Knowledge and Information Systems (2019) 60:145–178
https://doi.org/10.1007/s10115-018-1245-3

REGULAR PAPER

High-throughput and scalable protein function identification with Hadoop and Map-only pattern of the MapReduce processing model

Dariusz Mrozek[1] · **Marek Suwała**[1] · **Bożena Małysiak-Mrozek**[1]

Received: 9 September 2016 / Revised: 26 May 2018 / Accepted: 16 June 2018 /
Published online: 13 July 2018

Abstract
Efficient computational solutions for identification of protein functions or finding structural homologs of proteins gain importance in the era of structural genomics and in the face of growing volumes of biological data. Structural alignments, which underlie these two processes, take a lot of time to complete, especially when performed for large collections of 3D protein structures. Fortunately, structural alignments can be carried out on well-separable and independent subsets of the whole macromolecular data repository, which perfectly fits the MapReduce processing paradigm of bringing computations to data. In this paper, we show how the protein function identification and finding structural homologs can be efficiently accelerated with the use of the MapReduce procedure executed on Hadoop cluster established in a virtualized compute environment or a private cloud. For this purpose, we propose Map-only processing pattern of the MapReduce procedure, which is formally defined in this paper. The solution that we show joins advantages of performing computations in small virtualized compute environments with large-scale computations in public clouds, thus allowing to perform structural alignments for a number of usage scenarios, including comparison of pairs of 3D protein structures during evaluation of predicted protein models, one-to-many comparisons while identifying possible functions of the given structure, or all-to-all alignments while investigating the divergence between known protein structures and classifying proteins by their fold. In this paper, we also present results of performance tests when scaling up nodes of the Hadoop cluster and increasing the degree of parallelism with the intention of improving efficiency of the computations.

Keywords Bioinformatics · Big data · Proteins · Scalable computations and MapReduce · 3D protein structures · Cloud computing

1 Introduction

The volume of biological data collected in dedicated, frequently public, repositories increases every year. This is caused by international efforts to understand living organisms at many

✉ Dariusz Mrozek
dariusz.mrozek@polsl.pl

[1] Institute of Informatics, Silesian University of Technology, Akademicka 16, 44-100 Gliwice, Poland

levels determined by the biological information that is analyzed. These efforts are supported by various international projects, such as 1000 Genomes [43], aimed at sequencing genomes of at least one thousand anonymous participants from a number of different ethnic groups in order to establish a detailed catalog of human genetic variations, that generate terabytes of genetic data. This massive influx of biological data is also accelerated by newly developed technologies for DNA sequencing, which are getting faster and less expensive every year. Driven by advances in DNA sequencing techniques and the huge gap between the number of known protein sequences [19] and 3D protein structures [18], structural genomics tries to find the 3D structure of every protein that is encoded by a given sequenced genome. This is done by combining traditional experimental methods, like X-ray crystallography or nuclear magnetic resonance (NMR), with modeling approaches that use various prediction methods [10,23, 31,44]. These prediction methods, used for protein structure determination, may rely on sequence or structural homology [24,45] to a protein of the structure already determined and stored in a repository, such as the world-renowned Protein Data Bank [1]. The quality of protein models produced by structural genomics can be evaluated by performing structural alignments between these models and known protein structures. Structural alignment can also be used to find structural homologs and compare proteins with low sequence similarity, in order to detect evolutionary relationships between proteins that share very little common sequence.

Structural alignment methods can be used in comparisons of individual structures (single one-to-one comparisons), one-to-many comparisons against a set of protein structures collected in a repository, or many-to-many comparisons performed for all protein structures within the repository (Fig. 1). In the two latter cases, however, performing structural alignments can be very time-consuming and thus may require vast computational resources in order to complete the task in a reasonable time. Application of computationally efficient solutions for mining structural similarities by structural alignment becomes especially important in the face of the exponentially growing number of macromolecular data in the Protein Data Bank. Fortunately, recent advances in processing and analyzing data by utilization of the MapReduce processing model on the Hadoop [46,50] or the Spark platforms [48] allow to significantly accelerate the computations and scale them with the growing volume of data and the growing demand for computational power. Out of the two mentioned platforms we chose to use Hadoop and the MapReduce, since they are suitable for long-running batch processes. Indeed, the structural alignment is a long-running process, in which each single component comparison of proteins takes between a few seconds to a few minutes. Moreover, there are thousands of such comparisons in one-to-many comparison scenarios. There is also no need to reprocess the produced results, like in Spark, which keeps intermediate results in-memory for subsequent processing iterations.

1.1 MapReduce processing model and Hadoop

MapReduce is a data processing model that allows processing highly parallelizable data sets. The MapReduce defines how parallel processing of huge data sets will be performed in a distributed environment. There are generally two stages of performing calculations in systems that implement the MapReduce model: the Map phase, which comprises parallel processing of records of the input data set, and the Reduce phase, which aggregates related data. Hadoop implements the MapReduce computational model, thus allowing data analysts to easily create and run applications that process data in a distributed cluster of computers. Hadoop accepts MapReduce applications, called MapReduce jobs, as a set of specific processing work to

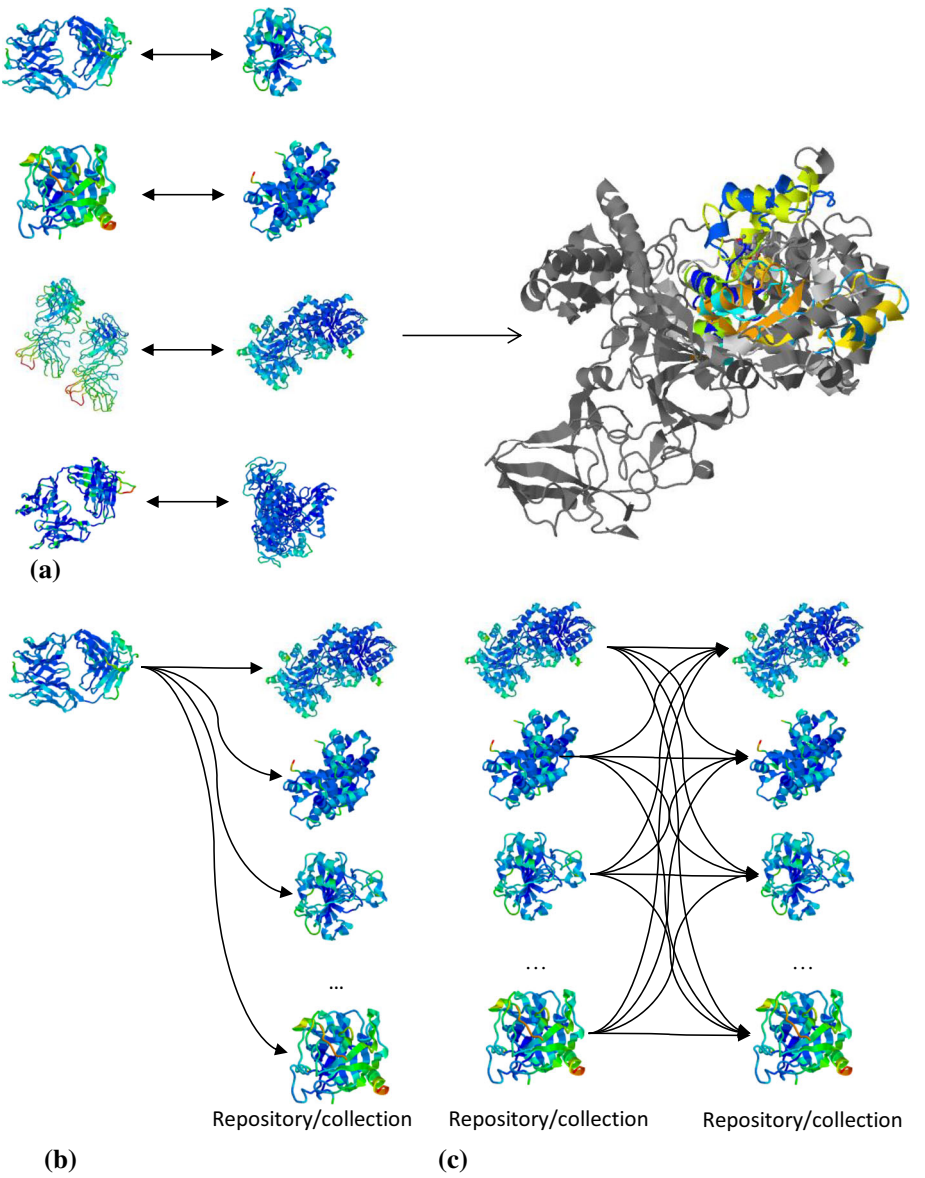

Fig. 1 Various comparison scenarios for 3D protein structures: **a** one-to-one comparisons between pairs of protein structures (left) and the result of a single structural alignment (right), **b** one-to-many comparison between given 3D protein structure and structures in the repository/collection, **c** many-to-many comparisons between 3D protein structures in the repository/collection

be done on the specified distributed data with some specified configuration parameters. The MapReduce job is then divided by the Hadoop into smaller tasks; there are two main types of these tasks: Map tasks and Reduce tasks. Such a division is directly related to the concept of performing two-stage calculations in the MapReduce model. Input data, subdivided into smaller fragments (Splits) containing particular data records, are processed in parallel by

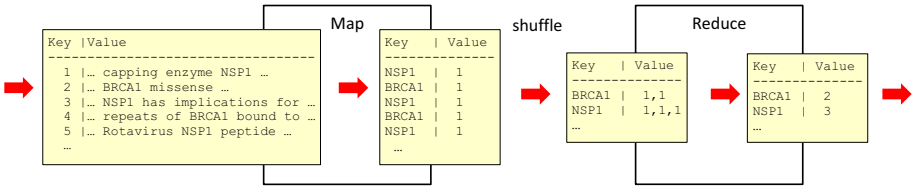

Fig. 2 Sample data flow in single Map and Reduce tasks for aggregating the number of occurrences of short names of proteins in titles of scientific papers

multiple Map tasks that generate results in the form of key-value pairs. These results are then sorted and grouped based on the key. Reduce tasks may then perform aggregations of values related to particular keys and generate the output in the form of a list of key-value pairs.

Sample MapReduce data flow for aggregating the number of occurrences of short names of proteins in titles of scientific papers is presented in Fig. 2. As the input, the Map phase accepts titles of scientific papers, for example, from the Protein Data Bank. The key is the row identifier that is further ignored as it is not needed in presented example. The Map function extracts short names of proteins and produces the list of the names (as the key) with the information that it occurred (as the value). For simplicity of further aggregation, the value of 1 is generated for each occurrence. The output from the Map function is then processed by the MapReduce framework (shuffle), which sorts and groups key-value pairs by key. This produces the input for the Reduce phase. The Reduce function aggregates provided values for each extracted name of the protein. It generates the list of short names (key) together with aggregated number of occurrences (value) as the output.

Large portions of data processed by the Hadoop are usually stored in the Hadoop Distributed File System (HDFS). The HDFS is a distributed file system dedicated to work with MapReduce-based applications. HDFS is designed for distributed storage of large datasets, while ensuring a reliable and quick access to data. The data stored in HDFS are divided into blocks, which are in turn distributed in all cluster nodes. Hadoop protects stored data against loss by repeated replication of every single block of data to multiple nodes of the cluster, thereby enabling to carry out reliable and fast computations. Until losing the last replica of the data, the owner does not become aware of any damage which may take place on the data server side.

1.2 Related works

3D protein structure alignments can be performed with the use of various methods. There are also various solutions to accelerate this task when it is performed on a large scale. In this section, we review methods for protein structure alignment and scalable solutions allowing to accelerate massive alignments performed on large data sets.

1.2.1 Methods for protein structure alignment

3D protein structure alignment is a complex and time-consuming process due to complexity of 3D protein structures, huge search space, and computational complexity of algorithms used to complete the process. In the last decades, various scientists and research centers have designed and developed a variety of methods for aligning 3D protein structures, finding similarities between proteins, or classifying protein structures into groups, including VAST [9,20],

DALI [11], LOCK2 [38], FATCAT [47], CE [40], FAST [49], MultiProt [39], MotifMiner [4], MICAN [26], APGM [14], DEDAL [5], CASSERT [29,33], and others. These methods rely on various representative features of 3D protein structures. For example, local geometric features and selected biological characteristics are used in the CTSS [2] algorithm. Shape signatures that include the information on C_α atom positions, torsional angles, and the type of the secondary structure are calculated for each residue in a protein structure. A similar idea is used in CASSERT [33] and GPU-CASSERT [29], but structural residue descriptors consist of the information on relative position of the C_α atom, the secondary structure type, and the residue type. Another feature-based approach is used in methods proposed by Fober and colleagues in [7,8] and Leinweber et al. in CavSimBase [17]. These approaches make use of graphs to model molecular structures and represent various features of protein structures. Then, comparison of protein structures is reduced to the problem of comparing graphs. These methods outperform many methods in terms of efficiency, but they mainly focus on the estimation of similarity (or distance) of protein molecules, especially binding sites. DALI algorithm [11,12], one of the most popular alignment methods, compares proteins based on distance matrices, which are built for each of the compared proteins. Each cell of such a distance matrix contains the distance between the C_α atoms of every two residues in the same structure (inter-residue distances). Fragments of 6×6 elements of the matrix constitute so-called *contact patterns*, which are then compared between two proteins to find the best match. On the other hand, the VAST algorithm [20], which is available through the Web site of the National Center for Biotechnology Information (NCBI), uses secondary structure elements (SSEs—α-helices and β-sheets) forming the cores of compared proteins. SSEs are then mapped to the representative vectors, which simplifies the analysis and comparison. During the comparison, the algorithm attempts to match a set of vectors for pairs of protein structures. Other methods, like LOCK2 [38], also apply the SSE representation of protein structures in the comparison process.

The CE [40] algorithm, which was applied in the solution presented in the paper, uses the combinatorial extension of the alignment path formed by aligned fragment pairs (AFPs). AFPs are fragments of both structures indicating a clear structural similarity and they are described by local geometrical features, including positions of C_α atoms. The advantage of new versions of the CE [36] over other methods is the capability of handling circular permutations. This problem is typical for many alignment algorithms that compute sequence order-dependent alignments. The idea of AFPs is also used in FATCAT [47], which is the second method implemented in our distributed solution presented in the paper. By entering twists in the alignment, the FATCAT allows for flexible alignments, eliminates drawbacks of many existing methods that treat proteins as rigid bodies, thus leading to better alignments and a decreased RMSD in a number of cases. Additionally, both methods provide the final superposition of 3D protein structures.

1.2.2 Scalable solutions for 3D protein structure similarity searching

The growing volume of biological data, the variety of data gathered in different areas of bioinformatics, together with the high rate at which the data are generated, caused the necessity to search for efficient computational solutions that would support parallel data exploration and increase the performance of the process [21,22]. Recent technological advances in parallel programming, distributed computing, and data processing brought several solutions for accelerating protein structure comparison with the use of GPU devices, farms of computers or virtual machines located on the Cloud, and Hadoop clusters.

The first group of GPU-based approaches contains solutions proposed by Leinweber et al. [15–17] for structural comparisons of protein binding sites. These approaches rely on the feature-based representation of protein structure and graph comparisons, mentioned in the previous section. Authors reported a significant superiority of GPU-based executions of the comparison process over the CPU-based ones in terms of runtimes of performed experiments. Mentioned works focus on comparison of protein binding sites, not protein structures as a whole. In contrast, fold-based methods for protein comparison focus on entire protein structures. Examples of GPU-accelerated fold-based methods are *SA Tableau Search* by Stivala et al. [42], *pssAlign* by Pang et al. [35], and GPU-CASSERT reported in one of our previous works [29]. The *SA Tableau Search* makes use of orientations of secondary structure elements and distance matrices to represent protein structures, and simulated annealing for optimal alignment. The *pssAlign* uses locations of the C_α atoms to represent proteins in the comparison, and a dynamic programming procedure for the alignment. Finally, the GPU-CASSERT represents proteins as reduced chains of secondary structure elements and chains of molecular residue descriptors, and it compares proteins with the use of the dynamic programming-based two-phase alignment method. All three GPU-based implementations of fold-based methods significantly accelerate calculations related to assessing the similarity of protein molecules. However, they also require appropriate GPU devices and specific preparation of data. Moreover, they only measure the similarity of protein molecules without providing the final superposition.

The second group of approaches consists of solutions that utilize farms of computers or virtual machines to perform protein structure comparison. Examples of such solutions are MAS4PSi [27], Cloud4Psi [28,34], and CloudPSR [32]. All of them rely on fold-based methods for 3D protein structure comparison. The MAS4PSi utilizes the JADE multi-agent system for reliable and efficient computations, in which software agents residing on the farm of physical computers perform pairwise alignments. On the other hand, the Cloud4PSi and the CloudPSR utilize a collection of virtual machines (worker roles) located on the public Cloud for protein structure alignment. Various scheduling schemes for computations were tested in both systems. In all presented systems, parallelization brought a significant acceleration of computations mainly by scaling the systems horizontally. However, they are specific for the public cloud provider.

The third group contains Hadoop- and MapReduce-based approaches for protein structure alignment. This group contains the system developed by Che-Lun Hung and Yaw-Ling Lin [13] and HDInsight4PSi [30]. The Hadoop-based system developed by Hung and Lin uses two popular fold-based alignment methods—DALI [11] and VAST [9]. The Map phase performs structural alignments for given pairs of protein structures, and the Reduce phase refines results of the produced alignments. The system was tested with one thousand 3D protein structures randomly chosen from the Protein Data Bank on a private virtualized computing environment containing eight data nodes showing good, linear scalability of performed computations. In our previous work, published in [30], we also proved that using sequential files (instead of processing individual structures) may increase the performance of the MapReduce-based parallel protein structure similarity searches. This was especially visible when processing large repositories of macromolecular structures. The HDInsight4PSi, presented in the work [30], was developed for HDInsight/HBase clusters deployed in Microsoft Azure public cloud. The system uses the MapReduce procedure when performing one-to-many protein structure comparisons against large collections of proteins. However, besides huge scaling capabilities, provisioning the HDInsight/HBase clusters from public cloud providers may produce a significant cost for potential users. Thus, some of them may want to make use of their own, existing compute environments or establish a private cloud [25,41]

due to economic or security reasons. For example, some companies or laboratories may treat their data as intellectual property and may want to use on-premises storage to keep the data within their data centers.

1.3 Scope of this work

Undoubtedly, for a variety of biological data and a variety of scenarios of how these data can be processed and analyzed, the MapReduce processing model brings the potential to make a step forward toward the development of solutions that will allow to get insights in various biological processes much faster. In this paper, we propose MapReduce-based computational solution, called H4P, for efficient mining of similarities in 3D protein structures and for structural superposition. The H4P benefits from the Map-only processing pattern of the MapReduce, which is presented and formally defined in this paper. The H4P service is implemented for the Hadoop framework with supposed deployment to private clouds. It was tested on the virtualized computer cluster with the Hadoop framework. The H4P joins advantages of the system developed by Hung and Lin for small virtualized compute environments with large-scale scanning capabilities of the HDInsight4PSi, thus allowing to perform structural alignments for a number of usage scenarios, including comparison of pairs of 3D protein structures during evaluation of predicted protein models, one-to-many comparisons while identifying possible functions of the given structure, or all-to-all alignments while investigating the divergence between known protein structures and classifying proteins by their fold. In this paper, we also present results of performance tests when scaling up nodes of the Hadoop cluster and increasing the degree of parallelism with the intention of improving efficiency of the computations.

2 Materials and methods

In this section, we provide foundations of the methods used for 3D protein structure alignment, formal definition of the Map-only pattern of MapReduce processing model, and details of the implementation of the Map-only processing pattern in the H4P system.

2.1 Algorithms for 3D protein structure alignment

The Map-only pattern of the MapReduce processing model was tested in the H4P system with the use of two popular methods for 3D protein structure alignment, namely jCE and jFATCAT [37]. These methods were chosen mainly due to a good quality of alignments that they provide and the capability of handling circular permutations. In the H4P, both methods were implemented as MapReduce procedures. These methods, the jCE and the jFATCAT, are enhanced versions of the Combinatorial Extension (CE) [40] and the Flexible structure AlignmenT by Chaining Aligned fragment pairs allowing Twists (FATCAT) [47]. CE and FATCAT methods are relatively slow, which justifies the use of highly scalable platforms for their parallelization, but on the other hand, they generate high-quality alignments with respect to the RMSD measure. Both methods are publicly available through the Protein Data Bank (PDB) Web site for those who want to search for structural neighbors [10], which confirms their good reputation in the structural bioinformatics community.

Combinatorial Extension (CE) is one of the methods used for finding common regions in two protein structures on the basis of structural alignment of protein molecules. In this

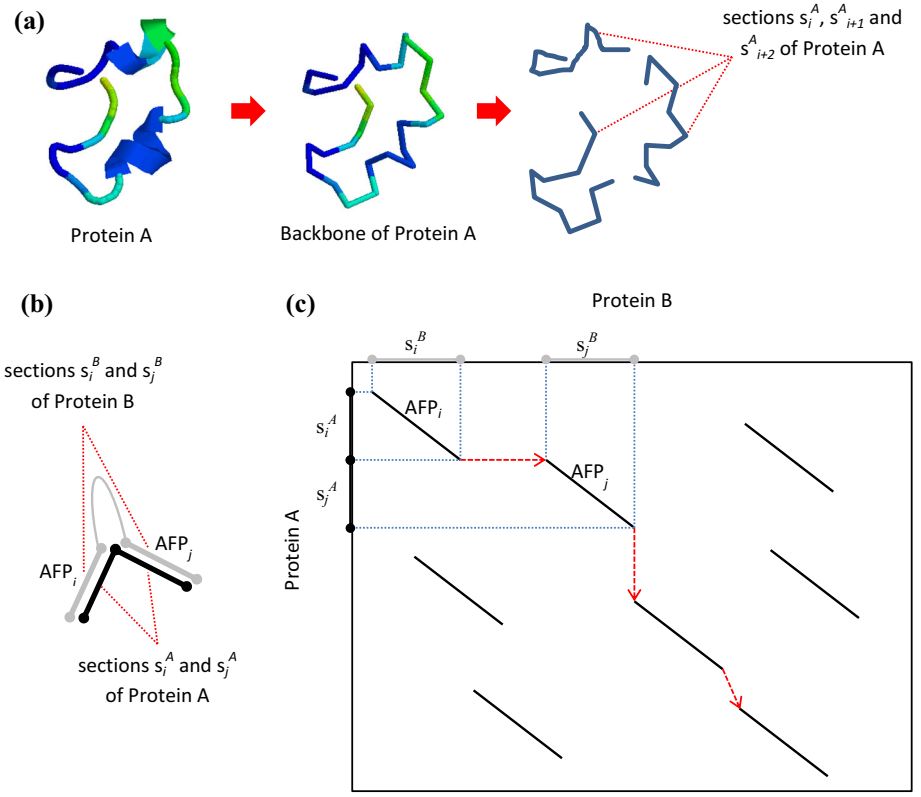

Fig. 3 3D protein structure alignment by combining AFP elements: **a** 3D structure of a sample protein, its backbone, and splitting into fixed-length fragments (sections), **b** sections of protein structures A (black) and B (gray) forming two aligned fragment pairs AFP_i and AFP_j, **c** alignment matrix showing various AFP elements and alignment path created by joining (red arrows) several AFPs

method, two 3D protein structures are first transformed to a reduced representation of *rigid bodies* on the basis of backbone atoms (C_α, Fig. 3a). Alignment of protein structures involves comparison of their individual fragments (Fig. 3b). The method initially seeks fragments of a fixed length (8 residues) and pairs them, if they satisfy a similarity condition. These paired fragments are called as *aligned fragment pairs* (AFPs). In particular, the AFPs are pairs of 8-residue fragments, which are considered similar if their corresponding internal distances are similar. Then, by linking together several aligned fragment pairs (AFPs) in the calculated similarity/alignment matrix the method creates so-called *alignment path* (Fig. 3c).

The alignment path P is the longest continuous path of AFPs of size m^{AFP} in a similarity/alignment matrix representing all possible AFPs that conform to the criteria for structure similarity. Various combinations of AFP elements, representing possible continuous alignment paths, are selectively extended or discarded, resulting in a single alignment path. The objective of the method is to determine the optimal, i.e., the longest possible and continuous alignment path for two protein structures. The best alignment path represents the alignment between two protein structures A and B of length n^A and n^B.

Decision on the extension of the alignment path is made based on several criteria, by evaluating (1) every single AFP, (2) AFP that pretends to be joined against the current path, and (3) the whole path itself, which results in three conditions that should be satisfied:

$$D_{rr} < D_0, \tag{1}$$

$$\frac{1}{r-1}\sum_{i=0}^{r-1} D_{ir} < D_1, \tag{2}$$

$$\frac{1}{r^2}\sum_{i=0}^{r}\sum_{j=0}^{r} D_{ij} < D_1, \tag{3}$$

where D_{ij} is the distance between aligned fragments defined by the AFP i and j in the alignment path and r is the next AFP that pretends to be added to the alignment path of $r-1$ AFPs in length. D_0 and D_1 are similarity thresholds with typical values of $D_0 = 3$Å and $D_1 = 4$Å.

The second of the methods implemented as parallel procedure in the H4P is FATCAT (*Flexible structure AlignmenT by Chaining AFPs with Twists*) [47]. The development of the method was motivated by the fact that existing methods for 3D protein structure similarity searching do not take into account one of the characteristics of protein structures, namely flexibility. It turns out that proteins are flexible molecules, which undergo significant structural reorganizations and conformational changes, depending on their function in the organism or current activity state. Alignment methods for protein structures, e.g., the CE, reduce complicated 3D protein structures to rigid bodies, but do not consider the flexibility. It was noted that the way of representing the elastic proteins in comparison of their structures may cause skipping significant structural similarities and thus incorrect or incomplete alignments. The FATCAT method solves this problem by incorporating the flexibility of protein structures into the alignment process. This method, similarly to the CE, seeks the optimal alignment of protein structures by linking successive aligned fragment pairs (AFPs). The difference lies in the fact that the FATCAT allows such combinations of the following AFP elements, which would not be possible without modifying one of the compared structures. Such a combination is obtained by introducing so-called *twists* in the reference protein structure in one of its points that is referred to as *hinge*. These points become also the place of connecting those successive AFP elements that constitute the alignment path. The objective of the method is to determine the optimal alignment of flexible 3D protein structures while minimizing the number of twists made in the reference protein.

Flexible structure alignment is performed by chaining AFPs with at most t twists. Optimal alignment path is obtained by using dynamic programming procedure and maximizing the score $S(j)$. The $S(j)$ score reflects the best scoring path ending at AFP j, and it can be calculated from the best ending at previous AFP (i) that can be connected with AFP j:

$$S(j) = a(j) + \max\left\{\max_{\substack{e^A(i)<b^A(j)\\ e^B(i)<b^B(j)}} [S(i) + c(i \longrightarrow j)], 0\right\}$$
$$\text{subject to } T(j) \leq t, \tag{4}$$

where $a(j)$ is the score of AFP j itself, $b^A(j)$, $b^B(j)$, $e^A(i)$, $e^B(i)$ are the starting and ending positions of AFP i and j in proteins A and B, $c(i \longrightarrow j)$ is the score for introducing the connection between AFP i and j, $T(j)$ is the number of twists required for connecting the chain of AFPs leading up to $S(j)$.

The score for connecting AFP i and j depends on the compatibility of the AFPs and the number of mismatched regions and gaps in the partial alignment path formed by both AFPs, according to the following equation:

$$c(i \longrightarrow j) = W(D_{ij}) \times P_c \times F(p, g), \tag{5}$$

$$W(D_{ij}) = \begin{cases} 1 & \text{if } D_{ij} > D_c \\ \left(\frac{D_{ij} - D_0}{D_c - D_0}\right) & \text{elseif } D_0 < D_{ij} \leq D_c \\ 0 & \text{otherwise} \end{cases}, \tag{6}$$

$$F(p, g) = M_c \times p + M_g \times g, \tag{7}$$

where D_{ij} is calculated as root-mean-square of the distance matrix between AFP i and j, D_c is the threshold for introducing a twist, D_0 is the threshold for penalizing the connection, P_c is the maximum penalty for connecting two AFPs, p is the number of mismatched regions, g is the number of gaps, M_c is the penalty involved with mismatching two positions, M_g is the penalty for a gap.

2.2 Map-only pattern of the MapReduce processing model

H4P uses the Hadoop framework (ver. 2.x) and the MapReduce processing model to parallelize computations related to massive 3D protein structure alignments. For the purpose of defining the Map-only pattern of the MapReduce processing model, let us consider a commonly used one-to-many comparison scenario, in which a dedicated implementation of the MapReduce application compares and aligns 3D structures of the given query protein(s) to 3D structures of candidate proteins stored in the repository. (Other comparison scenarios, i.e., one-to-one and many-to-many, will be considered as restrictions or extensions to the one-to-many scenario.) The collection of candidate structures can be described as follows:

$$C_P = \left\{c_{P,v} | v = 1, 2, \ldots, V\right\} \tag{8}$$

where V is the number of candidate protein structures in the collection (repository).

These candidate protein structures can be processed individually, while comparing pairs of 3D protein structures in one-to-one comparison scenario or one-to-many scenarios when the size of repository is relatively small. However, processing individual candidate protein structures in large repositories negatively affects performance due to small sizes of most of macromolecular data files ranging from kilobytes to several megabytes (comparing to 64 MB/128 MB block size of the HDFS). For this reason, for one-to-many and many-to-many scenarios with large collections of candidate proteins, we decided to group candidate protein structures in so- called *sequential files* that fit the size of the HDFS data block size and contain many records, each representing a candidate protein structure. When processing these data in the MapReduce application, the Hadoop logically represents sequential files as input splits. These input splits can be defined as follows:

$$s_H = \left\{c_{P,m}, c_{P,m+1}, \ldots, c_{P,n}\right\}, \text{where } 1 \leq m \leq n \leq V \text{ and } m, n, V \in \mathbb{N}_+, \tag{9}$$

where m and n determine boundaries of the input splits.

Each input split is a subset of the whole collection:

$$s_H \subseteq C_P \quad \text{and} \quad \forall k, l > 0, k \neq l \;\; s_{H,k} \cap s_{H,l} = \emptyset, \tag{10}$$

which means that each candidate protein structure belongs only to one sequential file.

We assume that the size of each input split fits the 64 MB block size of HDFS:

$$SizeOf(s_H) = 64\,\text{MB}. \tag{11}$$

The number of protein structures in Hadoop split may vary, and it depends on the sizes of particular protein structures included in the sequential files:

$$|s_H| = n - m, \tag{12}$$

where m and n determine boundaries of the input split.

The collection of input splits is defined as:

$$S_H = \left\{ s_{H,u} | u = 1, 2, \ldots, U \right\}, \tag{13}$$

where U is the number of input splits, equal to the number of sequential files with candidate protein structures.

Protein structures located in all input splits cover the whole collection of candidate proteins:

$$C_P = \bigcup_{k=1}^{U} s_{H,k}, \tag{14}$$

where U is the number of Hadoop input splits, and cardinality of the collection C_P:

$$|C_P| = \sum_{k=1}^{U} |s_{H,k}|. \tag{15}$$

The Hadoop framework processes data by creating and executing MapReduce jobs. A MapReduce job divides the whole work into smaller tasks. There are two types of tasks that can appear in the processing workflow—Map tasks, which are followed by Reduce tasks. Map tasks are used for processing data, and Reduce tasks are usually used to consolidate results produced in the Map phase. The Map and the Reduce phases are not necessarily sequential, but the Reduce phase depends on the output of the Map phase. For general purposes, from the viewpoint of implemented and executed tasks, a MapReduce job can therefore be defined as:

$$J_{MR} = T_M \cup T_R, \tag{16}$$

where T_M is the set of Map tasks and T_R is the set of Reduce tasks.

In the H4P, massive 3D protein structure alignments are performed in the Map phase as it is presented in Fig. 4. Results of the alignment processes (identifiers of aligned structures and a set of similarity measures) are then stored in a database for future reuse. Storing results does not involve any consolidation of results in the Reduce phase, like in the standard MapReduce procedure (Fig. 4a). Running the Reduce phase just for storing results of the alignment processes would introduce unnecessary overhead, since the output of the Map phase is written to local disks and then transferred across the network to the reducer nodes. Our early tests performed with the use of full MapReduce implementation showed that it is 15–20% less efficient than the Map-only implementation (see Sect. 3.4.1). Therefore, we assume that in the Map-only pattern the $T_R = \emptyset$ and then:

$$J_{MR} = T_M, \tag{17}$$

and in consequence, we can define the MapReduce job as a set of Map tasks:

$$J_{MR} = \left\{ t_{M,s} | s = 1, 2, \ldots, S \right\}, \tag{18}$$

where S is the number of all Map tasks. This defines the Map-only processing pattern of the MapReduce model (Fig. 4b).

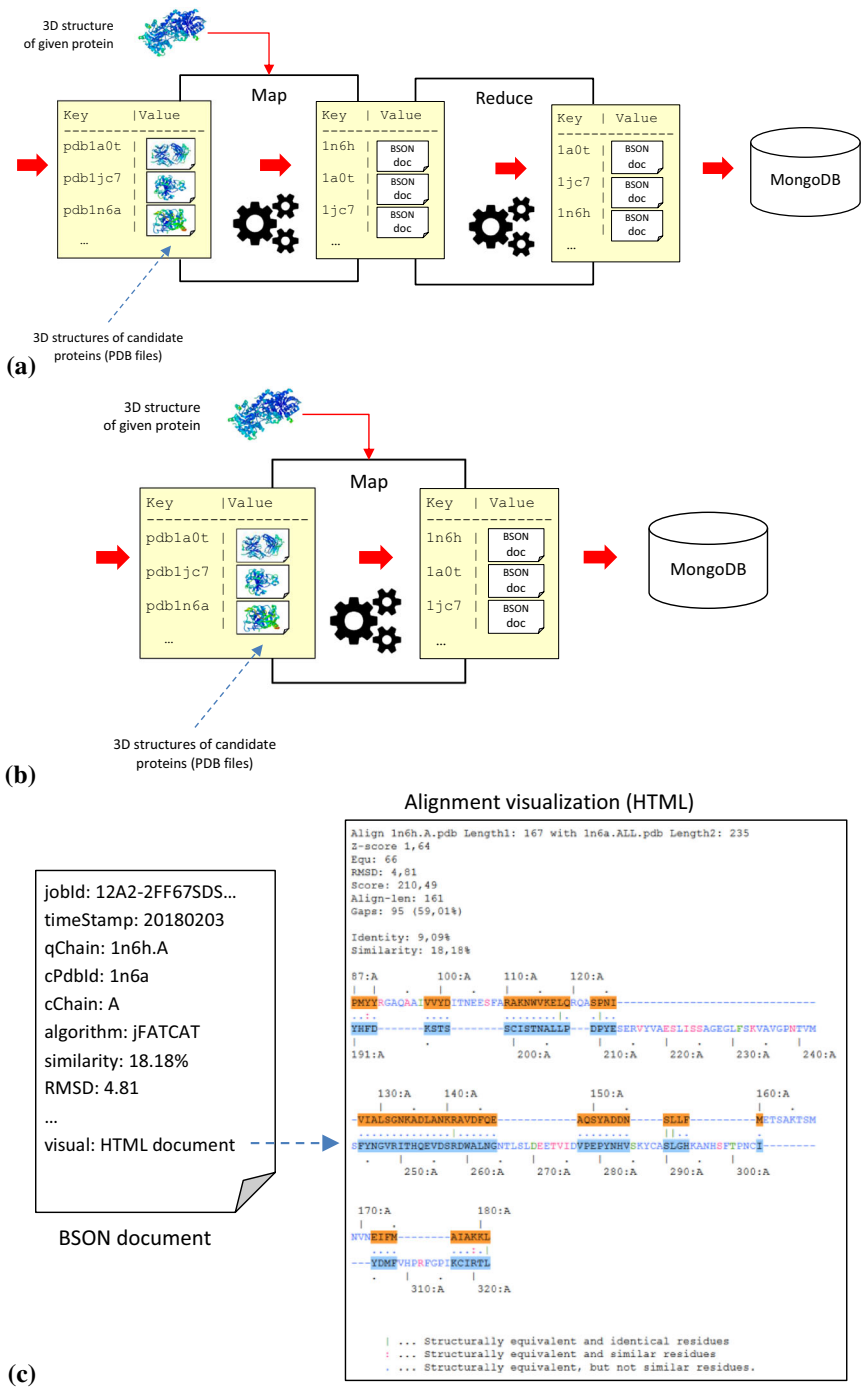

Fig. 4 Simplified idea of 3D protein structure alignment implemented as the MapReduce job with: full MapReduce (**a**) and Map-only (**b**) processing patterns. The alignment implements one-to-many comparison scenario with 3D structures of candidate proteins delivered as sequential files. (**c**) Single BSON document generated as the output of the Map phase

We can state that during the MapReduce job execution with the use of the Map-only processing pattern:

$$\exists f_1 \quad S_H \xrightarrow{f_1} T_M, \tag{19}$$

where f_1 is the function that assigns input splits (sequential files with candidate protein structures) to Map tasks in such a way that:

$$\begin{aligned} &\forall s_{H,u}, s_{H,w} \in S_H, \ u, w = 1, 2, \ldots, U, \\ &f_1(s_{H,u}) = f_1(s_{H,w}) \Longrightarrow s_{H,u} = s_{H,w}. \end{aligned} \tag{20}$$

The Hadoop cluster can be defined as a set of computing nodes:

$$C_H = \left\{ c_{H,d} | d = 1, 2, \ldots, D \right\}, \tag{21}$$

where D is the number of cluster nodes c_H.

We can state that during Hadoop computations with the use of the Map-only processing pattern:

$$\exists f_2 \quad J_{MR} \xrightarrow{f_2} C_H, \tag{22}$$

where f_2 is the function that assigns Map tasks to the Hadoop cluster nodes in such a way that:

$$\forall c_H \in C_H, \quad \exists t_M \in J_{MR}, \quad f_2(t_M) = c_H. \tag{23}$$

2.3 Implementation of the Map-only processing pattern in the H4P

The H4P uses the Map-only processing pattern when parallelizing computations related to massive 3D protein structure alignment. When the H4P Map-based application is executed on the Hadoop cluster (Fig. 5), it creates the MapReduce job that is parallelized on nodes of the cluster. The H4P application passes all configuration parameters to the MapReduce job and passes the job for execution. The job is executed under control of the ApplicationMaster service. The ApplicationMaster initializes the MapReduce job (step 3) and arranges data into input splits (step 4), which are logical representation of macromolecular data stored in file blocks (in the Hadoop Distributed File System). The number of input splits depends on the format and size of input data. The H4P accepts input splits with protein structures stored as individual files or sequential files. Sequential files contain many records, and each single record contains macromolecular data for one candidate 3D protein structure (exactly, the file name and compressed binary content). For each such an input split, the ApplicationMaster creates a Map task (step 5), which processes each record of the input split (each candidate protein structure) by execution of the dedicated `map` function. The pseudocode of the Map task with the `map` function is presented in Listing 1. Many Map tasks that are created for each of the input splits are executed in parallel on the available nodes of the Hadoop cluster (steps 6–8). Execution of Map tasks on a single compute node of the Hadoop cluster is controlled by NodeManager service. Appropriate algorithm of data processing is invoked in the `map` function, executed within the Map task. The `map` function is responsible for finding common fragments in two protein structures (given query structure and a candidate protein structure from the input split) by performing structural alignment and 3D superposition of compared protein structures. Once the Map task is completed (step 9), another Map task can be assigned to the worker node for execution, until all input splits are processed. The algorithm of processing data in a single Map task in one-to-many comparison scenario with the use of sequential files (against large repositories of candidate protein structures) is presented in Fig. 6 and Listing 1.

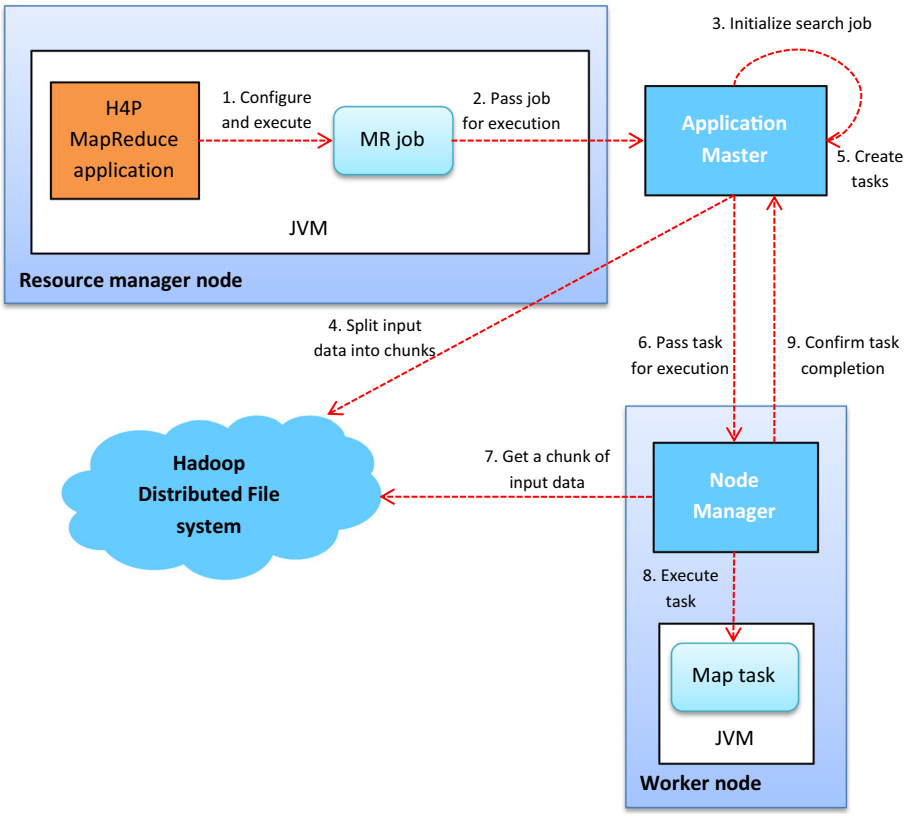

Fig. 5 Execution of the H4P MapReduce application in Hadoop

```
//prepare query protein structure for comparison
//and get alignment algorithm name from the job context
setup( context )
{
  algorithmName = context.getAlignAlgorithmName();
  qChainID = context.getQChainID();
  queryProtChain = getStructureHDFS(
        context.queryProtein(qChainID));
}

//key: name of the PDB file with a single candidate protein
//value: content of the PDB file with a single candidate protein
map( key, value, context )
{
  candidatePdbId = key;
  List isomerList = value.getIsomers();

  for each isomer in isomerList do
  {
    //get list of chains of the candidate protein
    List chainList = isomer.getChains();
    for each chain in chainsList do
    {
     afpChain = align(queryProtChain, chain, algorithmName);
     identity = afpChain.getIdentity();
```

```
        similarity = afpChain.getSimilarity();
        rsmd = afpChain.getRMSD();
        score = afpChain.getScore();
        probability = afpChain.getProbability();
        alignmentHTML = afpChain.getWebDisplayResult();
        docBSON = createBSONdocument(jobId, timeStamp,
            candidatePdbId, candidatePdbChainId, identity,
            similarity, rmsd, score, probability,
            alignmentHTML);
        writeToMongoDB(queryProteinChain, candidatePdb,
            docBSON);
      }
    }
}
```

Listing 1 Pseudocode of the Map task and the *map* function.

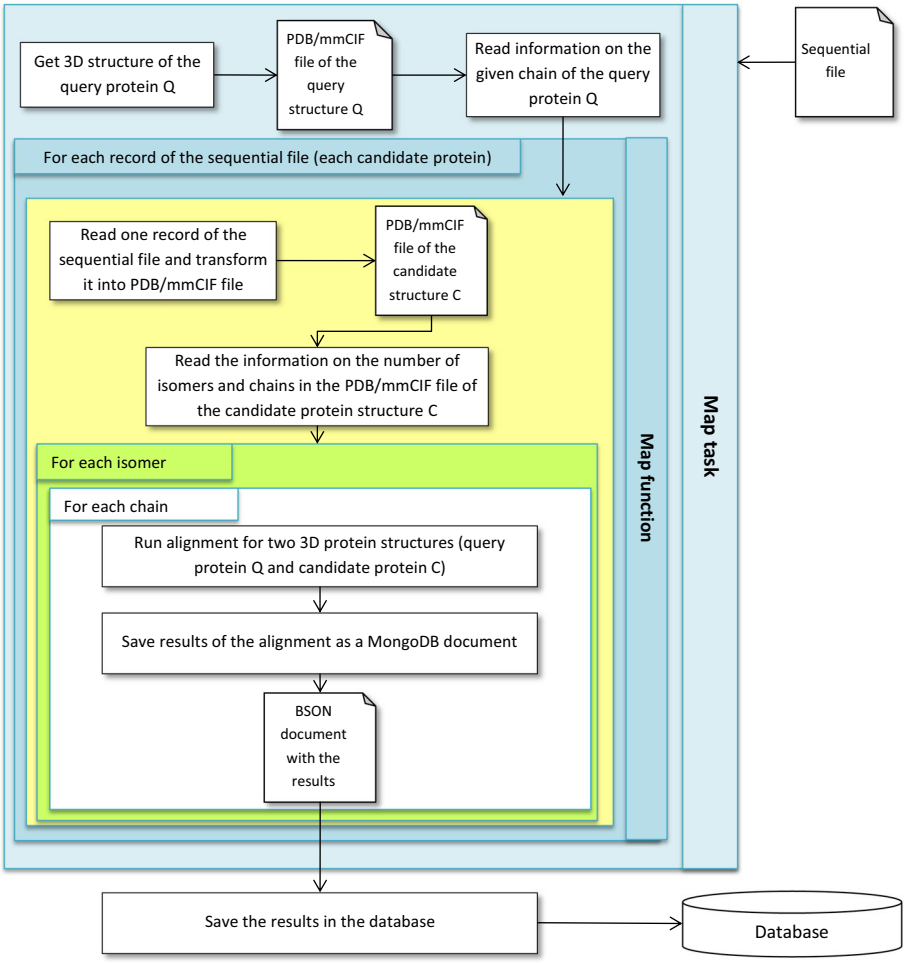

Fig. 6 Algorithm of macromolecular data processing, alignment and similarity searching in the Map task for one-to-many comparison scenario with the use of sequential files (against large repositories of candidate protein structures)

The first step of the Map task involves retrieving the name of the alignment algorithm and getting the 3D structure of the user's query protein (Q). This is done in the `setup` function on the basis of job configuration available in the job context (lines 5–6, Listing 1). Macromolecular data for the specified query protein structure are retrieved from an appropriate PDB/mmCIF file representing given protein structure (lines 7–8). The file should be located in the HDFS. The name of the file and path in the HDFS file system is given as parameters in the job context. The name of the algorithm used for structural alignment is also passed through the job context (line 5). The use of sequential files causes the `map` function to be invoked multiple times in a single Map task, once for every record of the sequential file (every candidate protein). For this reason, macromolecular data of the query structure and the alignment method are stored in the internal memory of the task (private attributes of the `Mapper` class), which allows for reusing them in each invocation of the `map` function (line 13). Such a solution ensures that macromolecular data of the query protein structure will be loaded and processed only once within the Map task, which positively affects the performance of the whole process.

The `map` function is executed individually for each record of the sequential file (each candidate protein structure). As input parameters, the `map` function accepts the name of the macromolecular data file for a candidate protein structure (as the `key` parameter) and the macromolecular data that is passed to it through the `value` (lines 11–15). This organization is also adapted in sequential files. This makes the sequential files compatible with the input of the `map` function.

The `map` function retrieves the list of isomers for the candidate protein structure. (Some proteins contain many isomers, line 16.) For each isomer, it then retrieves all chains (line 22) and performs alignment of the each chain to the chain of the query protein (lines 22–24). Comparison and alignment of two protein structures, which generates a set of similarity measures (lines 25–29), are performed by means of the jCE or the jFACTAT method. Importantly, the alignment comprises a pair of protein chains, which includes:

- a specified chain of the given, query protein Q (`queryProtChain`, line 24),
- all, successive chains of the candidate protein C (`chain`, line 24) as proteins may have many chains, and each of the chain has its own tertiary (3D) structure, and therefore, it is processed separately.

Each alignment generates the *chain of AFPs* (aligned fragment pairs), which is a result of used jCE or jFATCAT alignment methods. Depending on the number of chains in the candidate protein structure C, in each single invocation of the `map` function, the alignment can be executed more than once. As a result, single invocation of the `map` function can provide many similarity results (many sets of similarity measures), but each subsequent outcome is related to different pair of compared protein chains. Finally, results of each alignment process, including algorithm-specific similarity measures (like identity, similarity, RMSD, score, and probability), timestamp, and identifier of the candidate protein structure and its chain, are gathered and saved as BSON documents that are stored in the MongoDB database [3] (lines 31–35). Single entry in the MongoDB database contains the entry id, identifiers of compared proteins and their chains, and the binary BSON document with the outcome of a single alignment.

For one-to-one comparison scenarios (or more precisely, when comparing several pairs of proteins), the `map` function is invoked only once in the Map task, for a pair of compared proteins. For many-to-many comparison scenarios, each separate sets of Map tasks are created for each given query protein structure, and within each such a set, processing occurs in the same way as presented in Fig. 6 and Listing 1.

3 Experimental results

Parallel, Map-based procedures for massive 3D protein structure alignments were extensively tested in order to verify their performance. The main goal of the experiments was to address the following questions:

- What is the efficiency of massive structural alignments performed on the Hadoop with the use parallel, Map-based versions of the jCE and the jFATCAT methods?
- How the use of sequential files affects the performance of similarity searches?
- How scalable the H4P with the implemented parallel, Map-based methods is?
- What is the efficiency of H4P's procedures compared to other mentioned GPU-based and cloud-based systems, and MapReduce-based procedures for structural alignment?

3.1 Runtime environment

Runtime environment used for performed experiments was established on virtualized computer cluster equipped with two 4-core (8-thread) physical CPUs Intel Xeon 2.40 GHz and 96 GB RAM. In this environment, we created eight virtual machines—each one was assigned two logical CPUs (from the pool of 16 available) and 30 GB of local storage. One of the machines was assigned 4 GB of RAM, and remaining machines were assigned 2 GB of RAM each. Each of the virtual machines worked under the control of Ubuntu 12.04 LTS operating system, and each had the Hadoop framework and all required software for parallel structural alignments installed on it. All virtual machines were connected by a common network, which gave the possibility to configure and run the Hadoop cluster containing up to 8 nodes. In this computational cluster:

- one virtual machine served as both the Master node, as well as the Worker node,
- remaining (up to seven) virtual machines served exclusively as Worker nodes that performed protein comparisons.

Such a prepared environment was used to run the H4P system with parallel procedures for structural alignments and carry out assumed performance experiments.

3.2 Dataset

Preparation of the test data for performed research experiments involved sampling and downloading from the PDB repository a certain number of files in the PDB format. The set of test data was a subset of the PDB containing 1000 randomly selected protein structures (as in the tests performed by Hung and Lin). PDB files representing the selected protein structures have been downloaded from the PDB FTP server and stored hierarchically in 66 folders. (This corresponded to the way how these files were stored in the PDB repository.) Then, by using an auxiliary program (also MapReduce-based program implemented as an additional component of the H4P), the downloaded files have been converted and stored as sequential files. In this way, for the 66 folders containing a total of 1000 PDB files there were created 66 sequential files. These sequential files were then placed in the HDFS distributed file system of the running Hadoop cluster, enabling multiple processing of the files in the H4P system. For the purposes of our research, in the HDFS we also placed a set of test data in the form of a non-transformed, individual PDB files. All the data in the HDFS were subjected to automatic replication, wherein the degree was dependent on the number of cluster nodes used in particular test case.

Table 1 Values of the data replication parameter in HDFS for various numbers of cluster nodes

Number of Hadoop computing nodes	Data replication factor
1	1
2	2
4	3
6	3
8	3

3.3 A course of experiments

In our investigations over the scalability of the H4P system, we carried out many numerical experiments in which we used (Deoxy) Hemoglobin (Alpha Chain) [6] as a query protein structure (Q). The protein is identified by 2HHB PDB code in the Protein Data Bank (PDB). This is a medium-sized protein that consists of two amino acid chains, and structural alignment in each of experiments concerned the chain A. Structural alignments were performed with the use of parallel, Map-based versions of the jCE and the jFATCAT (flexible) methods.

For each experiment scheme, the process of protein similarity searching comprised the entire dataset of 1000 proteins, which gave a total of 3321 amino acid chains being aligned with a given, query protein chain. The number of computing nodes of the Hadoop cluster, on which experiments were conducted, ranged from one to eight, namely $d \in \{1, 2, 4, 6, 8\}$. Depending on the experiment scheme, each computing node of the cluster was adapted to carry out one or simultaneously two mapping operations (Map tasks). As a result, we distinguished two schemes for conducted experiments, where

- single compute cluster node was responsible for the execution of one Map task, i.e., the maximum number of parallel mapping operations running in the cluster was equal to the number of compute nodes of the cluster currently in use, i.e., $s' \in \{1, 2, 4, 6, 8\}$;
- single compute cluster node was responsible for the simultaneous execution of two Map tasks, i.e., the maximum number of parallel mapping operations running in the cluster was twice the number of compute nodes of the cluster currently in use, i.e., $s' \in \{2, 4, 8, 12, 16\}$.

In order to test various comparison scenarios, the experiment schemes described above were carried out for the test data provided both in the form of Hadoop sequential files, as well as in their direct form—as individual files in the PDB format. At this point, it should also be noted that the total number of Map tasks that were executed in the system every time was equal to the number of input data files. Therefore, for tests conducted using sequential files, the number of Map tasks was equal to the number of files, i.e., 66. And, in experiments carried out with the use of individual PDB files, the number of Map tasks executed in the system reached the value of 1000.

It is also important that in the course of experiments, we changed the degree of data replication. Macromolecular data were dispersed throughout the HDFS file system. Table 1 shows the values which the data replication parameter received in the system, depending on the number of active nodes in the cluster.

3.4 Results

Performance tests were conducted for both experiment schemes (one Map task per node versus two Map tasks per node) and for two comparison scenarios (one-to-one and one-to-

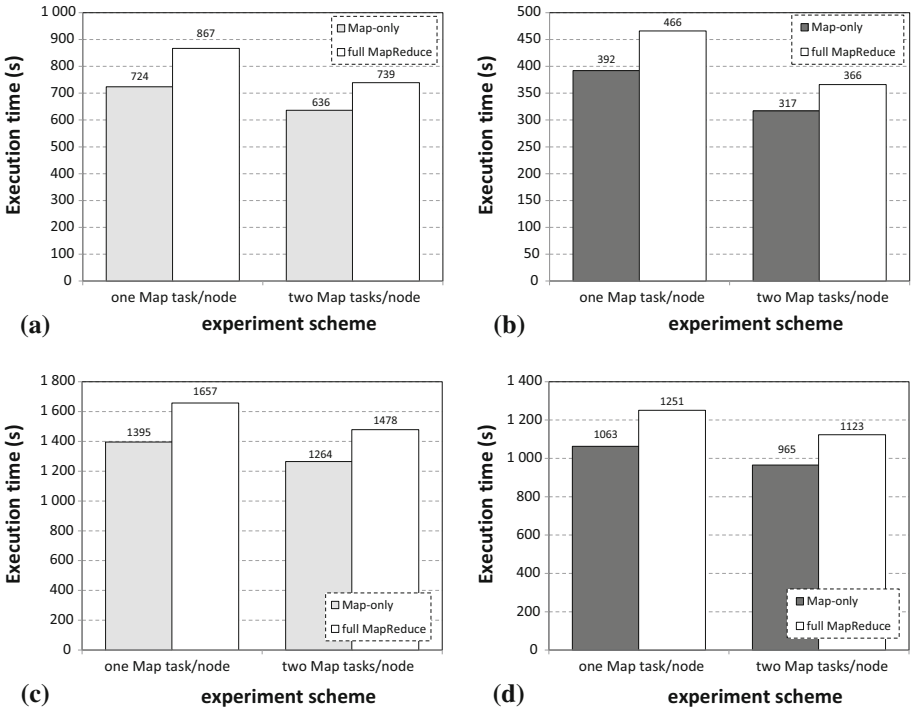

Fig. 7 Execution time for various MapReduce execution patterns (*Map-only* and *full MapReduce*) and various experiment schemes for the jFATCAT (**a**, **c**) and the jCE (**b**, **d**) structural alignment algorithms. Tests performed for the 8-node Hadoop cluster for one-to-many comparison scenario with sequential files (**a**, **b**) and batch one-to-one comparison scenario with individual PDB files (**c**, **d**)

many scenarios). The many-to-many comparison scenario is an extension of the one-to-many scenario with the use of sequential files. In all cases, performance has been assessed on the basis of execution time measurements. We have carried out at least two replicas for each measurement. Then, the obtained results were averaged and in such a way they are presented in the following sections. Averaged values of measurements were also used to determine n-fold speedups when scaling computations on Hadoop cluster.

3.4.1 Comparison of Map-only and full MapReduce execution patterns

In the first series of tests, we wanted to empirically verify the assumption that skipping the Reduce phase will bring improvement of performance of the whole massive alignment process. To this purpose, we performed experiments according to both experiment schemes (one map task per cluster node and two map tasks per cluster node) with sequential files (one-to-many comparison scenario) and individual PDB files (batch one-to-one comparison scenario) on the 8-node Hadoop cluster. Results of these experiments presented in Fig. 7 show the execution time for each of the comparison scenario and MapReduce implementation.

As can be observed from Fig. 7, the Map-only execution pattern of the MapReduce processing model is faster in all tested comparison scenarios (one-to-many and batch one-to-one) and experiment schemes (one Map task/node and two Map tasks/node). For example, for the one-to-many comparison scenario with sequential files, the jFATCAT algorithm, and two

Map tasks running on each node (Fig. 7a, right chart bars), the execution with the use of the Map-only pattern takes 636 s, while full MapReduce implementation takes 739 s (16% longer). Similarly, for the batch one-to-one comparison scenario with individual PDB files, the jCE algorithm, and one Map task running on each node (Fig. 7d, left chart bars), the execution with the use of the Map-only pattern takes 1063 s, while full MapReduce implementation takes 1251 s (18% longer). In all tested cases, the full MapReduce implementation is 15–20% less efficient than the Map-only execution pattern.

3.4.2 Results for one-to-many comparison scenario with sequential files

Hadoop sequential files are the default format of the data processed in the H4P system. The results presented in this chapter relate to experiments carried out with molecular data stored in the form of sequential files. Figure 8 shows the results of experiments performed according to both experiment schemes (one map task per cluster node and two map tasks per cluster node) and illustrates the change in computation time, depending on the number of Hadoop compute nodes.

As can be observed, with the growing number of Hadoop nodes the time needed to complete the whole MapReduce job drops significantly. For structural alignments with the jFATCAT algorithm, the execution time was reduced from 5167 s (on one node) to 724 s (on eight nodes of the Hadoop cluster) for one Map task/node experiment scheme, and from 2612 s (on one node) to 636 s (on eight nodes of the Hadoop cluster) for two Map tasks/node experiment scheme. For structural alignments with the jCE algorithm, the execution time was reduced from 2844 s (on one node) to 392 s (on eight nodes of the Hadoop cluster) for one Map task/node experiment scheme, and from 1574 s (on one node) to 317 s (on eight nodes of the Hadoop cluster) for two Map tasks/node experiment scheme.

In order to better illustrate the results, we show the acceleration of the computations in Table 2. The acceleration is calculated according to the following formula and obtained by comparing particular execution times to the case in which calculations are performed by only one Hadoop node executing one map task (this mimics the stationary/serial implementation of protein structure alignment):

$$S_{m,d} = \frac{T_{1,1}}{T_{m,d}}, \tag{24}$$

where $m = 1$ or 2 is the number of Map tasks performed in parallel on a single node of the Hadoop cluster, d is the number of data nodes used, $T_{1,1}$ is the execution time obtained while performing computations with the use of 1-node cluster and one Map task, and $T_{m,d}$ is the execution time obtained while performing computations on the d-node cluster configured to execute in parallel m Map tasks per node.

From the obtained results, it can be concluded that both the increase in number of nodes, and the number of Map tasks resulted in significant acceleration of calculations, and thus, increase in the overall system performance.

Table 3 shows how the average time of matching a pair of amino acid chains changed with the number of Hadoop nodes and the number of Map tasks executed in parallel on each node.

3.4.3 Results for batch one-to-one comparison scenario with individual PDB files

Stationary implementations of the jCE and the jFATCAT algorithms accept individual PDB/mmCIF files as input data. Our Map-based implementations of the algorithms also allow to perform structural alignments for pairs of unprocessed, individual files from PDB

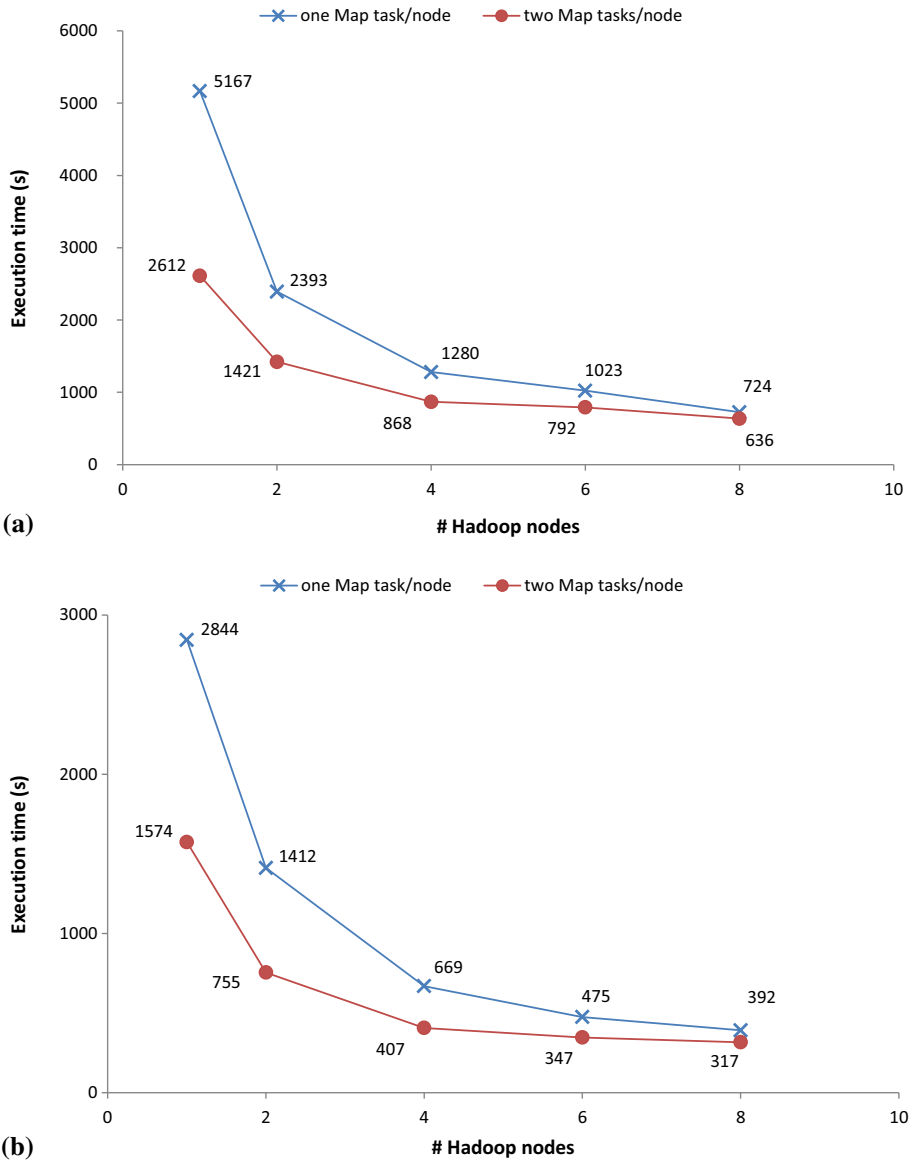

Fig. 8 Dependency between the execution time and the number of computing nodes of the Hadoop cluster (for one-to-many comparison scenario with sequential files) for parallel, Map-based versions of the jFATCAT (**a**) and the jCE (**b**) structural alignment algorithms

repository and individual protein structures from local collections stored in private compute environments, e.g., private clouds. In order to verify performance of such a comparison scenario, we repeated experiments described in the previous subsection, but this time for a different format of input data. In this case, the source of input data for the system was unprocessed, individual protein structures stored in the PDB/mmCIF format.

Results of experiments performed with the use of input data stored in a standard PDB format are presented in a manner analogous to that of the previous section. Batch (exactly,

Table 2 Acceleration of calculations for the growing number of Hadoop computing nodes (for one-to-many comparison scenario with sequential files) for parallel, Map-based versions of the jFATCAT and the jCE structural alignment algorithms

Number of Hadoop computing nodes	SpeedUp			
	jFATCAT		jCE	
	1 Map task/node	2 Map tasks/node	1 Map task/node	2 Map tasks/node
1	1.00	1.98	1.00	1.81
2	2.16	3.64	2.01	3.77
4	4.04	5.95	4.25	6.99
6	5.05	6.52	5.99	8.20
8	7.14	**8.12**	7.26	**8.97**

Bold indicates the highest speedup achieved for a particular algorithm

Table 3 Average time for matching pairs of amino acid chains (for the one-to-many comparison scenario with sequential files) for parallel, Map-based versions of the jFATCAT and the jCE structural alignment algorithms

Number of Hadoop computing nodes	Average time for matching pairs of amino acid chains (s)			
	jFATCAT		jCE	
	1 Map task/node	2 Map tasks/node	1 Map task/node	2 Map tasks/node
1	1.56	0.79	0.86	0.47
2	0.72	0.43	0.43	0.23
4	0.39	0.26	0.20	0.12
6	0.31	0.24	0.14	0.10
8	0.22	**0.19**	0.12	**0.095**

Bold indicates the shortest average execution time achieved for a particular algorithm

1000) one-to-one comparisons were performed with the use of individual protein structures. In Fig. 9, we show the dependency between the execution time and the number of compute nodes in the Hadoop cluster. Similarly to experiments with sequential files, we can observe a vast acceleration of similarity searches while increasing the number of Hadoop nodes. For structural alignments with the jFATCAT algorithm, the execution time was reduced from 9771 s (on one node) to 1395 s (on eight nodes of the Hadoop cluster) for the one Map task/node experiment scheme, and from 6062 s (on one node) to 1264 s (on eight nodes of the Hadoop cluster) for the two Map tasks/node experiment scheme. For structural alignments with the jCE algorithm, the execution time was reduced from 7106 s (on one node) to 1063 s (on eight nodes of the Hadoop cluster) for the one Map task/node experiment scheme, and from 4739 s (on one node) to 965 s (on eight nodes of the Hadoop cluster) for the two Map tasks/node experiment scheme.

Speedup calculated based on the execution time measurements for the particular Hadoop cluster configuration and the experiment scheme is shown in Table 4. From Table 4, for experiments with individual PDB files, we can draw similar conclusions to those related to the use of sequential files. During the course of experiments, it turned out that also in this case, with an increasing number of computing nodes of the Hadoop cluster, the time required to perform similarity searches is significantly reduced relative to the case of using a single node with one Map task ($T_{1,1}$).

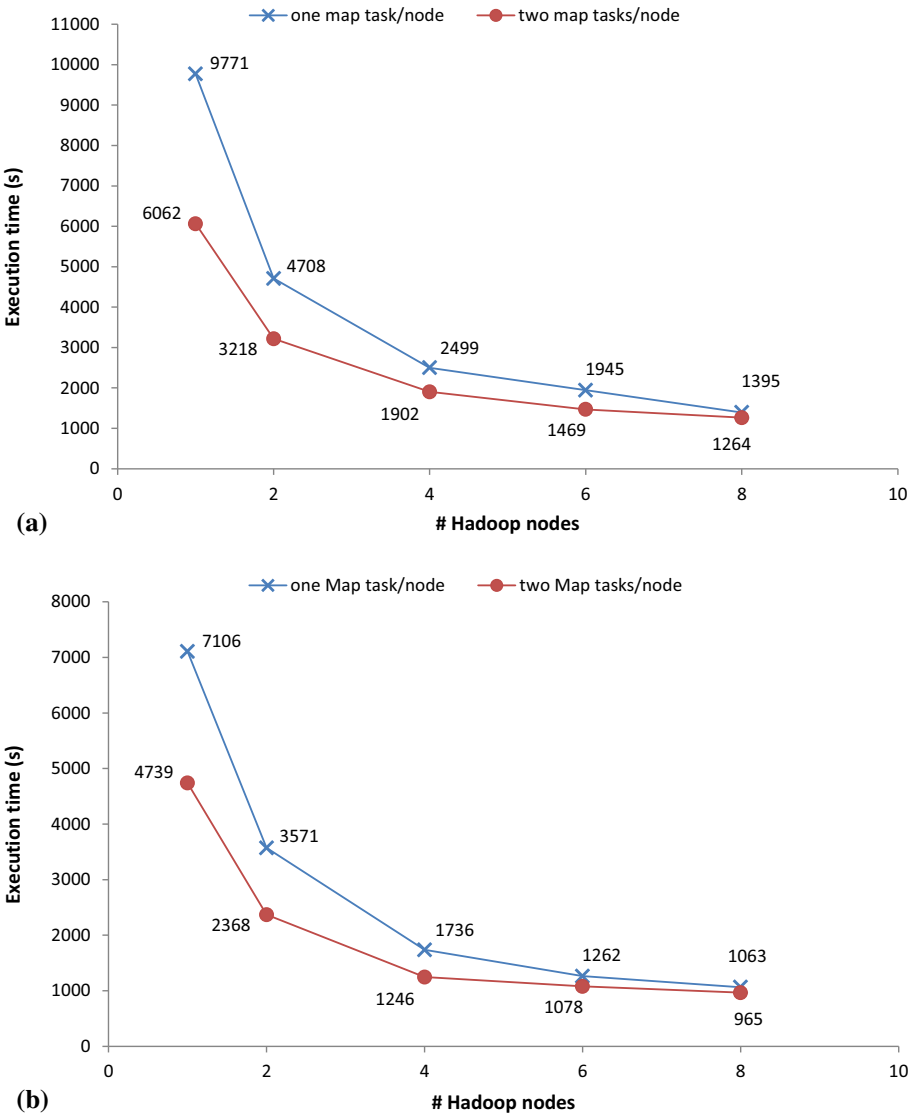

Fig. 9 Dependency between the execution time and the number of computing nodes of the Hadoop cluster (for the batch one-to-one comparison scenario with individual PDB files) for parallel, Map-based versions of the jFATCAT (**a**) and the jCE (**b**) structural alignment algorithms

Table 5 shows how the average time of matching a pair of amino acid chains changed with the number of Hadoop nodes and the number of Map tasks executed in parallel on each node.

3.4.4 Comparison of results for one-to-many and batch one-to-one comparison scenarios

In this section, we compare results obtained while performing one-to-many comparisons of protein structures with sequential files and batch one-to-one comparisons of protein structures

Table 4 Acceleration of calculations for the growing number of Hadoop computing nodes (for the batch one-to-one comparison scenario with individual PDB files) for parallel, Map-based versions of the jFATCAT and the jCE structural alignment algorithms

Number of Hadoop computing nodes	SpeedUp			
	jFATCAT		jCE	
	1 Map task/node	2 Map tasks/node	1 Map task/node	2 Map tasks/node
1	1.00	1.61	1.00	1.50
2	2.08	3.04	1.99	3.00
4	3.91	5.14	4.09	5.70
6	5.02	6.65	5.63	6.59
8	7.00	**7.73**	6.68	**7.36**

Bold indicates the highest speedup achieved for a particular algorithm

Table 5 Average time for matching pairs of amino acid chains (for the batch one-to-one comparison scenario with individual PDB files) for parallel, Map-based versions of the jFATCAT and the jCE structural alignment algorithms

Number of Hadoop computing nodes	Average time for matching pairs of amino acid chains (s)			
	jFATCAT		jCE	
	1 Map task/node	2 Map tasks/node	1 Map task/node	2 Map tasks/node
1	2.94	1.83	2.14	1.43
2	1.42	0.97	1.08	0.71
4	0.75	0.57	0.52	0.38
6	0.59	0.44	0.38	0.32
8	0.42	**0.38**	0.32	**0.29**

Bold indicates the shortest average execution time achieved for a particular algorithm

with individual PDB/mmCIF files. We also present the acceleration rates for both experiment schemes (one Map task/node and two Map tasks/node). Figure 10 illustrates the comparison of the runtime for both comparison scenarios, depending on the number of used computing nodes of the Hadoop cluster and for both experiment schemes. Figure 10a, c presents the results obtained for the experiment scheme where each node was responsible for execution of only one Map task per node. In turn, Fig. 10b, d presents the results of the same experiment, except that each node was adapted to execute in parallel a maximum of two Map tasks per node. The total number of comparisons of protein structures performed in both scenarios was equal to 1000 (with respect to the number of protein structures) or 3321 (with respect to the number of amino acid chains).

Analyzing results gathered in Fig. 10 we can see that by delivering macromolecular data in the form of sequential files gives nearly twofold to threefold increase in system performance in both described experiment schemes and in all tested cases. This is also visible in Table 6, which shows speedups for both experiment schemes and various configurations of the Hadoop cluster calculated according to the following equation:

$$S_{f,d} = \frac{T_{i,d}}{T_{s,d}}, \tag{25}$$

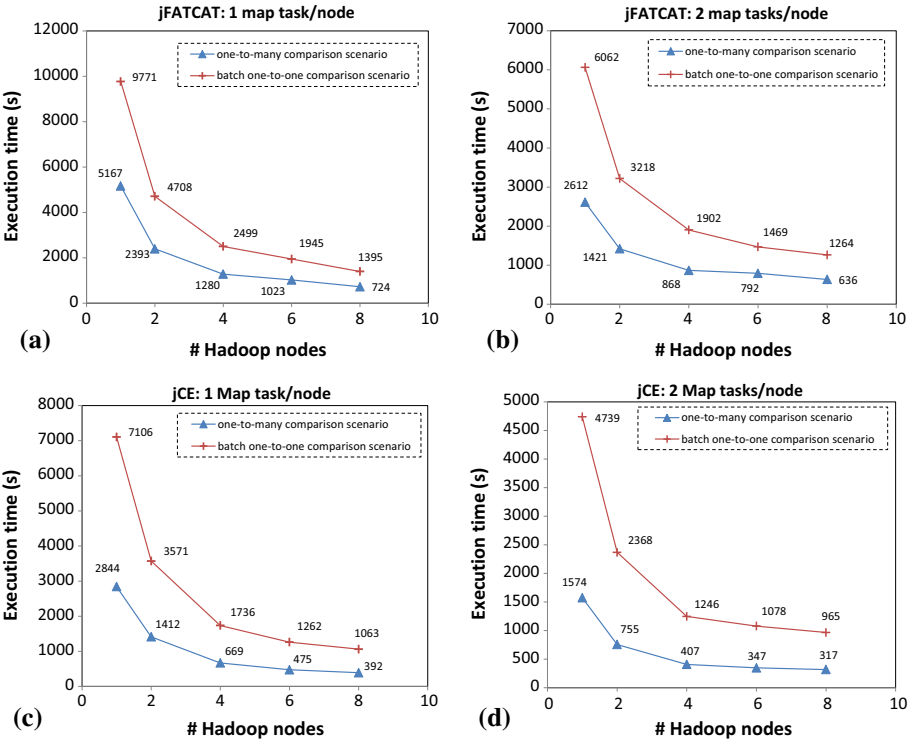

Fig. 10 Comparison of execution times for the varying numbers of computing nodes of the Hadoop cluster for one-to-many comparisons of protein structures with sequential files and batch one-to-one comparisons of protein structures with individual PDB/mmCIF files performed with the experiment scheme where each node was responsible for execution of only one Map task per node (**a**, **c**) and two Map tasks per node (**b**, **d**) for the jFATCAT (**a**, **b**) and the jCE (**c**, **d**) alignment algorithms

where d is the number of nodes of the Hadoop cluster, $T_{i,d}$ is the execution time obtained while performing computations with individual PDB/mmCIF files (in the batch one-to-one comparison scenario) on the d-node cluster, and $T_{s,d}$ is the execution time obtained while performing computations with sequential files (in the one-to-many comparison scenario) on the same d-node cluster.

On the basis of results presented in Table 6, we can draw the conclusion that the acceleration $S_{f,d}$ of the calculations obtained by using sequential files is quite stable for varying numbers of nodes of the Hadoop cluster in each experiment scheme—average speedups range between 1.93 and 3.07 with standard deviations ranging between 0.03 and 0.18, and the speedup is larger when computing nodes execute two Map tasks in parallel.

3.4.5 Influence of the number of map tasks on the acceleration of computations

To further investigate the behavior of the parallel, Map-based procedures for structural alignment of proteins implemented in the H4P, we conducted additional experiments. In these experiments, we tried to identify how the number of logical processors assigned to a single node of the Hadoop cluster and the number of parallel Map tasks influence the acceleration of the structural alignments. All tests were conducted for the one-to-many comparison scenario

Table 6 Acceleration of calculations performed with sequential files in relation to individual PDB files for varying numbers of Hadoop cluster nodes, for both experiment schemes (one map task per cluster node and two map tasks per cluster node), and for parallel, Map-based versions of the jFATCAT and the jCE structural alignment algorithms

Number of Hadoop computing nodes	SpeedUp			
	jFATCAT		jCE	
	1 Map task/node	2 Map tasks/node	1 Map task/node	2 Map tasks/node
1	1.89	2.32	2.50	3.01
2	1.97	2.26	2.53	3.14
4	1.95	2.19	2.59	3.06
6	1.90	1.85	2.66	3.11
8	1.93	1.99	2.71	3.04
Average	1.93	2.21	2.60	3.07
Standard deviation	0.03	0.18	0.08	0.04

Table 7 Acceleration of calculations performed for one-to-many comparison scenario with sequential files for varying numbers of logical processors and varying numbers of Map tasks performed in parallel on a single node of the Hadoop cluster

Number of Hadoop computing nodes	SpeedUp $S_{m,c}$				
	1 Map task	2 Map tasks	3 Map tasks	4 Map tasks	5 Map tasks
1	**1.00**	1.02	0.99	1.00	1.00
2	2.77	**5.48**	5.42	5.46	5.45
3	2.81	5.95	**8.01**	8.01	8.01
4	2.79	5.95	8.25	**9.99**	9.42

Bold indicates the highest speedup achieved for a particular configuration of the system

with the use of sequential files as input data. Table 7 shows the variability of the speedup for the varying numbers of logical processors per node and the varying numbers of Map tasks performed in parallel on a single node of the Hadoop cluster. The speedup was calculated according to the following formula:

$$S_{m,c} = \frac{T_{1,1}}{T_{m,c}}, \tag{26}$$

where $m = 1 \ldots 5$ is the number of Map tasks performed in parallel on a single data node of the Hadoop cluster, c is the number of logical CPUs assigned to a single node, $T_{1,1}$ is the execution time obtained while performing computations with the use of one logical CPU assigned to the cluster node and one Map task, and $T_{m,c}$ is the execution time obtained while performing computations with the use of c logical CPUs assigned to a single cluster node configured to execute m Map tasks in parallel.

The analysis of results of this experiment allowed us to conclude that increasing the number of Map tasks simultaneously executed in a single computing node is meaningful only to the point where this number reaches a value equal to the number of logical processors assigned to the node. After exceeding this threshold, the acceleration value stabilized at a certain level and despite continued increasing of the number of simultaneously executed Map tasks, we did not recorded any further growth of the acceleration ratio.

Table 8 Average execution time per protein chain for various parallelization approaches

Used approach	Average time per chain	
	jFATCAT	jCE
Cloud4PSi (8 VMs, single-core)	0.99	0.70
CloudPSR (8 VMs, single-core)	2.16	NA
HDInsight4PSi (8 nodes)	0.15	0.17
H4P-IF-2MT (8 nodes)	0.38	0.29
H4P-SF-2MT (8 nodes)	0.19	0.0955
GPU-CASSERT	1.76183E−05	
GPU-CASSERT with CPU superposition	0.67	

Results for the H4P performing alignments with individual (IF, many one-to-one comparison scenario) and sequential (SF, one-to-many comparison scenario) files with two Map tasks (2MT) performed in parallel on a single node of the Hadoop cluster

3.4.6 Comparison of H4P performance against other approaches

MapReduce-based approaches for massive structural alignments, like H4P, HDInsight4PSi, and Hung and Lin's system, are one of possible approaches to increase the performance of the alignment process. In this section, we compare the H4P to other Hadoop-based approaches, approaches that rely on a farm of single-core virtual machines, like the Cloud4PSi and the CloudPSR, and the GPU-CASSERT that makes use of GPU devices to accelerate computations. The Cloud4PSi and the CloudPSR have dedicated role-based computer architectures, and their processing model does not utilize MapReduce patterns at all. Both are dedicated for Microsoft Azure public cloud and utilize its virtual machines (VMs). These virtual machines can have various compute capabilities depending on the VM series used. However, we used one to eight single-core VMs and data packages with one candidate protein (by default the Cloud4PSi uses 10 proteins per data package) in order to perform many (exactly 1000) one-to-one comparisons and to investigate the performance while using this naïve parallelization approach. The HDInsight4PSi and the H4P utilize MapReduce processing model, but the HDInsight4PSi scales computations on HBase clusters in the Microsoft Azure public cloud and allows optional Reduce phase to take place. All four systems provide parallel procedures for structural alignments performed with the jCE and the jFATCAT algorithms. The Hung and Lin's system, which also utilizes the Hadoop and the MapReduce, scales refined DALI [11] alignments on the private 8-node Hadoop cluster. The GPU-CASSERT parallelizes similarity searches with the CASSERT algorithm [33] on many-core streaming multiprocessors of GPU devices.

In Table 8, we can observe the comparison of average execution times per protein chain when comparing 1000 protein structures in one-to-one and one-to-many comparison scenarios for various parallelization approaches. Results clearly show that the H4P (H4P-IF-2MT, one-to-one comparisons with two Map tasks per node) was faster than the farm of single-core virtual machines in the Cloud4PSi (0.99 s for the jFATCAT, 0.70 s for the jCE) and in the CloudPSR (2.16 s, only jFATCAT), even if it processed individual files (0.38 s for the jFATCAT, 0.29 for the jCE). When utilizing sequential files, the H4P (H4P-SF-2MT, one-to-many comparisons with two Map tasks per node) was twice faster (0.19 for the jFATCAT and 0.0955 for the jCE) than the H4P processing individual files (H4P-IF-2MT). The HDInsight4PSi was the fastest out of all MapReduce-based systems (0.15 for the jFATCAT and

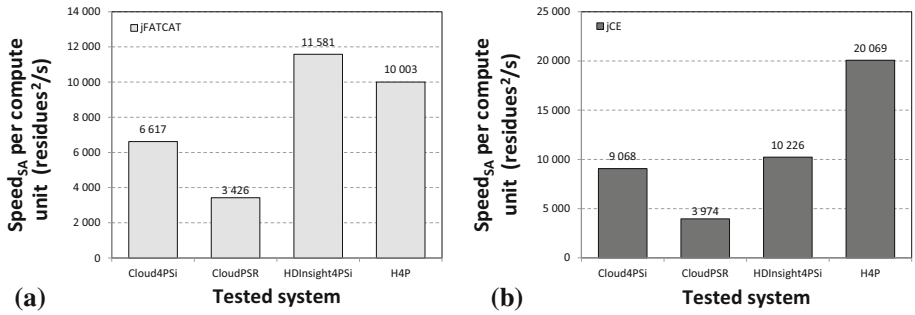

Fig. 11 Structural alignment speed per compute task achieved by parallelizing computations in four systems for massive structural alignments of 3D protein structures: **a** for jFATCAT alignment algorithm, **b** for jCE alignment algorithm. Tests performed for one-to-many comparison scenario in all systems, with sequential files (only HDInsight4PSi and H4P). H4P was configured for parallel execution of two Map tasks per node

0.17 for the jCE), which results from higher mapper-to-node ratio (31 parallel Map tasks on the 8-node HBase cluster versus 16 Map tasks on the H4P). The GPU-CASSERT outperformed all approaches with the average execution time 1.76183E−05 s per compared pair of protein chains. This results from using many cores of the GPU device and its streaming multiprocessors. However, the method only calculates the similarity of proteins on the GPU device. Together with the structural superposition, which is executed on the CPU, the average execution time was 0.67 s, which is worse than runtimes of the H4P in any configuration.

Implemented in the H4P, Map-only procedures for structural alignments of 3D protein structures are also among the most efficient cloud-dedicated solutions for the problem. In Fig. 11, we show performance comparison of four systems that implement parallel structural alignments in cloud environments: Cloud4PSi, CloudPSR, HDInsight4PSi, and H4P, expressed in terms of Structural Alignment Speed measure [30], which in Fig. 11 is averaged for compute unit/task performed in parallel. Structural Alignment Speed shows how many residue-to-residue comparisons are performed in a unit of time and allows measuring the performance of protein alignments regardless of the used collection of protein structures. As we can observe from Fig. 11, the H4P achieves the best average structural alignment speed of 20,069 (residues^2/s) per single Map task executed in parallel for the jCE alignment algorithm, which reflects almost two times better performance of the system than the HDInsight4PSi and the Cloud4PSi, and more than 5 times better performance than the CloudPSR. For the jFATCAT structural alignment algorithm, the performance of the H4P is slightly worse (10,003 residues^2/s) than performance of the HDInsight4PSi (11,581 residues^2/s), but much better than performance of the Cloud4PSi (6617 residues^2/s) and the CloudPSR (3426 residues^2/s). Slightly worse performance of the H4P for jFATCAT is a result of high memory consumption of the jFATCAT alignment algorithm and relatively limited memory resources that were available for Map tasks working in the virtualized environment of the established private cloud.

Map-only procedures for structural alignments implemented in the H4P scale very well in various Hadoop cluster configurations. In Fig. 12, we show speedups over serial executions achieved by approaches that utilize a farm of VMs (Cloud4PSi and CloudPSR) and three MapReduce-based implementations of procedures for structural alignment (Hung and Lin, HDInsight4PSi, and H4P), when scaling those systems from one to eight data nodes. Both, the Hung and Lin's and the H4P systems, were tested in the private clouds, while the HDInsight4PSi was tested in the Microsoft Azure public cloud. Hung and Lin reported linear

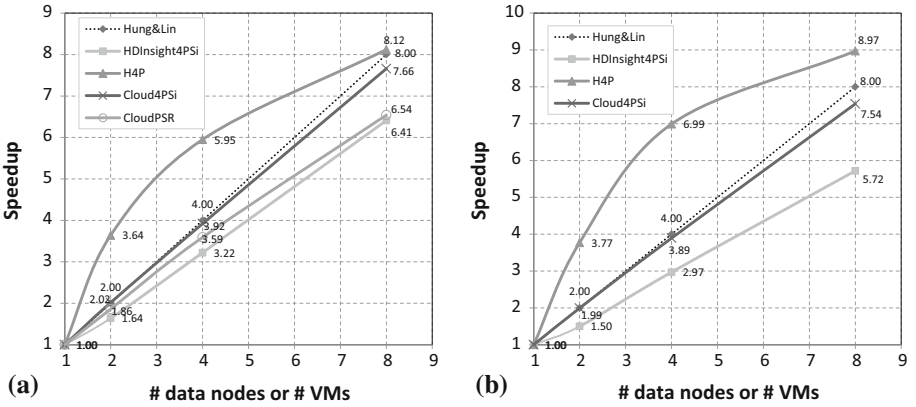

Fig. 12 Speedups achieved by three MapReduce-based implementations of structural alignment (Hung and Lin's, HDInsight4PSi, and H4P) and two VM-based approaches (Cloud4PSi and CloudPSR): (a) H4P, HDInsight4PSi, Cloud4PSi, and CloudPSR utilizing the jFATCAT alignment approach, (b) H4P, HDInsight4PSi, and Cloud4PSi utilizing the jCE alignment approach. Tests performed when comparing 1000 protein structures. The H4P was configured for parallel execution of two Map tasks per node

(ideal) speedup when scaling their system from one to eight Hadoop data nodes—4.0 for four data nodes, and 8.0 for eight data nodes. When scaling the HDInsight4PSi within the same range of data nodes of the HBase cluster, the system achieved sublinear speedup—5.72 for the jCE and 6.41 for the jFATCAT for eight data nodes. (Much better speedup was achieved when scaling the HDInsight4PSi above 16 data nodes, due to growing (*#Map tasks/#CPUs*) ratio, but both, Hung and Lin's system and the H4P, were not scaled out above eight data nodes. Moreover, in the Hung and Lin's system and the H4P, the (*#Map tasks/#CPUs*) ratio was constant.) The H4P achieved very good 8.97-fold (jCE) and 8.12-fold (jFATCAT) superlinear speedup on 8-node Hadoop cluster over serial execution of the alignment process in the experiment scheme with two Map tasks executed per one Hadoop node. In terms of efficiency gain, the approaches working on the basis a farm of single-core virtual machines, the Cloud4PSi achieved 7.66-fold speedup over the serial execution for the jFATCAT and 7.54-fold speedup for the jCE. The CloudPSR was much worse (6.54-fold speedup for jFATCAT) than the H4P and slightly better than HDInsight4PSi (Fig. 12a).

4 Discussion and conclusions

Parallel procedures for structural alignment of proteins become very important in the face of the growing number of 3D structures in repositories of macromolecular data, such as the Protein Data Bank, and in the face of structural genomics projects trying to predict protein structures on a large scale. These parallel procedures, including MapReduce-based or Map-only-based implementations, are extremely handy in one-to-many comparison scenarios, when we want to compare a given protein structure to a huge collection of structures, and in many-to-many comparison scenarios, when we want to compare all protein structures from the (growing) repository to each other. Hadoop-based implementations of the alignment process allow to bring computations to macromolecular structures of proteins stored in the Hadoop Distributed File System, where each of the structures is described by thousands of variables.

This paper shows one of possible applications of the Map-only pattern of the MapReduce processing model. Results of our experiments show that with the implementation of the pattern in the H4P, we can reduce average time of matching a pair of protein structures by, at least, one order of magnitude, from seconds to fractions of seconds, depending on the number of tasks executed in parallel, by scaling computations just from one to eight compute units. During the course of our experiments, we could also observe that by implementing the Map-only pattern (especially, in a hardware environment with limited compute resources) we could keep the number of Map tasks constant during the whole computation process. This was possible due to saving compute resources that are usually consumed by Reduce tasks (which can run in parallel after first Map tasks return their outcomes) that were omitted in the Map-only pattern implemented in the H4P.

As also shown in this paper, the number of Map tasks executed in parallel at each data node of the Hadoop cluster may vary depending on the hardware configuration. However, the best results were obtained for those hardware configurations where the number of parallel Map tasks did not exceed the number of logical CPUs assigned to the compute unit (Hadoop data node). Manipulation of the number of logical CPUs assigned to compute units is quite easy, especially in private clouds (to whom H4P is dedicated), where virtual machines can be flexibly tuned with respect to various parameters influencing performance of the computational cluster. The H4P is then fully adaptable to computational capabilities the Hadoop cluster established in such a virtualized environment of the private cloud.

Among unique features of the H4P's implementation of structural alignment, it is worth mentioning its flexibility in various comparison scenarios. Although the best performance was achieved for alignments performed with the use of sequential files, this approach of data feeding is suitable for one-to-many and many-to-many comparison scenarios. For single one-to-one and batch one-to-one comparison scenarios, it is better to use slower, but still parallel version that consumes individual PDB/mmCIF files with single protein structures. This slower implementation of parallel alignment procedure works similar to the one presented by Hung and Lin [13], which processed individual pairs of protein structures. However, the system developed by Hung and Lin was developed based on the full MapReduce processing pattern, with the Map phase for structural alignment and the Reduce phase for refinement of obtained results. The H4P also differs from the Hung and Lin's system with the implemented algorithms for structural alignment.

To summarize, we can draw several conclusions on the basis of the results of our research. First of all, we can notice that by implementing procedures for the structural alignment as parallel, Map-only jobs for Hadoop computing framework, we significantly increase the performance of the alignment process proportionally to the number of Hadoop data nodes used in computations. This results in shortening of the average time of the alignment processes performed in various comparison scenarios, especially in one-to-many and many-to-many structural alignments with the use of large collections of protein structures, but also in batch one-to-one alignments. Secondly, the Map-only pattern of the MapReduce processing model serves sufficiently well when performing computations, such as the alignment of 3D protein structure, that do not require secondary processing (sorting, grouping, and aggregating data) of primary results. Thirdly, for various comparison scenarios the H4P provides two versions of parallel, Map-only procedures for structural alignment—slower version that processes individual files is used in one-to-one comparison scenarios, and faster version that consumes sequential files is used in one-to-many and many-to-many comparison scenarios. Fourthly, Map-only procedures for structural alignments implemented in the H4P achieved much better average *Structural alignment speed* per compute task than most cloud-based systems. In particular, the H4P's parallel jCE-based alignment procedure outperforms other

Cloud-dedicated alignment procedures, and our parallel jFATCAT alignment procedure is only slightly worse than the procedure implemented in HDInsight4PSi for HBase clusters and much better than procedures implemented in other systems working on the Cloud as farms of virtual machines.

Our implementations are an example of a few rare solutions in the world that give a great promise showing how advances in computational solutions keep in pace with progress in large-scale data gaining in bioinformatics. Future works will be focused on development of the Spark-based procedures for massive structural alignments, mainly dedicated to many-to-many comparison scenarios. In many-to-many comparison scenarios, we need to use the same macromolecular data describing protein structures many times, while comparing proteins in various possible combinations. The Spark framework may be suitable for these scenarios as it allows to cache data and intermediate results in-memory for subsequent processing iterations. This would require re-implementation of the code of the procedures for structural alignment, but may bring additional time savings.

Acknowledgements This work was supported by Microsoft Research within Microsoft Azure for Research Award Grant, and Statutory Research funds of Institute of Informatics, Silesian University of Technology, Gliwice, Poland (Grant No. BK/213/RAU2/2018). A part of this work was supported by Polish National Centre for Research and Development (NCBR) within Applied Research Programme (PBS), project entitled: Platform for remote testing of scientific hypotheses and biomedical data analysis (BioTest), Grant No. DZP/PBS3/2441/2014/.

References

1. Berman H et al (2000) The protein data bank. Nucleic Acids Res 28:235–242
2. Can T, Wang YF (2003) CTSS: a robust and efficient method for protein structure alignment based on local geometrical and biological features. In: Computational systems bioinformatics. CSB2003. Proceedings of the 2003 IEEE bioinformatics conference. CSB2003, pp 169–179
3. Chodorow K (2013) MongoDB: the definitive guide, powerful and scalable data storage, 2nd edn. O'Reilly Media, Sebastopol
4. Coatney M, Parthasarathy S (2005) MotifMiner: efficient discovery of common substructures in biochemical molecules. Knowl Inf Syst 7(2):202–223. https://doi.org/10.1007/s10115-003-0119-4
5. Daniluk P, Lesyng B (2011) A novel method to compare protein structures using local descriptors. BMC Bioinform 12(1):344. https://doi.org/10.1186/1471-2105-12-344
6. Fermi G, Perutz M, Shaanan B, Fourme R (1984) The crystal structure of human deoxyhaemoglobin at 1.74 A resolution. J Mol Biol 175:159–174
7. Fober T, Hüllermeier E (2010) Similarity measures for protein structures based on fuzzy histogram comparison. In: International conference on fuzzy systems, pp 1–7
8. Fober T, Mernberger M, Klebe G, Hüllermeier E (2012) Fingerprint kernels for protein structure comparison. Mol Inform 31(6–7):443–452. https://doi.org/10.1002/minf.201100149
9. Gibrat J, Madej T, Bryant S (1996) Surprising similarities in structure comparison. Curr Opin Struct Biol 6(3):377–385
10. Gu J, Bourne P (2009) Structural bioinformatics (methods of biochemical analysis), 2nd edn. Wiley-Blackwell, Hoboken
11. Holm L, Kaariainen S, Rosenstrom P, Schenkel A (2008) Searching protein structure databases with DaliLite v. 3. Bioinformatics 24:2780–2781
12. Holm L, Sander C (1993) Surprising similarities in structure comparison. J Mol Biol 233(1):123–138
13. Hung CL, Lin YL (2013) Implementation of a parallel protein structure alignment service on Cloud. Int J Genomics 439681:1–8

14. Jia Y, Zhang J, Huan J (2011) An efficient graph-mining method for complicated and noisy data with real-world applications. Knowl Inf Syst 28(2):423–447. https://doi.org/10.1007/s10115-010-0376-y
15. Leinweber M, Baumgärtner L, Mernberger M, Fober T, Hüllermeier E, Klebe G, Freisleben B (2012) GPU-based cloud computing for comparing the structure of protein binding sites. In: 2012 6th IEEE international conference on digital ecosystems and technologies (DEST), pp 1–6
16. Leinweber M, Fober T, Freisleben B (2018) GPU-based point cloud superpositioning for structural comparisons of protein binding sites. IEEE/ACM Trans Comput Biol Bioinform PP(99), 1–14
17. Leinweber M, Fober T, Strickert M, Baumgärtner L, Klebe G, Freisleben B, Hüllermeier E (2016) CavSimBase: a database for large scale comparison of protein binding sites. IEEE Trans Knowl Data Eng 28(6):1423–1434
18. Lesk A (2010) Introduction to protein science: architecture, function, and genomics, 2nd edn. Oxford University Press, Oxford
19. Liao VCC, Chen MS (2014) DFSP: a Depth-First SPelling algorithm for sequential pattern mining of biological sequences. Knowl Inf Syst 38(3):623–639. https://doi.org/10.1007/s10115-012-0602-x
20. Madej T, Lanczycki C, Zhang D, Thiessen P, Geer R, Marchler-Bauer A, Bryant S (2014) MMDB and VAST+: tracking structural similarities between macromolecular complexes. Nucleic Acids Res 42(Database issue):D297–303
21. Małysiak-Mrozek B, Stabla M, Mrozek D (2018) Soft and declarative fishing of information in big data lake. IEEE Trans Fuzzy Syst PP(99), 1–1. https://doi.org/10.1109/TFUZZ.2018.2812157
22. Małysiak-Mrozek B, Zur K, Mrozek D (2018) In-memory management system for 3D protein macromolecular structures. Curr Proteom. https://doi.org/10.2174/1570164615666180320151452
23. Małysiak-Mrozek B, Baron T, Mrozek D (2018) Spark-IDPP: High-throughput and scalable prediction of intrinsically disordered protein regions with Spark clusters on the Cloud. Cluster Comput (**accepted**)
24. Marsolo K, Parthasarathy S (2008) On the use of structure and sequence-based features for protein classification and retrieval. Knowl Inf Syst 14(1):59–80. https://doi.org/10.1007/s10115-007-0088-0
25. Mell P, Grance T (2011) The NIST definition of Cloud Computing. Special Publication 800-145. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. Accessed on 7 May 2015
26. Minami S, Sawada K, Chikenji G (2013) MICAN: a protein structure alignment algorithm that can handle multiple-chains, inverse alignments, Ca only models, alternative alignments, and non-sequential alignments. BMC Bioinform 14(24):1–22
27. Momot A, Małysiak-Mrozek B, Kozielski S, Mrozek D, Hera Ł, Górczyńska-Kosiorz S, Momot M (2010) Improving performance of protein structure similarity searching by distributing computations in hierarchical multi-agent system. In: Pan JS, Chen SM, Nguyen NT (eds) Computational collective intelligence. Technologies and applications. Springer, Berlin, pp 320–329
28. Mrozek D (2014) High-performance computational solutions in protein bioinformatics. SpringerBriefs in Computer Science. Springer, Berlin
29. Mrozek D, Brozek M, Małysiak-Mrozek B (2014) Parallel implementation of 3D protein structure similarity searches using a GPU and the CUDA. J Mol Model 20:2067
30. Mrozek D, Daniłowicz P, Małysiak-Mrozek B (2016) HDInsight4PSi: boosting performance of 3D protein structure similarity searching with HDInsight clusters in Microsoft Azure cloud. Inf Sci 349–350:77–101
31. Mrozek D, Gosk P, Małysiak-Mrozek B (2015) Scaling ab initio predictions of 3D protein structures in Microsoft Azure cloud. J Grid Comput 13:561–585
32. Mrozek D, Kutyła T, Małysiak-Mrozek B (2016) Accelerating 3D protein structure similarity searching on Microsoft Azure Cloud with local replicas of macromolecular data. In: Wyrzykowski R (ed) Parallel processing and applied mathematics—PPAM 2015. Lecture Notes in Computer Science, vol 9574. Springer, Heidelberg, pp 1–12
33. Mrozek D, Małysiak-Mrozek B (2013) CASSERT: a two-phase alignment algorithm for matching 3D structures of proteins. In: Kwiecień A, Gaj P, Stera P (eds) Computer networks, communications in computer and information science, vol 370. Springer, Berlin, pp 334–343
34. Mrozek D, Małysiak-Mrozek B, Kłapciński A (2014) Cloud4Psi: cloud computing for 3D protein structure similarity searching. Bioinformatics 30(19):2822–2825
35. Pang B, Zhao N, Becchi M, Korkin D, Shyu CR (2012) Accelerating large-scale protein structure alignments with graphics processing units. BMC Res Notes 5(1):116. https://doi.org/10.1186/1756-0500-5-116
36. Prlić A, Bliven S, Rose P, Bluhm W, Bizon C, Godzik A, Bourne P (2010) Pre-calculated protein structure alignments at the RCSB PDB website. Bioinformatics 26:2983–2985
37. Prlić A, Yates A, Bliven S et al (2012) BioJava: an open-source framework for bioinformatics in 2012. Bioinformatics 28:2693–2695
38. Shapiro J, Brutlag D (2004) FoldMiner and LOCK2: protein structure comparison and motif discovery on the Web. Nucleic Acids Res 32:536–541

39. Shatsky M, Nussinov R, Wolfson HJ (2004) A method for simultaneous alignment of multiple protein structures. Proteins Struct Funct Bioinf 56(1):143–156. https://doi.org/10.1002/prot.10628
40. Shindyalov I, Bourne P (1998) Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. Protein Eng 11(9):739–747
41. Singh S, Chana I (2016) Cloud resource provisioning: survey, status and future research directions. Knowl Inf Syst 49:1–65. https://doi.org/10.1007/s10115-016-0922-3
42. Stivala AD, Stuckey PJ, Wirth AI (2010) Fast and accurate protein substructure searching with simulated annealing and GPUs. BMC Bioinform 11(1):446. https://doi.org/10.1186/1471-2105-11-446
43. The 1000 Genomes Project Consortium (2015) A global reference for human genetic variation. Nature 526:68–74
44. Wei L, Xing P, Tang J, Zou Q (2017) PhosPred-RF: a novel sequence-based predictor for phosphorylation sites using sequential information only. IEEE Trans Nanobiosci 16(4):240–247
45. Wei L, Tang J, Zou Q (2017) Local-DPP: An improved DNA-binding protein prediction method by exploring local evolutionary information. Inf Sci 384:135–144. http://www.sciencedirect.com/science/article/pii/S0020025516304509
46. White T (2012) Hadoop: the definitive guide—storage and analysis at internet scale, 3rd edn. O'Reilly, Sebastopol
47. Ye Y, Godzik A (2003) Flexible structure alignment by chaining aligned fragment pairs allowing twists. Bioinformatics 19(2):246–255
48. Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin MJ, Ghodsi A, Gonzalez J, Shenker S, Stoica I (2016) Apache Spark: a unified engine for big data processing. Commun ACM 59(11):56–65. https://doi.org/10.1145/2934664
49. Zhu J, Weng Z (2005) FAST: a novel protein structure alignment algorithm. Proteins 58:618–627
50. Zou Q, Li XB, Jiang WR, Lin ZY, Li GL, Chen K (2014) Survey of MapReduce frame operation in bioinformatics. Brief Bioinform 15(4):637–647. https://doi.org/10.1093/bib/bbs088

Dariusz Mrozek is currently an Associate Professor and Head of Division of Theory of Informatics in Institute of Informatics at the Silesian University of Technology (SUT) in Gliwice, Poland. He received his PhD degree from SUT in 2006. His research interests cover bioinformatics, information systems, parallel and Cloud computing, databases and Big data. He is now focused on the analysis of protein structures, functions and activities, and the use of novel computation techniques to get insights from biological data, including NGS and proteomics data. He is the author of 90+ papers published in conference proceedings and international journals, author of a book on the use of high-performance computational solutions in protein bioinformatics published by Springer, co-editor of thirteen other books devoted to databases and data processing, and editor of two special issues in reputable scientific journals. Working in different research projects, he cooperated with qualified institutions, e.g., Imperial College of London (on the Chernobyl Tissue Bank) and Microsoft Research in the USA.

Marek Suwała received the MSc degree in Computer Science from the Silesian University of Technology in Gliwice, Poland, in 2013. He currently works for Bank Zachodni WBK in Wrocław, Poland as system analyst. His interests cover business process modeling and Web Services technologies.

Bożena Małysiak-Mrozek received the MSc and PhD degrees, in Computer Science, from the Silesian University of Technology in Gliwice, Poland. She is an assistant professor in the Institute of Informatics at the Silesian University of Technology in Gliwice, Poland, and also a member of the IBM Competence Center. Her scientific interests cover information systems, computational intelligence, bioinformatics, databases, big data, cloud computing, and soft computing methods. She is also a member of the Organizing Committee of the International Conference Beyond Databases, Architectures, and Structures (BDAS) and co-editor of thirteen books devoted to databases and data processing.