Noname manuscript No.
(will be inserted by the editor)

Optimization of Consistency-Based Multiple Sequence Alignment using Big Data technologies

Jordi Lladós · Fernando Cores · Fernando Guirado

Received: date / Accepted: date

Abstract With the advent of new high-throughput next-generation sequencing technologies, the volume of genetic data processed has increased significantly. It is becoming essential for these applications to achieve large-scale alignments with thousands of sequences or even whole genomes. However, all current MSA tools have exhibited scalability issues when the number of sequences increases. The main drawback of these methods is that errors made in early pairwise alignments are propagated to the final result, affecting the accuracy of the global alignment. The use of consistency information enables the final result to be improved and makes it more stable from the accuracy point of view. However, such methods are severely limited by the memory required to store the consistency information. Authors in a previous work analyzed the structure and distribution of the data stored in the constraint library and demonstrated that it could be possible to reduce it without loosing accuracy and thus it is possible to increase the number of sequences to be aligned. However, the execution time for obtaining the constraint library for a bigger number of sequences also increases greatly. In the present paper, the authors apply Big Data technologies to take advantage of the high degree of parallelism provided by the MapReduce paradigm in order to reduce considerably the library calculation time. Moreover, Big Data infrastructure provides a distributed storage system to improve the library scalability and machine-learning algorithms to enhance the consistency selection policies.

Keywords Multiple Sequence Alignment · Consistency · Accuracy · Spark · Big Data · MapReduce

Jordi Lladós
INSPIRES Research Center, Universitat de Lleida. E-mail: jordi.llados@diei.udl.cat

Fernando Cores
INSPIRES Research Center, Universitat de Lleida. E-mail: fcores@diei.udl.cat

Fernando Guirado
INSPIRES Research Center, Universitat de Lleida. E-mail: f.guirado@diei.udl.cat

1 Introduction

Multiple Sequence Alignment (MSA) tools are really important in the analysis of biological sequence data and are a basic step in many bioinformatic analyses, Phylogenetic tree reconstruction ([4]), structure prediction of RNAs and proteins ([11]) or hidden Markov modeling ([3]).

The main idea in the alignment process is to place sequence residues into the same column according to a selected criterion. These criteria can be structural, evolutionary, functional or sequence similarity. However, due to the fact that the best computational match cannot correspond to the best biological meaning, it is well known that the problem leads to NP-hard [21], requiring the utilization of heuristic algorithms.

Among the different MSA approaches, progressive alignment is the most prevalent. Progressive alignment builds up a final MSA by combining pairwise alignments beginning with the most similar pair and progressing, following a guide tree, to the most distantly related. The main drawback of these methods is that errors made in early pairwise alignments are propagated to the final result, thus affecting the accuracy of the global alignment. To lessen the early-error propagation, consistency-based methods were proposed.

Consistency-based methods evaluate each individual pairwise prior to the alignment process to obtain relevant information about the best possible residue match. The data, known as consistency, is stored in a library and used in further steps to minimize the errors propagated in the alignment process. However, storing the consistency library (CL) requires a huge amount of memory. In T-Coffee (TC) [12], the most representative method in this category, the CL size is in the order of $O(N^2L^2)$, N being the number of sequences and L, the residue length of the sequence. This is an important issue that limits the performance and scalability of the consistency based MSA tools.

The authors in a previous work [10] analyzed the distribution and relationship of the constraints stored in the CL in order to determine; (1) which constraints are the most relevant in the alignment process and (2) how these constraints are related to the sequences depending on their position in the tree guide. The conclusions enabled an innovative method for building the CL to be developed. This was named MEL (Memory Efficient consistency Library) and it reduces the amount of memory needed by up to half compared with the initial requirements and, more importantly, without losing accuracy. This new method, allows bigger sequences sets to be processed and expands the viability of use of consistency-based MSA tools. However, increasing the number of sequences also means increasing the execution time needed to obtain the CL, this being a new challenge to tackle.

Thus, the main goal of the present paper is to take advantage of Big Data technologies based on the MapReduce paradigm to reduce the execution time considerably to obtain the CL when applying the library reduction. This new method can be achieved by redesigning the way in which the CL is built, adapting the process to the map and reduce stages. Furthermore, the library

will be stored in the distributed file system (HDFS), allowing an extended capacity and a parallel access for writing and reading the consistency.

The paper is organized as follows: Section 2 presents a brief state of the art of consistency-based MSA tools. In Section 3, we define the consistency library statement. Section 4 presents the development of our proposal. In Section 5 the experimental study is defined and done in order to evaluate the effectiveness of the new method and finally, the main conclusions are set out in Section 6.

2 State of the art

The consistency-based MSA has been shown to be able to increase final alignment accuracy. In [5], O. Gotoh first introduced consistency to identify anchor points to reduce the search space of an MSA. Since then, several MSA tools based on consistency have appeared in the literature.

Do et al. presented ProbCons in [2]. This was a modification of the traditional sum-of-pairs scoring system that incorporates Hidden Markov Models to specify the probability distribution over all alignments between a pair of sequences. Furthermore, Subramanian et al. developed a new tool, DIALIGN-T, in [19], which formulated consistency based on finding ungapped local alignments via segment-to-segment comparisons that determine new weights using consistency. Notredame et al. presented T-Coffee. This improves the alignment accuracy by seeking consistency from a set of global and local pairwise alignments. The scoring function for aligning two sequences or two pre-aligned groups is determined by the whole set of sequences via two processes called library generation and library extension. Although T-Coffee can produce high alignment accuracy, the CL is time and memory consuming when the number of sequences is large. Another method based on consistency, MAFFT, was presented by Katoh et al. in [7]. This uses a new objective function combining the WSP score from Gotoh and the COFFEE-like score ([13]) that evaluates the consistency between a multiple and pairwise alignments.

However, it is known that when the number of sequences to be aligned increases, there is a degradation of accuracy [17]. Other studies focusing on phylogeny estimation from nucleotide datasets have confirmed this hypothesis [9]. This situation can be mitigated by the use of consistency. However, consistency-based methods do not scale well because of the computational resources required to calculate and store the consistency information, which grows quadratically.

The recent use of Big Data technologies has given to bioinformatics researchers an opportunity to achieve scalable, efficient and reliable computing performance on clusters and cloud computing services. The open-source Apache Hadoop project [22], which adopts the MapReduce framework [1] and the distributed file system (HDFS [8]), are able to store and process Petabytes of information efficiently. Moreover, Hadoop has a complete stack of services and frameworks (Spark, Cassandra, etc) that provides a wide range of machine-learning and data-analysis tools to process any kind of workflow.

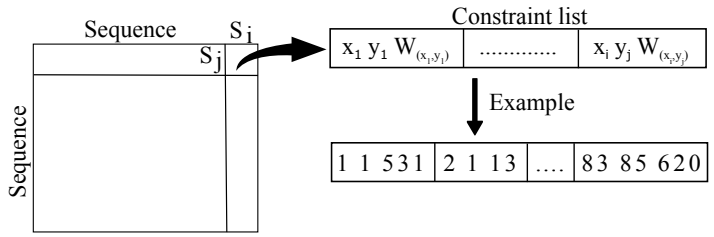

Fig. 1 Library structure.

Thus, applying these technologies in bioinformatics enables the performance and scalability of massive data processing applications to be improved. In [15], a novel approach is proposed that combines the dynamic programming algorithm with the computational parallelism of Hadoop data grids to improve accuracy and also accelerate Multiple Sequence Alignment. In [23], the authors developed a DNA MSA tool based on trie trees to accelerate the centre star MSA strategy. It was implemented using the MapReduce distributed framework.

3 Consistency Library Statement

Consistency-based methods use the point of view that prevention is the best way to avoid the possible errors in the early stages of the alignment when following the guide tree. However, next-generation sequencing applications are unable to take advantage of this improvement due to the bottleneck generated by the need for outsized amounts of memory. This is a huge problem for marker genes, like the ribosomal RNA (rRNA), where millions of sequences are already publicly available, or in biological research that easily produces hundreds of thousands of new sequences that must be aligned [14].

The consistency library (CL) information is a collection of data obtained from computing all-against-all possible pairwise alignments. It can be represented by an NxN matrix (see Figure 1), where each cell $S_i - S_j$ when $i \neq j$ contains a list of residue matches between those sequences. Each residue match is represented by a constraint/entry $\{x, y, W_{(x,y)}\}$, x being a residue of S_i matched with y a residue of S_j and a weight $W_{(x,y)}$ representing its correctness. Each constraint list is used in the progressive alignment stage in order to fill the dynamic programming matrix.

The size of the CL is in the order of $\mathrm{O}(N^2L^2)$, where N^2 is given by all the possible combinations of sequences without repetition and L^2 by the worst scenario in one pairwise (there are no matches between both sequences).

In [10], the authors provide a deep analysis of the CL in order to identify the structure and data that provide an optimal CL. The analysis was carried out based on the BAliBASE benchmark [20], a database of high-quality and manually-refined reference alignments based on 3D structural superpositions.

We study which are the main properties/features that and optimal consistency library must have to obtain a good alignment (the one that maximizes

the accuracy). The study evaluated different CL sizes using a genetic algorithm. It identified the best possible subset of constraints that must be stored in the CL to maximize the accuracy of the final alignment.

Once the subset of constraints was identified, the relationship of the constraints with the residue distribution over the length of the alignment was analyzed. It was observed that the selected constraints covered almost all the domain, being balanced and without repetitions. Next, from the point of view of the weight of the constraints, their distribution showed that the optimal library is composed mainly of the most weighted ones (64%), while the rest of them remain well distributed throughout the library. Finally, the relationship between the stored constraints and the guide tree that determines which sequences and which order that they must be aligned in was evaluated. This is relevant information because, in the end, the guide tree determines the propagation of the errors, and it is necessary to reduce those constraints that only provoke noise. It was observed that the sequences present in the leaves of the guide tree, meaning the first stages of the progressive alignment, contain a major number of constraints, and this number decreases for the sequences that must be processed later.

The main premises extracted from that study conclude the next following requirements that the constraints must meet to obtain an optimal CL:

- The higher the weight, the better.
- The most related leaves of the guide trees must have more constraints.
- The constraints have to cover all the domains of the alignment.

Using this knowledge, the authors designed an innovative method to build the CL, named Memory Efficient consistency Library (MEL). It is capable of deciding whether or not a constraint should be maintained and thus optimizing the amount of required memory.

We tested the accuracy achieved with MEL solving BAliBASE benchmark. The figures in Table 1 are total Sum-of-Pairs (SP) produced using bali score. The first column indicates the number of entries used in the run and the second the percentage reduction. The average score over all families is given in the third column. The results for BAliBASE subgroupings are in columns 4–9.

The results demonstrated that MEL was capable to filter the CL, maintaining only the most relevant constraints. Thus, allowing the amount of required memory to be reduced. Furthermore, the impact of this drop on the accuracy of the final alignment was negligible between 48.73 to 100 percent of consistency reduction ($\sigma = 0.001$).

4 Distributed Memory-Efficient consistency Library (DMEL)

The MEL algorithm was originally developed in the C programming language and was included in the TCoffee MSA tool [1]. As stated above, the new consis-

[1] T-Coffee-MEL sources and its installation instructions can be found at: github.com/jllados/TCoffee-MEL

Table 1 BAliBASE accuracy results with MEL.

Entries	%	*Average*	*RV*11	*RV*12	*RV*20	*RV*30	*RV*40	*RV*50
157M	100.00	**0.746**	0.534	**0.879**	**0.827**	0.718	0.758	**0.759**
134M	84.89	0.745	0.534	0.878	0.826	0.715	0.760	0.755
125M	79.09	0.745	**0.535**	0.877	0.826	0.716	0.760	0.756
114M	72.62	0.744	0.532	0.878	0.826	0.717	0.758	0.755
103M	65.41	0.745	0.528	0.875	0.825	0.717	0.767	0.757
90M	57.42	0.745	0.527	0.875	0.826	**0.723**	0.767	0.753
77M	48.73	0.745	0.525	0.872	0.822	0.721	**0.775**	0.752
62M	39.46	0.740	0.521	0.865	0.817	0.719	0.773	0.747
47M	29.74	0.725	0.511	0.833	0.794	0.715	0.766	0.730
31M	19.86	0.672	0.471	0.734	0.728	0.678	0.731	0.688
16M	9.93	0.512	0.349	0.494	0.553	0.552	0.579	0.546

tency library (CL) require less memory than the original, thus the free space can be used in order to maintain more constraints from a higher number of sequences. However, increasing the number of sequences provokes an important increase in the execution time, due to the fact that calculating the CL is a hugely demanding computational process. Accordingly, taking advantage of this method requires redefining the way the CL is obtained and then reducing the execution time.

Currently, there are new computing technologies commonly applied to Big Data designed to manage huge amounts of data using the MapReduce paradigm. These include such as Hadoop [22] and Spark [16]. Our proposal consists of using these technologies to partition the set of sequences into multiple simple tasks that can be processed in a parallel mode in order to calculate the CL.

The MapReduce is a programming paradigm that allows massive computational resources (processors, memory and disks) to be exploited in a simple and scalable way. This paradigm divides the computation into two stages: map and reduce. In the map stage, the input data is read and arranged as key-value $< K_1, V_1 >$ pairs, to then perform the map function on them, resulting in an output set of $< K_2, V_2 >$ pairs. The mapping function is executed in a distributed way using various mapping tasks. The output is then delivered to the reduce tasks which first shuffle the data by grouping all the values that belong to the same key together. Afterwards, the reduce function compute a single output from the grouped $< K_2, List(V_2) >$ tuples.

Spark is a fast engine for large-scale data processing in real-time executed over Hadoop. Apache Spark is able to reduce the disk overhead generated by MapReduce thanks to passing the results between the different stages through the memory instead of using HDFS files. In addition, Spark provides a more powerful set of transformations that facilitates the programming of applications. From point of view of the architecture, Spark uses a master/slave approach. It has one central coordinator (*Driver*) that communicates with many distributed workers (*Executors*). The driver is the process where the main method runs and the executors are those that process the data received.

In the present paper, the MEL method was implemented using Apache Spark. This new version has been named DMEL, which that stands for *Distributed-MEL*. The main programming language was Python and some critical parts were developed in C, using the Ctypes extension, which provides the ability to call up external shared libraries.

Algorithm 1 presents the DMEL method. The map stage is responsible for defining all the tasks in charge of computing the constraints for a set of pairs of sequences. The driver generates these tasks for all the $N * (N-1)/2$ pair combinations (line 1) and distributes them in a balanced way among all the Map tasks using a Resilient Distributed Dataset (RDD) (line 2). Then, in line 3, the map tasks are launched and scheduled for processing on the executors. As a result, each map generates a portion of the library in parallel, and this persists in the HDFS file system.

The executor, lines 4-23, performs a subset of the pairwise combinations. This is done in the double-nested loop in lines 4-5, which obtains the different combinations of sequences assigned to the *task*. It calculates the library for each of these combinations by calling the *pair_wise(S_i, S_j)* function of the shared library (PPCAS.so). This function calculates the consistency of these two sequences and pushes them into a temporary queue structure, in which all the library constraints are stored sorted by their weight, lines (8-10). After this, each constraint is evaluated in order to determine if it must be stored in the final list. The evaluation code, line 18, has to check two parameters, the first being the maximum bound previously generated. With this value, the method prunes the number of residues matching the same region of the alignment. The second parameter indicates whether or not a new entry has enough memory to be allocated. Also, if the evaluated pairwise (PA_{ij}) is a leaf node, the remainder from the division in the line 13 is added to the maximum number of constraints. As the queue is sorted by weight, the higher ones will be the first to be evaluated, thus ensuring that they have more chances of surviving than the lower ones. Finally, it writes this portion of the library to the disk (HDFS).

5 Experimentation

The experimental study is focused on three main parts; (1) defining the best granularity for the map-tasks, (2) demonstrating the scalability of the method by increasing the number of sequences and finally (3) studying the accuracy obtained when the CL is reduced.

In the previous work, the experimental study was done by using the Balibase benchmark. However, this benchmark does not provide a large enough number of sequences. Accordingly, we have used the HomFam [18] benchmarking suite, which provides large datasets using Pfam families that contain thousands of sequences. In order to validate the results of aligning a Pfam family, the Homstrad site includes some reference alignments and the corresponding Pfam family. These references are previously de-aligned and shuffled into the

Driver

tasks_list = generate_tasks();
rdd_tasks = sc.parallelize(tasks_list, len(tasks_list));
rdd_tasks.map(executor_function).saveAsTextFile(hdfs_path);

Executor

for each sequence $S_i \in task_i$ **do**
  for each sequence $S_j \in task_j$ **do**
    _libraryC = ctypes.CDLL("./PPCAS.so")
    PA_{ij}=_libraryC.pair_wise(S_i, S_j)
    for each residue $x \in S_i, y \in S_j|$ are aligned in $PA_{i,j}$ **do**
      $W_{(x,y)} = \frac{\sum OCCURRENCE(PA_{i,j})}{RESIDUES(PA_{i,j})}$;
      Q_Push_Sorted($x, y, W_{(x,y)}$));
    end for
    MAX_Constraints_$PA_{ij} = \frac{MAX_Library}{C_{N,2}}$;
    Bound= $\frac{MAX\_Constraints\_PA_{ij}}{MAX\_Length(S_i, S_j)}$;
    if isleaf(S_i, S_j) **then**
      MAX_Constraints_PA_{ij}+=remainder;
    end if
    for each constraint $x_i, y_j, W_{(xi,yi)} \in$ Q **do**
      if balanced($x_i, y_j, Bound$)| not full MAX_Constraints_PA_{ij} **then**
        $L(S_i^x, S_j^y) = W_{(S_i^x, S^y_j)}$;
      end if
    end for
  end for
end for

Algorithm 1: Distributed Memory-Efficient consistency Library (DMEL) construction

dataset. After the alignment process, the reference sequences are extracted and compared with the originals in Homstrad. The correlation between the obtained alignment and the references determines the accuracy of the alignment.

HomFam contains almost one hundred sets. We randomly selected five of the biggest families to evaluate the method (Acetyltransf, DEATH, PDZ, GEÑ and phc). The results for each experiment correspond to five iterations in order to show their robustness.

The execution environment is a distributed memory cluster made up of 20 nodes, each characterized by an Intel Core 2 Quad at 2.4GHz with 8GB DDR2 of RAM.

5.1 Study of map-task granularity

In this subsection, we evaluate the impact on the execution time of the number of tasks for each executor. The total number of tasks to assign is given by Equation 1, N being the number of sequences which represents all the pairwise

that must be evaluated to obtain the CL. Equation 1 divided by the number of partitions gives the number of tasks for each executor.

$$pair\_combinations = \frac{N * (N-1)}{2} \tag{1}$$

Table 2 MapReduce Tasks granularity - Execution time

	N° of Sequences.			
$N^o of Partitions$	100	200	500	1000
100	34.44	39.35	70.34	181.50
250	**33.79**	38.50	61.43	146.59
500	35.89	37.83	59.82	134.58
750	34.89	38.02	58.47	130.78
1000	34.94	**37.41**	58.26	129.47
1250	35.09	37.94	58.12	128.70
1500	35.88	38.36	58.09	128.88
1750	35.49	38.62	**57.82**	128.87
2000	35.91	37.87	58.44	**128.50**

Various numbers of partitions to determine their impact on performance are evaluated in Table 2. It can be observed that when the number of partitions increases, DMEL requires less time to obtain the CL. However, when the number of tasks becomes greater than the number of sequences, the execution time remains similar. This is because the executors saturate with a coarser granularity. Thus, the selected approach to define the partitions is based on the number of sequences. Doubling this value seems to give a good scenario for each test case. Finally, the number of partitions is defined by Equation 2, giving a granularity represented by Equation 3 for each executor.

$$n\_partitions = 2 * N \tag{2}$$

$$executor\_tasks = \frac{(N-1)}{4} \tag{3}$$

It is important to take into account that DMEL is only in charge of generating the CL and is not capable of aligning the sequences. So, in the next subsections we use the T-Coffee consistency-based MSA Tool, one of the most representative methods in this category, to obtain the final alignment.

5.2 DMEL scalability

As stated in the paper in which MEL was presented [10], the best compromise between accuracy and execution time is obtained when the applied reduction is approximately 50% of the CL. That is why in the present experimental study we set this value for the corresponding reduction in the CL.

Figure 2 shows the execution time needed for DMEL when increasing the number of sequences. The results have been compared with the original MEL

and T-Coffee (only the CL generation process). From the point of view of the number of sequences, DMEL and MEL are able to deal with a bigger number than T-Coffee. This is due to the lower amount of memory required to store the CL. T-Coffee reaches its memory limit with 500 sequences, while MEL is able to process up to 1000 and DMEL can generate thousands of them thanks to the fact that it is distributed among the nodes.

When the execution time is evaluated, MEL is shown to be the slowest method. It requires more evaluations than T-Coffee to determine if a constraint must be stored. As can be observed, DMEL is the fastest method. Although it does the same evaluations as MEL, it takes advantage of the Big Data infrastructure to calculate the CL in a parallel mode.

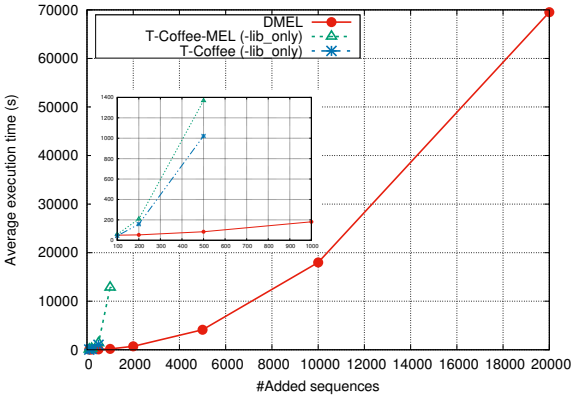

Fig. 2 Comparison of Library calculation time of DMEL and MEL using HomFam sets.

Figure 3 depicts the real benefits of using a Big-Data infrastructure with 500 sequences. The left axis shows the execution time, and the right one, the speedup obtained. It takes 1,005 seconds with a single node, while this can be reduced to 83 seconds when using 20 nodes. This represents an 16.3x speedup over the TC-MEL with the -lib_only flag (1,367 seconds). Between 1-10 nodes, DMEL is able to achieve super-linear speedups due to the library calculation on Spark is more efficient than the original version while adding more nodes is overkill for 500 sequences (it does not have enough parallel work) and the speedup grows less sharply.

5.3 Study of DMEL accuracy

When increasing the number of sequences, the accuracy of the final alignment can be easily degraded. For this reason, this parameter is the challenge for validating the viability of any MSA tool. Thus, in this subsection we analyze the results from the point of view of this parameter.

Figure 4 shows the accuracy results obtained for different sizes of the sequence sets. As MEL and DMEL apply the same algorithm to determine the

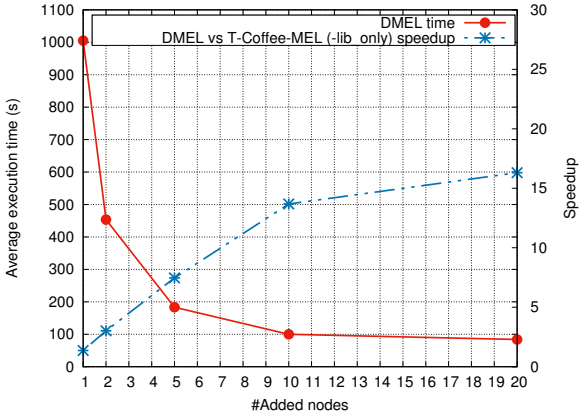

Fig. 3 Scalability of DMEL with HomFam sets

constraints to be stored in this experimental study only DMEL is evaluated. It can be seen that all methods have similar results despite the CL being reduced by up to 50%. The most relevant aspect is the fact that it is possible to increase the number of sequences further the number in the original T-Coffee but still maintain the accuracy. In this case, the limiting factor of the number of sequences to align is given by the main memory of a single node, because T-Coffee needs all the CL loaded at the same time. T-Coffee+DMEL has the same limitation because the amount of memory is the same, although it is possible achieve twice the number of sequences.

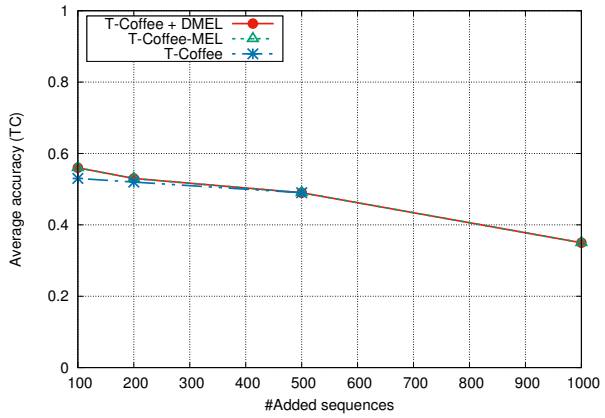

Fig. 4 Comparison of the accuracy of DMEL and T-Coffee using HomFam sets.

Finally, we evaluated the execution time needed to produce the alignment shown in Figure 5. In this experimentation we have included T-Coffee using both MEL and DMEL methods. As can be observed, T-Coffee obtained the

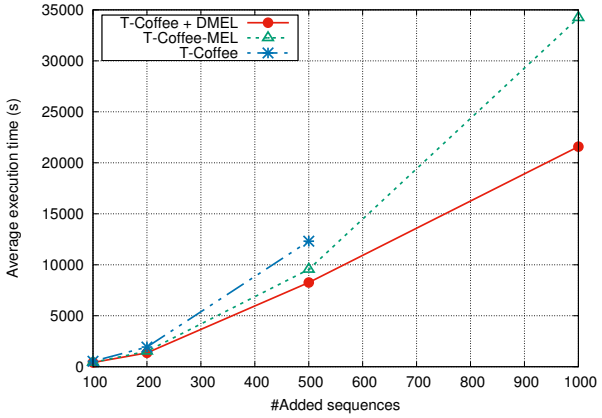

Fig. 5 Comparison of the execution time of DMEL, MEL and T-Coffee using HomFam sets.

worst execution times up to 500 sequences. MEL and DMEL treat a reduced library and that is why there is a lower number of accesses to the CL and it leads to a shorter execution time. Above 500 sequences, using the reduced CL, obtained from MEL and DMEL, T-Coffee can produce the final alignment. In this situation DMEL obtained the shortest execution times, because the calculation of the CL was done in parallel.

6 Conclusions

In this paper, the authors present an efficient Consistency Library for Multiple Sequence Alignment Tools designed over Apache Spark. The approach is applied during the process of building the consistency library in a Big Data infrastructure. Its goal is to maintain the best consistency information while reducing the execution time and the size of the memory that the library may use without affecting the accuracy.

We prove that DMEL is able to reduce the execution time and maintain the accuracy of an aligner with consistency. This is due the fact that; 1) it uses an engine for big data processing and 2) it reduces the number of constraints maintaining them at a certain number. Moreover, it produces a great benefit in scalability, because it enables the aligner to align more sequences than would be possible when using the complete library.

One of the most interesting characteristic of DMEL is the fact that the consistency library calculation is distributed among the Big Data infrastructure. By this reason, we will design a new method able to take profit of this parallel distribution, avoiding the necessity of storing locally the whole consistency library, meaning that the local memory restriction disappears.

Acknowledgements This work has been supported by the MEyC-Spain under contract TIN2014-53234-C2-2-R, TIN2017-84553-C2-2-R and TIN2016-81840-REDT.

References

1. Dean, J., Ghemawat, S.: MapReduce: A Flexible Data Processing Tool. *Communications of the ACM* 53(1), 72-77 (2010).
2. Do C, Brudno M and Batzoglou S, PROBCONS: Probabilistic Consistency-based Multiple Alignment of Amino Acid Sequences, Proceedings Nineteenth National Conference on Artificial Intelligence, pages 703-708 (2004)
3. Eddy SR, A new generation of homology search tools based on probabilistic inference, Genome Inform, volume 23, pages 205-211 (2009)
4. Gouy M, Guindon S and Gascuel O, Seaview version 4: A multiplatform graphical user interface for sequence alignment and phylogenetic tree building, Molecular Biology and Evolution, 27(2), 221-224 (2010)
5. Gotoh O, Consistency of optimal sequence alignments, Bulletin of Mathematical Biology, 52(4), 509-525 (1990)
6. Just W, Computational complexity of multiple sequence alignment with sp-score, Journal of computational biology, 8(6), 615-623 (2001)
7. Katoh K, Misawa K, Kuma K and Miyata T, Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform, Nucleic Acids Research, 30(14), 3059-3066 (2002)
8. Karun A.K. and Chitharanjan K., A review on hadoop - HDFS infrastructure extensions, IEEE Conference on Information & Communication Technologies, 132-137 (2013).
9. Liu K, Linder CR and Warnow T, Multiple sequence alignment: a major challenge to large-scale phylogenetics, PLoS currents, 2, (2010)
10. Lladós J, Cores F and Guirado F, Efficient Consistency Library for Multiple Sequence Alignment Tools, International Conference Computational and Mathematical Methods in Science and Engineering, 4, 1269-1280 (2017)
11. Marks, D.S., Hopf, T.A. and Sander C, Protein structure prediction from sequence variation, Nat Biotech, 30(11), 1072-1080 (2012)
12. Notredame C, Higgins DG and Heringa J, T-coffee: A novel method for fast and accurate multiple sequence alignment, Journal of molecular biology, 302(1), 205-217 (2000).
13. Notredame C, Holm L and Higgins DG, Coffee: an objective function for multiple sequence alignments, Bioinformatics, 14(5), 407-422 (1998)
14. Pruesse E, Peplies J and Glöckner FO, SINA: accurate high throughput multiple sequence alignment of ribosomal RNA genes, Bioinformatics, 28(14), 1823-1829 (2012)
15. Sadasivam, G., Baktavatchalam, G.: A novel approach to Multiple Sequence Alignment using hadoop data grids. *International Journal of Bioinformatics Research and Applications* 6(5), 472-483 (2010).
16. Sakr S, Big Data Processing Stacks, IT Professional, 19(1), 34-41 (2017)
17. Sievers F, Dineen D, Wilm A and Higgins DG, Making automated multiple alignments of very large numbers of protein sequences, Bioinformatics, 29(8), 989-995 (2013)
18. Sievers F, Dineen D, Wilm A, Higgins DG, Making automated multiple alignments of very large numbers of protein sequences, Bioinformatics, 29(8), 989-995 (2013)
19. Subramanian AR, Weyer-Menkhoff J, Kaufmann M, et al, Dialign-T: an improved algorithm for segment-based multiple sequence alignment, BMC Bioinformatics, 66(6), (2005)
20. Thompson J.D., Plewniak F. and Poch O, Balibase: a benchmark alignment database for the evaluation of multiple alignment programs, Bioinformatics, 15(1), 87-88 (1999)
21. Wang L. and Jiang T, On the complexity of multiple sequence alignment, Journal of Computational Biology, 1(4), 337-348 (1994)
22. Zhang Y, Cao T, Li S, Tian X, Yuan L, Jia H, Vasilakos, AV, Parallel Processing Systems for Big Data: A Survey, Proceedings of the IEEE, 104(11), 2114-2136 (2016)
23. Zou, Q., Hu, Q., Guo, M., Wang, G.: HAlign: Fast multiple similar DNA/RNA sequence alignment based on the centre star strategy. *Bioinformatics* 31(15), 2475–2481 (2015).