into a protein structure (lilac) where one computationally determines the optimal spatial configuration of ligand and protein. In this case the structure is an experimentally determined protein-ligand complex 2XJ1 [55], the kinase PIM-1 complexed with 3'-Guanylic Acid; in docking the structures for the protein and ligand are separately determined experimentally and an optimal complex must be computed. Image (b) represents structural alignment where a set of protein structures are structurally overlapped with each other to identify deep sequence relationships that may not be identified using sequence alignment methods. In this case the structure 1SAR (chain A) [56] a Ribonuclease, is overlapped with a variety of other structures using the Dali server [57]. Clustering of protein-ligand complexes represents a hybrid of these where different possible protein-ligand complexes are clustered by their structural similarity.

3.1.1 Docking of protein-ligand complexes on Hadoop

A number of molecular docking applications have been implemented to exploit the Hadoop platform. For example, [62] at the Oak Ridge National Laboratory in the US, have utilised AutoDock4 on a private 68 node Hadoop cluster to perform the docking of 2,637 compounds from the Directory of Useful Decoys (DUD) database [63], against the Human estrogen receptor alpha agonist protein (PDB entry 1L2I, [64]). They used the DUD database because it contains ligands that bind to the target, as well as chemically similar ligands that do not (decoys). This allowed them to test the reproducibility of the docking experiments - they found that the results of running AutoDock on Hadoop were consistent with the experiments of [63], specifically that the percentage of known binding ligands correlated with the percentage ranked in the DUD database. In their configuration they used 10 mappers per node on the 57 nodes of their cluster that were dedicated to run MapReduce Tasks giving 570 mappers running in parallel. This resulted in a 450x speed-up of AutoDock in performing the docking task on Hadoop as compared with utilising AutoDock itself to manage the parallelisation. Furthermore, they report that 95.59% of CPU time is used by AutoDock, and, therefore, there is less than a 5% overhead in running AutoDock in a Hadoop map process, and that, as the tails seen in the graphs of the CPU load were steep, this indicates that job initialisation and termination are not resource intensive.

As a comparison, [65] conducted the same experiment using the DUD database with MPI and a multi-threading parallel scheme at an extremely large scale (15,408 CPUs). They found that VinaLC scaled very well up to with an overhead of only 3.94%. 17 million flexible compound docking calculations were completed on 15,408 CPUs within 24 hours. 70% of the targets in the DUD data set were recovered using VinaLC.

Another system for protein-ligand binding on Hadoop, developed by [66] is a scalable docking service called Cloud-PLBS (Cloud Protein Ligand Binding Service), which utilises the SMAP docking tool [67]. Their system employs an additional virtualisation system, whereby the Hadoop slave nodes run on virtualised machines and are instantiated depending on the input job requirements. In terms of benchmarking performance, they compared stand-alone, sequential processing of protein-ligand pairs using SMAP, with parallel execution of SMAP within a Hadoop map function - specifically 2, 4, 6 and 8 mappers. They observed that in docking 40 protein-ligand pairs, reduction in execution time using Hadoop vs. stand-alone for 2, 4, 6 and 8 mappers was 33.92%, 56.97%, 70.21%, 77.65%, respectively.

They also tested the fault-tolerance of Hadoop in running their protein-ligand pair docking system by simulating task failures in 50% of the map steps by removing node(s) from service. They observed that the docking jobs still completed. As discussed previously, fault-tolerant distributed computation is a feature of Hadoop based applications, and this resilience in the execution of tasks is important, because the likelihood of a node failing increases with the scaling-up of a cluster. Fault-tolerance is also desirable in web-based services such as Cloud-PLBS, which serve to automate computational jobs and present the results to the user, without requiring third-party intervention to rectify failed jobs. However, it should be noted that no reference to source code for their system is provided in their paper, and the Cloud-PLBS service at *http://bioinfo.cs.pu.edu.tw/cloud-PLBS/index.html* is no longer available.

3.1.2 Clustering of protein-ligand complexes

One of the challenges in the field of molecular docking studies arises from the requirement to search the conformational space of protein-ligand complexes generated across docking experiments. This is necessary to select the most likely conformations of protein-ligand complexes, and, therefore, the putative ligands (potential drug molecules) which partake in these interactions. In such experiments large numbers of protein-ligand complexes are generated, docked, and scored [68], and it is, therefore, necessary to select a subset of putative ligands based on significant protein-ligand interactions.

Estrada *et al.* observe that selecting the native conformation, based on the assumption that the lowest energetically scored conformation (as computed by an energy function) represents the native binding of the ligand and protein, is not reliable, even in larger sets of conformations. This is often due to non-native ligand-protein complexes generating falsely low energy scores. They point out that, whilst hierarchical clustering techniques are a logical way of addressing this problem - as the lowest scoring, most densely populated clusters overlap with native conformation - most clustering algorithms are computationally expensive, and scale poorly with large datasets. They implemented a system using MapReduce on Hadoop to address this issue. They used two datasets, of size 5 TB (3,872 million ligand conformations) and 1 TB (768 million ligand conformations), generated from the Docking at Home volunteer grid computing project (Docking@Home) [69], which utilised CHARMM [70] .

The examples discussed in the section 3.1.1 did not fully implement their solutions in MapReduce. This would have involved implementing (or re-implementing) algorithms using MapReduce, but instead exploited Hadoop's *map* step to encapsulate and execute external applications. The method discussed in [68] however, is fully implemented in MapReduce.

A map step is used which geometrically reduces the conformational space. This is stored in an Octree data structure [71], together with a unique identifier (an Octkey) used for traversal. This is achieved by projecting the x, y, z components of the conformations onto a 2D plane, and calculating their gradients (for x, y, and z) which are then encoded into a single point in the Octree. A reduce step is used to aggregate conformations in the Octree. Further MapReduce operations are then used to traverse the Octree using the Octkey identifier.

In order to compare the accuracy of their Hadoop-based Octree method (for selecting native conformations from the ensemble of complexes) against other approaches, namely Hierarchical clustering and Minimum Energy selection, they docked 100,000 protein-ligand complexes each for HIV, Trypsin, and P38alpha. They obtained 80%, 75% and 25% accuracy for Hadoop based Octree, Hierarchical Clustering and Minimum Energy methods, respectively.

They also examined the accuracy of selecting native conformations from the cross-docking data in the Docking@Home datasets, whereby each conformation of the ligand in the set of complexes is docked with each conformation of protein. In doing so, they compared their Octree method with the Energy Minimum approach, and observed 43.8% and 5.8% accuracy, respectively.

In testing the scalability in processing the 5 TB dataset which, as discussed contains 3,872 million ligand conformations, they used a Hadoop cluster where each node possesses 32 cores (4x Octacore AMD Opteron 2.4 GHz), and up to a maximum of 32 nodes were available. The range of the scaling used was 1 node of 32 cores (to process 121 million conformations) to 32 nodes with a total of 1024 cores (to process the full dataset) and analysed 1, 2, 4, 8, 16, and 32 nodes - in all cases, the number of ligand conformations processed per core was 3.8 million.

It is not stated in their paper how many map processes were running per core, but it is assumed that it is 1 map step per core. Whilst they demonstrated their method was amenable to scaling, they observed an appreciable decrease in parallel efficiency with the increase in cores, from 99.1% down to 43.8% for 64 cores (2 nodes) and 1024 (32 nodes), respectively. This appears to be due to the increased overhead in communications between the processes as the number of processes increases (communications to computation ratio).

A similar application [72] was developed on the Hadoop platform that partitions the results of molecular dynamics simulations. The trajectories of atom positions, velocities and energies as a function of time are clustered, as large datasets. This method yields important information about the most probable conformations of proteins in ensembles. Their system employs the GROMOS algorithm [73], which is not inherently parallel, by implementing it as a series of map and reduce functions so as to utilise Hadoop. They tested their parallelised MapReduce implementation of the GROMOS algorithm on a Hadoop cluster comprising of 1 master and 3 slave nodes, each comprising two hexa-core Xeon E5645 CPUs 32 GB of RAM and 2 TB of disk space. They observed up to 10 and 7 times speed-up (over using sequential GROMOS) of the first and second phases, and final two phases of their algorithm, respectively.

A docking application also relevant to our discussion, although not implemented in Hadoop, has been developed by [74] using a scientific workflow management tool, SciCumulus deployed on AWS. Their system employed molecular docking, using AutoDock4 and Vina, on their platform to explore both drug discovery and scalability. Their drug discovery objective was the identification of putative drug ligands that bind to Cysteine Proteases of Protozoan genomes utilising 10,000 protein-ligand complexes. This aims to facilitate the development of drugs for the Neglected Tropical Diseases (NTDs). In investigating the scaling-up of the computational task, they utilised up to 32 heterogeneous nodes (containing varied numbers of cores) to include a total of 128 Amazon AWS EC2 cores. They observed an almost linear relationship between number of nodes and speed, but this plateaued as they approached the maximum of 32 nodes, suggesting less benefit in scaling beyond this. They point out that this is likely due to more complicated load balancing in the set of heterogeneous nodes. The result of their docking experiments using the Cysteine Protease-ligand complexes identified 287 and 355 putative ligands for AutoDock4 and Vina, respectively. It is important to note, however, that these potential drug molecules have, on average, RMSDs greater than $2-3$ Å (Angstroms) which is the maximum accepted value for a useful result.

3.2 Structural Alignment

The alignment of proteins by structure, as opposed to by sequence, is a computational technique used to identify homologous polymer structures within proteins that may be conserved between proteins. The technique facilitates the study of the structural and evolutionary relationships of proteins with low sequence similarity [75, 76]. A variety of of algorithms have been developed to perform structural alignment of proteins, such

as, for example, STRUCTAL (Structural Analysis Algorithm), DALI (Distant Alignment) [77], CE (Combinatorial Extension) [78], VAST (Vector Alignment Search Tool) [75], and FATCAT (Flexible structure Alignment by Chaining Aligned fragment pairs allowing Twists) [79], SSAP (Sequential Structure Alignment Program) [80], and MUSTANG (Multiple Sequence Alignment Algorithm) [81]. The technique has been applied to the study of protein binding sites, and solvent exposed surfaces (these effect the energetics of protein-ligand conformations) [82, 83, 84].

Structural alignment algorithms are usually computationally complex, [85] present a method which runs at best in approximately Polynomial time, but they also point out that approximations are often used. There is also a need to apply such techniques at scale. For example, Hadoop has been employed by [84] who implemented structural alignment for binding site prediction. Using test sets of 200 and 48 ligand-protein complexes, they were able to achieve 93% and 98% accuracy, respectively, and were able to improve the efficiency of the experiment by exploiting parallelisation. A service for structurally aligning pairs of proteins has also been implemented for the Hadoop platform by the developers of Cloud-PLBS [86] (discussed in the previous section 3.1). It utilises the same distributed architecture as Cloud-PLBS, that is, individual Hadoop nodes running within their own VM, and each VM running a *map* and *reduce* process. As with Cloud-PBS, a web-interface is used to enable the user to provide input of two PDB files by their PDB-ID. To perform the structural alignment they state their system uses the DALI and VAST algorithms. Whilst the authors do detail the basis of RMSD (Root Mean Square Deviation) in structural alignment algorithms, and discuss refinement methods in their paper, they claim to implement these algorithms in MapReduce. As previously noted with Cloud-PLBS, there is no source code available, and the corresponding web service is unavailable. It is highly likely that, given the complexity of these algorithms and that there are already implementations available, the same method used in Cloud-PLBS - execution of an external program within a map step - is employed.

A similar bioinformatics SaaS (Software-as-a-Service) for structural alignment of proteins, has been developed for the Microsoft Azure platform - Cloud4Psi developed by [87]. Their service utilises three newer algorithms that are implemented in the BioJava project, and which are derived from CE (jCE), and FATCAT (jFATCAT-rigid and jFATCAT-flexible) [88]. They tested their system on a subset of 1,000 PDB structures for both scalability and reproducibility. For scalability, two different scaling methods were compared: *horizontal-scaling* (i.e. by adding more nodes to the system) and *vertical-scaling* (i.e. by using nodes with more CPU cores). They found that, whilst both scaling methods increased the n-fold speed-up for each of the three algorithms, both suffered a decrease in the performance gains - for horizontal scaling, this was found to be the result of increased disk I/O due to multiple nodes utilising a shared VHD (Virtual Hard-Disk), and for vertical scaling this was due to an increase in processes on the same node (due to higher specification of each node), resulting in increased CPU utilisation. As the horizontal scaling method suffered less from this effect, it was the method chosen. For reproducibility of results, they found that each of the three algorithms produced the same results independent of cluster configuration and scaling used.

3.3 Other Structural Bioinformatics applications using Hadoop

Large-scale processing of molecular data is desirable in both applications. Such techniques facilitate the *in-silico* study of vast arrays of molecular compounds and macromolecular structures that are available from large molecular databases, which are also increasing in size and diversity [89, 90, 91, 92, 53, 7]. A number of scalable Structural Bioinformatics methods have been provided by the bioinformatics Group at UCL (University College London) as web-based services through their Protein Analysis Workbench [93]. These are accessible via SOAP (Simple Object Access Protocol), and XML-RPC (Extensible Markup Language-Remote Procedure Call) protocols. Importantly, the most commonly used methods have also been deployed as Java packages specifically for the Hadoop platform. This includes PSIPRED for protein structure prediction [94], GenTHREADER for protein fold recognition method using genomic sequences [95] and Disopred [96] for predicting protein disorder. However, other Workbench tools such as BioSerf, MEMSAT, DomPred, MetSite and FFPred appear not to have been deployed as Java packages for Hadoop and the Hadoop package is not available in their most up to date github repository `https://github.com/psipred` suggesting that it is not being updated.

4 Conclusions

This purpose of this review is to give an insight into the impact that Hadoop and the MapReduce formalism has in Structural Bioinformatics. This is a apposite moment to consider this as there have been a number of different applications of Hadoop in the field and Hadoop (and the wide variety of different applications that have been built on it) has become more stable and accessible. Spark is very likely to supplant Hadoop. Researchers working in Structural Bioinformatics have not yet begun to apply Spark to their own research and the same challenges (determining the trade off between entirely rewriting a legacy application to make use of the paradigm or using the formalism to launch the legacy application, understanding the algebraic formalism based on key-value pairs, benchmarking with traditional approaches such as batch schedulers and optimising configuration) apply there as well.

As noted previously the adoption of Hadoop is not a trivial step, particularly for a Structural Bioinformatics lab that already has extensive experience in using traditional batch schedulers running on a local cluster. Rewriting stable code to make them use the MapReduce formalism at the lowest level would require substantial effort though using MapReduce to call executables requires much less effort and can still make use of the fault-tolerance and data locality features of Hadoop.

Questions have been raised about whether the focus on scalability is masking other significant overheads for such systems and a metric has been developed to examine this [97]. The impact of this for Bioinformatics applications has not been explored in the literature and is an interesting and open question in this area.

This article has focussed on

- determining the breadth of cases where Hadoop has been used,
- how well it scales and
- how dependent the installation of Hadoop is on its configuration (which is indicative of the difficulty one would have in installing Hadoop).

In the first instance we see a range of traditional high-throughput applications in Structural Bioinformatics (e.g. docking and structural alignment) where Hadoop has been employed.

With respect to scaling, the publications reviewed here indicate that some adjustment of parameters have been made, but these largely focused on how the applications scale with the number of nodes. They show that performance is linear though the gains in performance tend to reduce as the systems are scaled up. In molecular docking, [68] observed a fall in parallel efficiency of 55.3% (99.1% - 43.8%) when scaling from 2 nodes to 32 nodes. In structural alignment, [84] observed that performance degraded slightly after 8 mappers was increased to 30. Furthermore, this trend has also been observed on the MS Azure platform we have discussed for comparison - in scaling Cloud4Psi, also a structural alignment application [87], observed that horizontal scaling resulted in performance degradation

as a result of an increase in nodes sharing a virtual disk, and that vertical scaling resulted in performance degradation as a result of increased CPU utilisation (due to more processes running per node).

The platform, and method of distribution, also dictate performance and scalability. In observing two identical docking operations on the DUD database, one using a 1,088 core Hadoop cluster [62], and the other using 15,408 cores with a mixed parallel MPI implementation of AutoDock [65], the Hadoop cluster took 69 hours, and the MPI implementation completed within 24 hours. Whilst this is certainly a result of the number of cores, the MPI system scaled better, with only very slight degradation in performance after 6,000 CPU cores. Although this comparison involved the same docking operation and dataset using different platforms, currently, there are no comparisons of performance between Hadoop and batch-schedulers on the same cluster apparatus in the literature.

The comparison of vertical and horizontal scaling in Cloud4Psi indicates significant change in performance, so configuration *is* important. As noted with the Cloud4Psi example, there can be a significant variation depending on configuration. Performance gains across applications, therefore, are dependent on configuration.

As Hadoop platforms stabilise, the significant advantage of its employment is of using a platform where the computation is expressed explicitly in terms of an algebra. This makes building workflows easier by allowing the developer to concentrate on the calculation, rather than the process such as WDL [41]. Nonetheless, there is a significant gap in take up as such systems remain difficult to deploy.

The Hadoop ecosystem is rapidly evolving and such gaps will reduce over time. Spark will improve the situation further. On the other hand, it will also require regular release upgrades. These can be potentially be difficult to deploy, and often add new components to the ecosystem which increase the potential to introduce bugs that may affect different areas of the system.

To address this, organisations such as Cloudera and Hortonworks [98, 99] provide supported Hadoop-stack releases, and cloud-service providers such as Amazon offer managed-services, for example Elastic Map Reduce (EMR) [100]. Implementing systems on the Hadoop platform, as discussed, also requires specialist programming knowledge of MapReduce, and if the relevant Hadoop cluster is maintained in-house, specialist skills in maintaining an Hadoop cluster are also required. For this reason, managed services are often utilised by enterprise companies because such systems have been deployed and tested by technical experts and therefore mitigates risks, and dispenses the need to employ or train in-house skilled personnel to maintain a Hadoop cluster.

In summary we see a complex pattern of results suggesting that while Hadoop is promising as a platform for Structural Bioinformatics, the benefits are uneven. The use of Spark should produce better and more consistent performance and we look forward to future work in this regard.

Key Points

- There exist a number of implementations using Hadoop of high-throughput analyses, e.g. ligand-protein docking and structural alignment.
- In general these deployments are based on using standard executables which are called from the MapReduce formalism rather than rewrites of the software.
- Scalability is complex and configuration-dependent though there is some evidence that MPI implementations scale better than Hadoop.
- The ease to which Hadoop can be used for Structural Bioinformatics will improve through further innovations such as Spark, easy to deploy instances of cloud platforms and the Workflow Languages such as WDL, CWL, and Nextflow.

Author Description:
Dr. Jamie Alnasir completed his PhD from University of London. His research interests are Next Generation sequencing, Distributed Computing, High Performance Computing, Computational Biology, and Bioinformatics. He currently works in Scientific Computing at the Institute of Cancer Research, London.
Dr. Hugh P. Shanahan is a Reader at the department of Computer Science, Royal Holloway, University of London. His research interests are Next Generation Sequencing, Metagenomics and High Throughput Computing.

Acknowledgements

The research in this article was made possible from support from the Department of Computer Science, Royal Holloway, University of London.

References

[1]Taylor, R. C. (2010) An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC bioinformatics,* **11**(Suppl 12), S1.

[2]Messerschmitt, D. G., Szyperski, C., et al. (2005) Software ecosystem: understanding an indispensable technology and industry. *MIT Press Books,* **1**.

[3]Nugent, T. and Jones, D. T. (September, 2012) Membrane protein structural bioinformatics. *Journal of Structural Biology,* **179**(3), 327–337.

[4]Petrey, D. and Honig, B. (2014) Structural Bioinformatics of the Interactome. *Annual Review of Biophysics,* **43**(1), 193–210.

[5]Sudha, G., Nussinov, R., and Srinivasan, N. a. (November, 2014) An overview of recent advances in structural bioinformatics of protein?protein interactions and a guide to their principles. *Progress in Biophysics and Molecular Biology,* **116**(2), 141–150.

[6]Choong, Y. S., Tye, G. J., and Lim, T. S. (October, 2013) Minireview: Applied Structural Bioinformatics in Pro teomics. *The Protein Journal,* **32**(7), 505–511.

[7]Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N., and Bourne, P. E. (jan, 2000) The Protein Data Bank.. *Nucleic acids research,* **28**(1), 235–42.

[8]Shanahan, J. G. and Dai, L. (2015) Large Scale Distributed Data Science Using Apache Spark. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* New York, NY, USA: ACM KDD '15 pp. 2323–2324.

[9]Wagener, J. L. (August, 1996) High performance fortran. *Computer Standards & Interfaces,* **18**(4), 371–377.

[10]Shanahan, H. P., Owen, A. M., and Harrison, A. P. (July, 2014) Bioinformatics on the Cloud Computing Platform Azure. *PLOS ONE,* **9**(7), e102642.

[11]Guo, Z., Fox, G., and Zhou, M. (2012) Investigation of data locality in mapreduce. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)* IEEE Computer Society pp. 419–426.

[12]Apache Software Foundation HDFS architecture documentation. `http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html` (2016) [Online; accessed 10-Jan-2017].

[13]Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al. (2013) Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing* ACM p. 5.

[14]Gunarathne, T., Wu, T.-L., Qiu, J., and Fox, G. (2010) MapReduce in the Clouds for Science. In *Cloud Computing Technology and Science*

(CloudCom), 2010 IEEE Second International Conference on IEEE pp. 565–572.

[15]Shanahan, J. G. and Dai, L. (2015) Large scale distributed data science using apache spark. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* ACM pp. 2323–2324.

[16]Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012) Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* USENIX Association pp. 2–2.

[17]Apache Software Foundation Spark 2.6 documentation. `http://spark.apache.org/docs/latest/programming-guide.html` (2014) [Online; accessed 26-Jan-2017].

[18]Apache Software Foundation SparkR documentation. `https://spark.apache.org/docs/latest/sparkr.html` (2015) [Online; accessed 21-October-2017].

[19]Kaplan, J. and Nelson, M. (January, 1993) A Comparison of Queueing, Cluster and Distributed Computing Systems. *NASA Technical Memorandum: 109025,*.

[20]Chun, B. N. and Culler, D. E. (2002) User-centric performance analysis of market-based cluster batch schedulers. In *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on* IEEE pp. 30–30.

[21]Message Passing Interface Forum MPI: a message passing interface standard. (1993).

[22]Jin, H. and Sun, X.-H. (2013) Performance comparison under failures of MPI and MapReduce: An analytical approach. *Future Generation Computer Systems,* **29**(7), 1808–1815.

[23]Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., and Murthy, R. (2009) Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment,* **2**(2), 1626–1629.

[24]Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. (2008) Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* ACM pp. 1099–1110.

[25]Lyubimov, D. and Palumbo, A. (2016) Apache Mahout: Beyond MapReduce, CreateSpace Independent Publishing Platform, .

[26]George, L. (2011) HBase: The Definitive Guide: Random Access to Your Planet-Size Data, " O'Reilly Media, Inc.", .

[27]Schatz, M. C. (2009) CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics,* **25**(11), 1363–1369.

[28]Nguyen, T., Shi, W., and Ruden, D. (2011) CloudAligner: A fast and full-featured MapReduce based tool for sequence mapping. *BMC research notes,* **4**(1), 171.

[29]Smith, A. D., Xuan, Z., and Zhang, M. Q. (2008) Using quality scores and longer reads improves accuracy of Solexa read mapping. *BMC bioinformatics,* **9**(1), 128.

[30]Matsunaga, A., Tsugawa, M., and Fortes, J. (2008) Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on* IEEE pp. 222–229.

[31]Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990) Basic local alignment search tool. *Journal of molecular biology,* **215**(3), 403–410.

[32]Schatz, M. C., Sommer, D., Kelley, D., and Pop, M. (2010) De Novo assembly of large genomes using cloud computing. In *Proceedings of the Cold Spring Harbor Biology of Genomes Conference*.

[33]Langmead, B., Schatz, M. C., Lin, J., Pop, M., and Salzberg, S. L. (2009) Searching for SNPs with cloud computing. *Genome biology,* **10**(11), R134.

[34]Trapnell, C., Pachter, L., and Salzberg, S. L. (2009) TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics,* **25**(9), 1105–1111.

[35]Li, R., Li, Y., Fang, X., Yang, H., Wang, J., Kristiansen, K., and Wang, J. (2009) SNP detection for massively parallel whole-genome resequencing. *Genome research,* **19**(6), 1124–1132.

[36]Langmead, B., Hansen, K. D., and Leek, J. T. (2010) Cloud-scale RNA-sequencing differential expression analysis with Myrna. *Genome biology,* **11**(8), R83.

[37]Leipzig, J. (2016) A review of bioinformatic pipeline frameworks. *Briefings in bioinformatics,* p. bbw020.

[38]McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., et al. (2010) The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome research,* **20**(9), 1297–1303.

[39]Niemenmaa, M., Kallio, A., Schumacher, A., Klemelä, P., Korpelainen, E., and Heljanko, K. (2012) Hadoop-BAM: directly manipulating next generation sequencing data in the cloud. *Bioinformatics,* **28**(6), 876–877.

[40]Wiewiórka, M. S., Messina, A., Pacholewska, A., Maffioletti, S., Gawrysiak, P., and Okoniewski, M. J. (2014) SparkSeq: fast, scalable, cloud-ready tool for the interactive genomic data analysis with nucleotide precision. *Bioinformatics,* p. btu343.

[41]Broad Institute WDL (Workflow Definition Language) specification and documentation. `https://software.broadinstitute.org/wdl/documentation/spec` (2016) [Online; accessed 21-Nov-2017].

[42]Broad Institute Cromwell, execution engine for WDL - Documentation via Forum. `https://gatkforums.broadinstitute.org/gatk/discussion/7349/the-art-of-the-pipeline-introducing-cromwell-wdl` (2016) [Online; accessed 21-Nov-2017].

[43]Amstutz, P., Andeer, R., Chapman, B., Chilton, J., Crusoe, M. R., Valls Guimera, R., Carrasco Hernandez, G., Ivkovic, S., Kartashov, A., Kern, J., et al. (2016) Common Workflow Language, Draft 3.

[44]Di Tommaso, P., Chatzou, M., Floden, E. W., Barja, P. P., Palumbo, E., and Notredame, C. (2017) Nextflow enables reproducible computational workflows. *Nature biotechnology,* **35**(4), 316.

[45]Stephens, Z. D., Lee, S. Y., Faghri, F., Campbell, R. H., Zhai, C., Efron, M. J., Iyer, R., Schatz, M. C., Sinha, S., and Robinson, G. E. (07, 2015) Big Data: Astronomical or Genomical?. *PLoS Biol,* **13**(7), 1–11.

[46]Ward, R. M., Schmieder, R., Highnam, G., and Mittelman, D. (2013) Big Data challenges and opportunities in high-throughput sequencing. *Systems Biomedicine,* **1**(1), 29–34.

[47]Guo, R., Zhao, Y., Zou, Q., Fang, X., and Peng, S. (August, 2018) Bioinformatics applications on Apache Spark. *GigaScience,* **7**(8).

[48]Nothaft, F. A., Massie, M., Danford, Timothy an d Zhang, Z., Laserson, U., Yeksigian, C., Kottalam, J., Ahuja, A., Hammerbacher, J., Linderman, M., Franklin, M. J., Joseph, A. D., and Patterson, D. A. (2015) Rethinking Data-Intensive Science Using Scalable Ana lytics Systems. In *Proceedings of the 2015 ACM SIGMOD International Con ference on Management of Data* New York, NY, USA: ACM SIGMOD '15 pp. 631–646.

[49]O?Brien, A. R., Saunders, N. F. W., Guo, Y., Buske, F. A., Scott, R. J., and Bauer, D. C. (December, 2015) VariantSpark: population scale clustering of genotype information. *BMC Genomics,* **16**(1), 1052.

[50]Bradley, A. R., Rose, A. S., Pavelka, A., Valasatava, Y., Duarte, J. M., Prli?, A., and Rose, P. W. (June, 2017) MMTF?An efficient file format for the transmission, visualization, and analysis of macromolecular structures. *PLOS Computational Biology,* **13**(6), e1005575.

[51]Umbrin, H. and Latif, S. (March, 2018) A survey on Protein Protein Interactions (PPI) methods, databases, challenges and future directions. In *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)* pp. 1–6.

[52]Mrozek, D., Suwała, M., and Małysiak-Mrozek, B. (July, 2018) High-throughput and scalable protein function identification with Hadoop and Map-only pattern of the MapReduce processing model. *Knowledge and Information Systems,*.

[53]Abola, E. E., Sussman, J. L., Prilusky, J., and Manning, N. O. (jan, 1997) Protein Data Bank archives of three-dimensional macromolecular structures.. *Methods in enzymology,* **277**, 556–71.

[54]Kouranov, A., Xie, L., de la Cruz, J., Chen, L., Westbrook, J., Bourne, P. E., and Berman, H. M. (2006) The RCSB PDB information portal for structural genomics. *Nucleic acids research,* **34**(suppl 1), D302–D305.

[55]Schulz, M. N., Fanghänel, J., Schäfer, M., Badock, V., Briem, H., Boemer, U., Nguyen, D., Husemann, M., and Hillig, R. C. (March, 2011) A crystallographic fragment screen identifies cinnamic acid d erivatives as starting points for potent Pim-1 inhibitors. *Acta Crystallographica Section D: Biological Crystallography,* **67**(3), 156–166.

[56]Sevcik, J., Dodson, E. J., and Dodson, G. G. (April, 1991) Determination and restrained least-squares refinement of the structures of ribonuclease Sa and its complex with 3'-guanylic acid at 1.8 Å resolution. *Acta Crystallographica Section B,* **47**(2), 240–253.

[57]Holm, L. and Rosenström, P. (July, 2010) Dali server: conservation mapping in 3D. *Nucleic Acids Research,* **38**(suppl_2), W545–W549.

[58]Morris, G. M. and Lim-Wilby, M. (2008) Molecular docking. *Molecular modeling of proteins,* pp. 365–382.

[59]Meng, X.-Y., Zhang, H.-X., Mezei, M., and Cui, M. (2011) Molecular docking: a powerful approach for structure-based drug discovery. *Current computer-aided drug design,* **7**(2), 146–157.

[60]Moses, H., Dorsey, E. R., Matheson, D. H., and Thier, S. O. (2005) Financial anatomy of biomedical research. *Jama,* **294**(11), 1333–1342.

[61]Rawlins, M. D. (2004) Cutting the cost of drug development?. *Nature reviews Drug discovery,* **3**(4), 360–364.

[62]Ellingson, S. R. and Baudry, J. (2011) High-throughput virtual molecular docking: Hadoop implementation of AutoDock4 on a private cloud. In *Proceedings of the second international workshop on Emerging computational methods for the life sciences* ACM pp. 33–38.

[63]Huang, N., Shoichet, B. K., and Irwin, J. J. (2006) Benchmarking sets for molecular docking. *Journal of medicinal chemistry,* **49**(23), 6789–6801.

[64]Shiau, A. K., Katzenellenbogen, B. S., Barstad, D., Agard, D. A., Greene, G. L., Radek, J. T., Katzenellenbogen, J. A., Nettles, K. W., and Meyers, M. J. (2002) Structural characterization of a subtype-selective ligand reveals a novel mode of estrogen receptor antagonism. *Nature Structural and Molecular Biology,* **9**(5), 359.

[65]Zhang, X., Wong, S. E., and Lightstone, F. C. (2013) Message passing interface and multithreading hybrid for parallel molecular docking of large databases on petascale high performance computing machines. *Journal of computational chemistry,* **34**(11), 915–927.

[66]Hung, C.-L. and Hua, G.-J. (2013) Cloud computing for protein-ligand binding site comparison. *BioMed research international,* **2013**.

[67]Xie, L. and Bourne, P. E. (2007) A robust and efficient algorithm for the shape description of protein structures and its application in predicting ligand binding sites. *BMC bioinformatics,* **8**(4), S9.

[68]Estrada, T., Zhang, B., Cicotti, P., Armen, R. S., and Taufer, M. (2012) A scalable and accurate method for classifying protein–ligand binding geometries using a MapReduce approach. *Computers in biology and medicine,* **42**(7), 758–771.

[69]Estrada, T., Armen, R., and Taufer, M. (2010) Automatic selection of near-native protein-ligand conformations using a hierarchical clustering and volunteer computing. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology* ACM pp. 204–213.

[70]Brooks, B. R., Bruccoleri, R. E., Olafson, B. D., States, D. J., Swaminathan, S. a., and Karplus, M. (1983) CHARMM: a program for macromolecular energy, minimization, and dynamics calculations. *Journal of computational chemistry,* **4**(2), 187–217.

[71]Samet, H. (1988) An overview of quadtrees, octrees, and related hierarchical data structures. *NATO ASI Series,* **40**, 51–68.

[72]Paschina, G., Roverelli, L., D'Ł™Agostino, D., Chiappori, F., and Merelli, I. (2015) Clustering Protein Structures with Hadoop. In *International Meeting on Computational Intelligence Methods for Bioinformatics and Biostatistics* Springer pp. 141–153.

[73]Scott, W. R., Hünenberger, P. H., Tironi, I. G., Mark, A. E., Billeter, S. R., Fennen, J., Torda, A. E., Huber, T., Krüger, P., and van Gunsteren, W. F. (1999) The GROMOS biomolecular simulation program package. *The Journal of Physical Chemistry A,* **103**(19), 3596–3607.

[74]Ocaña, K., Benza, S., de Oliveira, D., Dias, J., and Mattoso, M. (2014) Exploring large scale receptor-ligand pairs in molecular docking workflows in HPC clouds. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International* IEEE pp. 536–545.

[75]Gibrat, J.-F., Madej, T., and Bryant, S. H. (1996) Surprising similarities in structure comparison. *Current opinion in structural biology,* **6**(3), 377–385.

[76]Orengo, C. A., Michie, A., Jones, S., Jones, D. T., Swindells, M., and Thornton, J. M. (1997) CATH–a hierarchic classification of protein domain structures. *Structure,* **5**(8), 1093–1109.

[77]Holm, L. and Sander, C. (1998) Touring protein fold space with Dali/FSSP. *Nucleic acids research,* **26**(1), 316–319.

[78]Shindyalov, I. N. and Bourne, P. E. (1998) Protein structure alignment by incremental combinatorial extension (CE) of the optimal path.. *Protein engineering,* **11**(9), 739–747.

[79]Ye, Y. and Godzik, A. (2003) Flexible structure alignment by chaining aligned fragment pairs allowing twists. *Bioinformatics,* **19**(suppl_2), ii246–ii255.

[80]Orengo, C. A. and Taylor, W. R. (1996) [36] SSAP: sequential structure alignment program for protein structure comparison. *Methods in enzymology,* **266**, 617–635.

[81]Konagurthu, A. S., Whisstock, J. C., Stuckey, P. J., and Lesk, A. M. (2006) MUSTANG: a multiple structural alignment algorithm. *Proteins: Structure, Function, and Bioinformatics,* **64**(3), 559–574.

[82]Ma, B., Elkayam, T., Wolfson, H., and Nussinov, R. (2003) Protein–protein interactions: structurally conserved residues distinguish between binding sites and exposed protein surfaces. *Proceedings of the National Academy of Sciences,* **100**(10), 5772–5777.

[83]Konc, J. and Janežič, D. (2010) ProBiS algorithm for detection of structurally similar protein binding sites by local structural alignment. *Bioinformatics,* **26**(9), 1160–1168.

[84]Liu, G., Liu, M., Chen, D., Chen, L., Zhu, J., Zhou, B., and Gao, J. (2016) Predicting Protein Ligand Binding Sites with Structure Alignment Method on Hadoop. *Current Proteomics,* **13**(2), 113–121.

[85]Kolodny, R. and Linial, N. (2004) Approximate protein structural alignment in polynomial time. *Proceedings of the National Academy of Sciences of the United States of America,* **101**(33), 12201–12206.

[86]Hung, C.-L. and Lin, Y.-L. (2013) Implementation of a parallel protein structure alignment service on cloud. *International journal of genomics,* **2013**.

[87]Mrozek, D., Małysiak-Mrozek, B., and Kłapciński, A. (2014) Cloud4Psi: cloud computing for 3D protein structure similarity searching. *Bioinformatics,* **30**(19), 2822–2825.

[88]Prlić, A., Yates, A., Bliven, S. E., Rose, P. W., Jacobsen, J., Troshin, P. V., Chapman, M., Gao, J., Koh, C. H., Foisy, S., et al. (2012) BioJava: an open-source framework for bioinformatics in 2012. *Bioinformatics,* **28**(20), 2693–2695.

[89]Degtyarenko, K., De Matos, P., Ennis, M., Hastings, J., Zbinden, M., McNaught, A., Alcántara, R., Darsow, M., Guedj, M., and Ashburner, M. (2007) ChEBI: a database and ontology for chemical entities of biological interest. *Nucleic acids research,* **36**(suppl_1), D344–D350.

[90]Pence, H. E. and Williams, A. ChemSpider: an online chemical information resource. (2010).

[91]Allen, F. H. (2002) The Cambridge Structural Database: a quarter of a million crystal structures and rising. *Acta Crystallographica Section B: Structural Science,* **58**(3), 380–388.

[92]Wang, Y., Xiao, J., Suzek, T. O., Zhang, J., Wang, J., and Bryant, S. H. (2009) PubChem: a public information system for analyzing bioactivities of small molecules. *Nucleic acids research,* **37**(suppl_2), W623–W633.

[93]Buchan, D. W., Minneci, F., Nugent, T. C., Bryson, K., and Jones, D. T. (2013) Scalable web services for the PSIPRED Protein Analysis Workbench. *Nucleic acids research,* **41**(W1), W349–W357.

[94]McGuffin, L. J., Bryson, K., and Jones, D. T. (2000) The PSIPRED protein structure prediction server. *Bioinformatics,* **16**(4), 404–405.

[95]Jones, D. T. (1999) GenTHREADER: an efficient and reliable protein fold recognition method for genomic sequences. *Journal of molecular biology,* **287**(4), 797–815.

[96]Ward, J. J., McGuffin, L. J., Bryson, K., Buxton, B. F., and Jones, D. T. (2004) The DISOPRED server for the prediction of protein disorder. *Bioinformatics,* **20**(13), 2138–2139.

[97]Mcsherry, F., Isard, M., and Murray, D. G. (2015) Scalability! but at what cost. In *In 15th Workshop on Hot Topics in Operating Systems (HotOS XV), Kartause Ittingen* USENIX Association.

[98]Cloudera About Cloudera. `https://www.cloudera.com/more/about.html` (2016) [Online; accessed 01-February-2018].

[99]Hortonworks About Hortonworks. `https://hortonworks.com/about-us/` (2016) [Online; accessed 01-February-2018].

[100]Amazon Amazon EMR (Elastic MapReduce). `https://aws.amazon.com/emr/` (2016) [Online; accessed 14-April-2017].