Final Year Project Report

Full Unit - Final Report

Structural Bioinformatics Framework using a MapReduce formalism

Vinay Kakkar

A report submitted in part fulfilment of the degree of

MSc in Computer Science with a Year In Industry

Supervisor: Hugh P. Shanahan

Department of Computer Science
Royal Holloway, University of London

March 25, 2023

Abstract

Structural bioinformatics is a rapidly growing field that aims to understand biological processes at the molecular level. In this dissertation, we present a novel framework for structural bioinformatics using a MapReduce formalism. Our framework allows for efficient processing of large-scale structural data by distributing computation across a cluster of computers. We demonstrate the effectiveness of our approach through benchmarking and comparisons with current implemented solutions. Our framework not only provides faster computation but also offers improved scalability and fault tolerance, making it a valuable tool for large-scale structural bioinformatics analyses. Furthermore, we highlight the potential of our framework for facilitating collaboration and data sharing among researchers, which is crucial for advancing our understanding of complex biological systems. Overall, our proposed framework presents a significant step forward in the field of structural bioinformatics, enabling the efficient and scalable analysis of complex structural data.

1 Introduction

Structural bioinformatics plays a critical role in the development of new drugs, as well as in understanding the molecular basis of biological processes. It involves the analysis and interpretation of large amounts of complex data, including the three-dimensional structures of proteins, DNA, and other macromolecules. As the size and complexity of these data sets continue to increase, the need for efficient and scalable computational tools for their analysis and processing becomes ever more pressing.

One promising approach to addressing this challenge is the use of MapReduce, a programming model for large-scale data processing in distributed computing environments. MapReduce enables the parallel processing of large data sets across multiple nodes in a cluster, which can significantly improve the speed and efficiency of data analysis tasks.

In this report, we present a structural bioinformatics framework that utilizes a MapReduce formalism for the efficient analysis of large-scale structural data. Our framework is based on a distributed computing architecture that allows for the parallel processing of structural data, including protein structures, protein-protein interactions, and other molecular structures. We describe the key components of our framework, including the data preprocessing, map, and reduce phases, as well as the parallel algorithms and data structures used to implement these phases.

We demonstrate the effectiveness of our framework through a series of experiments on real-world structural data sets, showing that our approach can significantly reduce the time and resources required for complex bioinformatics tasks. Our results highlight the potential of MapReduce-based approaches for accelerating the analysis of large-scale structural data in the field of bioinformatics and facilitating breakthroughs in drug development and biological research.

2 Aims and Objectives

2.1 Aim provided in the Project Description

"The aim of this project is to be provide a framework where large numbers of protein structures can be analysed using a user-provided executable in the MapReduce formalism."

Aim

Analysing protein structures can provide insights into their properties and interactions with other molecules. However, analysing large numbers of protein structures can be computationally intensive, requiring significant computational resources and time. Thus, the aim is to provide a framework that allows for the analysis of large numbers of protein structures using a MapReduce formalism.

MapReduce: is a programming model for processing and generating large data sets. Allowing for parallel processing of data across multiple nodes in a cluster, making it well-suited for analysing large datasets.

More specifically the aim is to develop a framework that can take a user-provided executable and apply it to a large number of protein structures in a parallel, distributed manner using MapReduce. This will enable researchers to perform complex analyses on large datasets of protein structures more efficiently, saving time and computational resources. Essentially enabling researchers to analyse large numbers of protein structures more quickly and effectively than before.

Motivation

The analysis of protein structures is a critical area of research, providing insights into their properties and interactions with other molecules. However, as mentioned before the computational demands of analysing large datasets of protein structures can be a significant obstacle, requiring substantial time and resources. To address this challenge, this project aims to develop a framework that utilizes the MapReduce programming model to enable researchers to efficiently analyse large numbers of protein structures using a user-provided executable. By doing so, this framework has the potential to revolutionize the field, allowing researchers to perform complex analyses on massive datasets in a parallel, distributed manner. This will ultimately save valuable time and computational resources, ultimately enabling researchers to make more rapid progress in their investigations of protein structures and potentially leading to new discoveries in the field.

2.2 Objectives

To achieve the aim of the project, the following objectives have been formulated:

1. Develop a software framework that supports the MapReduce formalism and can process large numbers of protein structures.
2. Implement a distributed computing system using MapReduce to parallelize protein structure analysis across multiple computing nodes.
3. Optimize the software framework to reduce the processing time required for protein structure analysis.
4. Design an interface that allows users to manipulate the PDBs that are being passed into the executable for protein structure analysis.
5. Ensure the software framework is scalable and can handle increasingly large datasets.
6. Validate the software framework by testing it with a variety of protein structure analysis tools and evaluating its performance in comparison to other available tools.
7. Ensure the software framework can be easily updated to keep pace with advancements in protein structure analysis techniques and computing technology.
8. Provide documentation and user support to enable researchers to use the software framework effectively

MapReduce Framework for Protein Analysis

> Develop a software framework that suppots the Mapreduce formalism and can process large numbers of protein structures.

With the increase in the amount of data being generated in Structural Bioinformatics, there is a need for frameworks that can process large amounts of data efficiently. The MapReduce programming model is a powerful tool for processing large datasets across a distributed computing cluster. It allows for parallel processing of data by dividing it into smaller chunks, which are then processed in parallel across multiple computing nodes. Thus, developing a software framework that supports MapReduce allows for the efficient processing of large numbers of protein structures, as it can handle the distributed processing of data across a cluster of computers.

Parallelized Protein Analysis with MapReduce

> Implement a distributed computing system using MapReduce to parallelize protein structure analysis across multiple computing nodes.

Protein structure analysis is a computationally intensive task that can take a long time to complete on a single computer. By implementing a distributed computing system using MapReduce, the analysis can be parallelized across multiple computing nodes, significantly reducing the time required to process the data. This allows for faster analysis of protein structures, which is essential in research fields such as drug discovery.

Optimization for Faster Protein Analysis

> Optimize the software framework to reduce the processing time required for protein structure analysis.

Even with a distributed computing system, the processing time required for protein structure analysis can be significant. Thus, the software framework must be optimized to reduce the processing time required for protein structure analysis. This can be achieved by improving the efficiency of the algorithms used in the analysis, minimizing the amount of data transferred between nodes, and optimizing the hardware used in the computing cluster. Optimizing the software framework ensures that protein structure analysis can be done as quickly and efficiently as possible.

User Interface for Protein Analysis

> Design an interface that allows users to manipulate the pdbs that are being passed into the executable for protein structure analysis.

As the framework revolves around using and manipulatting PDB file. An interface for manipulating the input data is necessary to provide users with the flexibility to customize their protein structure analysis. This interface should be intuitive and easy to use, allowing researchers to select the protein structures they want to analyze and adjust the analysis parameters as needed. Providing this interface ensures that researchers can tailor the analysis to their specific needs and allows for greater flexibility in the types of analysis that can be performed.

Validation via Performance Comparison

> Validate the software framework by testing it with a variety of protein structure analysis tools and evaluating its performance in comparison to other available tools.

As a part of the aim is to provide the ability to perform quicker and efficiently protein data analysis. Validation of the software framework is necessary to ensure that it is working correctly and producing accurate results. Testing the framework with a variety of protein structure analysis tools allows for a comprehensive evaluation of its performance. Comparing the performance of the software framework with other available tools helps to identify areas for improvement and ensures that the framework is competitive with other solutions in the field. This validation ensures that the software framework is a reliable and effective tool for protein structure analysis.

Scalable Data Handling for Protein Analysis

> Ensure the software framework is scalable and can handle increasingly large datasets.

As the size of protein structure datasets continues to increase, it is essential that the software framework can scale to handle this growth without a significant decrease in performance. The system should be designed to handle datasets of various sizes and be able to scale up or down based on the demands of the analysis. This ensures that the software framework can handle the ever-increasing amounts of data generated by modern research techniques.

Upgradable Framework for Advanced Analysis

> Ensure the software framework can be easily updated to keep pace with advancements in protein structure analysis techniques and computing technology.

As the field of protein structure analysis continues to evolve, it is important that the software framework can keep pace with the latest advancements. This means that the framework should be designed to be modular and extensible, allowing for new analysis techniques to be easily integrated. It should also be designed to take advantage of the latest computing technologies, high-performance computing clusters. This ensures that the software framework can remain relevant and effective as new research techniques are developed.

User Support for Protein Analysis Software

> Provide documentation and user support to enable researchers to use the software framework effectively.

To ensure that researchers can use the software framework effectively, comprehensive documentation and user support are essential. This includes detailed documentation on how to install and configure the software, as well as guides on how to use the various functions provided by the framework. Providing documentation and user support helps to ensure that researchers can use the framework for their research purposeses and obtain accurate results from their protein structure analyses.

Further Objective: Intergrate API for PDB access

> Intergrate a PDB file download and search from the RCSB API for efficient data access.

The PDB is a crucial resource for structural bioinformatics, and researchers need to be able to access and search for specific PDB files relevant to their research. By Intergrating an API that allows users to download and search for PDB files, researchers will have an easier time accessing and analysing the relevant structural data. This will increase the efficiency of the research process and allow researchers to make more informed decisions. Additionally, making this data more accessible to researchers can also promote collaboration and sharing of data, furthering scientific progress in the field of structural bioinformatics.

3 Protien Structures, PDB Bank and Big Data Frameworks

3.1 Protein Structures

Introduction

Amino acids are molecules that when combined forms proteins. All of the 20 amino acids, see table 1 have in common a central carbon atom which is attached to a hydrogen atom, an amino group, and a carboxyl group [BT98].

Proteins are responsible for catalysing most of the chemical reactions in cells. They can function as enzymes catalysing a wide variety of reactions important for life and thus also important for the structure of living systems such as proteins involved in the cytoskeleton. The size of protein can vary [Zve08].

Definition 3.1 (Catalysing) *Catalysing is to make a chemical reaction happen or happen more quickly by acting as a catalyst.*

Definition 3.2 (Cytoskeleton) *A dynamic network of interlinking protein filaments present in the cytoplasm of all cells [Zve08].*

Primary, Secondary, Tertiary and Quaternary Structure

Please refer to 1 for a visual representation.

The **primary structure** of a peptide or protein is the linear sequence of its amino acids. It is read and written from the amino-terminal to the carboxyl-terminal end. [SFB04].

The **secondary structure** refers to the local arrangement of a peptide chain. Where several common secondary structures have been identified in proteins [SFB04].

Tertiary structure is a three-dimensional structure of a protein the formation is built up of bonds and interactions that serve to change the shape of the overall protein [God22].

The **quaternary structure** of a protein is built-up of several protein chains/subunits. Each of the subunits has its primary, secondary, and tertiary structure [OR15].

Considering Protein structure on several different levels

The fold of the protein plays part in determining the way the protein will function, and also whether it will function correctly. As there are Protein structures on different levels we need to consider the analysis of protein structure by experimental techniques such as X-ray crystallography, nuclear magnetic resonance, and RNAseq which show that proteins adopt distinct structural elements [Zve08].

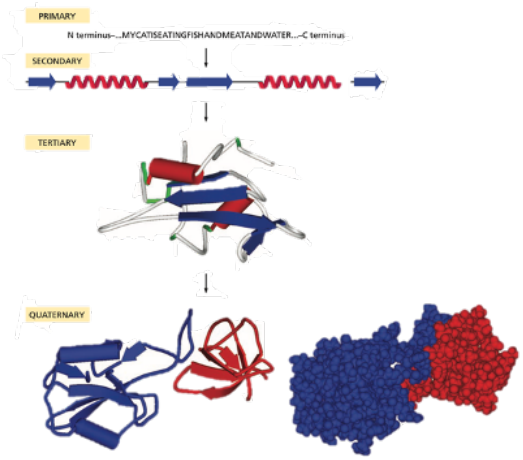


Figure 1: From the sequence alone, the primary structure to secondary structure, to tertiary structure(3D), to finally quaternary structure found when several tertiary structures form a multisubunit complex [Zve08].

Amino Acids

Sequence of amino acids 1 will build up the linear protein chain [Zve08]. Amino acids are different from each other due to their side chains and due to this the functional properties of various different proteins are different [Zve08]. You can see the amino acids grouped here 1.

Amino acid	Three-letter code	One-letter code
Glycine	Gly	G
Alanine	Ala	A
Valine	Val	V
Leucine	Leu	L
Isoleucine	Ile	I
Proline	Pro	P
Phenylalanine	Phe	F
Methionine	Met	M
Tryptophan	Trp	W
Cysteine	Cys	C
Asparagine	Asn	N
Glutamine	Gln	Q
Serine	Ser	S
Threonine	Thr	T
Tyrosine	Tyr	Y
Aspartic acid	Asp	D
Glutamic acid	Glu	E
Histidine	His	H
Lysine	Lys	K
Arginine	Arg	R

Table 1: The 20 amino acids. The amino acid name, the three-letter code, and the one-letter code are given. The Amino acids are split up into Nonpolar, Polar, Acidic and Basic respectfully

3.2 Large Scale Experssion

Gene expression begins when genes are transcribed into messenger RNAs, which are then translated to produce proteins.

Total gene expression in cultured cells or a tissue sample can be detected in three main ways:

1. DNA microarray technology.
2. Two-dimensional Gel electrophoresis or Chromatography.
3. RNAseq

Both DNA microarray technology and Two-dimensional Gel electrophoresis, produce enormous amounts of raw data [Zve08] due to this, many proteins currently evade high-resolution structure determination.

Structural mass spectrometry

Structural mass spectrometry is a powerful approach used to determine the 3D structure of biological protiens it has nearly an unlimited size constraint and speed. Although the data provided by mass spectrometry is vague for full high-resolution structure elucidation, structural mass spectrometry can be used to examine the size, solvent accessibility, and topography of proteins [LLV18] [LZG20].

We can have computational methods that aid experimental technique intending to elucidate protein structures [SL20] [LWL$^+$20]. Software packages can be used to combine data with advanced structure sampling and scoring techniques. Computational tools for protein structure modeling, include the Rosetta software suite [LWL$^+$20] [ALFJ$^+$17], I-TASSER [YYR$^+$15], Phyre2 [KMY$^+$15], Integrative Modeling Platform [RLW$^+$12], HADDOCK [DBB03], and MODELLER [EWMR$^+$06] [BL22].

Large Scale Gene Expression

Genome DNA microarray experiments produce large amount of data can be computationaly heavy on where methods can yield alternative conclusions from inceasing the computational effort.

The goal of these experiments is to determine biological or functional meaning from the lists of genes, either by:

1. Identify critical genes that are responsible for a biological effect.
2. Find patterns within the genes that point to an underlying biological process.

[Zve08]

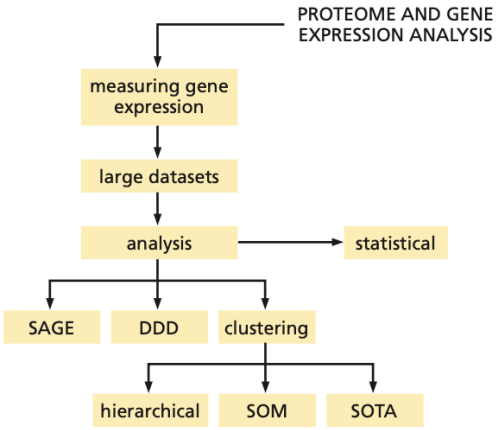
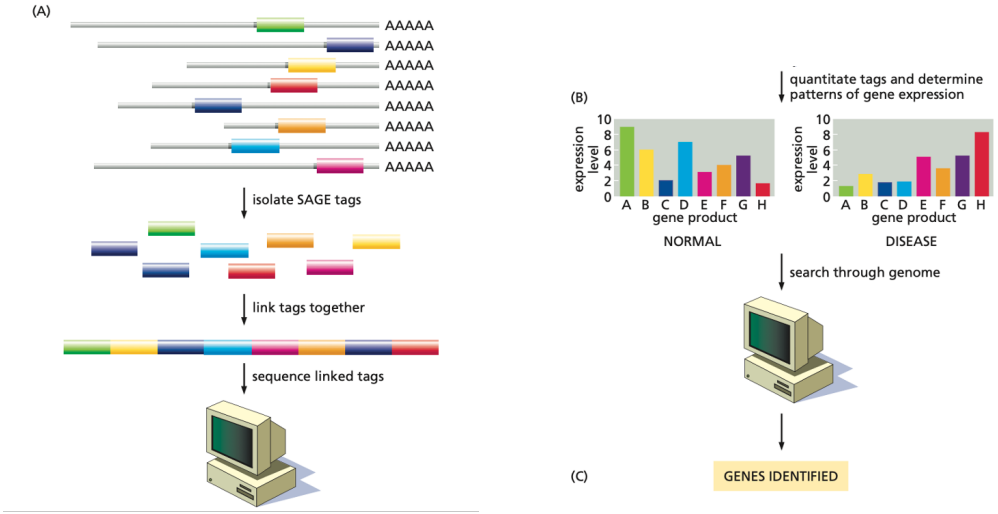


Figure 2: Describing Common experimental aspects of gene expression and of the analysis of the resulting data [Zve08].

Serial analysis of gene expression

Serial analysis of gene expression is the alternative compared to microarrays when trying to investigate patterns of gene expression.



(a) (b)

Figure 3: *An outline of the SAGE method for comparing levels of gene expression. (A) Short sequence tags. The sequence tags are isolated and are linked together to produce long DNA molecules that can be cloned and sequenced. (B) Once sequenced, each tag can be calculated, resulting in a value that gives the expression level of the corresponding transcript [Zve08].*

A short sequence contains enough information to uniquely identify a gene. The sequence tags from the total cellular RNA can be linked together to form long DNA molecules. The total number of times a particular tag is observed the concatemers approximates the expression level of the corresponding gene. The data produced by SAGE include a list of the tags with their corresponding counts, providing a digital output of cellular gene expression. Which allows the user to specify which organ is to be investigated. Libraries consisting of gene lists organized by the various types of tissues or cell lines are provided for further choice. The output from SAGE provides the SAGE tag, the UniGene ID, the gene description, and color and letter-coded differences in expression levels [Zve08].

Clustered gene expression data

Clustered pattern data obtained from gene expression microarrays/genome bioinformatics can be used as a tool to identify new transcription factors or other cell-regulatory proteins.

The clustered genes/proteins can be analyzed. Leading to a vast collection of data from many gene/protein expression experiments being available on the Web [Zve08].

Large Scale Protein Expression

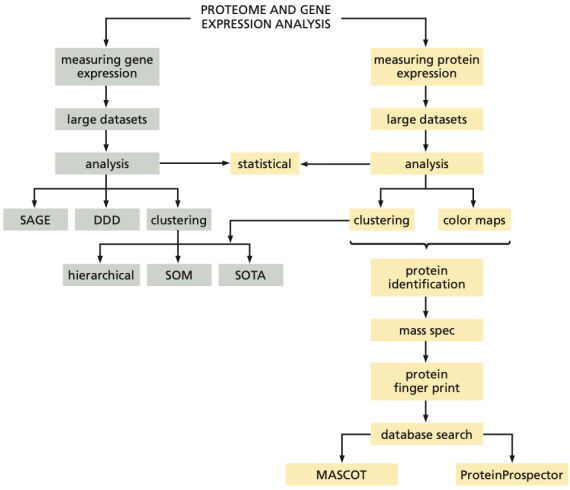


Figure 4: Describing some experimental aspects of protein expression and of the analysis of the resulting data. [Zve08].

For functional protein, mRNAs need to be translated, whilst the protein products can change which influence their function. For this reason we can measure and anlayse different proteins.

There is more proteins than there are genes in a genome. Transcripts can be spliced in various ways to give different mRNAs, providing different protein products, from the same gene. However, proteins that can be modified after translation giving more different protien products.

Protien expressions can vary in a organism depending on the origin and it will also differ between the separate stages of an organism's life cycle and under different environmental conditions [Zve08].

Definition 3.3 (proteome) *The proteome refers to all the proteins that make up an organism at a specific point in time and under specific conditions.*

RNAseq

The transcriptome is important for revealing the molecular constituents of cells and tissues, interpreting the functional aspects of the genome, also for understanding development and disease [WGS09].

Many methods deduce and quantify the transcriptome, including hybridization or sequence-based approaches. For example, hybridization-based approaches involve incubating fluorescently labeled cDNA with microarrays or commercial high-density oligo microarrays [WGS09].

However, these methods have several limitations, such as:

- Dreliance upon existing knowledge about genome sequence.
- Limited dynamic range of detection owing to both background.
- High background levels owing to cross-hybridization [OM06] [RRG07].
- saturation of signals.

Definition 3.4 (transcriptome) *The transcriptome is the complete set of transcripts in a cell, and their quantity, for a specific developmental stage or physiological condition.*

Sequence-based approaches directly determine the cDNA sequence such as Tag-based methods which include SAGE, CAGE [KKN$^+$06], MPSS [RBL$^+$02].

Each approach is high throughput and can provide precise, gene expression levels. However, a significant portion of the short tags can not be uniquely mapped to the reference genome [WGS09].

RNA-Seq RNA sequencing has clear advantages over existing approaches it uses deep sequencing technologies where a population of RNA is converted to a library of cDNA fragments with adaptors attached to one or both ends. Each molecule is then sequenced in a high-throughput manner to obtain short sequences from one or both ends [WGS09].

Bioinformatic Difficulties with Predictions on Proteins

It is difficult to define the precise ends of the helices(The secondary structure of proteins is made up of a-helices and b-strands) for structures found in globular proteins that are not perfectly regular. Making it one step more difficult when trying to predict these structures [Zve08].

To Note:

- Several different types of b-sheet are found in protein structures.
- Turns, hairpins, and loops connect helices and strands.
- Any chain between two regular structures is referred to as a loop.
- Mostly a loop will contain a turn (or even several).

In antibody recognition, immunoglobulins employ loops at the edge of a b-sheet. All immunoglobulin structures with the same overall chain fold, but it is the difference at these loops that results in different results. Loops take up one of a limited number of structures called canonical forms. This type of classification is another reason why trying to predict both the structure and function of the protein is difficult [Zve08].

Definition 3.5 (Immunoglobulin) *Immunoglobulins are heterodimeric proteins composed of two heavy and two light chains. Types of white blood cells that helps the body fight infection [SC10].*

Alpha Fold

AlphaFolds' goal is to predict the 3D coordinates of all heavy atoms for a given protein using the primary amino acid sequence and aligned sequences of homologues as inputs [JEP+21].

Mutations in proteins can lead to misfolding which is often associated with disease states, for example, Alzheimer's and Parkinson's which is one of the challenges for alphaFold [Fel].

The output is a file containing the 3D coordinates for every non-hydrogen atom in the protein. whilst showing the confidence levels for every amino acid residue, providing the reliability of the predicted structure [Fel].

Bioinformatics with Alpha Fold

In July 2021, AlphaFold was developed by DeepMind and was made available to the public [TAW+21].

Where it tries to solve the issue of invariant protein structures that are under translations and rotations [BP03].

AlphaFold is trained on protein chains from the PDB using the input sequence to query databases of protein sequences to generate a multiple sequence alignment [JEP+21]. Although we still do not exactly know how a protein sequence folds and alpha fold do not help in figuring this out its impact will likely be in accelerating and improving the production of new medications [NZLJ22].

AlphaFold 2

The CASP14 was recently held which is a blind trial that critically assesses techniques for protein structure prediction [DITS22], AlphaFold2 was entered and out-performed all competitors.

Recently, RoseTTAFold was developed, trying to implement similar principles. Since then, other end-to-end structure predictors have emerged using different principles such as fast multiple sequence alignment processing in DMPFold218 and language model representations.[BPE22].

We use the root mean square deviation, to calculate the similarity between the two structures, AlphaFold models had an accuracy of 0.96 compared to 2.80 which was the second-best score. AlphaFold models also had a high level of accuracy in predicting the position of residue side chains when the protein backbone prediction was accurate [DITS22] [JEP+21].

3.3 The Protein Data Bank and the File Formats

Protein Data Bank

The Protein Data Bank was established at Brookhaven National Laboratories [BKW+77] in 1971 as an archive for biological macromolecular crystal structures [BWF+00].

Definition 3.6 (Macromolecular) *Macromolecular is any very large molecule, usually with a diameter ranging from about 100 to 10,000 angstroms*

It is an information source for data retrieved from atomic structures, crystallography, and three-dimensional structures of biomolecules, including nucleic acids and proteins [BG21].

At the time this was the first open-access digital data resource in biology which started with just seven protein structures [BBB+22b].

Various groups such as the Protein Data Bank in Europe, Protein Data Bank Japan help manage the Protein Data Bank archive. Current wwPDB members also include the ElectronMicroscopy Data Bank and the Biological Magnetic Resonance Bank [BBB+22b].

Protein Data Bank China has recently joined the wwPDB as an Associate Member with its role as wwPDBdesignated PDB Archive Keeper. Where they are responsible for weekly updates of the archive and safeguarding both digital information and a physical archive of correspondence [BBB+22a].

The management of PDB must comply with FAIR (the acronym depicts: Findable, Accessible, Interoperable, Reusable) and FACT [vdABH17] guiding principles for scientific data [WDA+16] [WSHB20].

The FAIR Guiding Principles	
To be Findable:	F1. (meta)data are assigned a globally unique and persistent identifier F2. data are described with rich metadata (defined by R1 below) F3. metadata clearly and explicitly include the identifier of the data it describes F4. (meta)data are registered or indexed in a searchable resource
To be Accessible:	A1. (meta)data are retrievable by their identifier using a standardized communications protoco A1.1 the protocol is open, free, and universally implementable A1.2 the protocol allows for an authentication and authorization procedure, where necessary A2. metadata are accessible, even when the data are no longer available
To be Interoperable:	I1. (meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation. I2. (meta)data use vocabularies that follow FAIR principles I3. (meta)data include qualified references to other (meta)data
To be Reusable:	R1. meta(data) are richly described with a plurality of accurate and relevant attributes R1.1. (meta)data are released with a clear and accessible data usage license R1.2. (meta)data are associated with detailed provenance R1.3. (meta)data meet domain-relevant community standards

Table 2: The guidlines to what builds up the FAIR principles [WDA+16]

Aims and Objectives of PDB

Enzymology, electron microscopy, computational chemistry small molecule crystallography, biochemistry, biophysics, macromolecular crystallography and nuclear magnetic resonance spectrometry all help the aims and goals of the PDB archive [BG21].

Definition 3.7 (Enzymology) *Enzymology is the branch of biochemistry aiming to understand how enzymes work*

Definition 3.8 (Electron Microscopy) *Electron microscopy is a technique for obtaining high resolution images of biological and non-biological specimens.*

PDBs provide open access to nearly 200 000 archived, validated, and biocurated experimentally determined three-dimensional structures of biological macromolecules.3D structures archived in the PDB have enabled important scientific breakthroughs by basic and applied researchers [Bur21]. Open access to PDB data without restrictions on usage has also aided structural bioinformatics in areas such as computational biology.

Recent Project

A project was undertaken to change the information management services for RCSB.org. The idea was to have developed a primary place for studying 3D biostructures by extending RCSB.org web portal functionality to support parallel delivery of more than one million CSMs publicly available from AlphaFold DB and ModelArchive together [BBB+22a].

Covid

During the COVID-19 pandemic, more than 2000 structures associated with the agent of the coronavirus disease were released and have become accessible to global users for free. The properties of these structures give us this opportunity to find out the ligand binding sites, the spatial conformation of ligands, protein-to-protein interactions, and amino acid substitutions regarding different viral proteins. Moreover, chemical, functional and energetic characteristics can also be gained to describe the potential capabilities of each molecule. These properties might aid us to determine the potential drug targets for drug design and vaccine preparation [LZD+20].

PDB Currently

As of 2022, the PDB has a vast number of 3D biostructures, eukaryotic protein structures exceeded 105 000. Bacterial protein structures were also numerous, totaling nearly 66 000. Archaeal protein structures were the least numerous totaling 5500. However the PDB coverage is decidedly limited, with mouse protein structures being most numerous at 8000 structures [BB21].

We have powerful tools developed by RCSB PDB for searching and analysis which include structure, sequence, sequence motif, structure motif, and visualization [BBB+22a].

Upon reaching the RCSB.org home page, users can query, organize, visualize, analyse, compare, and explore PDB structures and CSMs side-by-side. Searching 3D structure information can encompass PDB structures and CSMs or be limited to PDB structures only. Either PDB structures or CSMs can be excluded from the search results. The two types of structure information accessible via RCSB.org are clearly distinguished from each other. Top bar searching and data delivery for PDB structures and CSMs [BBB+22a].

Figure 5: Search options at RCSB.org include Top Bar or Basic Search; Advanced Search; and Browse Annotations [BBB+22a].

Figure 6: *Top Bar or Basic Search options available from every RCSB.org web page. Examples of searching for 3D structures using (A) simple text string insulin receptor; (B) drop down autosuggestions based on the text string insulin receptor; (C) Boolean operators to combine insulin + receptor (+ = AND); or (D) an amino acid sequence. (E) Searching RCSB.org documentation using a text string biological assembly [BBB+22a]*

Recent RCSB PDB data architecture improvements

In 2020, RCSB PDB had an upgrade of its delivery architecture [RDL+21]at RCSB.org [BBB+21].

The legacy monolithic data delivery application was changed into a distributed deployment of individual microservices, each with a single responsibility.

Data access services provide both Representational State Transfer and GraphQL API access to a data warehouse hosted in a MongoDB documentoriented database. Originally, advanced Search QueryBuilder functionality encompassed text, PDB data attributes, 3D structure, sequence, biopolymer sequence motif, and chemical similarity. Every search function is implemented as an independent service.

A separate search service is responsible for launching each search function, combining and delivering their integrated results to public programmatic search APIs. When each service has a single responsibility, we have greater flexibility in scaling the deployment of services in response to changes in user load and significant reductions in the time required to develop, test, and deploy new features. The Sequence Motif search function has been extended with a new 3D Structure Motifsearch capability [BBR20].

Recent advances in RCSB PDB data integration

RCSB PDB integrates the content of each expertly biocurated Entry with information from more than 50 external data resources.

Integrated external data needs to follow a data schema that defines the organization of the RCSB PDB data warehouse. Finally, it is available to RCSB PDB front-end services, public data access APIs, and our text search indexing service [BBB+22b].

Recent PDBx/mmCIF data standard improvements

The PDBx/mmCIF data standard is maintained by the wwPDB organization in collaboration with wwPDBPDBx/mmCIF Working Group domain experts recruited from the scientific community. The PDBx/mmCIF web resource supports browse and search access to standard terminology. The Working Group includes developers for many of the widely used structure determination software systems, who ensure that data produced by these programs comply with the PDBx/mmCIF data standard, generating complete and correct data files for PDB deposition. The wwPDB and the Working Group collaborate on developing terminologies for new and rapidly evolving methodologies such as Free Electron Laser, 3DEM, Serial Crystallography, and X-ray, whilst improving representations for existing data content. Most recently, the Working Group has focused on modernizing content descriptions for processed X-ray diffraction data, including extensions describing anisotropic diffraction limits, unmerged reflection data, and new quality metrics of anomalous diffraction data. Deposition and delivery improve our ability to assess experimental data quality, and every PDB data consumer's ability to Find and Reuse relevant PDB Entries [BBB+22b].

External Resources	
AlphaFold DB	Computed Structure Models by AlphaFold2
ATC	Anatomical Therapeutic Chemical (ATC) Classification System from World Health Organization
Binding MOAD	Binding affinities
Binding DB	Binding affinities
BMRB	BMRB-to-PDB mappings
Cambridge structural Database	Crystallographic small molecule data from the Cambridge Crystallographic Data Centre
CATH	Protein structure classification- Class, Architecture, Topology/fold, and Homologous superfamily
ChEMBL	Manually curated database of bioactive molecules with drug-like properties
CSD	Cambridge Structural Database: Validated and curated small-molecule organic and metal-organic crystal structures from the Cambridge Crystallographic Data Centre
DrugBank	Drug and drug target data
ECOD	Evolutionary Classification of Protein Domains
EMDB	3DEM density maps and associated metadata
ExplorEnz	IUBMB Enzyme nomenclature and classification
Gencode	Human and Mouse Gene annotations

Table 3: Some of the External Resources Integrated Into RCSB PDB

Future and struggles of PDB

Future

As the PDB archive has started its 52nd year, it gives open access to analyses of structures and much more to: basic and applied researchers, educators, and students spanning fundamental biology, biomedicine, bioenergy, bioengineering, and biotechnology, with key points that help many communities that use this facility. Firstly It delivers Data In and Data Out services efficiently to a user base that is now numbering many millions worldwide. Secondly, it has wwPDB partners that process, validate, and biocurate the growing number of increasingly complex PDB depositions received. Manages and safeguards the growing PDB archive in its role as wwPDB designated Archive Keeper. Thirdly it enables searching, visualization, exploration, and analysis of experimentally-determined PDB structures integrated with more than one million CSMs through its web portal. [BBB+22a].

Struggles

Even after all the advancments PDB has gone through there are still additional challenges lying ahead which include:

- Rapid growth in public-domain CSMs of individual polypeptide chains, already numbering ¿200 million at the time of writing.
- Anticipated advances in AI/ML-based prediction of structures of multi-protein complexes.
- Continued development of biomolecular structure determination methods using X-ray Free Electron Lasers, revealing the microscopic details of chemical reactions in real time.
- Growth in the number and complexity of atomic-level cryoelectrontomography structures of macromolecular machines.
- Integration of PDB structures and CSMs with complementary information coming from correlative light microscopy and related imaging methods across length scales ranging from atoms to small molecules to individual biomolecules to macromolecular assemblies to organelles to cells and ultimately tissues
- Merging of the PDB-Dev prototype archiving system for integrative methods structures with the PDB archive
- Federating other biodata resources, such as the SmallAngle Scattering Database and the Proteomics Identification Database, with the PDB, EMDB and BMRB core archives jointly managed by the wwPDB partnership

[BBB+22a].

File Formats

The PDB archive holds a few different types of file types that hold data such as atomic coordinates and other information describing proteins and other biological macromolecules. Depending on what the data is created from it can fall into a different category.

PDB Data

The main information in the PDB archive is coordinate files for biological molecules. These files list the atoms in each protein and their 3D coordinates.

These files are available in several formats:

- PDB
- mmCIF
- XML

The header section of the text summarizes the protein, citation information, and the details of the structure solution, which is then followed by the sequence and a long list of the atoms and their coordinates. It also contains the experimental observations used to determine atomic coordinates [Goo].

.pdb Files

The PB format consists of a collection of records that describe the atomic coordinates, chemical and biochemical features, and experimental details of the structure determination [WF03].

Each item of data in the PDB format is assigned to a one of PDB record types (HEADER. SOURCE. REMARK, etc.). The ATOM records the atomic coordinate data [WF03].

PDB format has been extended with new REMARK records. For example, REMARK 3 that encodes refinement information has been modified and extended for each new refinement program and program version [WF03].

The PB format uses fixed-width fields to represent data, so we have limits on the size of certain items of data. For example, we cant have more then 99,999 atoms and polymer chain can be only one character. This means some structures are devided into multiple files [WF03].

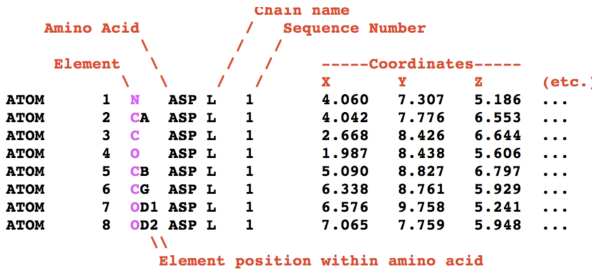


Figure 7: Showing contents of a PDB file for the Atom values [AAB+19].

.mmCIF Files

Mmcif is a dictionary-based approach to data extracted from crystallographic experiments [WF03].

It includes all the data we can find in a pdb file. Also, we have sufficient data names so that the experimental section of a structure paper can be written automatically and to facilitate the development of tools i.e. computer programs could easily access and validate mmCIF data files [WF03].

.xml

XML builds from a PDB Exchange dictionary. Although presented in very different syntaxes, the PDB Exchange and XML representations use the same logical data organization. [WIN+05].

The dictionary data block is mapped to the standard top-level XML schema element, and the data file data block is mapped to a datablock element. Category or table definitions in the Exchange dictionary are described as XML complex types. The category definition. [WIN+05].

PDB Exchange data dictionary attributes	XML schema mapping
Data block	Root level *schema element*
Category groups	
Categories	*complexTypes*
Definition	*annotation* and *documentation* elements
Examples	*annotation* and *documentation* elements
Primary keys	*attributes* of the data category
Items	*elements* of the data category
Definition	*annotation* and *documentation* elements
Examples	annotation and documentation elements
Data types	*simpleTypes*
Range restrictions and allowed values	*restrictions* within *simpleTypes* or *unions* of *simpleTypes*
Mandatory data code	Element attributes *minOccurs* and *nillable*
Parent-child relationships	*key/keyref* elements
Interdependency/exclusivity	
Units of measurement	Additional *fixed attributes*
Subcategory membership	

Figure 8: Summary of the correspondences between PDB Exchange data dictionary and XML schema metadata [WIN+05].

Visualizing Structures

PDB files can be viewed from text editors but we can also use a browsing or visualization program. RCSB PDB allows you to search and explore the information, including information on experimental methods and the chemistry and biology of the protein. Visualization programs allow to read of the PDB file and, display the protein structure generating custom pictures of it. These programs can contain analysis tools that allow you to measure distances and bond angles, and identify interesting structural features [Goo].

Reading Coordinate Files

Before exploring structures in the PDB archive we need some prior understanding of the coordinate files. For example, we can find a diverse mixture of biological molecules, small molecules, ions, and water which can get confusing we can use the names and chain IDs to help sort these out. In structures determined from crystallography, atoms are annotated with temperature factors that describe their vibration and occupancies that show if they are seen in several conformations. NMR structures often include several different models of the molecule [Goo].

Potential Challanges

There are some things to note as you could fall into some challenges when browsing through the PDB archive. Many structures, particularly those determined by crystallography, only include information about part of the functional biological assembly. One thing to note is that the PDB can aid with this. Another note is many PDB entries are missing portions of the molecule that were not observed in the experiment. These include structures that include only alpha carbon positions, structures with missing loops, structures of individual domains, or subunits from a larger molecule. In addition, most of the crystallographic structure entries do not have information on hydrogen atoms [Goo].

3.4 Hadoop spark and pyspark

What is Hadoop

Hadoop is an open-source framework for writing and running distributed applications that process large amounts of data. Key aspects making it valuable such 1. Accessible 2.Robust 3. Scalable 4.simple [Lam10].

HDFS is used in haddop which is a filessystem and a MapReduce engine. With one master node and many worker nodes. The master node provides instructions to the worker nodes and computations are performed on the worker nodes. [HRJ17].

Mapper

Input key/value pairs are mapped to a set of key/value pairs. The mapper then sorts the key-value pairs by the keys. Partitioners are mainly responsible for providing intermediate key/values to the reducers [PBN12] [HRJ17].

Reducer

Firstly, the reducer combines data having the same key from different map functions. The values having the same key are reduced to a smaller set of values and output is produced [HRJ17].

What is Spark

Apache Spark is a popular open-source platform for large-scale data processing used for iterative machine learning tasks [MBY+16].

Spark is a cluster computing system providing APIs in Java, Scala, Python (pySpark), and R, along with an optimized engine that supports general execution graphs. Moreover, Spark is efficient at iterative computations so it is suited for the development of large-scale machine learning applications [MBY+16].

Spark is a quick and general engine used for analysing large-scale data stored across a cluster of computers. Spark uses in-memory cluster computing which is its most important feature for increasing the processing speed of an application. It combines SQL streaming and complex analytics [HRJ17].

Spark Architecture

There are five core components that make Spark so powerful and easy to use. The core architecture of Spark consists of the following layers:

- Storage
- Resource management
- Engine
- Ecosystem
- APIs

[Sin22].

Storage

Before using Spark, data must be made available in order to process it. This data can reside in any kind of database. Spark offers multiple options to use different categories of data sources, to be able to process it on a large scale. Spark allows you to use traditional relational databases as well as NoSQL, such as Cassandra and MongoDB [Sin22].

Resource Management

The next layer consists of a resource manager. As Spark works on a set of machines (it also can work on a single machine with multiple cores), it is known as a Spark cluster. Typically, there is a resource manager in any cluster that efficiently handles the workload between these resources. The two most widely used resource managers are YARN and Mesos. The resource manager has two main components internally [Sin22]:

- Cluster manager
- Worker

It's kind of like master-slave architecture, in which the cluster manager acts as a master node, and the worker acts as a slave node in the cluster. The cluster manager keeps track of all information pertaining to the worker nodes and their current status. Cluster managers always maintain the following information [Sin22]:

- Status of worker node (busy/available)
- Location of worker node
- Memory of worker node
- Total CPU cores of worker node

The main role of the cluster manager is to manage the worker nodes and assign them tasks, based on the availability and capacity of the worker node. On the other hand, a worker node is only responsible for executing the task it's given by the cluster manager [Sin22].

The tasks that are given to the worker nodes are generally the individual pieces of the overall Spark application. The Spark application contains two parts [Sin22]:

- Task
- Spark driver

The task is the data processing logic that has been written in either PySpark or Spark R code. It can be as simple as taking a total frequency count of words to a very complex set of instructions on an unstructured dataset. The second component is Spark driver, the main controller of a Spark application, which consistently interacts with a cluster manager to find out which worker nodes can be used to execute the request. The role of the Spark driver is to request the cluster manager to initiate the Spark executor for every worker node [Sin22].

Engine and Ecosystem

The base of the Spark architecture is its core, which is built on top of RDDs (Resilient Distributed Datasets) and offers multiple APIs for building other libraries and ecosystems by Spark contributors. It contains two parts: the distributed computing infrastructure and the RDD programming abstraction. The default libraries in the Spark toolkit come as four different offerings [Sin22].

Spark SQL

SQL being used by most of the ETL operators across the globe makes it a logical choice to be part of Spark offerings. It allows Spark users to perform structured data processing by running SQL queries. In actuality, Spark SQL leverages the catalyst optimizer to perform the optimizations during the execution of SQL queries. Another advantage of using Spark SQL is that it can easily deal with multiple database files and storage systems such as SQL, NoSQL, Parquet, etc [Sin22].

MLlib

Training machine learning models on big datasets was starting to become a huge challenge, until Spark's MLlib (Machine Learning library) came into existence. MLlib gives you the ability to train machine learning models on huge datasets, using Spark clusters. It allows you to build in supervised, unsupervised, and recommender systems; NLP-based models; and deep learning, as well as within the Spark ML library [Sin22].

Structured Streaming

The Spark Streaming library provides the functionality to read and process real-time streaming data. The incoming data can be batch data or near real-time data from different sources. Structured Streaming is capable of ingesting real-time data from such sources as Flume, Kafka, Twitter, etc [Sin22].

Graph X

This is a library that sits on top of the Spark core and allows users to process specific types of data (graph dataframes), which consists of nodes and edges. A typical graph is used to model the relationship between the different objects involved. The nodes represent the object, and the edge between the nodes represents the relationship between them. Graph dataframes are mainly used in network analysis, and Graph X makes it possible to have distributed processing of such graph dataframes [Sin22].

Programming Language APIs

Spark is available in four languages. Because Spark is built using Scala, that becomes the native language. Apart from Scala, we can also use Python, Java, and R [Sin22].

Spark Execution

Any Spark application spins off a single driver process (that can contain multiple jobs) on the master node that then directs executor processes (that contain multiple tasks) distributed to a number of worker nodes shown 9.

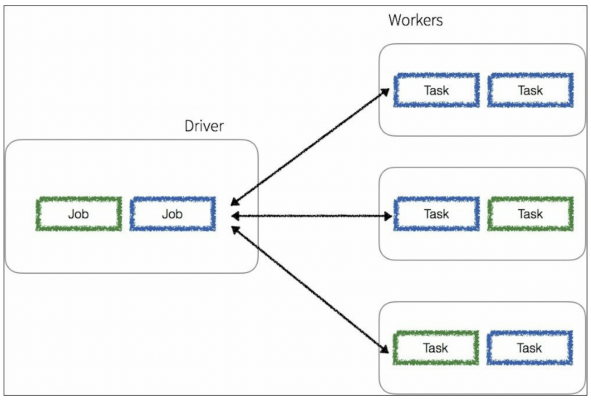


Figure 9: Think of somthing to say [DL17].

The driver process determines the number and the composition of the task processes directed to the executor nodes based on the graph generated for the given job. Note, that any worker node can execute tasks from a number of different jobs [DL17].

Spark vs Hadoop

Hadoop Map Reduce	Spark
For Applications that repeatedly reuse the same set of data, map reduce is very inefficient.	Spark uses in-memory processing, reusing it for faster computation.
MapReduce is quite faster in batch processing.	As memory size is limited, it would be quite slower in batch processing of huge data set.
Data is stored in disk for processing.	Data is stored in main memory. As it is an inmemory computation engine entire data is copied.
Difficulty in processing and modifying data in real time due to its high latency.	Used to process and modify data in real time due to its low latency.
Predominantly used to process from bygone datasets.	Predominantly used for streaming, batch processing and machine learning
For fault tolerance, MapReduce uses replication.	For fault tolerance, Spark uses RDDs.
It merges and partitions shuffle files.	It does not merges and partition shuffle files.
Primarily disk based computation.	Primarily RAM based computation.

Table 4: Showing the differences between haddop and spark [HRJ17].

Number of words	Hadoop (Sec)	Spark(Sec)
100	79	28.841
1000	91	31.185
10000	96	35.181
100000	103	36.969
1000000	116	39.569

Table 5: Comparision of Execution time for wordcount program [HRJ17].

Number of words	Hadoop (Sec)	Spark(Sec)
5	2.541	0.9030
10	3.370	1.459
50	6.420	2.840
100	9.383	3.452
200	10.100	5.749

Table 6: Comparison of Execution time for logistic regression program [HRJ17].

Summarising the results shows Spark to be quicker in both experiments. Spark also provides an API for python which will be very helpful in this project seeing its easy nature to be able to read files and work with text-based files. Therefore I have decided to work with Pyspark for this project.

Software Architectural Bottlenecks

HDFS has scheduling delays in the architecture which results in cluster nodes waiting for new tasks as the access pattern is periodic.HDFS client code, serializes computation and I/O instead of decoupling and pipelining those operations. [SRC10].

Definition 3.9 (HDFS) *The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware*

Portability Limitations

Some performance-enhancing features in the filesystem are not available such as bypassing the filesystem page cache and transferring data directly from the disk into user buffers. Thus, HDFS implementation runs less efficiently and has higher processor usage than would otherwise be necessary [SRC10].

3.5 Analysis of existing systems that solve similar tasks

There are a few systems that solve similar tasks these include:

- SparkMD: This is a Spark-based framework for molecular dynamics simulations. It uses a distributed version of the GROMACS simulation engine to enable large-scale simulations of protein systems.
- BioSpark: This is a Spark-based framework for processing and analyzing large-scale genomics and proteomics datasets. It provides a set of APIs for analyzing DNA sequences, protein structures, and other biological data.
- Pysparkling: This is a Python-based library for parallelizing computations using Spark. It can be used for a range of bioinformatics analyses, including sequence alignment, protein structure prediction, and gene expression analysis.
- SparkProt: This is a Spark-based framework for protein structure prediction. It uses a combination of machine learning algorithms and structural bioinformatics tools to predict the 3D structure of proteins from their amino acid sequences.
- DeepChem: This is a deep learning-based framework for drug discovery and development. It uses Spark to parallelize computations and can be used for a range of tasks, including protein-ligand docking, virtual screening, and compound synthesis planning.

4 Software Engineering

4.1 Objective Implementation

- Develop a software framework that suppots the Mapreduce formalism and can process large numbers of protein structures.

To implement this we need to break down the objective into two parts the first part is to achieve a framework that supports a mapreduce formalism. In order to do this i need to decide between hadoop and apache spark. Please refer to my Literature review on Hadoop spark and pyspark where spark performes quicker whilst having having an API for python.

Implementing PySpark

Install Java: PySpark requires Java to run. You can download and install the latest version of Java from the official website (https://www.oracle.com/java/technologies/javase-downloads.html).

Install Apache Spark: PySpark is built on top of Apache Spark, so you need to install Apache Spark first. You can download the latest version of Apache Spark from the official website (https://spark.apache.org/downloads.html). Choose a pre-built package for Apache Spark and select the version that matches your installed Java version. Extract the downloaded package to a location of your choice.

Install Python: PySpark requires Python 3.x to run. You can download and install the latest version of Python from the official website (https://www.python.org/downloads/).

Set environment variables: To use PySpark, you need to set environment variables for Java and Spark. As I am implementing this on MAC OS to do so we need to open the terminal and run the following commands:

```
JAVA_HOME=/Library/Java/JavaVirtualMachines/JDKversion/Contents/Home
PATH=$JAVA_HOME/bin:$PATH
SPARK_HOME=<path_to_spark_folder>
PATH=$SPARK_HOME/bin:$PATH
```

Replace 'JDK version' with the version of Java you installed and 'path to spark folder' with the path to the folder where you extracted Apache Spark.

Install PySpark: You can install PySpark using pip. Open the terminal and run the following command:

```
pip install pyspark
```

Finally Test PySpark: You can test PySpark to check if all is set uop correctly by opening the Python shell and running the following commands:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("test").getOrCreate()
df = spark.range(10)
df.show()
```

This should create a Spark session, create a DataFrame with 10 rows, and display it in the console.

The second part of the objective is to ensure that we can process large numbers of protein structures. To do this we need to optimize our Spark job and cluster configuration. There are some actions we can not implement, some actions we can consider before the code and finally some actions we can implement in to our code.

Can not implement

Cluster Configuration: I initially aimed to use a YARN implementation which allows you to dynamically share and centrally configure the same pool of cluster resources between all frameworks that run on YARN. [goy18] These are the issues i ran across:

1. Configuration: Yarn requires additional configuration beyond what is required for a standalone Spark cluster. You need to configure Yarn-specific settings such as memory allocation and container sizes, as well as Spark-specific settings such as executor and driver memory.
2. Resource Management: Yarn is responsible for managing resources for multiple applications running on a shared cluster. This can make it more difficult to manage resource allocation and ensure that each application has access to the resources it needs.
3. Monitoring: Because Yarn is managing resources for multiple applications, it can be more difficult to monitor the performance of individual applications. You need to look at both Yarn-level metrics and Spark-level metrics to get a complete picture of performance.
4. Troubleshooting: When issues arise in a Yarn-managed cluster, it can be more difficult to troubleshoot the problem. You need to determine whether the issue is related to Yarn or Spark, and then diagnose the problem accordingly.

Due to a smaller-scale workload and a simpler, more flexible, and more cost-effective option, a standalone Spark cluster may be a better fit.

Data Serialization: Serialization is the process of converting data structures into a format that can be easily transmitted over the network. By default, PySpark uses Python's pickle library for serialization, which can be slow for large datasets. You can improve serialization performance by using more efficient serialization formats like Avro or Parquet. However these are the issues i came across:

Serialization and deserialization can introduce performance overhead depending on the size and complexity of the data being serialized.

While both are widely used, they may not have the same level of tooling and support as other formats, like CSV or JSON. This can make it more difficult to work with data in these formats.

Both can add complexity to your data processing pipeline. For example, you may need to write custom serialization and deserialization logic, or configure your data processing systems to work with these formats.

For this reason i have decided to stick with the provided Python's pickle library for serialization.

Before implementing the code

Hardware Configuration: Ensure that your cluster has sufficient CPU, RAM, and disk resources to handle the workload. You can also use SSDs (Solid State Drives) for faster disk access and faster I/O operations.

To do this we need to look at the minimum requirements for running PySpark.

- A 64-bit operating system e.g., macOS, Linux, or Windows
- At least 8 GB of RAM
- A multi-core processor e.g., Intel Core i5 or i7

Whilst Implementing code

These are some functions that we can implement into our code to ensure that optimize our Spark job and cluster configuration Partitioning: Partitioning the input data can significantly improve performance, especially for large datasets. Partitioning involves splitting the data into smaller chunks and processing them in parallel. You can specify the number of partitions using the repartition function in PySpark.

Caching: Caching is an optimization technique that involves storing intermediate results in memory to avoid recomputation. Caching is particularly useful when you have iterative algorithms that reuse the same data multiple times. You can use the cache function in PySpark to cache Resilient Distributed Datasets or DataFrames.

- Implement a distributed computing system using MapReduce to parallelize protein structure analysis across multiple computing nodes.

To implement a distributed computing system using MapReduce to parallelize protein structure analysis across multiple computing nodes using PySpark, we need complete the following steps:

1. Setup a PySpark cluster with multiple worker nodes.
2. Write a Map function to parse the protein structure data and extract the relevant features.
3. Write a Reduce function to combine the results from the Map function.
4. Load the protein structure data into PySpark RDDs.
5. Apply the Map function to each protein structure RDD in parallel.
6. Apply the Reduce function to combine the results from the Map function.
7. Save the output to a distributed file system.

The following code has been taken from my lines program which is inteanded to print the number of lines each pdb file contains which complete the steps above

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StringType
import os
from pathlib import Path
import tempfile

def main(spark):
    directory = ("/PROJECT/Programs/Lines/PDBsDirectory/*")
    tempdirectory = "/PROJECT/Programs/Lines/PDBsfromRDD"

    rddkeyvalue = spark.sparkContext.wholeTextFiles(directory)

    def numberoflines(filename, filecontent):
        with tempfile.NamedTemporaryFile() as tmp:
            tmp.write(filecontent)
            os.system("wc -l "+ tmp.name)

    r = rddkeyvalue.map(lambda x: numberoflines(x[0], x[1]))
    r = r.collect()
if __name__ == '__main__':
    spark = SparkSession.builder.appName('PDB').getOrCreate()
    main(spark)
```

PySpark cluster setup

The SparkSession object is a higher-level interface to create a Spark application with SparkContext, which provides a unified entry point to interact with the Spark cluster.

We use the builder API to create a SparkSession instance and set the application name to "PDB". The getOrCreate method creates a new SparkSession instance or reuses an existing one if one already exists. This creates a SparkSession with all the necessary configurations and settings for connecting to the Spark cluster.

When we execute the code, the SparkSession instance is created and the worker nodes are launched, enabling distributed computing across the cluster.

Map function

The numberoflines function takes a folders path and the values within the protien structure and creates a temp file in which it writes the contents of the protien structure. The temp file is then used to get the number of lines with the shell command "wc -l 'tempfilename'".

Once the numberoflines function has been defined, we can apply it to each protein structure file in parallel using the Map function in Spark.

Reduce function

The code provided does not include an explicit implementation of a Reduce function. However, you can write a Reduce function using PySpark's reduce function, which would result in the same respone if needed to return the number of lines.

PySpark RDDs

The code provided, loads protein structure data by using the wholeTextFiles function to read in all the files in a directory as an RDD of key-value pairs, where the key is the filename and the value is the contents of the file.

Save to distributed file system

The code provided does not include any code to save the output to a distributed file system. However, you can use PySpark's saveAsTextFile function to save the output as text files in a distributed file system.

Issues

When implementing this objective i came across some difficulties more specifily to do with maping the pdb file. Initially i had the idea of reading a pdb file and trying to map the atoms and there coordinates into the rdd. However with some dilebrations with my supervisor most user executables designed for proten analysis work with the whole pdb file with the additional information provided above the coordinates. So for that reason the program maps the entier pdb file.

- Optimize the software framework to reduce the processing time required for protein structure analysis.

I will now present the first version of my lines programs and explain features i changed so that the framework will be more optimized for reduce the processing time required for protein structure analysis.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StringType
import os
import shutil
from pathlib import Path


def main(spark):
    directory = ("/Programs/Lines/PDBsDirectory")
    tempdirectory = "/PROJECT/Programs/Lines/PDBsfromRDD"

    for file in os.listdir(os.fsencode(directory)):
        filename = os.fsdecode(file)
        if filename.endswith(".DS_Store"):
            continue
        rdd = spark.read.text("/PDBsDirectory/"+filename)
        rdd = rdd.map(lambda x: x[0])
        if os.path.exists(tempdirectory):
            shutil.rmtree(tempdirectory)
        rdd.saveAsTextFile(tempdirectory)
        os.system("wc -l "+tempdirectory+"/part-00000")
```

```
if __name__ == '__main__':
    # This creates a local cluster
    spark = SparkSession.builder.appName('PDB').getOrCreate()
    main(spark)
```

Issues with code

We have instasiated a for loop which loops through all the files in the PDBsDirectory the code then reads each file in the PDBsDirectory directory and loads it into an RDD using spark.read.text ignoring any files that end with DSStore which is one of the outputs returned from saveAsTextFile. This operation can be time-consuming for large files because it involves reading the entire file into memory as a single string. This is not efficient for processing large files.

The code then saves the RDD to the PDBsfromRDD directory using rdd.saveAsTextFile but we need to check if this directory exists already, deleting it if so, as for the next pdb file in the PDBsDirectory we will need to cast saveAsTextFile to this which requires that folder to not exist in the first place. SaveAsTextFile function can be expensive because it involves writing the entire RDD to disk.

Finally if there is a lot of files in PDBsDirectory the the for loop will need to go through each one whilst also running shutil.rmtree which is not effeciant as there are some operations being run each iteration which we can remove.

Improvment Using wholeTextFiles()

wholeTextFiles() can be used instead of textFile() to read all files within a directory as a single RDD. wholeTextFiles() reads the files in the specified directory and returns an RDD where each element is a tuple of (filename, contents), where filename is the name of the file and contents is the entire contents of the file as a single string.

This eliminates the need to have a for loop to go through the directory as we can read each file in the directory using wholeTextFiles() and a wildcard '*' in the directory path for example directory = "/path/*" which will select all the files in that path.

As we have the name and content of the file we can remove saveAsTextFile() as we can pass in the file name into the map function thus casting the shell command on the file.

This will result in the following code:

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StringType
import os
import shutil
from pathlib import Path


def main(spark):
    start = time.time()
    directory = ("/PROJECT/Programs/Lines/PDBsDirectory/*")

    rddkeyvalue = spark.sparkContext.wholeTextFiles(directory)
```

```
    def numberoflines(k):
        os.system("wc -l "+ k[40:])

    rddkeyvalue.map(lambda x: numberoflines(x[0])).collect()


if __name__ == '__main__':
    # This creates a local cluster
    spark = SparkSession.builder.appName('PDB').getOrCreate()
    main(spark)
```

- Validate the software framework by testing it with a variety of protein structure analysis tools and evaluating its performance in comparison to other available tools.

Need help with the above bullet point not sure what to write about

- Design an interface that allows users to manipulate the pdbs that are being passed into the executable for protein structure analysis.

Need to hold out on this one as we could potentially have a ui

- Ensure the software framework is scalable and can handle increasingly large datasets.

There are several things we can consider to esnure the software framework is scalable and can handle increasingly large datasets:

1. Partitioning data
2. Distributed computing
3. Efficient algorithms
4. Testing and validation

I will be using my TMalign program to show how i have implemented each one of these:

```
def main(spark):
    directory1 = ("/PROJECT/Programs/TMalign/PDBsDirectory1/*")
    directory2 = ("/PROJECT/Programs/TMalign/PDBsDirectory2/*")

    rddkeyvalue1 = spark.sparkContext.wholeTextFiles(directory1)
    rddkeyvalue2 = spark.sparkContext.wholeTextFiles(directory2)

    rdd = rddkeyvalue1.cartesian(rddkeyvalue2)

    def TMalign(tuple1, tuple2):
        with tempfile.NamedTemporaryFile() as tmp1,
        tempfile.NamedTemporaryFile() as tmp2:
            tmp1.write(tuple1[1])
            tmp2.write(tuple2[1])
            os.system("./TMalign " + tmp1.name + " " + tmp2.name)
```

```
    rdd.map(lambda tuple: TMalign(tuple[0], tuple[1])).collect()

if __name__ == '__main__':
    # This creates a local cluster
    spark = SparkSession.builder.appName('PDB').getOrCreate()
    main(spark)
```

Partitioning data

The cartesian() operation results in an RDD with a number of partitions equal to the number of partitions in the two input RDDs multiplied together. By default, Spark uses hash partitioning for cartesian() operation, which hashes the keys of the input RDDs and distributes them evenly across partitions to ensure a roughly equal workload per partition.

Distributed computing

As we are using MapReduce, this can help with scalability as the data is being broken down and performed in parrellel. This can be seen with the map step where each node will be given a pair of pdb files which need tmalign run on. Overall the processing workload is spread out across multiple machines, allowing for faster processing and the ability to handle larger datasets. Additionally, if one machine fails or experiences an error, the rest of the system can continue running without interruption.

Efficient algorithms

Some issues with the a earlier version of the tmalign program:

```
def main(spark):
    directory1 = ("/PROJECT/Programs/TMalign/PDBsDirectory1")
    directory2 = ("/PROJECT/Programs/TMalign/PDBsDirectory2")
    tempdirectory1 = ("/PROJECT/Programs/TMalign/PDBsfromRDD1")
    tempdirectory2 = ("/PROJECT/Programs/TMalign/PDBsfromRDD2")

    for file in os.listdir(os.fsencode(directory1)):
        filename = os.fsdecode(file)
        if filename.endswith(".DS_Store"):
            continue
        rdd = spark.read.text(filename).rdd.map(lambda x: x[0])
        if os.path.exists(tempdirectory1):
            shutil.rmtree(tempdirectory1)
        rdd.saveAsTextFile(tempdirectory1)

        for file in os.listdir(os.fsencode(directory2)):
            filename = os.fsdecode(file)
            if filename.endswith(".DS_Store"):
                continue
            rdd = spark.read.text(filename).rdd.map(lambda x: x[0])
            if os.path.exists(tempdirectory2):
                shutil.rmtree(tempdirectory2)
            rdd.saveAsTextFile(tempdirectory2)
```

```
                os.system("./TMalign part-00000 part-00000")

if __name__ == '__main__':
    # This creates a local cluster
    spark = SparkSession.builder.appName('PDB').getOrCreate()
    main(spark)
```

Here we are going through each file in the first folder and saving each file into a rdd. We then save this rdd into a tempfile before iterating through to the next file in the folder we have another for loop for the second folder. This is to loop through and repeat the steps above for the second folder. As a result we end with two temp files which we can run tmalign on iterating through all the files in the second folder before moving on to the next file in the first folder.

This is not effeciant as we have nested for loops and we are having to read through the second file eberytime we go the the next file in the first folder.

Introducing wholeTextFiles() method allowed the program to only need to read each folder once. We then get the values from each key-value pair and perform the cartesean product on them. Giving us an rdd where each value in the first folder is paired with each value in the second folder. This helped utilize the memory and compute resources of a cluster efficiently which improved overall performance.

Testing and validation: Whilst developing the framework, using python unittests i was able to test and validate its performance on different sizes of datasets also looking at catching unxepected errors occuring. This helped identify areas where the code got improvments and ensured that it can handle increasingly large datasets.

An example change is shown where we use tempfiles over saveAsTextFile. As in the eleair version we did not have a key value pair we needed to use the saveAsTextFile method which spits out the contents of the rdd for us to be able to then run into the tmalign function. However we now have the contents of the pdv so we can write these to a temp file where once used the directory is closed and not used again. This saves reasorces as we are not creating and deleting a permanent directory for each pdb file each iteration.

- Ensure the software framework can be eaisly updated to keep pace with advancements in protein structure analysis techniques and computing technology.

In order to achive this objective we need to consider implementing a modular design that allows for easy integration of updates and improvements. We can also incorporate version control tools such as Git to manage changes in the codebase and ensure that the latest updates are available to users.

modular design

Within my software i have ensured that:

- Each module should perform one specific task or functionality independently
- Modules should have well-defined interfaces for communication with other modules
- Modules should be relatively self-contained, i.e., they should not have dependencies on other modules

- Each module should have unit tests to ensure its correct functionality
- Modules should be designed to be reusable

Git

Within git these actions have sure the software framework can be easily updated to keep pace with advancements:

- Create a repository where the framework code will be uploaded to and managed.
- After each update or modification to the codebase, commit the changes to the repository with a clear commit message describing the changes made.
- Merge code to the repository's main branch after verifying that it is stable.
- Publish documentation in the form of a README file or wiki, providing detailed information about how to interact with the software and its requirements.

By implementing modular design and using Git best practices, you can ensure that your codebase remains a high-quality, reliable, and scalable software framework with the latest updates available to users.

- Provide documentation and user support to enable researchers to use the software framework effectively

To complete this objective i need to provide a readme file that completes these points:

- Overview of framework
- Installation
- Usage
- Examples
- Contact Information

To do this i have created a readme file that fulfils each point. The Readme file will Provide a brief overview of your software framework, including its purpose and functionality.Provide clear and concise instructions on how to install your software framework, including any prerequisites that must be installed. Describe how to use your software framework, including any input/output file formats and command line options. Provide examples that demonstrate the use of your software framework, including sample input files and the expected output and finally, provide contact information for users to reach out to you in case they have any questions or issues.

5 Critical analysis and discussion: 10 marks

5.1 A discussion of actual project achievements

A list of all the things achived in my project:

- PySpark
- Cluster Setup
- Key Value Pair
- TMalign
- API Post
- API Get
- Benchmarking

5.2 how successful the project was

The project was quite successful with me completing most to all project set objectives. Being able to search and download pdb files from the framework. The user is also able to call out two user executables on the pdbs they require. At the end of this project we have a software framework that suppots the Mapreduce formalism and can process large numbers of protein structures. Whilst being able to do so by parallelize protein structure analysis across multiple computing nodes. The framework is optimized to reduce the processing time required for protein structure analysis.

The framework has a interface that allows users to manipulate the pdbs that are being passed into the executable for protein structure analysis.

Finally the software framework is scalable and can handle increasingly large datasets and can be eaisly updated to keep pace with advancements in protein structure analysis techniques and computing technology.

5.3 Reflection on the project process

The project went according to plan by completing most to all the set out objectives however seting up the clusetr with correct rdd and key value pairs took much more time then expected. For this reason i didnt have enough time to create a good interface keeping at a bash level of interaction for the user. A readme has been created which helps the user understand the functions within the framework however a more user frienly user interface would be benifitial for eaxmple when running the functions that implement the api to search or download pdbs a better user interafce can be created that would be easier to use to be able to set up the pdb files the user would like to run the executables on to.

5.4 conclusions or results analysed or discussed

Need to include benchmarking here

6 Professional Issues: 10 marks

6.1 Should be a topic relevant to the project undertaken.

Structural bioinformatics involves the use of complex data analysis techniques and tools that require a high level of accuracy and reliability. As such, it is important to consider ethical issues related to data privacy, ownership, and sharing. It means that there could potentially impact on public health.

Public health is an important consideration in the field of structural bioinformatics, as the analysis of biological molecules at a molecular level can lead to a greater understanding of diseases and potential treatments. For example, structural bioinformatics has been used to develop new drugs, vaccines, and therapies that can improve public health outcomes.

Structural bioinformatics combines computational and experimental techniques to study the structure and function of biological molecules at a molecular level, requiring the analysis of large amounts of complex data and the use of advanced tools and techniques to extract meaningful information.

Given the sensitive nature of the data involved, ethical issues related to data privacy, ownership, and sharing are important.

Data privacy is a crucial aspect of structural bioinformatics as it involves the handling of personal data. Researchers must ensure that the data they collect is kept confidential and used only for the purpose for which it was collected. Appropriate measures must also be taken to protect the data from unauthorized access or disclosure. These measures can consist of researchers must ensure that any sensitive data they collect (such as genomic data) is kept confidential and not disclosed to unauthorized parties. Appropriate measures must be taken to protect the data from unauthorized access, such as using encryption or secure storage methods. This means that researchers must take appropriate measures to prevent data breaches and ensure that the data they collect is secure. If a breach does occur, researchers must take prompt action to minimize the harm to individuals and organizations whose data has been compromised.

Ownership of data is another ethical issue that must be carefully considered. Researchers must respect the rights of the individuals or organizations that provide the data and ensure that they are given appropriate credit for their contributions. The potential commercial value of the data must also be considered to ensure that researchers do not exploit it for their own gain. To do so Researchers must respect the rights of the individuals or organizations that provide the data, and ensure that they are properly credited for their contributions. For example, Researchers need to obtain informed consent from participants before collecting their data, and clearly explain the purpose and potential risks of the study. Participants should also be informed about their rights to access and control their data. If the data has commercial value, researchers must consider the ethical implications of using it for their own gain, and whether they need to obtain consent or provide compensation to the data providers. Another important aspect to ownership is transparency such that researchers should be transparent about their data collection and analysis methods, and make their findings publicly available whenever possible. This can help ensure that the scientific community and the public at large are aware of the potential benefits and risks associated with structural bioinformatics research.

Sharing of data is a fundamental aspect of structural bioinformatics, as it allows researchers to collaborate and share knowledge. However, sharing of data must be done in a responsible and ethical manner, with appropriate safeguards in place to protect the privacy and confidentiality

of the data. When sharing data, researchers should establish data sharing agreements that outline the terms and conditions of the data sharing arrangement, including who has access to the data and how it can be used. Researchers must ensure that appropriate safeguards are in place to protect the privacy and confidentiality of the data, such as using data use agreements or anonymization techniques. Such that, anonymization techniques can be used to protect the privacy of individuals and organizations whose data is being used in structural bioinformatics research. Researchers can remove identifying information from the data, such as names or addresses, to prevent it from being traced back to specific individuals.

Policies are important to concider too where we have:

Data access policies: Institutions and organizations that collect and share data should establish data access policies that outline who can access the data and under what circumstances. These policies can help ensure that the data is used in a responsible and ethical manner.

Data retention policies: Institutions and organizations should establish data retention policies that dictate how long data can be stored and when it should be destroyed. This can help ensure that data is not used for unintended purposes or retained longer than necessary.

In conclusion, as the Structural Bioinformatics Framework is examined within the context of a MapReduce formalism, it is crucial to consider the ethical implications of this field. The potential impact of structural bioinformatics on public health is significant, but the responsible management of data privacy, ownership, and sharing must be prioritized to ensure the ethical and responsible advancement of this field.

Bibliography

[AAB+19] Paul D. Adams, Pavel V. Afonine, Kumaran Baskaran, Helen M. Berman, John Berrisford, Gerard Bricogne, David G. Brown, Stephen K. Burley, Minyu Chen, Zukang Feng, Claus Flensburg, Aleksandras Gutmanas, Jeffrey C. Hoch, Yasuyo Ikegawa, Yumiko Kengaku, Eugene Krissinel, Genji Kurisu, Yuhe Liang, Dorothee Liebschner, Lora Mak, John L. Markley, Nigel W. Moriarty, Garib N. Murshudov, Martin Noble, Ezra Peisach, Irina Persikova, Billy K. Poon, Oleg V. Sobolev, Eldon L. Ulrich, Sameer Velankar, Clemens Vonrhein, John Westbrook, Marcin Wojdyr, Masashi Yokochi, and Jasmine Y. Young. Announcing mandatory submission of PDBx/mmCIF format files for crystallographic depositions to the Protein Data Bank (PDB). *Acta Crystallographica Section D Structural Biology*, 75(4):451–454, April 2019.

[ALFJ+17] Rebecca F. Alford, Andrew Leaver-Fay, Jeliazko R. Jeliazkov, Matthew J. O'Meara, Frank P. DiMaio, Hahnbeom Park, Maxim V. Shapovalov, P. Douglas Renfrew, Vikram K. Mulligan, Kalli Kappel, Jason W. Labonte, Michael S. Pacella, Richard Bonneau, Philip Bradley, Roland L. Jr. Dunbrack, Rhiju Das, David Baker, Brian Kuhlman, Tanja Kortemme, and Jeffrey J. Gray. The Rosetta All-Atom Energy Function for Macromolecular Modeling and Design. *Journal of Chemical Theory and Computation*, 13(6):3031–3048, June 2017. Publisher: American Chemical Society.

[BB21] Stephen K. Burley and Helen M. Berman. Open-access data: A cornerstone for artificial intelligence approaches to protein structure prediction. *Structure (London, England: 1993)*, 29(6):515–520, June 2021.

[BBB+21] Stephen K. Burley, Charmi Bhikadiya, Chunxiao Bi, Sebastian Bittrich, Li Chen, Gregg V. Crichlow, Cole H. Christie, Kenneth Dalenberg, Luigi Di Costanzo, Jose M. Duarte, Shuchismita Dutta, Zukang Feng, Sai Ganesan, David S. Goodsell, Sutapa Ghosh, Rachel Kramer Green, Vladimir Guranović, Dmytro Guzenko, Brian P. Hudson, Catherine L. Lawson, Yuhe Liang, Robert Lowe, Harry Namkoong, Ezra Peisach, Irina Persikova, Chris Randle, Alexander Rose, Yana Rose, Andrej Sali, Joan Segura, Monica Sekharan, Chenghua Shao, Yi-Ping Tao, Maria Voigt, John D. Westbrook, Jasmine Y. Young, Christine Zardecki, and Marina Zhuravleva. RCSB Protein Data Bank: powerful new tools for exploring 3D structures of biological macromolecules for basic and applied research and education in fundamental biology, biomedicine, biotechnology, bioengineering and energy sciences. *Nucleic Acids Research*, 49(D1):D437–D451, January 2021.

[BBB+22a] Stephen Burley, Charmi Bhikadiya, Chunxiao Bi, Sebastian Bittrich, Henry Chao, Li Chen, Paul Craig, Gregg Crichlow, Kenneth Dalenberg, Jose Duarte, Shuchismita Dutta, Maryam Fayazi, Zukang Feng, Justin Flatt, Sai Ganesan, Sutapa Ghosh, David Goodsell, Rachel Kramer, Vladimir Guranovic, and Christine Zardecki. RCSB Protein Data Bank (RCSB.org): delivery of

experimentally-determined PDB structures alongside one million computed structure models of proteins from artificial intelligence/machine learning. *Nucleic Acids Research*, November 2022.

[BBB+22b] Stephen K. Burley, Charmi Bhikadiya, Chunxiao Bi, Sebastian Bittrich, Li Chen, Gregg V. Crichlow, Jose M. Duarte, Shuchismita Dutta, Maryam Fayazi, Zukang Feng, Justin W. Flatt, Sai J. Ganesan, David S. Goodsell, Sutapa Ghosh, Rachel Kramer Green, Vladimir Guranovic, Jeremy Henry, Brian P. Hudson, Catherine L. Lawson, Yuhe Liang, Robert Lowe, Ezra Peisach, Irina Persikova, Dennis W. Piehl, Yana Rose, Andrej Sali, Joan Segura, Monica Sekharan, Chenghua Shao, Brinda Vallat, Maria Voigt, John D. Westbrook, Shamara Whetstone, Jasmine Y. Young, and Christine Zardecki. RCSB Protein Data Bank: Celebrating 50 years of the PDB with new tools for understanding and visualizing biological macromolecules in 3D. *Protein Science*, 31(1):187–208, 2022. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/pro.4213.

[BBR20] Sebastian Bittrich, Stephen K. Burley, and Alexander S. Rose. Real-time structural motif searching in proteins using an inverted index strategy. *PLOS Computational Biology*, 16(12):e1008502, December 2020. Publisher: Public Library of Science.

[BG21] Payam Behzadi and Márió Gajdács. Worldwide Protein Data Bank (wwPDB): A virtual treasure for research in biotechnology. *European Journal of Microbiology and Immunology*, 11(4):77–86, December 2021. Publisher: Akadémiai Kiadó Section: European Journal of Microbiology and Immunology.

[BKW+77] F. C. Bernstein, T. F. Koetzle, G. J. Williams, E. F. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi. The Protein Data Bank: a computer-based archival file for macromolecular structures. *Journal of Molecular Biology*, 112(3):535–542, May 1977.

[BL22] Sarah E. Biehn and Steffen Lindert. Protein Structure Prediction with Mass Spectrometry Data. *Annual Review of Physical Chemistry*, 73(1):1–19, 2022. _eprint: https://doi.org/10.1146/annurev-physchem-082720-123928.

[BP03] Pierre Baldi and Gianluca Pollastri. The Principled Design of Large-Scale Recursive Neural Network Architectures–DAG-RNNs and the Protein Structure Prediction Problem. *NaN*, page 28, 2003.

[BPE22] Patrick Bryant, Gabriele Pozzati, and Arne Elofsson. Improved prediction of protein-protein interactions using AlphaFold2. *Nature Communications*, 13:1265, March 2022.

[BT98] Carl Ivar Branden and John Tooze. *Introduction to Protein Structure*. Garland Science, New York, 2 edition, December 1998.

[Bur21] Stephen K. Burley. Impact of structural biologists and the Protein Data Bank on small-molecule drug discovery and development. *The Journal of Biological Chemistry*, 296:100559, 2021.

[BWF+00] Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, January 2000.

[DBB03] Cyril Dominguez, Rolf Boelens, and Alexandre M. J. J. Bonvin. HADDOCK: A Protein Protein Docking Approach Based on Biochemical or Biophysical Information. *Journal of the American Chemical Society*, 125(7):1731–1737, February 2003. Publisher: American Chemical Society.

[DITS22] Alessia David, Suhail Islam, Evgeny Tankhilevich, and Michael J. E. Sternberg. The AlphaFold Database of Protein Structures: A Biologist's Guide. *Journal of Molecular Biology*, 434(2):167336, January 2022.

[DL17] Tomasz Drabas and Denny Lee. *Learning PySpark*. Packt Publishing Ltd, February 2017. Google-Books-ID: HVQoDwAAQBAJ.

[EWMR+06] Narayanan Eswar, Ben Webb, Marc A. Marti-Renom, M. S. Madhusudhan, David Eramian, Min-Yi Shen, Ursula Pieper, and Andrej Sali. Comparative protein structure modeling using Modeller. *Current Protocols in Bioinformatics*, Chapter 5:Unit–5.6, October 2006.

[Fel] Felix. A brief introduction to AlphaFold | Science | Felix Online.

[God22] W T. Godbey. Chapter 3 - Proteins. In W T. Godbey, editor, *Biotechnology and its Applications (Second Edition)*, pages 47–72. Academic Press, January 2022.

[Goo] David S. Goodsell. PDB101: Learn: Guide to Understanding PDB Data: Introduction.

[goy18] saurabh goyal. Spark Architecture and Deployment, October 2018.

[HRJ17] Akaash Vishal Hazarika, G Jagadeesh Sai Raghu Ram, and Eeti Jain. Performance comparision of Hadoop and spark engine. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 671–674, February 2017.

[JEP+21] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, August 2021. Number: 7873 Publisher: Nature Publishing Group.

[KKN+06] Rimantas Kodzius, Miki Kojima, Hiromi Nishiyori, Mari Nakamura, Shiro Fukuda, Michihira Tagami, Daisuke Sasaki, Kengo Imamura, Chikatoshi Kai, Matthias Harbers, Yoshihide Hayashizaki, and Piero Carninci. CAGE: cap analysis of gene expression. *Nature Methods*, 3(3):211–222, March 2006. Number: 3 Publisher: Nature Publishing Group.

[KMY+15] Lawrence A. Kelley, Stefans Mezulis, Christopher M. Yates, Mark N. Wass, and Michael J. E. Sternberg. The Phyre2 web portal for protein modeling, prediction and analysis. *Nature Protocols*, 10(6):845–858, June 2015. Number: 6 Publisher: Nature Publishing Group.

[Lam10] Chuck Lam. *Hadoop in Action*. Simon and Schuster, November 2010. Google-Books-ID: 8DozEAAAQBAJ.

[LLV18] Patanachai Limpikirati, Tianying Liu, and Richard W. Vachet. Covalent labeling-mass spectrometry with non-specific reagents for studying protein structure and interactions. *Methods (San Diego, Calif.)*, 144:79–93, July 2018.

[LWL+20] Julia Koehler Leman, Brian D. Weitzner, Steven M. Lewis, Jared Adolf-Bryfogle, Nawsad Alam, Rebecca F. Alford, Melanie Aprahamian, David Baker, Kyle A. Barlow, Patrick Barth, Benjamin Basanta, Brian J. Bender, Kristin Blacklock, Jaume Bonet, Scott E. Boyken, Phil Bradley, Chris Bystroff, Patrick Conway, Seth Cooper, Bruno E. Correia, Brian Coventry, Rhiju Das, René M. De Jong, Frank DiMaio, Lorna Dsilva, Roland Dunbrack, Alexander S. Ford, Brandon Frenz, Darwin Y. Fu, Caleb Geniesse, Lukasz Goldschmidt, Ragul Gowthaman, Jeffrey J. Gray, Dominik Gront, Sharon Guffy, Scott Horowitz, Po-Ssu Huang, Thomas Huber, Tim M. Jacobs, Jeliazko R. Jeliazkov, David K. Johnson, Kalli Kappel, John Karanicolas, Hamed Khakzad, Karen R. Khar, Sagar D. Khare, Firas Khatib, Alisa Khramushin, Indigo C. King, Robert Kleffner, Brian Koepnick, Tanja Kortemme, Georg Kuenze, Brian Kuhlman, Daisuke Kuroda, Jason W. Labonte, Jason K. Lai, Gideon Lapidoth, Andrew Leaver-Fay, Steffen Lindert, Thomas Linsky, Nir London, Joseph H. Lubin, Sergey Lyskov, Jack Maguire, Lars Malmström, Enrique Marcos, Orly Marcu, Nicholas A. Marze, Jens Meiler, Rocco Moretti, Vikram Khipple Mulligan, Santrupti Nerli, Christoffer Norn, Shane Ó'Conchúir, Noah Ollikainen, Sergey Ovchinnikov, Michael S. Pacella, Xingjie Pan, Hahnbeom Park, Ryan E. Pavlovicz, Manasi Pethe, Brian G. Pierce, Kala Bharath Pilla, Barak Raveh, P. Douglas Renfrew, Shourya S. Roy Burman, Aliza Rubenstein, Marion F. Sauer, Andreas Scheck, William Schief, Ora Schueler-Furman, Yuval Sedan, Alexander M. Sevy, Nikolaos G. Sgourakis, Lei Shi, Justin B. Siegel, Daniel-Adriano Silva, Shannon Smith, Yifan Song, Amelie Stein, Maria Szegedy, Frank D. Teets, Summer B. Thyme, Ray Yu-Ruei Wang, Andrew Watkins, Lior Zimmerman, and Richard Bonneau. Macromolecular modeling and design in Rosetta: recent methods and frameworks. *Nature Methods*, 17(7):665–680, July 2020. Number: 7 Publisher: Nature Publishing Group.

[LZD+20] Joseph H. Lubin, Christine Zardecki, Elliott M. Dolan, Changpeng Lu, Zhuofan Shen, Shuchismita Dutta, John D. Westbrook, Brian P. Hudson, David S. Goodsell, Jonathan K. Williams, Maria Voigt, Vidur Sarma, Lingjun Xie, Thejasvi Venkatachalam, Steven Arnold, Luz Helena Alfaro Alvarado, Kevin Catalfano, Aaliyah Khan, Erika McCarthy, Sophia Staggers, Brea Tinsley, Alan Trudeau, Jitendra Singh, Lindsey Whitmore, Helen Zheng, Matthew Benedek, Jenna Currier, Mark Dresel, Ashish Duvvuru, Britney Dyszel, Emily Fingar, Elizabeth M. Hennen, Michael Kirsch, Ali A. Khan, Charlotte Labrie-Cleary, Stephanie Laporte, Evan Lenkeit, Kailey Martin, Marilyn Orellana, Melanie Ortiz-Alvarez de la Campa, Isaac Paredes, Baleigh Wheeler, Allison Rupert, Andrew Sam, Katherine See, Santiago Soto Zapata, Paul A. Craig, Bonnie L. Hall, Jennifer Jiang, Julia R. Koeppe, Stephen A. Mills, Michael J. Pikaart, Rebecca Roberts, Yana Bromberg, J. Steen Hoyer, Siobain Duffy, Jay Tischfield, Francesc X. Ruiz, Eddy Arnold, Jean Baum, Jesse Sandberg, Grace Brannigan, Sagar D. Khare, and Stephen K. Burley. Evolution of the SARS-CoV-2 proteome in three dimensions (3D) during the first six months of the COVID-19 pandemic. *bioRxiv: The Preprint Server for Biology*, page 2020.12.01.406637, December 2020.

[LZG20] Xiaoran Roger Liu, Mengru Mira Zhang, and Michael L. Gross. Mass Spectrometry-Based Protein Footprinting for Higher-Order Structure Analysis: Fundamentals and Applications. *Chemical Reviews*, 120(10):4355–4454, May 2020. Publisher: American Chemical Society.

[MBY+16] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D. B. Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Za-

haria, and Ameet Talwalkar. MLlib: Machine Learning in Apache Spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

[NZLJ22] Ruth Nussinov, Mingzhen Zhang, Yonglan Liu, and Hyunbum Jang. AlphaFold, Artificial Intelligence (AI), and Allostery. *The Journal of Physical Chemistry B*, 126(34):6372–6383, September 2022. Publisher: American Chemical Society.

[OM06] Michał J. Okoniewski and Crispin J. Miller. Hybridization interactions between probesets in short oligo microarrays lead to spurious correlations. *BMC Bioinformatics*, 7(1):276, June 2006.

[OR15] Robert J. Ouellette and J. David Rawn. 14 - Amino Acids, Peptides, and Proteins. In Robert J. Ouellette and J. David Rawn, editors, *Principles of Organic Chemistry*, pages 371–396. Elsevier, Boston, January 2015.

[PBN12] Aditya B. Patel, Manashvi Birla, and Ushma Nair. Addressing big data problem using Hadoop and Map Reduce. In *2012 Nirma University International Conference on Engineering (NUiCONE)*, pages 1–5, December 2012. ISSN: 2375-1282.

[RBL+02] Jeannette Reinartz, Eddy Bruyns, Jing-Zhong Lin, Tim Burcham, Sydney Brenner, Ben Bowen, Michael Kramer, and Rick Woychik. Massively parallel signature sequencing (MPSS) as a tool for in-depth quantitative gene expression profiling in all organisms. *Briefings in Functional Genomics*, 1(1):95–104, February 2002.

[RDL+21] Yana Rose, Jose M. Duarte, Robert Lowe, Joan Segura, Chunxiao Bi, Charmi Bhikadiya, Li Chen, Alexander S. Rose, Sebastian Bittrich, Stephen K. Burley, and John D. Westbrook. RCSB Protein Data Bank: Architectural Advances Towards Integrated Searching and Efficient Access to Macromolecular Structure Data from the PDB Archive. *Journal of Molecular Biology*, 433(11):166704, May 2021.

[RLW+12] Daniel Russel, Keren Lasker, Ben Webb, Javier Velázquez-Muriel, Elina Tjioe, Dina Schneidman-Duhovny, Bret Peterson, and Andrej Sali. Putting the Pieces Together: Integrative Modeling Platform Software for Structure Determination of Macromolecular Assemblies. *PLOS Biology*, 10(1):e1001244, January 2012. Publisher: Public Library of Science.

[RRG07] Thomas E. Royce, Joel S. Rozowsky, and Mark B. Gerstein. Toward a universal microarray: prediction of gene expression through nearest-neighbor probe sequence identification. *Nucleic Acids Research*, 35(15):e99, 2007.

[SC10] Harry W Schroeder and Lisa Cavacini. Structure and Function of Immunoglobulins. *The Journal of allergy and clinical immunology*, 125(2 0 2):S41–S52, February 2010.

[SFB04] Peter D. Sun, Christine E. Foster, and Jeffrey C. Boyington. Overview of Protein Structural and Functional Folds. *Current Protocols in Protein Science*, 35(1):1711–171189, February 2004.

[Sin22] Pramod Singh. Manage Data with PySpark. In Pramod Singh, editor, *Machine Learning with PySpark: With Natural Language Processing and Recommender Systems*, pages 15–37. Apress, Berkeley, CA, 2022.

[SL20] Justin T. Seffernick and Steffen Lindert. Hybrid methods for combined experimental and computational determination of protein structure. *The Journal of Chemical Physics*, 153(24):240901, December 2020. Publisher: American Institute of Physics.

[SRC10] Jeffrey Shafer, Scott Rixner, and Alan L. Cox. The Hadoop distributed filesystem: Balancing portability and performance. In *2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*, pages 122–133, March 2010.

[TAW+21] Kathryn Tunyasuvunakool, Jonas Adler, Zachary Wu, Tim Green, Michal Zielinski, Augustin Žídek, Alex Bridgland, Andrew Cowie, Clemens Meyer, Agata Laydon, Sameer Velankar, Gerard J. Kleywegt, Alex Bateman, Richard Evans, Alexander Pritzel, Michael Figurnov, Olaf Ronneberger, Russ Bates, Simon A. A. Kohl, Anna Potapenko, Andrew J. Ballard, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Ellen Clancy, David Reiman, Stig Petersen, Andrew W. Senior, Koray Kavukcuoglu, Ewan Birney, Pushmeet Kohli, John Jumper, and Demis Hassabis. Highly accurate protein structure prediction for the human proteome. *Nature*, 596(7873):590–596, August 2021. Number: 7873 Publisher: Nature Publishing Group.

[vdABH17] Wil M. P. van der Aalst, Martin Bichler, and Armin Heinzl. Responsible Data Science. *Business & Information Systems Engineering*, 59(5):311–313, October 2017.

[WDA+16] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J. G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A. C. 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1):160018, March 2016. Number: 1 Publisher: Nature Publishing Group.

[WF03] John D. Westbrook and Paula M. D. Fitzgerald. The PDB Format, mmCIF Formats, and Other Data Formats. In *Structural Bioinformatics*, pages 159–179. John Wiley & Sons, Ltd, 2003. Section: 8 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471721204.ch8.

[WGS09] Zhong Wang, Mark Gerstein, and Michael Snyder. RNA-Seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, 10(1):57–63, January 2009. Number: 1 Publisher: Nature Publishing Group.

[WIN+05] John Westbrook, Nobutoshi Ito, Haruki Nakamura, Kim Henrick, and Helen M. Berman. PDBML: the representation of archival macromolecular structure data in XML. *Bioinformatics*, 21(7):988–992, April 2005.

[WSHB20] John D. Westbrook, Rose Soskind, Brian P. Hudson, and Stephen K. Burley. Impact of the Protein Data Bank on antineoplastic approvals. *Drug Discovery Today*, 25(5):837–850, May 2020.

[YYR+15] Jianyi Yang, Renxiang Yan, Ambrish Roy, Dong Xu, Jonathan Poisson, and Yang Zhang. The I-TASSER Suite: protein structure and function prediction. *Nature Methods*, 12(1):7–8, January 2015. Number: 1 Publisher: Nature Publishing Group.

[Zve08] Marketa J. Zvelebil. *Understanding bioinformatics / Marketa Zvelebil & Jeremy O. Baum.* Garland Science/Taylor & Francis Group, Garland Science, Taylor & Francis Group, New York, 2008.