PH2: An Hadoop-based framework for mining structural properties from the PDB Database

Scott Hazelhurst
School of Electrical and Information Engineering
University of the Witwatersrand, Johannesburg
Private Bag 3, 2050 Wits, South Africa
Scott.Hazelhurst@wits.ac.za

ABSTRACT

PH2 is an Hadoop and SQL-based tool for extracting information out of the Protein Database (PDB) quickly. The PDB database is stored as a set of Hadoop sequence files in a replicated way on the Hadoop Distributed File System. PH2 then allows a user to provide queries about 3D structures (and other properties) in SQL, and for these queries to be run in a highly-parallel manner using the Hadoop framework. PDB is an important source of information about structural and other properties of proteins, and it currently contains about 65000 protein structures. Determining which proteins have particular shapes is an important bioinformatics application. PH2 parses each PDB file, creates a SQL database for it and then performs the appropriate queries. Experiments performed on a small local cluster and a large shared cluster show that the application is highly-scalable. On the large cluster, a complex real query takes less than 4 minutes to search the whole of PDB.

Categories and Subject Descriptors

D1.3 [**Software**]: Concurrent programming; H2.8 [**Databases**]: Database applications; J.3 [**Computer applications**]: Life and medical sciences

General Terms

Algorithms, Performance, Experimentation

Keywords

Hadoop, PDB, parallel computing, structural information

1. INTRODUCTION

The determination of the three dimensional structure of a protein is one of the most important questions in a range of problems across the molecular sciences. Since the shape of a protein helps determine its function, knowing a protein's shape is important for both fundamental science and applied science (e.g., drug design [10]).

Predicting the three dimensional structure of a protein from its linear sequence through computational means remains one of the big open questions of bioinformatics, despite impressive progress. And being able to "design" a protein to have a particular shape is *a fortiori* more difficult. Thus, in many applications, the first choice for determining the structure of a protein is to look in a database of known protein structures. A number of algorithms have been developed to search for proteins with particular properties — see [6] for an overview.

The Protein Database (PDB) [3] is perhaps the oldest of all bioinformatics databases. It contains the structural properties of about 65000 protein and nucleotide sequences (the overwhelming majority being protein). These structures have been determined experimentally — typically using X-ray crystallography and nuclear magnetic resonance (NMR). Despite shortcomings in resolution and the static nature of the obtained structure (a protein may change shape many times per second), the quality and quantity of data in PDB makes this the pre-eminent source of three dimensional structure information.

This paper presents an algorithm and tool that attempts to solve the following problem:

> *Given a desired structure, find the proteins which match the structure within certain user-specified parameters.*

The idea is to allow the user to provide a query over the (3D) structural properties, and to return the list of proteins together with meta-information of all proteins that match the query. The constraints on the design tool are:

- completeness — all real matches should be found;
- efficiency, or at least small running time — PDB is very large, and often scientists will have to play with different parameters and so many searches may be required;
- usability — a protein expert rather than a computer scientist should be able to specify queries of interest.

This paper addresses the first two points and suggests future research on the third point.

Motivation and background. This project started in response to a request from colleagues from Elevation Biotech, a biotechnology company, to help them find proteins that had certain structural properties. We developed prototype versions 0 and 1 to search PDB (version 1 is discussed briefly below). Version 1 solved the specific questions they had adequately (taking just over an hour to run), and so showed the value of the concept. But as it stands, version 1 is not a generally usable tool, without the program's author at your side.

An example of the type of query which a molecular scientist might want to make is:

> Find PDB files which contain an amino acid of type A on one chain, and an amino acid of type B on another chain, where an atom x_a of the first amino acid is within a certain distance and 3D orientation of atom x_b of the second amino acid.

As a command-line tool, PH1 could be parameterised, but only to some extent. To make the computations happen in reasonable time, a bespoke data structure was created that indexed certain data points. Some changes to the query required hand-editing relatively intricate parts of the C-code. And it was easy to think of small extensions to the query which would require rewriting of the program — hence, the need for the program's author.

Thus the primary goal of the research presented here was to design a tool that was more general — requiring much less computing expertise to run.

The computational cost was also a factor — although a turn-around time of roughly an hour is acceptable, it makes experimentation and fine-tuning difficult. Of course, the goal of writing a much more general solution conflicts with the need for more efficiency (or at least a shorter running program). Ideally we would like to reduce run-time and in generalising the solution we do not want to increase computational costs.

Outline of solution. The solution adopted in PH2 was two fold: (1) rather than using a bespoke data structure, use SQL; and (2) make the program run faster by parallelising.

1. The tool parses PDB files and stores them internally in SQL tables. The user specifies the query as an SQL query which is run against the SQL database. Although this still requires a significant degree of expertise, it is much more accessible since the specifier needs only the relevant biological knowledge, an understanding of the principles of SQL, and the very simple schema used, and not the specifics of a particular program. Moreover, this becomes amenable to providing a GUI which generates the required SQL.
2. Parallelise using the Hadoop. Hadoop is a powerful and flexible scheme for parallelisation and supports very large files. The experiments were carried out on a large public cluster (provided as part of the IBM/Google Cloud Computing Academic Initiative).

Structure of paper. The rest of this paper is structured as follows. Section 2 gives background, describing the biology in a little more detail and describing the structure of PDB, as well as an overview of the Hadoop framework. Section 3 presents PH2's design and implementation. Section 4 evaluates PH2. Finally, Section 5 concludes and suggests future research.

2. BACKGROUND

2.1 Proteins

Proteins are the essential building blocks of life [9]. Proteins perform a wide variety of functions in biological organisms: some directly form part of the organic material (e.g., the protein keratin is an important part of skin); some proteins are responsible for signalling and communication; some proteins act as enzymes; some are critical components of the immune system.

Proteins are *macromolecules*, which are in turn composed of molecules known as amino acids. There are 20 common, naturally occurring amino acids [9], although there are other naturally occurring and more exotic amino acids too. Different amino acids have different properties (e.g., size, charge, hydrophobicity).

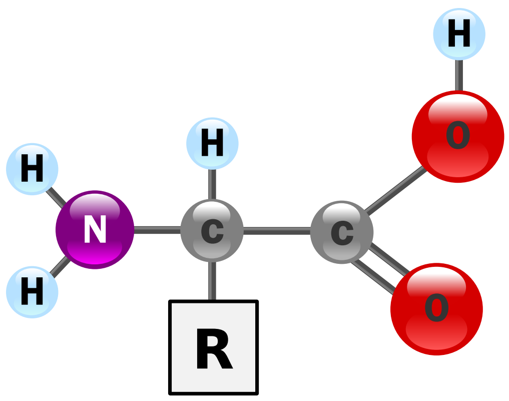
The general structure of an amino acid can be seen in Figure 1(a): they have an amine group (shown on the left), a carboxyl group (on the right), and a side-chain (shown as R). The amino acids differ significantly in their side chains. For example, the amino acid *alanine*, $CH_3CH(NH2)COOH$, depicted in Figure 1(b) has a relatively simple side-chain of a carbon atom and three hydrogen atoms.

An individual protein has a primary, linear structure — the linear sequence of the amino acids that comprise it. The secondary structure is the local three-dimensional structure of amino acids. There are different classifications such as α-helices and β-sheets. The tertiary structure is the overall three dimensional structure of the protein, typically represented by the three dimensional coordinates of the amino acids' atoms. For example, the shape of the protein myoglobin can be seen in Figure 2, with a number of α-helices clearly visible.

Finally, proteins often occur in protein complexes, where different proteins interact with each other. The 3D shape of a protein may change, depending on which complex it is in. This is known as the *quaternary* structure of the protein.

Determining 3D structure is difficult, and it should be borne in mind that proteins are dynamic, not static objects. While determining primary structure is relatively straight-forward, computing the 3D structure from primary structure is not, and the most accurate

(a) General structure



(b) Alanine

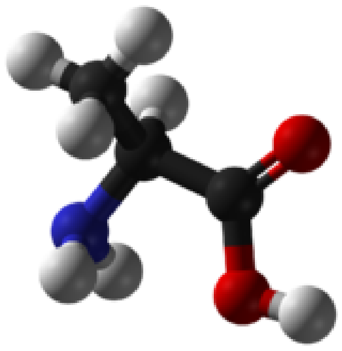


Figure 1: amino acid structure (figures courtesy of Wikimedia)

techniques for doing so are experimental using techniques such as X-ray crystallography and nuclear magnetic resonance. [4].

2.2 Structure of PDB

PDB contains 66234 structures (as at 6 July 2010) having grown from 13600 in 2000 to 34000 in 2005. In 2009, an average of 143 new structures were added into PDB each week[1], with a sightly higher rate of additions in the first half of 2010.

For historical reasons, each protein structure is represented in multiple formats. For this research, the PDB File Format collection was used (approximately 10GB in size, compressed and gzipped — uncompressed it is about four times bigger).

Each PDB file describes a protein or protein complex. It will contain meta-information about the protein (name, who determined the structure and how, journal information, source organism(s), etc.) as well as structural information. The bulk of each PDB file consists of a table which shows for each atom in the protein, which amino acid it belongs to, and the 3D coordinates of that atom. The primary structure of the protein can be determined directly from the order of the amino acids. An assignment

[1]`http://www.pdb.org/pdb/statistics/contentGrowthChart.do?content=total&seqid=100`

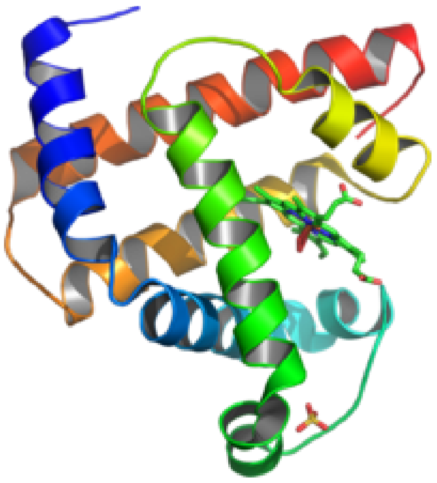


Figure 2: Secondary and tertiary structure of myoglobin (courtesy of Wikimedia)

of secondary structure is often also given in the file. The tertiary structure can be determined from the 3D coordinates of the atoms. Where a protein or protein complex consists of multiple chains, each chain's data is given too. For structures which are determined by NMR, typically different models are given which show the shape of the protein at different times. Figure 3 shows a small extract (see `http://www.rcsb.org/pdb/files/2J0U.pdb` for the full file).

2.3 Related work

Since PDB is such a useful source of information and there are so many different types of questions one could ask, a number of tools have been developed that can extract structural information from PDB. A comprehensive review of the related literature is not within the scope of this paper.

PSST-2.0 [2] allows searching of PDB in a variety of ways, including on amino acid composition, sequence length, and molecular weight. Structural searching is based on primary structure.

FATCAT [18] allows pairwise comparison between proteins, as well as searching for sequences based on structural similarity. The user provides a PDB file, and a structural similarity algorithm determines which proteins match it, to which degree. As can be imagined, it is a very expensive search.

PAST [15] is a web-based tool for searching for local structural motifs. The user provides a PDB file or extract from a PDB file as input, and the tool searches for other PDB files that have similar, local structures. SALAMI [11] is a similar tool.

The Protein Segment Finder [14] allows searching on segment size, amino acid sequence, secondary structure sequence and contacts (amino acids being close to each other). This last facility is closest to ours. However, for performance reasons they pre-compute possible contacts and so restrict contacts to being within 30 residues of each other. PH2 does an *ab initio* search of PDB in each search

```
HEADER    HYDROLASE                               04-AUG-06   2JOU
Title     THE CRYSTAL STRUCTURE OF EIF4AIII-BARENTSZ COMPLEX AT 3.0 A
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: ATP-DEPENDENT RNA HELICASE DDX48;
COMPND   3 CHAIN: A;
....
COMPND  10 MOL_ID: 2;
COMPND  11 MOLECULE: ATP-DEPENDENT RNA HELICASE DDX48;
COMPND  12 CHAIN: B;
.....
COMPND  19 MOL_ID: 3;
COMPND  20 MOLECULE: PROTEIN CASC3;
COMPND  21 CHAIN: T;
COMPND  22 FRAGMENT: RESIDUES 137-250;
COMPND  23 SYNONYM: BARENTSZ, CANCER SUSCEPTIBILITY CANDIDATE GENE 3
...
SOURCE    MOL_ID: 1;
SOURCE   2 ORGANISM_SCIENTIFIC: HOMO SAPIENS;
...
SOURCE  18 EXPRESSION_SYSTEM_VECTOR: PETMCN
KEYWDS    ATP-BINDING, DNA-BINDING, NUCLEAR PROTEIN, RRNA PROCESSING,
....
EXPDTA    X-RAY DIFFRACTION
AUTHOR    F.BONO,J.EBERT,E.LORENTZEN,E.CONTI
REVDAT   1   06-SEP-06 2JOU    0
JRNL        AUTH   F.BONO,J.EBERT,E.LORENTZEN,E.CONTI
JRNL        TITL   THE CRYSTAL STRUCTURE OF THE EXON JUNCTION COMPLEX
....
JRNL        REFN   ASTM CELLB5  US ISSN 0092-8674
REMARK   2 RESOLUTION. 3.00 ANGSTROMS.
.....
ATOM 58 CD2 LEU A  45 -4.123 -19.232  18.681  1.00 55.98 C
ATOM 59 N   ARG A  46 -7.658 -22.192  20.815  1.00 55.08 N
ATOM 60 CA  ARG A  46 -8.616 -22.967  21.590  1.00 55.42 C
```

Figure 3: Extract from the protein with PDB ID *2jou*. It shows that the δ-carbon 4 atom of a leucine amino acid can be found at position $(-4.123, -19.232, 18.681)$ and a nitrogen atom of an arginine can be found at $(-7.658, -22.192, 20.815)$. The leucine and arginine are neighbours being the 45th and 46th amino acids in the protein's primary structure. There are 3 chains in this complex: the two amino acids shown here are in chain A.

(obviously paying a significant computational cost which we pay for through parallelism).

Our tool, PH2, differs in the type of query that can be asked. It was designed to answer questions about structural properties of proteins or protein complexes which are not inherently local. In our real examples, we may be looking for amino acids or fragments (and more likely particular atoms) which are close to each other in the 3D space but are very far apart in primary structure. However, since queries can be given in SQL, the type of question that can be asked is very general.

The PDB-SQL tool [13] stores the contents of PDB in an SQL database. The database schema is very structured and normalised. However, it only contains the 3D coordinates of the α-carbon atoms of each residue, thus the amount of data stored and accessible is relatively small. Moreover, the database schema is designed for queries such as *find the coordinates of amino acids that match some property* rather than to find amino acids that are near some coordinate.

Two student projects explored the same problem [1, 12] using different approaches and technologies. These were useful in showing that it was feasible to solve the problem. An earlier, unpublished version of this tool, PH1, was written in C, using a bespoke data structure to index the PDB files as they were processed.

2.4 Hadoop Overview

Hadoop [16, 17] is an Apache project (`http://hadoop.apache.org/`) for reliable computing. There are a number of sub-projects, most of which are beyond the scope of this paper. For the purpose of this paper, there are two components that are important:

- A reliable, distributed file system. A number of file systems support Hadoop, but PH2 uses the Hadoop Distributed File System (`http://hadoop.apache.org/hdfs/`). Influenced by the design of the Google File System [7], the HDFS provides large-scale, distributed, redundant storage of data. Designed for peta-scale computing, it is highly-scalable.
- An implementation of the Map-Reduce framework for computation [5]. The Map-Reduce framework is an old one in computer science, dating back to the introduction of functional programming. It provides a simple but powerful paradigm for programming. Computation is divided into two phases. In the map-phase, the data is broken up and a *map* function is applied to each section of data. In the *reduce* phase, the results of the individual maps are combined together. For more complex computations, map-reduce computations can be chained.

A running Hadoop system will have nodes for the following:

- A *jobtracker*, which is responsible for allocating jobs to nodes and monitoring their behaviour;
- A *namenode*, which is responsible for storing the meta-information of the file system;
- A *secondary namenode*, which acts as a backup for the *namenode*;
- A set of worker nodes which do the work; these typically also act as data nodes, so that the Hadoop Distributed File System is distributed across the worker/data nodes.

Files are replicated in the distributed file system, which is important for reliability, availability and performance. Large files are split into blocks (by default 64MB in size), and replication happens at the block level. Different parts of a file may be stored on different physical computers, thereby improving scalability and performance (since different cheap, local file systems can read different parts of the same file in parallel).

For this project, the scalability of HDFS is not particularly important since although PDB is large, it is well within the range that conventional file systems can support. However, the data replication is very useful. Replication supports parallel access to the data, which improves performance and simplifies programming. The map-reduce framework simplifies application development.

3. TOOL DESIGN AND IMPLEMENTATION

3.1 Creation of PDB Sequence Files

PDB files in their raw form are not the most convenient for Hadoop. Hadoop is designed for dealing with very large files, and individual PDB files are relatively small (most are well under a megabyte). Moreover with over 60k files in the database, the number of files would impose significant extra book-keeping with adverse performance implications [17].

For this reason, we batch a number of PDB files together. Conveniently, the PDB database is stored hierarchically into about 1000 directories, which for ease of experimentation was used for processing (however, for a production system it might be appropriate to use a different way of dividing the files since we need to cater for new contributions to the database for dynamic update). An auxiliary Hadoop application batches all the PDB files in one PDB directory into an Hadoop sequence[2] file. Each sequence file consists of a number of records, each record representing one PDB file. The key values are the names of the PDB files and the values are the bytes of the compressed PDB file.

[2]*Sequence* here is an Hadoop term, referring to a sequence of bytes, and has nothing to with the fact that the underlying data contains protein sequences.

This process leaves approximately 1000 sequence files, which vary from about 500KB to 30MB in size. Alternative approaches are discussed in Section 5.

3.2 Database design

Based on the preliminary versions of the tool the following design decisions were made.

- Queries should be specified in SQL. Although still requiring a high-degree of computing expertise, it does not require in-depth knowledge of a particular program and its data structures. Moreover, as described later, it is amenable to providing a GUI which generates the actual SQL.
- The memory requirements of storing all atom data *and* the fact that queries will seldom if ever refer to two PDB files at the same time guided the decision not to store PDB as a monolithic SQL database. (1) For structural properties we do not need to do queries between PDB files (e.g., we do joins within a PDB file but not across). Therefore, there is no functional need for a monolithic database. Second, the Hadoop framework takes care of data replication and work allocation across a distributed environment. Finally, in deciding against trying a monolithic database, we observed that the work of [13] was a substantial computational feat even though most atom data was omitted. Moreover, at that time PDB was just over half the current size, and so PDB-SQL's database was about 7% of the size of our database (and it was still a computational feat). For this reason, each file is stored in its own database.

The schema for the PDB file databases is shown in Figure 4. There is only one table, *PDBTable*. Each record in the database describes one atom in a PDB file. The fields are as follows: *model* says which model of the protein it belongs to (for structures resolved with NMR) or 0 (for X-ray crystallography); *chain* states to which chain of the protein the atom belongs; *rcode* is the index of the atom in the PDB file; *residue* is the type of amino acid (e.g., leucine); *serial* is the index of the amino acid in the PDB file; *secstruc* is the assigned secondary structure; *atom* is the type of atom (e.g., C for Carbon) *aname* is the full name of the atom (e.g., CD4 for the δ-carbon 4 atom); x, y, z are the 3D coordinates; and *occ* and *tf* are the occupancy and temperature factors (crudely estimates of variation and accuracy).

The table is not normalised and contains redundant information. This choice was made since the database is created once and no modifications are made. Therefore, speed of access to the data rather than consistency of update is of paramount importance. A secondary factor is the size of any one database is not particularly big (tens of megabytes) so there is no need to conserve space for the sake of conserving space.

HSQLDB (*http://hsqldb.org/*) was used as the SQL database engine (version 1.8.1). It is a Java implementation of SQL, which allows convenient integration into the overall project JAR file, but other database engines could be used too. HSQLDB was run in in-process, pure

Field	Type
model	int
chain	char
rcode	varchar(5)
residue	varchar(4)
serial	int
atom	char
secstruc	char
aname	varchar(8)
x,y,z	double
occ, tf	double

Figure 4: Schema for PDBTable

memory mode which means that a separate SQL process server is not run, and that the database is kept in RAM rather than on disk.

The database was not indexed. There is a tension between taking the time to create appropriate indexes versus the time to make queries. For our experimentation, it did not make much difference, though some preliminary experimentation with other SQL database engines indicated that indexing can be an issue. This is something that could in principle be left to the user rather than hard-coding into the system, and is discussed later.

3.3 The PH2 application

The program is run using Hadoop and is given three parameters: the name of an SQL file with the query, the path of the directory containing all the sequence files, and a path where output should go.

The Hadoop system then executes the application on the cluster worker nodes. The number of workers used depends largely on the size of the input. Although configurable by the user, Hadoop by default divides the work up according to the number of input files and their size. Typically, each input file becomes a *split* on which work is done (although very big files might be split too). The number of physical processors on which these maps run depends on the number available and the overall load of the system.

Each split consists of a number of records, each record storing data for a PDB file. The key of a record is the name of the underlying PDB file, the value is the sequence of raw bytes representing the compressed data. The map function of PH2 is called for each record. PH2 runs Hadoop under the mode where the Java Virtual Machine for the map function can be re-used. Hadoop is run in non-multithreaded mode since this simplified programming, though this comes at the cost of extra memory on multi-core machines as each core has to have its own JVM.

3.4 The Map function: Parsing and querying

The core of the algorithm is the map function of the Hadoop framework, which is responsible for parsing a PDB file, populating the PDBTable, performing a query, returning results and then emptying the PDBTable.

Each call of a map function handles one record of a sequence file. The key value of the underlying PDB file name and an array of raw bytes of the compressed PDB file are passed as the parameters to the map function.

Parsing of the PDB was done using BioJava [8] (version 1.7). Once parsed, the information in the PDB file is inserted into PDB Table using the appropriate HSQLDB API. The query is then invoked on the database. If there are any results, they are output for the mapper to collate: the key is the name of the PDB file; the value is the result of the query.

3.5 The Reducer

In the current version, the reducer is the identity reducer of the Hadoop framework. Essentially this means that all the results from all the Hadoop calls are merged into one file, ordered by the file name. However, it would be possible to be more sophisticated. For example, the reducer could add the results of a query into another database which would allow more sophisticated queries.

4. EXPERIMENTS AND EVALUATION

This section presents some preliminary experimentation and evaluation.

4.1 Performance

To evaluate performance, PH2 was used to run a realistic, non-trivial SQL search over PDB and subsets of PDB. The sample SQL-script is shown in Figure 6 on page 111 (the details of particular amino acids and atom types searched for have been removed).

Base-line. As a baseline our tool PH1 takes about 3900s (just over an hour) to search the whole of PDB on a 2.3 GHz Xeon (E5345). However, its functionality and extensibility is much more limited than PH2.

Small, private cluster. The first experiment ran PH2 on our small, heterogeneous cluster of five 8-core machines. Figure 5 shows that the performance of PH2 scales linearly with data set size. Note that although nominally there are 40 cores in this cluster, memory and cache contention is likely to limit speed-up.

Large, shared cluster: IBM/Google Cloud . PH2 was run on the cluster of the IBM/Google Cloud Computing University Initiative. The cluster has 419 dual processor 2.8GHz Xeon machines (15,4), each with 1MB of L2 cache. On this cluster, there is no direct control over the number of physical processors used: this is determined by the number of data files (and in principle, system load — although we have been able to mitigate this by only running when the system is not loaded).

On this cluster, searching the whole of PDB takes 205s. Since the performance will vary on this multi-user cluster

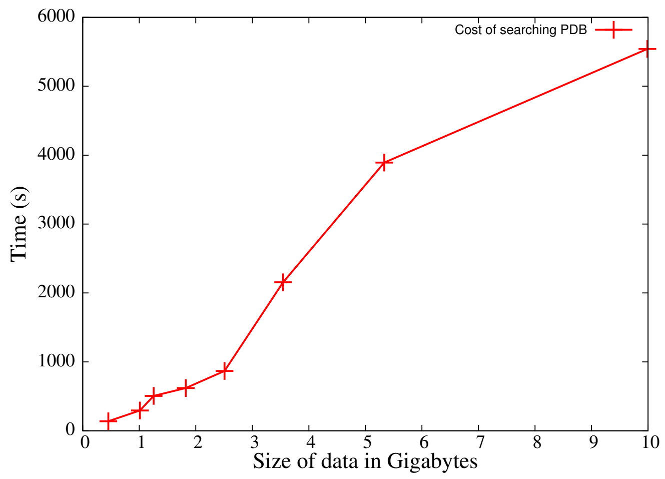


Figure 5: Performance of PH2 on different subsets of (compressed) PDB: running on small, local cluster

we do not try to produce a graph analogous to 5

4.2 Evaluation

The benefit gained is that we have a general purpose tool that can answer a wide range of queries over PDB, and a much more useful tool than our specialised PH1.

PH2 is about two orders of magnitude more expensive than PH1. This computational price needs to be traded-off against the increased power of the tool. The computational price can also be paid for by using parallelism. The application is highly-scalable and can be run in single-processor mode (probably 15–20 hours) or on a cluster of machines.

Evaluation of Hadoop

Hadoop was chosen as a means to an end, but the tool also allows us to evaluate Hadoop. On the downside, there is a steep learning curve to using Hadoop (say compared to MPI). On the other hand, once learned, it gives a powerful framework for developing further applications. One practical issue is that the Hadoop API is not stable and material published in 2009 is to some extent obsolete as new versions of the API have been released.

Second, although Hadoop is designed for fault tolerance, we experienced some level of unreliability on our local cluster. For example, we would find a run of a program taking significantly longer than expected, and tasks starting to crash. Rebooting the Hadoop system on our local cluster made the problem go away. This will no doubt be fixed as Hadoop matures.

5. CONCLUSION

This work shows the viability of performing complex SQL queries on PDB, and most importantly doing queries on structure. Although this requires substantial computational investments, small private clusters, large shared clusters and virtual clusters are readily available. For biologists, the cost of accessing commercial Hadoop facilities is very small in comparison to the costs of wet lab research, and minuscule in relation to the costs of the time of researchers who will use these facilities.

Nevertheless, there are a number of aspects for improvement.

5.1 User interface

Although the use of SQL as the query language is a significant improvement over changing a C program, its use still limits the ability of the system by its intended audience. We would like to explore the use of a user-friendly GUI, which biologists could easily specify a wide-range of queries. Of course, an expert user would still be able to give their own queries in SQL, but we believe that the majority of users could do so with a GUI. Figure 7 shows a mock-up of a web-based form that could be used to generate a typical query.

A subsidiary activity would be the development of a SQL library that supports common queries. This would also likely improve performance.

5.2 Performance improvement

There are number of areas in which performance can be improved. Some are relatively easy to do — for example, exploring the performance-tuning parameters of Hadoop, which can make a substantial difference in performance of programs [17]: little of this has been done. Other areas for improvement may require a more substantial redesign.

Hadoop is a Java-based framework, and so the most obvious design is to have a pure Java solution, which PH2 has done (using BioJava and HSQLDB). The main advantage of this is portability. In development, we were able to develop using Hadoop pseudo-mode on an Apple, experiment on a 32-bit Linux architecture and move to different 64-bit Linux architectures without recompilation. However, this likely comes at a significant cost. Using Hadoop's streaming facilities, it is possible to run arbitrary native-code binaries. It would be worth exploring how this could improve performance.

An important design decision was to store the PDB data in Hadoop sequence files. Each of these sequence files contained approximately 60 PDB files, each in a record (key: name, value: PDB data). The primary motivation for this decision was expedience, based on the APIs of BioJava, HSQLDB and Hadoop. A better option would be to store the SQL database directly. This would not only save on parsing and database creation costs, but would also reduce the trade-off in indexing, since we could pre-index (at the cost of an increase in memory costs). Indexing would improve the costs of doing queries. In principle, the change of mechanism of PDB storage is straight-forward, but there are some technical details related to the various API calls available that make this move tricky.

While the use of BioJava was very useful for program development, given how core the parsing of PDB files is,

```
select * from
   (select AMINO1_A_chain, AMINO1_A_model, AMINO1_AT1_res,
           'AMINO1_AT1',AMINO1_AT1_ser,AMINO1_AT1_x,AMINO1_AT1_y,AMINO1_AT1_z,
   'AMINO1_AT2',AMINO1_AT2_ser,AMINO1_AT2_x,AMINO1_AT2_y,AMINO1_AT2_z from
(select
   chain as AMINO1_A_chain, model as AMINO1_A_model,
           serial as AMINO1_AT1_ser, rcode as AMINO1_AT1_res,
           x as AMINO1_AT1_x, y as AMINO1_AT1_y, z as AMINO1_AT1_z from PDBTable
   where residue='AMINO1' and aname = 'AT1' ) AS CAT1 join
(select
           chain as AMINO1_B_chain, model as AMINO1_B_model,
   serial as AMINO1_AT2_ser, rcode as AMINO1_AT2_res,
   x as AMINO1_AT2_x, y as AMINO1_AT2_y, z as AMINO1_AT2_z from PDBTable
  where residue='AMINO1' and aname = 'AT2') AS CAT2
  on AMINO1_AT1_res=AMINO1_AT2_res and
     AMINO1_A_chain = AMINO1_B_chain and AMINO1_A_model=AMINO1_B_model)
   AS CC
   join
   (select AMINO2_A_chain, AMINO2_A_model, AMINO2_AT1_res,
           'AMINO2_AT1',AMINO2_AT1_ser,AMINO2_AT1_x,AMINO2_AT1_y,AMINO2_AT1_z,
   'AMINO2_AT2',AMINO2_AT2_ser,AMINO2_AT2_x,AMINO2_AT2_y,AMINO2_AT2_z from
(select
   chain as AMINO2_A_chain, model as AMINO2_A_model,
           serial as AMINO2_AT1_ser, rcode as AMINO2_AT1_res,
           x as AMINO2_AT1_x, y as AMINO2_AT1_y, z as AMINO2_AT1_z from PDBTable
   where residue='AMINO2' and aname = 'AT1' ) AS CAT1
join
(select
   chain as AMINO2_B_chain, model as AMINO2_B_model,
   serial as AMINO2_AT2_ser, rcode as AMINO2_AT2_res,
 x as AMINO2_AT2_x, y as AMINO2_AT2_y, z as AMINO2_AT2_z from PDBTable
  where residue='AMINO2' and aname = 'AT2') AS CAT2
  on AMINO2_AT1_res=AMINO2_AT2_res and AMINO2_A_model=AMINO2_B_model and
            AMINO2_A_chain = AMINO2_B_chain)
    AS SC
    ON CC.AMINO1_A_chain<>SC.AMINO2_A_chain and
       CC.AMINO1_A_model = SC.AMINO2_A_model and
  ((AMINO2_AT1_x-AMINO1_AT1_x)*(AMINO2_AT1_x-AMINO1_AT1_x)+
   (AMINO2_AT1_y-AMINO1_AT1_y)*(AMINO2_AT1_y-AMINO1_AT1_y)+
   (AMINO2_AT1_z-AMINO1_AT1_z)*(AMINO2_AT1_z-AMINO1_AT1_z) between 70 and 90);
```

Figure 6: Sample SQL script

it would be better to write a specialised parser (as done with PH1) that does exactly what we need rather doing more general purpose parsing.

The changes above would also allow the tool to have greater control of memory. Given the size of the individual data files, RAM should not be a problem. However, especially when running in a multi-core environment, we had to take care to tune the JVM memory allocation very carefully (too little and the JVM would run out of stack or heap space; too much, the machine would run out of physical memory).

5.3 Use of meta-data

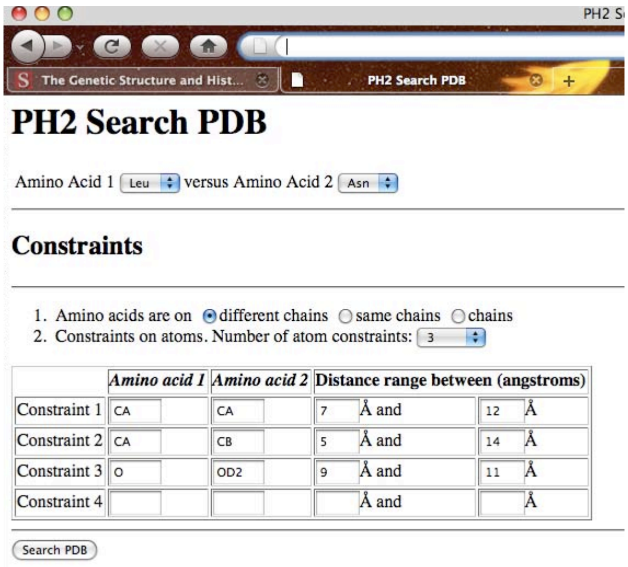
Although it has been argued that the use of many small databases is more appropriate than having one monolithic database, there is a point in having a database of the sort described in [13], since this would allow querying of meta-data (organism, type of protein, author, journal articles). Not only would this give more functionality to the user, it would also allow significant performance improvement since it could be used to restrict which data was searched. For example, by restricting a search to entries to proteins from a particular species, the size of the overall data that needed to be searched could be restricted significantly.

5.4 Extending functionality

The tools described in the literature section, such as [11, 14], provide very useful functionality that complement what our tool provides. How this functionality could be integrated into our tool is worth exploring.

5.5 Using higher-level Hadoop tools

Hadoop is a general framework for programming – powerful but with a complex API for programming. Higher level tools have been built on top of Hadoop. In particular, HBASE, Hive and HadoopDB are all systems that are built on top of Hadoop to provide data



PH2 Search PDB

Amino Acid 1 Leu versus Amino Acid 2 Asn

Constraints

1. Amino acids are on [✓ different chains] same chains chains
2. Constraints on atoms. Number of atom constraints: 3

	Amino acid 1	*Amino acid 2*	Distance range between (angstroms)	
Constraint 1	CA	CA	7 Å and	12 Å
Constraint 2	CA	CB	5 Å and	14 Å
Constraint 3	O	OD2	9 Å and	11 Å
Constraint 4			Å and	Å

Search PDB

Figure 7: GUI Mockup: This shows a simple web-based interface that could generate the required SQL as a web-service. This example shows the query: *find proteins in which leucine is on one chain and aspargine on another chain and the carbon alpha atom of the leucine is within some distance of the carbon alpha of the aspargine and some other distance of the carbon beta of the aspargine, and the oxygen atom of the leucine is within some other distance of the oxygen delta 4 atom of the aspargine. A simple CGI script could generate SQL queries of the form in Figure 6 easily. The challenge is to generalise this while keeping the user interface intuitive.*

management and querying facilities. They do not provide the querying capacity of SQL, but they support large scale data storage and a high-level way to do queries. In particular, the partitioned nature of the data would support the paradigms of these frameworks.

Acknowledgement

The research was conducted on the Hadoop cluster made available by IBM and Google as part of the IBM/Google Cloud Computing University Initiative. I thank Sean McLean, Angie Krackeler and Eugene Hung of IBM for their help. My thanks to Elevation Biotech, and in particular Grant Napier, Nichole Kinsey and Alexio Capovilla. Abdelkrim Rachedi of Wits Bioinformatics read a draft of this paper and made useful suggestions. Anonymous referees also made useful suggestions. This work was partially supported by the National Research Foundation of South Africa. Thapelo Agus, Richard Klein and Phetsa Ndlangamandla worked on as interns for the National Bioinformatics Network on an earlier version of the tool. I thank these students and Yair Mark for interesting discussions.

6. REFERENCES

[1] T. Agus, R. Klein, and P. Ndlangamandla. PDB Data Miner. Unpublished code, 2008.

[2] P. Ananthalakshmi, K. Samayamohan, C. Chokalingam, C. Mayilarasi, and K. Sekar. PSST-2.0: Protein Data Bank sequence search tool. *Applied Bioinformatics*, 4(2):141–5, 2005.

[3] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, Jan. 2000.

[4] J. Cohen. Bioinformatics — an introduction for computer scientists. *ACM Computing Surveys*, 36(2):122–158, 2004.

[5] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[6] I. Eidhammer, I. Jonassen, and W. Taylor. Structure Comparison and Structure Patterns. *Journal of Computational Biology*, 7(5):685–716, Oct. 2000.

[7] S. Ghemawat and H. G. S.-T. Leung. The Google File System. In *SOSP '03: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 29–43, New York, NY, USA, 2003. ACM.

[8] R. Holland, T. Down, M. Pocock, A. Prlić, D. Huen, K. James, S. Foisy, A. Dräger, A. Yates, M. Heuer, and M. Schreiber. BioJava: an open-source framework for bioinformatics. *Bioinformatics*, 24(18):2096–7, Sept. 2008.

[9] L. Hunter. *Molecular Biology for Computer Scientists*, pages 1–46. MIT Press, 1993.

[10] J. Kirchmair, P. Markt, S. Disinto, D. Schuster, G. Spitzer, K. Liedel, T. Langer, and G. Wolber. The protein data bank (PDB), its related services and software tools as key components for in silico guided drug discovery. *Journal of Medicinal Chemistry*, 51(22):7021–7040, Oct. 2008.

[11] T. Margraf, G. Schenk, and A. Torda. The SALAMI protein structure search server. *Nucleic Acids Research*, 37(Web Server issue):W480–4, July 2009.

[12] Y. Mark. Parallel-PDB: OpenMP for Bioinformatics. Honours Research Report, School of Computer Science, University of the Witwatersrand, 2009.

[13] E. Pryor and J. Fetrow. Pdb-sql: a storage engine for macromolecular data. In *ACM-SE 45: Proceedings of the 45th Annual Southeast Regional Conference*, pages 260–265, New York, NY, USA, 2007. ACM.

[14] A. Samson and M. Levitt. Protein segment finder: an online search engine for segment motifs in the PDB. *Nucleic Acids Research*, 37(Database Issue):D224–D–228, 2009.

[15] H. Täubig, A. Buchner, and J. Griebsch. PAST: fast structure-based searching in the PDB. *Nucleic Acids Research*, 34:W20–3, July 2006.

[16] J. Venner. *Pro Hadoop.* Apress, 2009.

[17] T. White. *Hadoop: The Definitive Guide.* O'Reilly Media, 2009.

[18] Y. Ye and A. Godzik. FATCAT: a web server for flexible structure comparison and structure similarity searching. *Nucleic Acids Research*, 32:W582–5, July 2004.