molecules

MDPI

Article

Scalable Extraction of Big Macromolecular Data in Azure Data Lake Environment

Dariusz Mrozek *, Tomasz Dąbek and Bożena Małysiak-Mrozek

Institute of Informatics, Silesian University of Technology, Akademicka 16, 44-100 Gliwice, Poland; dabek.t@gmail.com (T.D.); bozena.malysiak@polsl.pl (B.M.-M.)

* Correspondence: dariusz.mrozek@polsl.pl; Tel.: +48-32-237-1339

Received: 8 November 2018; Accepted: 1 January 2019; Published: 5 January 2019

Abstract: Calculation of structural features of proteins, nucleic acids, and nucleic acid-protein complexes on the basis of their geometries and studying various interactions within these macromolecules, for which high-resolution structures are stored in Protein Data Bank (PDB), require parsing and extraction of suitable data stored in text files. To perform these operations on large scale in the face of the growing amount of macromolecular data in public repositories, we propose to perform them in the distributed environment of Azure Data Lake and scale the calculations on the Cloud. In this paper, we present dedicated data extractors for PDB files that can be used in various types of calculations performed over protein and nucleic acids structures in the Azure Data Lake. Results of our tests show that the Cloud storage space occupied by the macromolecular data can be successfully reduced by using compression of PDB files without significant loss of data processing efficiency. Moreover, our experiments show that the performed calculations can be significantly accelerated when using large sequential files for storing macromolecular data and by parallelizing the calculations and data extractions that precede them. Finally, the paper shows how all the calculations can be performed in a declarative way in U-SQL scripts for Data Lake Analytics.

Keywords: data processing; Cloud computing; Big Data; data extraction; data lake; parallel computing; querying; proteins; nucleic acids, macromolecules; 3D structure; structural bioinformatics

1. Introduction

Parsing macromolecular data files is one of the first processes preceding many operations performed on 3D structures of proteins and nucleic acids, including statistical calculations over geometry of proteins, investigations of protein inter-residue contacts, docking prediction, 3D protein structure alignment, studying intra-protein interactions that stabilize protein molecules, like disulphide bonds, aromatic-aromatic interactions, sulphur–aromatic interactions, ionic interactions. Macromolecular data describing 3D structures of proteins and nucleic acids are usually stored in files that have various formats. Protein Data Bank (PDB) [1], the most popular repository established to collect data describing 3D structures of macromolecules, stores and exchanges the data in three formats—PDB [2], mmCIF [3], and PDBML [4]. All of them are text files with sections and records that hold a particular type of information. The descriptions of macromolecular structures in these files include primarily their geometries, but also many other features that determine the physical and chemical properties of DNA, RNA, and protein molecules. These features can be directly extracted from the files or derived by calculations on the basis of macromolecular data stored in them.

However, the exponential growth of macromolecular data in the Protein Data Bank may pose pressure on the existing hardware equipment, as compute capabilities of desktop computers are usually limited [5]. With this hardware, some calculations performed over protein geometries and atomic interactions can be successfully completed in several minutes only for smaller collections of

macromolecular data. However, as the repositories grow quickly every year, desktop computers may become a bottleneck in the entire discovery process [6,7]. This results in focusing on the use of various scalable platforms that would support efficient data processing and calculations [8–10]. Taking into account the complex nature of biological data, including macromolecular data of proteins, various storage formats for the data, the growing amount of the data, and finally, the complexity of some calculation processes performed over the data, we may find out that we meet challenges of Big Data processing and analysis [11–13]. Performing many calculations with 3D structures of proteins and nucleic acids meets the 5V model of Big Data at least in terms of the *volume* and the *variety* of data, and maybe for some calculations (e.g., those related to drug design) also the *value*. This leads to the use of various Big Data platforms, like Hadoop [14] or Spark [15], to perform these calculations for large collections of data, and make us switching to Cloud computing platforms that enable almost unlimited compute resources for scaling the calculations broadly [16].

In this paper, we show how parsing and extracting big macromolecular data of proteins and nucleic acids can be effectively performed in highly scalable Azure Data Lake cloud environment. The solution presented here mitigates the problem of limited compute resources of desktop computers, in terms of data storing and processing. Moreover, by using the declarative U-SQL query language we now simplify the manipulation of macromolecular data and performing calculations over 3D structures of macromolecules. Together with the methods, in the paper, we show sample U-SQL scripts that can be used in scalable, parallel calculations performed in the Azure Data Lake environment.

2. Related Works

The spectrum of calculations performed over 3D structures of macromolecules, including proteins, can be very broad. It may include various exploration operations that are based on simple calculations on particular atoms, but also more complex calculations related to structure similarity searching, structural alignment, structural superposition, or computational protein structure determination through prediction. Among the tools that were reported for the exploration of various features of proteins and their geometries, there have been developed several ones that are focused on the recognition and the analysis of different types of interactions in proteins carried out on experimentally determined, high-resolution protein structures from the Protein Data Bank. For example, the protein interactions calculator (PIC) server reported by Tina et al. [17] was developed for studying different types of interactions that occur within a given protein structure. The aromatic-aromatic Interactions Database (A2ID) reported by Chourasia et al. [18] allows studying the aromatic-aromatic networks within proteins. IntGeom [19] can be used for the calculation of interaction geometry between planar groups in proteins. Mentioned examples, however, neither focus on the performance of the operations nor provide large flexibility on performed data explorations.

In this regard and in terms of big data processing, interesting ideas were proposed by Hazelhurst in [20]. In the work, the author reported a Hadoop-based PH2 system that enables the exploration of various features of 3D protein structures, e.g., calculation of distances between particular atoms within single protein structures. For this purpose, the system uses the Structured Query Language (SQL) [21] as a query language, which provides flexibility in querying macromolecular data. 3D protein structures, as raw PDB files, are stored in a replicated way on the Hadoop Distributed File System (HDFS). PH2 system relies on massive parallelism provided by Hadoop computational framework in order to improve the efficiency of the exploration process. SpeeDB reported by Robillard et al. in [22] also tackles similar problems. It provides in-memory database structure to investigate various types of interactions in proteins, including hydrogen bonds, ionic interactions, disulfide bonds, and aromatic interactions but seems to be less flexible in the exploration capabilities. In-memory protein structure management system was also used by us in the IMPSMS for fast calculations over 3D protein structures [23]. However, the system suffered the problem of limited memory, leading to a limited number of proteins that could be stored in it.

Since in the paper we show a solution for various types of calculations performed over protein structures that works on the basis of declarative queries, it is also worth mentioning SQL-based approaches for querying protein data stored in relational databases. For example, procedural extensions to Oracle RDBMS for aligning and matching protein sequences, called the ODM BLAST, were reported in [24]. The BioSQL [25], which incorporates modules of the BioJava project [26], focuses on biomolecular sequences and features, their annotation, a reference taxonomy, and ontologies. Several extensions to the SQL language, including PSS-SQL [27,28] and the query language developed by Hammel and Patel [29] and Tata et al. [30], were proposed for searching protein similarities on the basis of protein secondary structures. These works show how protein data can be stored in relational tables. They also present searching techniques that can be applied to explore the data, and how these data are indexed in order to speed up the searching process. Although these extensions allow to operate on the primary (ODM BLAST) and the secondary structures of proteins (the rest), these tools do not directly address processing 3D protein structures. In terms of processing and querying 3D protein structures in relational databases, Mrozek et al. [31] developed the P3D-SQL extension for the Oracle PL/SQL language that allows invoking 3D protein structure similarity searching in SQL queries and performing the process against the whole relational database of 3D protein structures. However, the performance of the solution is worse than the performance of the process executed on raw PDB files stored on hard disc drives (HDD), since the process is executed within Oracle memory pool, which is also limited.

To overcome the problems of limited computational power and limited memory, several cloud-based solutions for exploration processes performed over protein structures were proposed. For example, the system developed by Che-Lun Hung and Yaw-Ling Lin [32] uses Hadoop-based implementations of two popular fold-based alignment methods—DALI [33] and VAST [34]. The system is scaled on a private cloud. Hadoop and the MapReduce processing model is also used in our previous works [35–37] for the same purpose. These works also show that the use of sequential files (instead of processing individual structures) may increase the performance of parallel protein structure similarity searches. Sequential files will also be used in the approach presented in the paper. Dedicated cloud-based architectures were also developed for scalable protein structure similarity searching [38–41] and protein structure prediction [42,43]. These works prove that cloud resources may significantly simplify and accelerate many calculations related to protein structures.

The approach presented in the paper combines wide scaling capabilities offered by the Cloud computing model, Big Data techniques for efficient data processing, and declarative capabilities of SQL-based solutions that simplify various data explorations by querying Big macromolecular Data sets.

3. Methods and Technologies

To mitigate the above-mention problems we have developed the *PDBUSQLExtractor* that allows parsing and extracting data from PDB files describing structures of proteins, nucleic acids, and their complexes. In this section, we provide implementation details of the scalable parser for the PDB macromolecular data files developed for the Azure Data Lake.

3.1. Extensions to the Azure Data Lake Environment

Our *PDBUSQLExtractor* parser with all extraction methods was developed for the Azure Data Lake environment. Azure Data Lake (ADL) is a scalable, cloud environment that enables storing and analyzing large data sets. The interactive batch analysis in the ADL is possible in real time for various types of data, including structured, semi-structured, and unstructured data [44]. The architecture of our solution for extracting and processing Big macromolecular Data in Azure Data Lake is presented in Figure 1. The Azure Data Lake consists of two main components:

1. Data Lake Store (DLS), which constitutes a petabyte scale, unlimited storage for a domain-related data lake, in which large collections of data located in various files are distributed across

many storage servers. This enables performing read operations in parallel and improves the performance of data read operations.

2. Data Lake Analytics (DLA), which enables efficient and scalable analysis of data stored in Big Data Lakes by parallelizing the analysis on a distributed infrastructure in the Azure cloud. It provides the U-SQL compiler and distributed execution environment for declarative processing and analysis of data stored in the Data Lake Store.

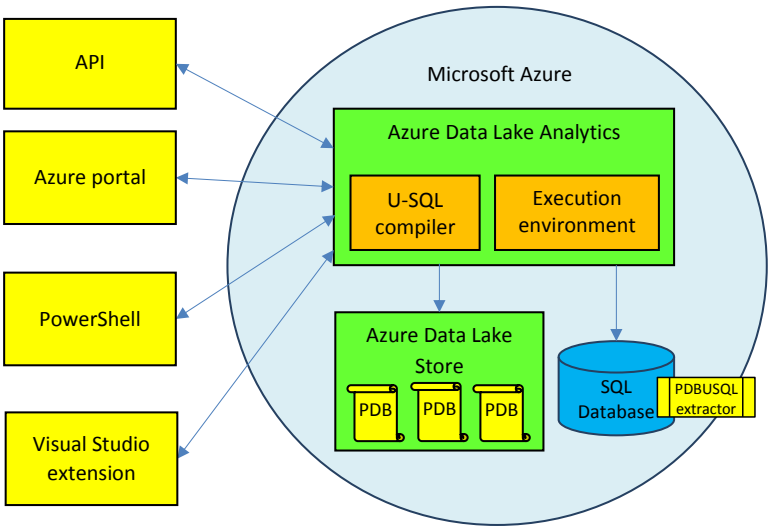

Figure 1. Architecture of our solution for extracting and processing Big macromolecular Data in the Azure Data Lake environment with the *PDBUSQLExtractor*. PDB files describing macromolecular data of proteins and nucleic acids are stored in the Data Lake Store. Efficient processing occurs in the Data Lake Analytics equipped with *PDBUSQLExtractor* library.

The U-SQL is a big data query language that combines the declarative capabilities of the SQL query language and expressive power of C# code. Data analysis, preceded by extraction, processing, and transformation of data from the data lake, is performed with the use of the U-SQL scripts containing query expressions. U-SQL scripts can be executed through four available channels, including application programming interfaces (APIs), Azure portal, PowerShell environment, and Visual Studio programming platform. U-SQL programmers and data analysts may use various expressions in data analyses they perform, including the most popular SELECT expression, and also PROCESS, REDUCE, and COMBINE expressions that apply custom or user-defined operators (UDOs). These query expressions produce rowsets that can be assigned to rowset variables. The rowset variables are populated in the EXTRACT U-SQL phrase with the use of appropriate *data extractor*. Built-in extractors, like CSV, TSV, or Text can be used for data extraction. However, specific file formats, like the PDB, require dedicated, custom extractors that know how to extract the data, how a single record looks like, how atomic the information stored in it is, and how to read it. We have developed the *PDBUSQLExtractor* for parsing and extracting data from PDB macromolecular data files describing structures of proteins and nucleic acids. Custom extractors, as .NET assemblies are stored in the Azure SQL Database. Each execution of the U-SQL script invoking the *PDBUSQLExtractor* causes loading the extractor-related DLL library into the execution environment of the Azure Data Lake Analytics. Macromolecular data of proteins and nucleic acids are stored in the Azure Data Lake Store. The Data Lake Analytics has wide access to the data stored in the repository (DLS). Results of data extraction and processing are also stored in the Data Lake Store by invoking the U-SQL OUTPUT expression that uses appropriate *data outputter*. Likewise in the EXTRACT phase, data outputters can be built-in or custom.

3.2. Setting Up the Azure Data Lake for Scalable Extraction of Macromolecular Data

To use the *PDBUSQLExtractor* a user must initially complete 3 steps:

1. set up the Azure Data Lake environment,
2. upload PDB files with macromolecular structures into the Azure Data Lake Store,
3. register the *PDBUSQLExtractor* library in the Azure Data Lake Analytics.

All of the mentioned operations can be completed through the Azure portal, Azure PowerShell, Azure CLI, or appropriate Application Programming Interfaces (APIs). Users should follow the Azure Data Lake Analytics Documentation [45] for the particular method. Setting up the Azure Data Lake environment in the Microsoft Azure cloud (step 1) can be easily done through the Azure portal (http://portal.azure.com/), which provides a graphical interface that simplifies the management and configuration of the ADL platform. The portal is available through web browsers in any operating system (OS). What is important, users must set up the ADL environment within their own Azure subscriptions, since they will bear the costs of using the platform (costs of storing the data and costs of computations performed). The same portal can be used for uploading the processed and analyzed data describing macromolecular structures (step 2). However, in the case of many files with macromolecular data, the preferable way may be using one of the command-line tools, like the Azure PowerShell or the Azure CLI. Azure PowerShell is an extension of Windows PowerShell that provides cmdlets for simplifying and automating the management of Azure cloud services. Azure CLI is another Microsoft's command-line tool for managing Azure resources, but in contrast to the Azure PowerShell, it can operate on various OS platforms. Registering the *PDBUSQLExtractor* library (*PDBUSQLExtractor.dll*, please refer to the Availability section at the end of the paper) in Azure Data Lake environment (step 3) is necessary for using the created data extractors in U-SQL scripts, since the library extends standard capabilities of the ADL environment. The library must be registered as an assembly in one of the Azure SQL Databases (*master* by default) available in the Azure Data Lake environment. Code editors, like Microsoft Visual Studio, greatly facilitate the library registering task, though all mentioned methods can be used for this.

After completing the three necessary steps, the user can start developing the U-SQL scripts for data extraction and secondary data analysis. For the development of the U-SQL scripts, he can use any code editor, preferably Microsoft Visual Studio or Visual Studio Code. The latter one is an open-source and free source code editor developed by Microsoft for Windows, Linux, and macOS. Both editors include support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. The developed U-SQL scripts can be executed in the Cloud, in the configured Azure Data Lake environment, or tested locally. Cloud executions allow for parallelization of computations related to data extraction and secondary analysis and scaling the computations on many compute units. Local executions, used for testing the U-SQL scripts, are supported by Azure Data Lake Tools for Visual Studio or by Azure Data Lake U-SQL SDK.

3.3. Modules and Methods for Parsing PDB Files

In the U-SQL scripts, parsing and extracting of macromolecular data of proteins and nucleic acids is possible by using a dedicated extractor from the created class `PDBUSQLExtractor`. The extractor is invoked in the EXTRACT U-SQL statement as it is shown in Listing 1 in line 22. In the presented data extraction, we parse and extract data from the ATOM sections of PDB files (*.ent) located in the indicated folder in the Data Lake Store. The dedicated extractor parses each *.ent file in the folder and fetches data from an appropriate section. At the moment, we have implemented the extraction process for the following sections of the PDB files:

- ATOM that presents the atomic coordinates for standard amino acids and nucleotides. It also presents the occupancy and temperature factor for each atom.

- HETATM that presents non-polymer or other non-standard chemical coordinates, such as water molecules or atoms presented in HET groups, together with the occupancy and temperature factor for each atom.
- SHEET that is used to identify the position of β-sheets in protein molecules.
- HELIX that is used to identify the position of α-helices in protein molecules.
- SEQRES that contains a listing of the consecutive chemical components (amino acids for proteins) covalently linked in a linear fashion to form a polymer.

Although we developed the *PDBUSQLExtractor* extractor mainly for parsing PDB files with macromolecular data describing 3D structures of proteins, it can be also used to extract data from PDB files describing nucleic acids and DNA/RNA-protein complexes. For the DNA/RNA molecules, it is possible to extract data from the ATOM, the HETATM, and the SEQRES sections.

Listing 1. A part of a U-SQL script parsing and extracting data from the *ATOM* sections of PDB files.

```
1  USE DATABASE [master];
2  REFERENCE ASSEMBLY [PDBUSQLExtractor];
3
4  @test =
5  EXTRACT protId string,
6  model int?,
7   serial  int ?,
8  name string,
9  altLoc  string ,
10 resNm string,
11 chain  string ,
12 resSq  int ?,
13 iCode string,
14 x  double,
15 y  double,
16 z  double,
17 occupancy double,
18 tempFact double,
19 element string,
20 charge string
21 FROM "/input/decompressed/{*.ent}"
22 USING new PDBUSQLExtractor.PDBExtractor("ATOM");
23
24 OUTPUT @test
25 TO "/output/resultatom.tsv"
26 USING Outputters.Tsv();
```

Attributes fetched during the execution of the extraction script and data types for the attributes are specified dynamically in the EXTRACT clause of the U-SQL statement (lines 5–20). They should be adjusted to the section of the PDB files that is currently extracted. While developing our dedicated extractors we followed the PDB file format specification [46]. Execution of the whole EXTRACT statement produces the @test rowset (line 4) that can be further processed and analyzed (example will be shown in Section 4) or stored in the Data Lake Store for future processing. A part of the rowset produced by the presented U-SQL script has the form shown in Figure 2 (column headers were formatted manually). Storing the produced rowset is performed in the OUTPUT statement (lines 24–26).

protId	model	serial	name	altLoc	resNm	chain	resSq	iCode	x	y	z	occupancy	tempFact	element	charge
1A00		1	N		VAL	A	1		101.601	38.534	-1.962	1	53.29	N	
1A00		2	CA		VAL	A	1		103.062	38.513	-2.159	1	47.99	C	
1A00		3	C		VAL	A	1		103.354	38.323	-3.656	1	43.68	C	
1A00		4	O		VAL	A	1		103.025	37.252	-4.204	1	48.4	O	
1A00		5	CB		VAL	A	1		103.8	37.438	-1.355	1	50.88	C	
1A00		6	CG1		VAL	A	1		105.274	37.822	-1.182	1	46.89	C	
1A00		7	CG2		VAL	A	1		103.166	37.064	-0.034	1	61.95	C	
1A00		8	N		LEU	A	2		103.962	39.357	-4.22	1	40.4	N	
1A00		9	CA		LEU	A	2		104.276	39.328	-5.665	1	35.83	C	
1A00		10	C		LEU	A	2		105.482	38.451	-5.963	1	34.34	C	
1A00		11	O		LEU	A	2		106.575	38.644	-5.401	1	38.15	O	
1A00		12	CB		LEU	A	2		104.396	40.769	-6.186	1	19.71	C	
1A00		13	CG		LEU	A	2		103.175	41.676	-6.005	1	16.01	C	
1A00		14	CD1		LEU	A	2		103.509	43.08	-6.464	1	25.3	C	
1A00		15	CD2		LEU	A	2		102.047	41.113	-6.821	1	13.69	C	
...															
.															
.															
.															
1A5E	1	1	N		MET	A	1		127.117	33.271	-15.101	1	0	N	
1A5E	1	2	CA		MET	A	1		126.476	34.059	-14.008	1	0	C	
1A5E	1	3	C		MET	A	1		124.981	34.233	-14.286	1	0	C	
1A5E	1	4	O		MET	A	1		124.194	34.449	-13.383	1	0	O	
1A5E	1	5	CB		MET	A	1		127.188	35.411	-14.022	1	0	C	
1A5E	1	6	CG		MET	A	1		128.474	35.312	-13.199	1	0	C	
1A5E	1	7	SD		MET	A	1		129.210	36.956	-13.022	1	0	S	
1A5E	1	8	CE		MET	A	1		127.941	37.662	-11.940	1	0	C	
1A5E	1	9	H1		MET	A	1		126.562	32.411	-15.279	1	0	H	
1A5E	1	10	H2		MET	A	1		127.153	33.844	-15.969	1	0	H	
1A5E	1	11	H3		MET	A	1		128.083	33.008	-14.818	1	0	H	
1A5E	1	12	HA		MET	A	1		126.626	33.574	-13.055	1	0	H	
1A5E	1	13	HB2		MET	A	1		127.429	35.678	-15.041	1	0	H	
1A5E	1	14	HB3		MET	A	1		126.543	36.163	-13.594	1	0	H	
1A5E	1	15	HG2		MET	A	1		128.246	34.912	-12.220	1	0	H	
1A5E	1	16	HG3		MET	A	1		129.172	34.659	-13.700	1	0	H	
1A5E	1	17	HE1		MET	A	1		127.058	37.039	-11.965	1	0	H	
1A5E	1	18	HE2		MET	A	1		128.314	37.712	-10.930	1	0	H	
1A5E	1	19	HE3		MET	A	1		127.695	38.658	-12.280	1	0	H	
...															
.															
.															
.															

Figure 2. Results of the extraction of the ATOM section performed with the U-SQL script from Listing 1 showing a part of produced rowset.

Similar scripts can be implemented to extract other sections of the PDB macromolecular data files for proteins, nucleic acids, and nucleic acid-protein complexes. For example, Listing 2 shows a sample script for parsing and extracting data from the `SEQRES` sections of PDB files. The only difference from the previous script is the `SEQRES` value of the parameter of the *PDBExtractor* used in the `USING` clause of the `EXTRACT` statement (line 23) and attributes fetched in the `EXTRACT` clause (lines 5–21). The data from the `SEQRES` records of the PDB files are extracted according to the PDB file format documentation and the list of attributes contains: a serial number of the `SEQRES` record for the current chain (`serNum`), chain identifier (`chainId`), number of residues in the chain (`numRes`), and names of residues returned in thirteen following columns (`resName1`–`resName13`). Each returned row in the produced rowset (`@test`, line 4) is preceded by the protein PDB ID identifier (`proteinId`) since we process many PDB files.

Listing 2. Sample U-SQL script parsing and extracting data from the *SEQRES* sections of PDB files.

```
1  USE DATABASE [master];
2  REFERENCE ASSEMBLY [PDBUSQLExtractor];
3
4  @test =
5  EXTRACT proteinId string,
6  serNum int?,
7  chainId string,
8  numRes int?,
9  resName1 string,
10 resName2 string,
11 resName3 string,
12 resName4 string,
13 resName5 string,
14 resName6 string,
15 resName7 string,
16 resName8 string,
17 resName9 string,
18 resName10 string,
19 resName11 string,
20 resName12 string,
21 resName13 string
22 FROM "/input/decompressed/{*.ent}"
23 USING new PDBUSQLExtractor.PDBExtractor("SEQRES");
24
25 OUTPUT @test
26 TO "/output/resultatom.tsv"
27 USING Outputters.Tsv();
```

Execution of the sample script produces the `@test` rowset (line 4). Attributes of the rowset that are needed for further analysis can be selected in the SELECT U-SQL expression (example will be shown in Section 4). A part of the rowset produced by the presented U-SQL script has the the form shown in Figure 3.

```
...
 5YYN  1 A 586 HIS HIS HIS HIS HIS HIS GLY GLY ALA MET ASN ILE GLN
 5YYN  2 A 586 ALA LEU LEU SER GLU LYS VAL ARG GLN ALA MET ILE ALA
 5YYN  3 A 586 ALA GLY ALA PRO ALA ASP CYS GLU PRO GLN VAL ARG GLN
...
 5YYN  1 B  77   G   C   G   C   U   C   G   U   A   G   C   U   C
 5YYN  2 B  77   A   A   U   U   G   G   A   U   A   G   A   G   C
 5YYN  3 B  77   A   U   C   U   G   A   C   U   A   C   G   G   A
 5YYN  4 B  77   U   C   A   G   A   A   G   G   U   U   A   G   G
 5YYN  5 B  77   G   G   U   U   C   G   A   A   U   C   C   U   C
 5YYN  6 B  77   U   C   G   A   G   C   G   C   G   C   C   A
 5YYN  1 C 586 HIS HIS HIS HIS HIS HIS GLY GLY ALA MET ASN ILE GLN
 5YYN  2 C 586 ALA LEU LEU SER GLU LYS VAL ARG GLN ALA MET ILE ALA
 5YYN  3 C 586 ALA GLY ALA PRO ALA ASP CYS GLU PRO GLN VAL ARG GLN
...
 6DCB  1 A 309 MET GLY SER SER HIS HIS HIS HIS HIS HIS SER SER GLY
 6DCB  2 A 309 LEU VAL PRO ARG GLY SER PRO LEU PRO ALA ALA GLY PHE
 6DCB  3 A 309 LYS LYS GLN GLN ARG LYS PHE GLN TYR GLY ASN TYR CYS
 6DCB  4 A 309 LYS TYR TYR GLY TYR ARG ASN PRO SER CYS GLU ASP GLY
 6DCB  5 A 309 ARG LEU ARG VAL LEU LYS PRO GLU TRP PHE ARG GLY ARG
 6DCB  6 A 309 ASP VAL LEU ASP LEU GLY CYS ASN VAL GLY HIS LEU THR
...
```

Figure 3. Results of the extraction of the *SEQRES* section performed with the U-SQL script from Listing 2 showing a part of produced rowset.

3.4. Optimizations

The *PDBUSQLExtractor* parser and extractor was developed in such a way that it supports space-optimized and performance-optimized storage of macromolecular data in the Cloud. In terms of use of storage space for macromolecular data, the extractor allows to work with:

- decompressed PDB macromolecular files,

- compressed PDB macromolecular files.

Using the compression allows reducing the storage space consumed on the Cloud, which has a positive impact on the costs of the Cloud storage. The *PDBUSQLExtractor* supports the standard *.gz compression for the PDB files. However, those users that possess the collections of decompressed macromolecular data in the form of PDB text files may also use the parser in data extraction and processing.

In terms of performance of the parsing and data extraction process, the *PDBUSQLExtractor* works on two types of data:

- individual macromolecular data files that are extracted in parallel,
- joined, sequential macromolecular data files that are also extracted in parallel.

We assume that in a regular approach the user operates on the repository of standard, individual PDB macromolecular data files. These files will be extracted in parallel according to the parallelism factor set by the user while executing the U-SQL script in the Azure Data Lake environment. However, we also noticed that much better performance can be gained while processing the macromolecular data assembled in sequential macromolecular data files of larger sizes. Therefore, we also gave the possibility to process the data in such a way for these users that prioritize the performance.

A part of a sample sequential file with several protein structures (in the PDB format) treated as single records in the ADL-based extraction and parsing is shown in Figure 4. The developed extraction method of the *PDBUSQLExtractor* parser uses the END section of each PDB data set to recognize the end of the record in the sequential file. Such a sequential file can be simply produced by using the *copy* command of the Microsoft Windows command line console.

Extraction of the sequential files is performed with the use of the dedicated `PDBConcatExtractor` extractor, invoked in the `USING` clause of the `EXTRACT` statement in the U-SQL processing script, as it is presented in Listing 3, line 8. The extractor accepts the same parameters as the `PDBExtractor` extractor used for processing macromolecular data stored in individual PDB files.

Listing 3. Extraction of the *ATOM* section data from PDB data sets assembled in sequential files with the use of the *PDBConcatExtractor* extractor.

```
1  USE DATABASE [master];
2  REFERENCE ASSEMBLY [PDBUSQLExtractor];
3
4  @test =
5  EXTRACT proteinId string,
6   ...
7  FROM "/input/pdbdataset.ent.gz"
8  USING new PDBUSQLExtractor.PDBConcatExtractor("ATOM");
9  OUTPUT @test
10 TO "/output/resultatom.txt"
11 USING Outputters.Tsv();
```

The advantage of the processing with the use of sequential files is that compute units used by the Data Lake Analytics for parallel extraction and processing will be initiated once. The initiation time is negligible in relation to the extraction and processing time of macromolecular data. The limitation of this solution is the maximum size of the row (record) the extractor can process, which is fixed to 4 MB. This means that the input collection should be divided into files of the size larger than 4 MB and those that are smaller than 4 MB. The smaller files should be concatenated into larger sequential files, and the larger ones should be extracted individually.

Both types of files, individual and sequential, can be also stored as compressed and decompressed in the Azure Data Lake Storage and processed in the Data Lake Analytics. Influence of this optimization on the performance of the extraction will be studied in Section 4.

```
HEADER    OXYGEN TRANSPORT                        13-DEC-97   101M
TITLE     SPERM WHALE MYOGLOBIN F46V N-BUTYL ISOCYANIDE AT PH 9.0
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: MYOGLOBIN;
COMPND   3 CHAIN: A;
COMPND   4 ENGINEERED: YES;
COMPND   5 MUTATION: YES
...
ATOM      1  N   MET A   0      24.277   8.374  -9.854  1.00 38.41           N
ATOM      2  CA  MET A   0      24.404   9.859  -9.939  1.00 37.90           C
ATOM      3  C   MET A   0      25.814  10.249 -10.359  1.00 36.65           C
ATOM      4  O   MET A   0      26.748   9.469 -10.197  1.00 37.13           O
ATOM      5  CB  MET A   0      24.070  10.495  -8.596  1.00 39.58           C
ATOM      6  CG  MET A   0      24.880   9.939  -7.442  1.00 41.49           C
ATOM      7  SD  MET A   0      24.262  10.555  -5.873  1.00 44.70           S
ATOM      8  CE  MET A   0      24.822  12.266  -5.967  1.00 41.59           C
...
.
.
.
END
HEADER    HYDROLASE(O-GLYCOSYL)                   29-SEP-92   102L
TITLE     HOW AMINO-ACID INSERTIONS ARE ALLOWED IN AN ALPHA-HELIX OF
TITLE    2 T4 LYSOZYME
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: T4 LYSOZYME;
COMPND   3 CHAIN: A;
COMPND   4 ENGINEERED: YES
...
ATOM      1  N   MET A   1      44.061  -3.277   8.755  1.00 22.03           N
ATOM      2  CA  MET A   1      43.619  -1.924   8.869  1.00 14.21           C
ATOM      3  C   MET A   1      42.195  -1.991   9.394  1.00 18.00           C
ATOM      4  O   MET A   1      41.523  -2.983   9.193  1.00 16.68           O
ATOM      5  CB  MET A   1      43.727  -1.353   7.458  1.00 17.98           C
ATOM      6  CG  MET A   1      42.994  -0.069   7.291  1.00 31.76           C
ATOM      7  SD  MET A   1      44.005   1.333   7.794  1.00 35.36           S
ATOM      8  CE  MET A   1      45.515   0.985   6.912  1.00 39.74           C
...
.
.
.
END
HEADER    MERCURY DETOXIFICATION                  07-MAR-97   1AFJ
TITLE     STRUCTURE OF THE MERCURY-BOUND FORM OF MERP, THE
TITLE    2 PERIPLASMIC PROTEIN FROM THE BACTERIAL MERCURY
TITLE    3 DETOXIFICATION SYSTEM, NMR, 20 STRUCTURES
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: MERP;
COMPND   3 CHAIN: A;
...
ATOM      1  N   ALA A   1     -17.870   2.452   0.507  1.00  0.00           N
ATOM      2  CA  ALA A   1     -16.847   1.374   0.640  1.00  0.00           C
ATOM      3  C   ALA A   1     -15.859   1.437  -0.526  1.00  0.00           C
ATOM      4  O   ALA A   1     -15.662   0.474  -1.235  1.00  0.00           O
ATOM      5  CB  ALA A   1     -17.639   0.072   0.594  1.00  0.00           C
ATOM      6  H1  ALA A   1     -18.232   2.469  -0.468  1.00  0.00           H
ATOM      7  H2  ALA A   1     -18.656   2.267   1.162  1.00  0.00           H
ATOM      8  H3  ALA A   1     -17.437   3.370   0.734  1.00  0.00           H
ATOM      9  HA  ALA A   1     -16.330   1.459   1.581  1.00  0.00           H
ATOM     10  HB1 ALA A   1     -18.248   0.054  -0.296  1.00  0.00           H
ATOM     11  HB2 ALA A   1     -16.955  -0.764   0.581  1.00  0.00           H
ATOM     12  HB3 ALA A   1     -18.272   0.006   1.466  1.00  0.00           H
...
.
.
.
END
```

Figure 4. A part of a sample sequential file with many protein structures stored as records for extraction by the *PDBUSQLExtractor* parallel parser. Each `END` section determines the border for a single record processed in the U-SQL script for Data Lake Analytics.

4. Results

Azure Data Lake allows parallelizing many jobs related to data processing. Parallelization of macromolecular data extraction should bring significant improvements in the performance of the *PDBUSQLExtractor* and reduction of the execution time. We tested the reduction of the execution time in a series of experiments performed in the Azure Data Lake environment.

4.1. Experimental Setup

The performance of the *PDBUSQLExtractor* was tested with the use of macromolecular data of 3D protein structures taken from the Protein Data Bank. We used two data sets in our tests:

- data set DS1 containing 1475 files,
- data set DS2 containing 14,750 files.

The amount of data constituted 1% (DS1) and 10% (DS2) of 3D structures stored in the repository at the time of experiments. Data extractions were performed for three different sections of PDB files, i.e.,:

- ATOM,
- SHEET,
- SEQRES.

Tests were performed with compressed and decompressed PDB macromolecular files stored as individual files and sequential files in all combinations. This gave four scenarios of performed experiments:

- decompressed-individual files (DI),
- compressed-individual files (CI),
- decompressed-sequential files (DS),
- compressed-sequential files (CS).

Parallelization was regulated by the *parallelization factor*, which was changed from one to 1475. This means that in the performance experiments we changed the number of compute units, in Azure Data Lake called *Allocation Units* (AUs), performing extractions and calculations from one up to 1475 (exactly, $\#AU = 1, 2, 8, 32, 128, 512, 1024, 1475$).

To compare obtained results with traditional, local processing, we also used a PC workstation with CPU Core i7 4700MQ 2.4GHz (4 cores, 8 threads), RAM 16GB, storage HDD 1TB, working under control of the Microsoft Windows 7 64-bit operating system for performing pure extractions of ATOM sections of processed data sets.

4.2. Execution Times

In the first series of tests, we verified the performance of the extraction process for 3D protein structures stored separately in individual files. This is the standard format that scientists from the whole world can use when downloading macromolecular data of proteins from the Protein Data Bank repository. In this series of tests, we used the data set DS1 with 1475 protein structures (files) and scaled the extraction process from 1 up to 1475 allocation units (AUs) used by the Azure Data Lake while executing U-SQL extraction scripts. Results of these tests are presented in Table 1.

Table 1. Execution time (s) for local (PC) and distributed (Azure Data Lake) extractions of various sections of the PDB files and the varying number of allocation units (AUs) for 3D protein structures stored separately as compressed (CI) and decompressed (DI) individual files.

	Azure Data Lake								**PC**
# AUs	**1**	**2**	**8**	**32**	**128**	**512**	**1024**	**1475**	**1**
DI-ATOM	12,400	6225	1606	459	171	100	94	94	464
CI-ATOM	11,740	5892	1526	441	170	103	97	97	516
DI-SHEET	12,225	6115	1548	412	128	60	60	60	–
CI-SHEET	11,793	5898	1484	391	120	60	56	56	–
DI-SEQRES	11,831	5922	1494	393	129	68	61	61	–
CI-SEQRES	11,746	5879	1482	396	124	57	57	57	–

As can be observed in Table 1 the execution time gradually decreases with the number of AUs (increasing parallelization factor). The execution time consists of the extraction time, the time needed for aggregating data from parallel streams, and the time of storing results. For a single AU, it takes more than 3 hours to extract appropriate sections and store extracted data in a new CSV file. The execution time is much better for 32 AUs—the extraction, aggregation, and storing data takes between 6 and 8 min (391 s to 459 s, depending on the section and compression used). For 1024 AUs the process takes less than 2 min (e.g., 97 s for the CI-ATOM). Extraction of the ATOM sections takes more time than the extraction from other sections. This is due to the larger amount of data that must be extracted, aggregated, and finally, stored after the extraction. Generally, the extraction phase is parallelized better than aggregating and storing data. Therefore, speedup gains are not huge for the increasing number of used AUs. Shapes of the n-fold speedup curves can be observed in Figure 5.

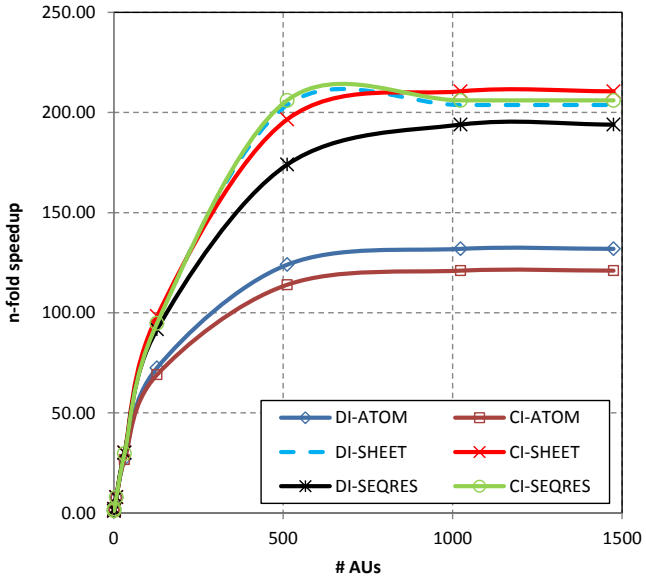

Figure 5. n-fold speedups achieved when extracting data from various sections (ATOM, SHEET, SEQRES) of compressed (CI) and decompressed (DI) individually stored PDB files for the varying number of allocation units (AUs).

More details on the use of assigned compute resources can be seen in Figure 6. Figure 6a shows that after parallelizing the process on 1024 AUs most of the AUs are used only in the extraction phase in the first 20 s. The aggregation of extracted data and storing takes the rest of the 94 s taken by the whole process. When assigning less AUs, the use of the resources is more effective. For example, for the 32 AUs assigned, most of the resources are used through the whole execution process (until the 400th s, Figure 6b).

Compression of data has almost no effect on the execution time. Nevertheless, the compressed data are extracted a little bit faster than uncompressed data in most cases (e.g., 459 s for DI-ATOM and 441 s for CI-ATOM, for 32 AUs in use, see Table 1). The compression, however, has a significant impact on the storage space occupied. For example, the compressed data set DS1 used in this series of tests occupied 139 MB, and the decompressed one occupied 614 MB.

Unfortunately, we could observe that the operating time was generally long for a relatively small total data size. Although the extraction of individual PDB files is well-parallelizable, it appeared to be a time-consuming process, especially for the executions that use a small number of AUs. The same extraction process performed on the PC workstation took between 7 and 9 min (Table 1, the PC column). This execution time is comparable to the time achieved by 32 AUs processing the same data set DS1 on the Azure Data Lake. This shows that processing individual files in highly scalable Big Data environment is possible, but not impressive in terms of execution times. There are two factors

deciding about this. First of all, PDB files must be analyzed atomically. This means that the Azure Data Lake cannot split the analysis of one file between several AUs, because the relevant information, such as model number and protein ID, are stored in other sections of the file. The second factor is the specificity of the Data Lake Analytics platform. It is adapted and optimized to process large data sets. The internal implementation of the ADLA development engine favors the code written directly in the U-SQL from the code written through extensions in the C# language. This is due to the fact that the U-SQL code is compiled to the C++ language and the managed code is executed in the Common Language Runtime (CLR) virtual machine. The CLR virtual machine initialization process is relatively long. The larger the data size, the more negligible the process is. Unfortunately, in the series of tests, the average file size taken from the Protein Data Bank was 96 KB for compressed files and 417 KB for uncompressed files. Initialization of CLR virtual machines consumes most of the time of AU units. At the same time, the ADL management process allocates one analytical unit for one input stream. This results in a huge time surplus dedicated to the initialization, finalization, and cleaning of CLR machine artifacts.

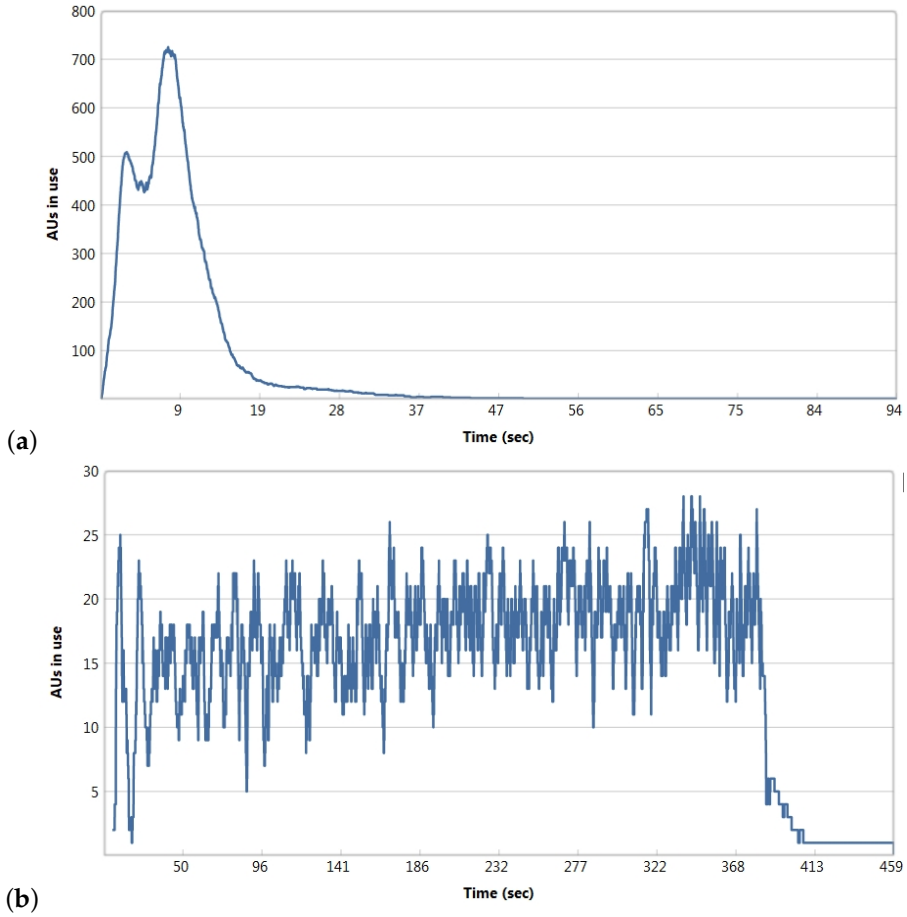

Figure 6. Use of compute resources (AUs) during the whole extraction process on data set DS1 for (**a**) 1024 AUs, and (**b**) 32 AUs assigned to the execution.

The proposed solution for this problem is to pre-process the macromolecular data and combine multiple files into sequential files, as we proposed in Section 3.4. In this case, each component PDB file of the sequential file can be treated as a separate input row for the extractor. Another advantage in this approach is the ability to opt out of the atomic processing of the sequential file. Then, compute units of the Azure Data Lake Analytics have to be initialized only once, which should additionally reduce the execution time. We checked this approach in the second series of tests.

For the second series of experiments, we created one sequential file with all macromolecular data from the data set DS1. Then, we run the data extraction process again in the Azure Data Lake. It was enough to use only one AU for the extraction process. Results of the execution are presented in Table 2. The execution time of the extraction, aggregation, and storing data in output files takes less than 2 min (116 s) when extracting the ATOM section of the PDF files, and less than 1 min when extracting data from the SHEET (46 s) and the SEQRES (51 s) sections. When processing individual PDB files, such a short execution time is achievable when using hundreds of AUs (see execution times in Table 1 for 128 and 512 AUs). This confirms that the Azure Data Lake and our *PDBUSQLExtractor* are adjusted to processing Big Data files.

Table 2. Extraction time (s) for various sections of the PDB files for 3D protein structures stored in compressed (CS) and decompressed (DS) sequential files (data set DS1) for the execution with one allocation unit (AU).

	ATOM	SHEET	SEQRES
DS	116	46	51
CS	116	46	51

Again, the compression (CS) has no effect on the execution time, but on the storage space occupied by the sequential files. The occupied storage space was similar to the one provided for individual files, as the files were just concatenated.

In the third series of experiments, we decided to use a larger set of protein structures. For this purpose, we used the data set DS2 with 14,750 protein structures, randomly selected from the Protein Data Bank. The set of proteins was arranged in 10 sequential files of similar sizes. The parallelization factor was changed from 1 to ten (we used up to 10 AUs) during the data extraction. Results are presented only for the extraction of the atomic positions from the ATOM section, as the extraction of this section was the longest one among all the sections that were processed.

Results of the execution of the whole extraction from the ATOM section of PDB files stored in sequential files for the varying number of allocation units (AUs) is presented in Figure 7. As can be observed the best execution time was achieved when extracting atomic data from 10 sequential files storing approximately 10% of the Protein Data Bank content with the use of 10 AUs—the process took only 330 s.

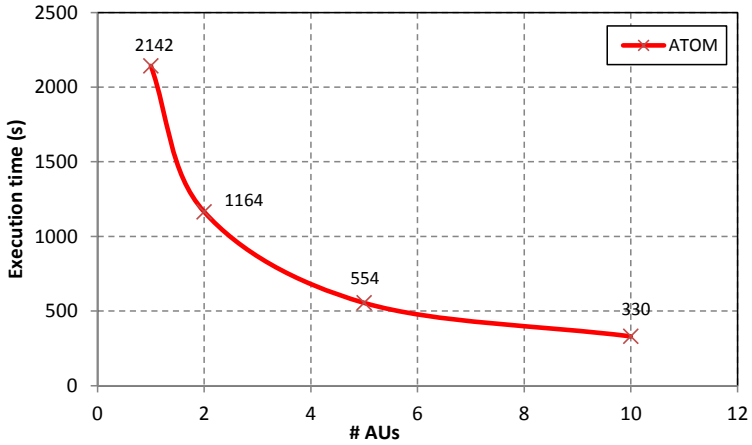

Figure 7. Execution time when extracting data from the *ATOM* section of PDB files stored in 10 sequential files (data set DS2) for the varying number of allocation units (AUs).

This confirms that our optimization in storage format by combining multiple PDB files into sequential files, and development of appropriate U-SQL data extractor, gives measurable benefits for

the efficiency of the extraction process. The execution time has significantly decreased when comparing it to processing individual files (see Table 1), and the cost of the processing according to the Data Lake Analytics price list has been significantly reduced (in the Cloud users pay for the time of used compute resources). The execution time of the U-SQL scripts became also more predictable compared to the first solution.

4.3. Sample Calculations

In this section, we show sample calculations that are accompanied by the extraction of atomic positions. In the presented example, we calculate distances between successive C_α atoms in each chain of successive proteins. Then, we select only these residues for which the distance to C_α atoms in the following residue is around 3.81Å. For two C_α atoms from a protein chain:

$$a_i^{C_\alpha} = (x_i, y_i, z_i)^T \quad \text{and} \quad a_{i+1}^{C_\alpha} = (x_{i+1}, y_{i+1}, z_{i+1})^T, \tag{1}$$

where x, y, z are Cartesian coordinates of particular C_α atoms, the distance between them can be calculated as the norm:

$$d_{i,i+1}^{C_\alpha} = \|a_i^{C_\alpha} - a_{i+1}^{C_\alpha}\| = \sqrt{(a_i^{C_\alpha} - a_{i+1}^{C_\alpha})^T (a_i^{C_\alpha} - a_{i+1}^{C_\alpha})}, \tag{2}$$

or by using the Euclidean distance:

$$d_{i,i+1}^{C_\alpha} = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2}. \tag{3}$$

Unless the data on the positions of C_α is prepared, the extraction process precedes the calculations and the *PDBUSQLExtractor* must be used. Listing 4 shows a sample U-SQL script that may implement the whole data processing task.

The U-SQL script references two external libraries. The *PDBUSQLExtractor* library contains implementations of extractors for individual and sequential PDB files (line 1). The FuzzySearchLib library (referenced in line 2) contains modules with classes and methods that allow using fuzzy search conditions to find all those distances that are around 3.81Å. Data extraction is performed by the EXTRACT expression executed in section S1 of the script (lines 6–25). The expression uses the `PDBConcatExtractor` (line 25) for fetching data from the ATOM sections of the PDB files assembled in a sequential file indicated in the FROM clause (line 24). Appropriate data from the ATOM sections of each protein, including the x, y, z coordinates of atoms, are extracted and supplemented by the `proteinId` and `modelId` (lines 8–9). The SELECT expression in section S2 of the U-SQL script (lines 27–39) retrieves only these fields of the produced rowset that are further needed for the calculations (lines 29–37), and only these rows of the rowset that correspond to the C_α atoms by applying appropriate filtering condition in the WHERE clause (line 39). The SELECT expression in section S3 of the U-SQL script performs the calculation of distances between two successive C_α atoms in each protein chain and produces the new rowset with distances, called `@filteredDistances`. Finally, the rowset is used in the SELECT expression in section S4 of the U-SQL script (lines 56–61). This expression implements a fuzzy query with the fuzzy search condition *Distance around 3.81Å*. The *Udfs.Around* function processes *CaCaDistance* values and compares them flexibly with the fuzzy set *Atomic distance around 3.81Å* defined by the Gaussian membership function presented in Figure 8. The fuzzy selection in the WHERE clause accepts only those residues that satisfy the fuzzy condition *CaCaDistance around 3.81* with a minimum degree of truth (similarity or membership degree) $\lambda = 0.5$ (line 61). Visual interpretation of the fuzzy selection is presented in Figure 8.

Listing 4. Sample U-SQL script that extracts data from the PDB files and produces the output rowset containing residues that are distant from the next residue by around 3.81Å.

```
REFERENCE ASSEMBLY [PDBUSQLExtractor];
REFERENCE ASSEMBLY FuzzySearchLib;
USING Udfs = FuzzySearchLib.Udfs;
USING Udl = FuzzySearchLib.Udl;

/* S1.Data extraction */
@extracted =
EXTRACT proteinId string,
modelId int,
serial int,
name string,
altLoc string,
resName string,
chainID string,
resSeq int,
iCode string,
x double,
y double,
z double,
occupancy double,
tempFactor double,
element string,
charge string
FROM "/input/pdbdataset.ent.gz"
USING new PDBUSQLExtractor.PDBConcatExtractor("ATOM");

/* S2.Selecting C-alpha atoms */
@filteredAtoms =
SELECT proteinId,
resSeq,
x,
y,
z,
modelId,
iCode,
chainID,
resName
FROM @extracted
WHERE name.Trim() == "CA";

/* S3.Calculating the distance between C-alpha atoms and their coordinates,
together with the index of the previous residue */
@filteredDistances =
SELECT a1.proteinId, a1.chainID, a1.resName,
Math.Sqrt(Math.Pow(a2.x - a1.x, 2) + Math.Pow(a2.y - a1.y, 2)
+ Math.Pow(a2.z - a1.z, 2)) AS CaCaDistance,
a1.resSeq AS resSeq1,
a2.resSeq AS resSeq2,
a1.modelId
FROM @filteredAtoms AS a1 JOIN @filteredAtoms AS a2
ON a1.proteinId == a2.proteinId
WHERE a1.resSeq == a2.resSeq + 1 AND a1.modelId == a2.modelId
AND a1.chainID == a2.chainID;


/* S4.Fuzzy selection CaCaDistance around 3.81 */
@cadist =
SELECT proteinId, resName, chainID, resSeq1, CaCaDistance,
Udfs.Around(CaCaDistance, 3.81, 3.77).Value AS MembershipDegree
FROM @filteredDistances
WHERE Udfs.Around(CaCaDistance, 3.81, 3.77) >= 0.5;


/* S5.Storing results of S4 in csv file */
OUTPUT @cadist
TO "/output/pdb_select.csv"
USING Outputters.Csv();
```

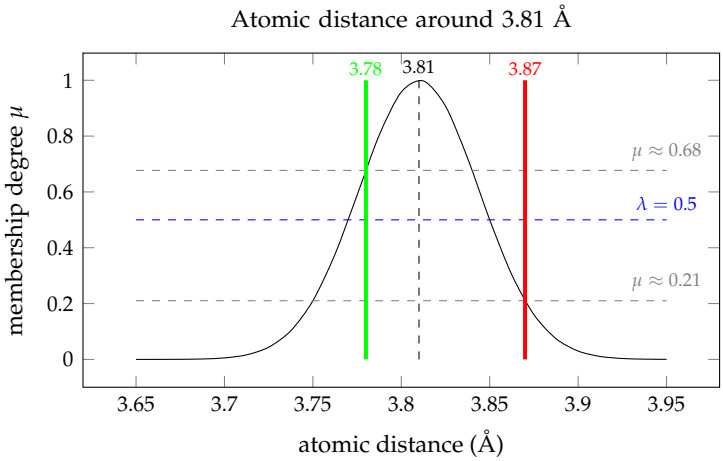

Figure 8. Visual interpretation of fuzzy selection with the fuzzy search condition *Atomic distance around 3.81 Å* showing two values of the inter-atomic distance in protein structures: 3.78 that satisfies the condition ($\mu(3.78) > \lambda$), and 3.87 that does not satisfy the condition ($\mu(3.87) < \lambda$) for $\lambda = 0.5$.

Such a U-SQL code generates the report shown in Figure 9, which is saved to CSV files in the Azure cloud storage by executing the OUTPUT statements *S*5 in lines 65–67. This report shows residues with values of the *Cα*–*Cα* distance around 3.81Å accompanied by membership degrees, which constitute degrees of truth how the particular distance value matches the fuzzy selection condition.

proteinId	chainID	resSeq	resName	CaCaDistance	MembershipDegree
1A00	A	2	LEU	3.798	0.946
1A00	A	3	SER	3.799	0.948
1A00	A	4	PRO	3.819	0.961
1A00	A	5	ALA	3.798	0.948
1A00	A	6	ASP	3.779	0.672
1A00	A	8	THR	3.826	0.890
1A00	A	9	ASN	3.815	0.986
1A00	A	10	VAL	3.799	0.949
1A00	A	11	LYS	3.792	0.871
...					
1A01	A	2	LEU	3.774	0.581
1A01	A	3	SER	3.789	0.831
1A01	A	4	PRO	3.810	0.999
1A01	A	5	ALA	3.799	0.952
1A01	A	6	ASP	3.812	0.997
1A01	A	7	LYS	3.821	0.945
1A01	A	8	THR	3.792	0.882
1A01	A	9	ASN	3.816	0.979
...					
1AOJ	A	19	GLY	3.848	0.531
1AOJ	A	20	TYR	3.805	0.992
1AOJ	A	21	GLU	3.780	0.680
...					
1AOJ	B	18	GLY	3.798	0.940
1AOJ	B	19	GLY	3.810	0.999
1AOJ	B	20	TYR	3.798	0.939
...					
1AOJ	C	17	VAL	3.819	0.964
1AOJ	C	18	GLY	3.841	0.644
1AOJ	C	19	GLY	3.801	0.968
...					
1AOJ	D	18	GLY	3.774	0.583
1AOJ	D	19	GLY	3.842	0.628
1AOJ	D	20	TYR	3.783	0.731
...					

Figure 9. Results of the extraction and the analysis performed with the U-SQL script from Listing 4 showing a part of produced rowset.

The execution of the presented U-SQL script for 14,750 protein structure from the DS2 data set stored in ten sequential files with 10 AUs took 370 s. In Figure 10 we can see the computational costs of the data extraction and the data analysis. We can see clearly that, in case of the presented analysis encoded in the U-SQL script, the computational cost of the data extraction is much higher (330 s) than the computational cost of the analysis performed (40 s). However, the proportions depend on the type of the analysis and complexity of the U-SQL code nested between the EXTRACT and the OUTPUT expressions.

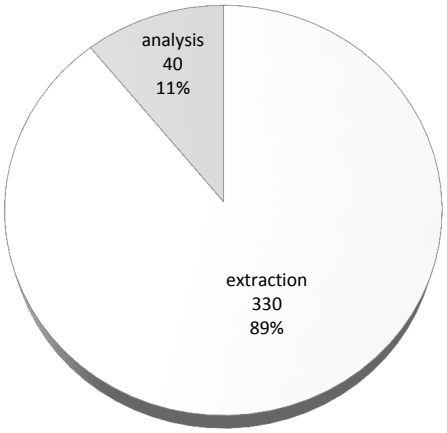

Figure 10. Computational costs of data extraction and data analysis for the sample U-SQL script executed for proteins from the data set DS2 stored in ten sequential files with 10 AUs.

5. Discussion

Efficient parsing and extracting of information from macromolecular data files describing 3D structures of proteins, nucleic acids, and nucleic acid-protein complexes is important for performing various sophisticated calculations on the structures on large scale. This will be especially attractive for all specialists working in the field of structural bioinformatics, computational biology, and drug design. The solution presented in the paper satisfies the requirements of the mentioned groups of specialists two-fold: (1) by providing a huge storage space for macromolecular data in the Data Lake Store in the Azure cloud, (2) by enabling scalable, parallel calculations on macromolecular structures in the Data Lake Analytics.

Results of performed experiments confirmed that the presented data extractors can be successfully used in retrieving data from particular sections of the PDB files stored in a variety of ways—(1) as individual compressed or decompressed files, or (2) as sequential compressed or decompressed files. The first approach does not require any initial processing of the data—they can be loaded to the cloud storage space as they are retrieved from the Protein Data Bank repository servers or from the local equipment or installed software tools, compressed or decompressed. Compression does not cause significant delays in data extraction in the Data Lake Analytics, but allows to save the storage space. The second approach requires some preprocessing prior to storing the data in the cloud storage space—sequential files must be created by linking many smaller PDB files into larger ones. This preprocessing may take some small amount of time, but causes that the extraction process is extremely fast comparing it to the one performed with individual files. This allows to extract more macromolecular data with a lower parallelization factor (less compute units are used) and, as a consequence, to reduce the costs of data processing in the Cloud platform. The results of our performance tests show that an important factor when working with the Azure Data Lake platform is to design tasks according to the specificity of the platform and its characteristics. The platform is intended to operate on large input, which favors processing sequential macromolecular data files, rather than individual ones. In less than two minutes, the platform with one compute unit (allocation unit) is

able to extract and process the sequentially stored macromolecular data of proteins, which would be extracted by hundreds of compute units for the same number of proteins stored individually.

Our solution complements various calculators for atomic interactions or inter-residue contacts presented in Section 2. It is less specific in terms of performed operations, but is more flexible. It does not provide any ready solution to calculate the particular type of interaction. However, users are able to encode their calculations in the U-SQL scripts after performing the extraction process. The U-SQL language gives the flexibility that allows performing any operation on macromolecular data in parallel once the data is extracted. This is completely different than the tools that provide dedicated, friendly, but closed web front-ends. In terms of the user interface, the presented solution is similar to the one presented in the PH2 system, but it is based on a different computational framework. The PH2 system uses Hadoop, while our solution is based on the Azure Data Lake. It is less efficient than in-memory solutions, like SpeeDB and IMPSMS, but gives more freedom in managing the underlying data and have a friendly U-SQL interface giving flexibility in various calculations.

Among important advantages of the presented solution, it is also worth noting that it can be broadly scaled out in the Azure cloud for petabytes of macromolecular data that we are awaiting due to the exponential growth of data in the field of bioinformatics. As a disadvantage, we should mention that the *PDBUSQLExtractor* is able to operate only within the Azure Data Lake platform. It is not possible to use it in other data processing platforms, like Apache Hadoop or Apache Spark. This flows mainly from the programming model of the Azure Data Lake, which *PDBUSQLExtractor* was adjusted to. Another disadvantage is the necessity to pay for the used storage space and, especially, for used compute resources of the Cloud platform. However, according to the principles of Cloud computing, users have to pay only for what they use and are excepted from the maintenance of the hardware. Taking into account current rates and unavoidable dynamics of the data growth in bioinformatics, we have to prepare efficient tools for such situations and our solution perfectly fits this scenario.

Future works will be focused on further development of the presented data extractors that would enable parsing and extracting data from other file formats, like mmCIF and PDBML. We also encourage readers to use the *PDBUSQLExtractor* while performing various sophisticated calculations on macromolecular structures in their research works. The Azure Data Lake also seems to be a good environment for performing efficient analyzes of other types of biological data, including data obtained in Next Generation Sequencing or mass spectrometry-based proteomics experiments. We are carrying works on the use of the platform in these areas of bioinformatics as they need such scalable solutions due to the enormity of data.

6. Availability

The *PDBUSQLExtractor* library is available at GitHub (https://github.com/dmrozek/repo-PDBUSQLExtractor) and at the project homepage http://zti.polsl.pl/w3/dmrozek/science/pdbusqlext.htm. The library can be used by the academic society for free, i.e., the software is free, but users have to set up their own Azure Data Lake execution environment in the Azure cloud to use the library for processing and analysis of their own data. Setting up steps are provided at the *PDBUSQLExtractor* project home page.

Further development of the system will be carried out by the Cloud4Proteins non-profit, scientific group (http://www.zti.aei.polsl.pl/w3/dmrozek/science/cloud4proteins.htm).

Author Contributions: D.M. conceived and designed the experiments; T.D. and D.M. performed the experiments; D.M. and B.M.-M. analyzed the data; T.D. and D.M. designed and implemented the tools; B.M.-M. implemented methods for fuzzy data processing; all authors contributed to the preparation of the U-SQL scripts; B.M.-M. and D.M. wrote the paper.

Funding: This research and the APC were funded within habilitation grant of the Rector of the Silesian University of Technology, Gliwice, Poland (grant No 02/020/RGH18/0148), and partially, within Statutory Research funds of Institute of Informatics, Silesian University of Technology, Gliwice, Poland (grant No BK/213/RAU2/2018). Access to the Azure Data Lake platform was funded by Microsoft Research within Microsoft Azure for Research Award grant.

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

ADL	Azure Data Lake
API	Application Programming Interface
AUs	allocation units
CI	compressed-individual files
CLR	common language runtime
CS	compressed-sequential files
CSV	comma-separated values
DI	decompressed-individual files
DLA	Data Lake Analytics
DLL	Dynamic-Link Library
DLS	Data Lake Store
DS	decompressed-sequential files
DS1	data set 1
DS2	data set 2
IMPSMS	In-Memory Protein Structure Management System
OS	operating system
PDB	Protein Data Bank
SDK	Software Development Kit
SQL	Structured Query Language
TSV	tab-separated values
UDO	user-defined operator
VMs	virtual machines

References

1. Berman, H.M.; Westbrook, J.; Feng, Z.; Gilliland, G.; Bhat, T.N.; Weissig, H.; Shindyalov, I.N.; Bourne, P.E. The Protein Data Bank. *Nucleic Acids Res.* **2000**, *28*, 235–242. [CrossRef] [PubMed]
2. Westbrook, J.; Fitzgerald, P. The PDB format, mmCIF, and other data formats. *Methods Biochem. Anal.* **2003**, *44*, 161–179. [PubMed]
3. Bourne, P.E.; Berman, H.M.; McMahon, B.; Watenpaugh, K.D.; Westbrook, J.D.; Fitzgerald, P.M. The macromolecular Crystallographic Information File (mmCIF). *Methods Enzymol.* **1997**, *277*, 571–590. [PubMed]
4. Wesbrook, J.; Ito, N.; Nakamura, H.; Henrick, K.; Berman, H. PDBML: The representation of archival macromolecular structure data in XML. *Bioinformatics* **2005**, *21*, 988–992. [CrossRef] [PubMed]
5. Mrozek, D.; Brozek, M.; Małysiak-Mrozek, B. Parallel implementation of 3D protein structure similarity searches using a GPU and the CUDA. *J Mol Model* **2014**, *20*, 2067. [CrossRef] [PubMed]
6. Jia, C.; Zuo, Y.; Zou, Q. O-GlcNAcPRED-II: An integrated classification algorithm for identifying O-GlcNAcylation sites based on fuzzy undersampling and a K-means PCA oversampling technique. *Bioinformatics* **2018**, *34*, 2029–2036. [CrossRef] [PubMed]
7. Masseroli, M.; Canakoglu, A.; Ceri, S. Integration and Querying of Genomic and Proteomic Semantic Annotations for Biomedical Knowledge Extraction. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2016**, *13*, 209–219. [CrossRef] [PubMed]
8. Ceri, S.; Kaitoua, A.; Masseroli, M.; Pinoli, P.; Venco, F. Data Management for Heterogeneous Genomic Datasets. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2017**, *14*, 1251–1264. [CrossRef]
9. Wei, L.; Tang, J.; Zou, Q. Local-DPP: An improved DNA-binding protein prediction method by exploring local evolutionary information. *Inf. Sci.* **2017**, *384*, 135–144. [CrossRef]
10. Hung, C.L.; Lin, C.Y. Open Reading Frame Phylogenetic Analysis on the Cloud. *Int. J. Genom.* **2013**, *2013*. [CrossRef]

11. Macalino, S.J.Y.; Basith, S.; Clavio, N.A.B.; Chang, H.; Kang, S.; Choi, S. Evolution of In Silico Strategies for Protein-Protein Interaction Drug Discovery. *Molecules* **2018**, *23*, 1963. [CrossRef] [PubMed]
12. Yang, X.; Wu, C.; Lu, K.; Fang, L.; Zhang, Y.; Li, S.; Guo, G.; Du, Y. An Interface for Biomedical Big Data Processing on the Tianhe-2 Supercomputer. *Molecules* **2017**, *22*, 2116. [CrossRef] [PubMed]
13. Mrozek, D. Scalable Big Data Analytics for Protein Bioinformatics *Computational Biology*; Springer: Berlin, Germany, 2018; Volume 28.
14. White, T. *Hadoop—The Definitive Guide: Storage and Analysis at Internet Scale*, 3rd ed.; O-Reilly: Sebastopol, CA, USA, 2012.
15. Zaharia, M.; Xin, R.S.; Wendell, P.; Das, T.; Armbrust, M.; Dave, A.; Meng, X.; Rosen, J.; Venkataraman, S.; Franklin, M.J.; Ghodsi, A.; Gonzalez, J.; Shenker, S.; Stoica, I. Apache Spark: A Unified Engine for Big Data Processing. *Commun. ACM* **2016**, *59*, 56–65. [CrossRef]
16. Mell, P.; Grance, T. The NIST Definition of Cloud Computing. Special Publication 800-145. 2011. Available online: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf (accessed on 10 October 2017).
17. Tina, K.G.; Bhadra, R.; Srinivasan, N. PIC: Protein Interactions Calculator. *Nucleic Acids Res.* **2007**, *35*, W473–W476. [CrossRef] [PubMed]
18. Chourasia, M.; Sastry, G.M.; Sastry, G.N. Aromatic–Aromatic Interactions Database, A2ID: An analysis of aromatic ΠΙ-networks in proteins. *Int. J. Biol. Macromol.* **2011**, *48*, 540–552. [CrossRef] [PubMed]
19. Pal, A.; Bhattacharyya, R.; Dasgupta, M.; Mandal, S.; Chakrabarti, P. IntGeom: A Server for the Calculation of the Interaction Geometry between Planar Groups in Proteins. *J. Proteom. Bioinform.* **2009**, *2*, 60–63. [CrossRef]
20. Hazelhurst, S. PH2: An Hadoop-based framework for mining structural properties from the PDB database. In Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists, Bela Bela, South Africa, 11–13 October 2010; pp. 104–112.
21. Date, C. *An Introduction to Database Systems*, 8th ed.; Addison-Wesley: Boston, MA, USA, 2003.
22. Robillard, D.E.; Mpangase, P.T.; Hazelhurst, S.; Dehne, F. SpeeDB: Fast structural protein searches. *Bioinformatics* **2015**, *31*, 3027–3034. [CrossRef] [PubMed]
23. Małysiak-Mrozek, B.; Żur, K.; Mrozek, D. In-Memory Management System for 3D Protein Macromolecular Structures. *Curr. Proteom.* **2018**, *15*, 175–189. [CrossRef]
24. Stephens, S.M.; Chen, J.Y.; Davidson, M.G.; Thomas, S.; Trute, B.M. Oracle Database 10g: A platform for BLAST search and Regular Expression pattern matching in life sciences. *Nucleic Acids Res.* **2005**, *33*, D675–D679. [CrossRef] [PubMed]
25. BioSQL Homepage. Available online: http://biosql.org/ (accessed on 2 November 2018).
26. Prlić, A.; Yates, A.; Bliven, S.E.; Rose, P.W.; Jacobsen, J.; Troshin, P.V.; Chapman, M.; Gao, J.; Koh, C.H.; Foisy, S.; et al. BioJava: An open-source framework for bioinformatics in 2012. *Bioinformatics* **2012**, *28*, 2693–2695. [PubMed]
27. Mrozek, D.; Wieczorek, D.; Małysiak-Mrozek, B.; Kozielski, S. PSS-SQL: Protein Secondary Structure—Structured Query Language. In Proceedings of the 2010 Annual International Conference of the IEEE Engineering in Medicine and Biology, Buenos Aires, Argentina, 31 August–4 September 2010; pp. 1073–1076.
28. Mrozek, D.; Socha, B.; Kozielski, S.; Małysiak-Mrozek, B. An efficient and flexible scanning of databases of protein secondary structures. *J. Intell. Inf. Syst.* **2016**, *46*, 213–233. [CrossRef]
29. Hammel, L.; Patel, J.M. Searching on the Secondary Structure of Protein Sequences. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, Hong Kong, China, 20–23 August 2002; Bernstein, P.A., Ioannidis, Y.E., Ramakrishnan, R., Papadias, D., Eds.; Morgan Kaufmann: San Francisco, CA, USA, 2002; pp. 634–645.
30. Tata, S.; Friedman, J.S.; Swaroop, A. Declarative Querying for Biological Sequences. In Proceedings of the 22nd International Conference on Data Engineering (ICDE'06), Atlanta, GA, USA, 3–7 April 2006; pp. 87–98.
31. Mrozek, D.; Małysiak-Mrozek, B.; Adamek, R. P3D-SQL: Extending Oracle PL/SQL Capabilities Towards 3D Protein Structure Similarity Searching. In *Bioinformatics and Biomedical Engineering*; Ortuño, F., Rojas, I., Eds.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2015; Volume 9043, pp. 548–556.
32. Hung, C.L.; Lin, Y.L. Implementation of a Parallel Protein Structure Alignment Service on Cloud. *Int. J. Genom.* **2013**. [CrossRef]

33. Holm, L.; Kaariainen, S.; Rosenstrom, P.; Schenkel, A. Searching protein structure databases with DaliLite v.3. *Bioinformatics* **2008**, *24*, 2780–2781. [CrossRef]
34. Gibrat, J.; Madej, T.; Bryant, S. Surprising similarities in structure comparison. *Curr. Opin. Struct. Biol.* **1996**, *6*, 377–385. [CrossRef]
35. Mrozek, D.; Daniłowicz, P.; Małysiak-Mrozek, B. HDInsight4PSi: Boosting performance of 3D protein structure similarity searching with HDInsight clusters in Microsoft Azure cloud. *Inf. Sci.* **2016**, *349*, 77–101. [CrossRef]
36. Mrozek, D.; Suwała, M.; Małysiak-Mrozek, B. High-throughput and scalable protein function identification with Hadoop and Map-only pattern of the MapReduce processing model. *J. Knowl. Inf. Syst.* **2018**, 1–34. [CrossRef]
37. Małysiak-Mrozek, B.; Daniłowicz, P.; Mrozek, D. Efficient 3D Protein Structure Alignment on Large Hadoop Clusters in Microsoft Azure Cloud. Beyond Databases, Architectures and Structures. In *Facing the Challenges of Data Proliferation and Growing Variety*; Kozielski, S., Mrozek, D., Kasprowski, P., Małysiak-Mrozek, B., Kostrzewa, D., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 33–46.
38. Mrozek, D.; Małysiak-Mrozek, B.; Kłapciński, A. Cloud4Psi: Cloud computing for 3D protein structure similarity searching. *Bioinformatics* **2014**, *30*, 2822–2825. [CrossRef]
39. Mrozek, D. *High-Performance Computational Solutions in Protein Bioinformatics*; SpringerBriefs in Computer Science; Springer: Cham, Switzerland, 2014.
40. Mrozek, D.; Kutyła, T.; Małysiak-Mrozek, B. Accelerating 3D Protein Structure Similarity Searching on Microsoft Azure Cloud with Local Replicas of Macromolecular Data. In *Parallel Processing and Applied Mathematics—PPAM 2015*; Wyrzykowski, R., Ed.; Lecture Notes in Computer Science; Springer: Heidelberg, Germany, 2016; Volume 9574, pp. 1–12.
41. Hung, C.L.; Hua, G.J. Cloud Computing for Protein-Ligand Binding Site Comparison. *Biomed. Res. Int.* **2013**. [CrossRef]
42. Mrozek, D.; Gosk, P.; Małysiak-Mrozek, B. Scaling *Ab Initio* Predictions of 3D Protein Structures in Microsoft Azure Cloud. *J. Grid Comput.* **2015**, *13*, 561–585. [CrossRef]
43. Zou, Q.; Wan, S.; Ju, Y.; Tang, J.; Zeng, X. Pretata: Predicting TATA binding proteins with novel features and dimensionality reduction strategy. *BMC Syst. Biol.* **2016**, *10*, 114. [CrossRef]
44. Microsoft Azure. Overview of Microsoft Azure Data Lake Analytics. Available online: https://docs.microsoft.com/en-us/azure/data-lake-analytics/data-lake-analytics-overview (accessed on 7 November 2018).
45. Microsoft Azure. Azure Data Lake Analytics Documentation. Available online: https://docs.microsoft.com/en-us/azure/data-lake-analytics/ (accessed on 18 December 2018).
46. Protein Data Bank Contents Guide. Atomic Coordinate Entry Format Description, Version 3.3. Available online: http://www.wwpdb.org/documentation/file-format-content/format33/v3.3.html (accessed on 7 November 2018).