J Supercomput (2017) 73:1467–1483
DOI 10.1007/s11227-016-1835-3

An effective extension of the applicability of alignment-free biological sequence comparison algorithms with Hadoop

Giuseppe Cattaneo[1] · **Umberto Ferraro Petrillo**[2] · **Raffaele Giancarlo**[3] · **Gianluca Roscigno**[1]

Published online: 8 August 2016

Abstract Alignment-free methods are one of the mainstays of biological sequence comparison, i.e., the assessment of how similar two biological sequences are to each other, a fundamental and routine task in computational biology and bioinformatics. They have gained popularity since, even on standard desktop machines, they are faster than methods based on alignments. However, with the advent of Next-Generation Sequencing Technologies, datasets whose size, i.e., number of sequences and their total length, is a challenge to the execution of alignment-free methods on those standard machines are quite common. Here, we propose the first paradigm for the computation of k-mer-based alignment-free methods for Apache Hadoop that extends the problem sizes that can be processed with respect to a standard sequential machine while also

A paper related to this work was presented at the *8th International Workshop on Parallel Programming Models and Systems Software for High-End Computing* [6].

✉ Umberto Ferraro Petrillo
umberto.ferraro@uniroma1.it

Giuseppe Cattaneo
cattaneo@unisa.it

Raffaele Giancarlo
raffaele.giancarlo@unipa.it

Gianluca Roscigno
giroscigno@unisa.it

1 Dipartimento di Informatica, Università degli Studi di Salerno, Via Giovanni Paolo II, 132, 84084 Fisciano, SA, Italy

2 Dipartimento di Scienze Statistiche, Università di Roma "Sapienza", P.le Aldo Moro 5, 00185 Rome, Italy

3 Dipartimento di Matematica ed Informatica, Università degli Studi di Palermo, Via Archirafi, 34, 90123 Palermo, PA, Italy

granting a good time performance. Technically, as opposed to a standard Hadoop implementation, its effectiveness is achieved thanks to the incremental management of a persistent hash table during the map phase, a task not contemplated by the basic Hadoop functions and that can be useful also in other contexts.

Keywords Alignment-free sequence comparison and analysis · Distributed computing · MapReduce · Hadoop

1 Introduction

High-performance computing (HPC, for short) in Bioinformatics, and more in general in the Life Sciences, has a long history of successes, although it has been classically associated with "Big Science Bioinformatics" tasks such as protein structure prediction, e.g., [1], or genome assembly, e.g., [30]. However, the growing amount of data that the new sequencing technologies provide poses unprecedented computational challenges that even the most basic and routine bioinformatics applications become amenable for HPC implementations. Nowadays, it is clear that most scientific data analyses can benefit from parallel frameworks to achieve speedup and scalability [35]. Following that trend, the *MapReduce* paradigm and its reference implementation Apache Hadoop are becoming a standard for solving bioinformatics problems using a distributed approach (see, e.g., [12,36]). Particularly relevant for this paper is the assessment of how similar to each other biological sequences in a set are. Such an information may then be used for various further investigations, e.g., phylogenetic studies. For brevity, we refer to this area with the classic term of *Sequence Comparison*: it is a central part of Sequence analysis [11,21].

1.1 Sequence comparison and HPC

Sequence comparison methods can be broadly divided into two main branches. The first, which can be considered to be the *Holy Grail* of Sequence analysis [21], consists of methods that assess the similarity among sequences via alignments. Further information can be found in [2]. Unfortunately, most of the alignment methods require more and more significant computation time, due to their intrinsic time complexity that compounds with the growing quantity of sequence data they have to process in each run. To address, at least in part, such a drawback, a second branch of *Sequence Comparison* methods has emerged, referred to as Alignment free. Although fairly recent [39] with respect to alignment algorithms, it has grown very rapidly [38], becoming very quickly populated with methods that are particularly appealing because their running time is proportional to the length of the input sequences, even if they are usually less accurate than traditional alignment-based approaches. Moreover, they have been proven to be effective and significant for biological investigations [7,15]. Further information can be found in [38].

Relevant for HPC is the following state of the art. Due to their routine use in *Sequence Analysis*, in particular for large database searches, a considerable effort to parallelize the reference alignment methods has taken place, with some degree of suc-

cess. Here, we limit ourselves to mention [20,27,28,40] and references therein. On the other hand, the design of alignment-free methods for HPC has not received analogous attention: to the best of our knowledge, only one effort is present in the literature [19] and it deals almost exclusively with implementations on a GRID infrastructure, leaving open performance assessments.

To appreciate our work, an important distinction has to be made between alignment-free methods and existing methods that deal with the collection of k-mer statistics in biological sequences, i.e., how may times each sequence of length k appears in a longer sequence. Due to its fundamental nature, many algorithms computing those statistics have been developed, supported by various architectures. We limit ourselves to mention the latest of them [3,10,29]. Although many alignment-free methods are based on k-mer statistics, it is not clear that the available algorithms collecting those statistics in an HPC framework can be profitably used as base to develop HPC alignment-free methods. For that matter, it is not even clear that those methods take full advantage of the computer architecture supporting them. Those aspects, certainly related to this research, deserve to be carefully investigated on their own and, in fact, this research group has work in progress in this area. For the sake of completeness, the problem of performing read mapping in a distributed setting using the *MapReduce* paradigm, a topic somewhat related to the one of this paper, is considered in [32].

1.2 A synopsis of our contributions and organization of the paper

Typically, an alignment-free method takes as input a collection of sequences and returns, as output, a matrix reporting, for each pair, a numerical score assessing how similar these are. We refer to the size of the input collection as problem size. It has two parameters: the number of sequences and their total length. Given the growing amount of sequence data that is produced nowadays in the life sciences and, in particular, for meta-genomic and population-genomic studies [22,24], it is important both to (a) extend the range of problem sizes to which alignment-free methods can be successfully applied and (b) reduce the time required for their computation. Those are the goals of this paper.

From the many available [38,39], we have chosen a representative sample of alignment-free methods: the ones that are based on collecting subword statistics in sequences, e.g., the number of occurrences of substring of length k which are referred to as k-mers, to assess sequence similarity. They are reported in Sect. 2.1.

We have tested those methods extensively in a sequential setting to assess their limits in terms of problem sizes that can be handled on a conventional sequential machine. With use of a specific hardware configuration, we obtain a benchmark for the quantitative evaluation of how well the distributed solution proposed here meets goal (a). Since the memory size available on a sequential machine is the key resource constraint limiting the execution of those methods, our experiments are performed with varying memory sizes. The corresponding results are provided in Sect. 4. A performance bottleneck is identified, due to the considerable amount of memory required by the data structures those algorithms use to maintain the k-mer occurrence counts, i.e., mainly a hash table. This suggest that, to extend the problem sizes that can be handled by those methods, it is convenient to resort to a distributed approach.

At a technical level, we propose a *MapReduce* paradigm, based on Hadoop, for the computation of k-mer-based alignment-free methods. It is reported in Sect. 3. One of its features is its ability to increase the span of the problems that can be solved [goal (a)] by virtually spreading over several nodes of a Hadoop cluster the core data structure used by these algorithms, i.e., the already mentioned hash table. Essential for its efficiency is the proper management of this data structure. Indeed, although one could seamlessly delegate its management to the underlying Hadoop framework, the performance of the resulting solution has turned out to be disappointing. Therefore, we propose a new solution where this data structure is managed explicitly by our code. This technique may be of use also in other bioinformatics contexts (see again [22,24]). Moreover, the proposed distributed solution is able to speed up the processing of sequences by splitting them into pieces that can be processed in parallel [goal (b)]. Section 3 provides details about this Hadoop implementation.

In addition to the technical contribution just mentioned, the experiments reported in Sect. 4 shows that the Hadoop solution proposed here is quite effective in extending the state of the art with respect to goals (a) and (b). A quantification, being quite dependent on the hardware that has been used for the experiments, is provided in Sect. 4.

2 Background

To make the presentation in this paper self-contained, we provide some background and references regarding alignment-free sequence comparison and the *MapReduce* paradigm and Hadoop.

2.1 A selection of alignment-free sequence comparison methods

One of the main goals of Biology and, more in general, the Life Sciences in the study of biological sequences is to assess either homology or function or both. The first consists of establishing the evolution of a biological sequence, which can be an entire genome or even a single gene. The second consists of discovering what is the function of a biological sequence, usually newly discovered.

Both homology and function are nearly impossible to formalize in mathematical terms since they are inherently related to the evolution of living species, which is a process that can be described only in part by Mathematics. Yet, it has been observed and validated experimentally that a mathematical estimate of "similarity" among a set of biological sequences gives in most cases good indications about common ancestry and function [21].

An example may be of help in illustrating the impact of similarity functions on biological research. Assume one is given a set of species for which one is interested in knowing their common ancestry, i.e., an evolutionary taxonomy, which is usually represented via a phylogenetic tree. Usually a reliable taxonomy requires many years of investigation and deep biological knowledge. Figure 1a provides a taxonomy of 15 species that has been obtained solely with the use of biological knowledge, with very little computational work. It would be certainly of great benefit to the biologists to start from a working hypothesis for their classification. Here clustering, in particular

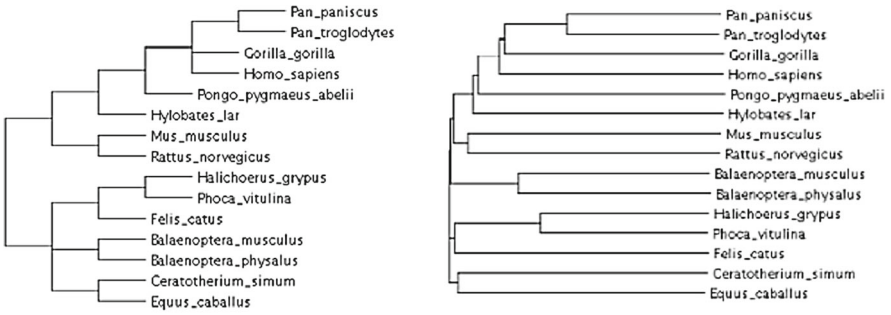

Fig. 1 **a** The taxonomy of 15 species obtained by the National Center for Biotechnology Information (NCBI); **b** Hierarchical Clustering of the same 15 species as in (**a**). It is based on a quantification of the similarity between each pair of mitochondrial genomes of the species listed at the leaves of the tree. Its construction took only a few seconds on a conventional computer. The sequence similarity function is based on Kolmogorov complexity and data compression [15]

Hierarchical, can be of great help, in particular with the use as distance function of the similarity between pairs of genomes. Figure 1b provides an example of Hierarchical Clustering with the same 15 species as in (a). The two trees are remarkably close; therefore, the tree built with computational techniques is a good starting point for a more refined taxonomy.

As already mentioned in the Introduction, alignment-free methods are now a standard for sequence comparison. Among the many available, [38,39], we have chosen a representative sample of alignment-free methods: the ones that are based on subword statistics in sequences to assess sequence similarity. The first three methods that we present use exact k-mer occurrence counts while the remaining one uses approximate k-mer occurrence counts. Mathematically, they are all distance or dissimilarity measures (see [17] for definitions). Moreover, with the use of a hash table, it is simple to obtain algorithms computing them in linear time. The details are left to the reader.

Consider an alphabet Σ of n symbols and an integer $k \geq 1$. Let Σ^k be the corresponding set of k-mers. It is convenient to associate to each k-mer x its rank in the lexicographic order of k-mers.

Squared Euclidean Dissimilarity Measure [39,43]. It is defined as:

$$d_{SE}(S, Q) = \sum_{i=1}^{n^k} (s_i - q_i)^2 \tag{1}$$

where S and Q are two sequences, and s_i and q_i are the number of occurrences of the ith k-mer in S and in Q, respectively.

D_2 *Score*[37]. It is defined as:

$$D_2(S, Q) = \sum_{i=1}^{n^k} s_i \times q_i \tag{2}$$

For completeness, we mention that notable variants of D_2 are D_2^S and D_2^*, defined in [7] and [37], respectively. They will not be considered in this study since computationally they are as demanding as D_2.

Feature Frequency Profile (FFP) [34]. Let S_k and Q_k be the empirical probability distribution vectors of the k-mers in S and Q, respectively. FFP is the Jensen–Shannon divergence [42] between them; that is,

$$JS_k(S_k, Q_k) = \frac{1}{2}KL(S_k, M_k) + \frac{1}{2}KL(Q_k, M_k) \tag{3}$$

where $M_k = (S_k + Q_k)/2$ and KL is the Kullback–Leibler divergence [25].

Spaced-word frequencies [4,23]. A spaced-word over an alphabet Σ is a word composed of symbols from Σ and wild-card symbols. This approach computes the relative frequencies of spaced words with respect to a single fixed *spaced pattern* P of match and don't care positions. Then, it computes the distance between two sequences by combining their spaced-word frequencies using a proper distance function measure such as Squared Euclidean dissimilarity measure.

2.2 *MapReduce* and Hadoop

A computing paradigm that has become very popular is *MapReduce* [9], where a generic computation takes a set of input ⟨ key, value ⟩ pairs, and produces a set of output ⟨key, value⟩ pairs. The computations to carry on these pairs are expressed through the definition of two functions: *map* and *reduce*. The map function takes an input pair and produces a set of intermediate ⟨key, value⟩ pairs. All the intermediate values having the same *key* are grouped and passed to the reduce function. This function accepts a *key* and a set of values for that *key*. It aggregates those values to form another set of pairs (possibly, a smaller set; in general just one output pair is produced per each reduce function). Map and reduce functions are executed, as tasks, on the nodes of a distributed system.

Here, we make use of Apache Hadoop, a java-based open source distributed computing environment that is currently the most popular framework supporting *MapReduce*. It allows for reliable, scalable and distributed computing. The newer version is mainly composed of two components: a data processing framework called *Yet Another Resource Negotiator* and the *Hadoop Distributed File System* (HDFS) [33]. The data processing framework organizes a computation as a sequence of user-defined *MapReduce* operations on datasets of ⟨key, value⟩ pairs. These operations are executed, as tasks, across the nodes of a cluster. Each node may exploit core level parallelism by running several tasks at the same time by means of Hadoop containers. The HDFS is a distributed and block-structured file system optimized to run on commodity hardware and able to provide fault tolerance through replication of data. A v2.x Hadoop simple cluster consists of a single *master node* and multiple *slave nodes*.

3 Hadoop alignment-free sequence comparison

For the methods described in Sect. 3.1, the basic paradigm adopted here for their computation is an extension of the algorithm proposed by Elsayed et al. [13] to assess similarity in large document collections. We refer to it as a paradigm since it is described independently of the dissimilarity measure that is actually used to compare sequences. It requires the execution of two *MapReduce* jobs. The first one, described in Sect. 3.1.1, extracts all the k-mers, either exact or approximate, in a sequence (or part of it). The second one, described in Sect. 3.1.2, uses that information to compute a dissimilarity measure provided in input.

Since the performance of the basic paradigm turns out to be disappointing (see Sect. 4), an improved version of it is proposed here and it is described in Sect. 3.2.

3.1 The basic paradigm and its straightforward Hadoop implementation

3.1.1 Indexing

This job is used to extract, for each input sequence, the k-mers that occur in that sequence and that are later used to compute the dissimilarity between sequences. For completeness, we mention that this is the same strategy used by BioPig [31].

Mapper The map function takes as an input a pair $\langle idSeq,\ S\rangle$, where *idSeq* is a unique identifier for the input sequence S. Then, for each k-mer x present in S, it outputs the pair $\langle x,\ (idSeq, 1)\rangle$. The extraction of the k-mers from S is performed incrementally scanning it from left to right via shift operations (details left to the reader). At the end of that task, finally, each map function communicates to the reducers the length of the sequence via the pair $\langle idSeq,\ |S|\rangle$.

Reducer Based on the $\langle kmer,\ (idSeq, 1)\rangle$ values produced by the map function, the reduce function receives by the Hadoop framework a set of pairs $\langle x,\ L\rangle$, where L is the list of $(idSeq, 1)$ pairs corresponding to that particular k-mer. Input pairs are aggregated so as to produce as output the number of times each k-mer x appears in every input sequence. That is encoded as a record $\langle x,\ L'\rangle$, where L' is the mentioned statistics. Each record is saved in a distinct Hadoop `SequenceFile` to be processed in the second step. Moreover, each reduce task returns the size of each sequence it processed using the HDFS cache mechanism.

3.1.2 Dissimilarity measurement

Given as input one of the measures defined in Sect. 2.1, identified by means of a set of standard labels, and the output of job 1, this job is used to evaluate the pairwise dissimilarity for each pair of input sequences.

Mapper Given a pair $\langle x,\ L'\rangle$, the map function computes the partial dissimilarity for each distinct pair of sequences in L' according to the input-provided measure identified

by the label D. As output, the map function emits a $\langle (idSeq_A, idSeq_B, D),\ pdiss\rangle$ pair, where $idSeq_A$ and $idSeq_B$ are the identifiers of two input sequences, while $pdiss$ is the partial dissimilarity, that is, a partial evaluation of the measure D (as limited to only the statistics available to the map function).

Reducer Based on the $\langle (idSeq_A, idSeq_B, D),\ pdiss\rangle$ values produced by the map function, the reduce function receives by the Hadoop framework a set of pairs $\langle (idSeq_A, idSeq_B, D),\ list\{pdiss'\}\rangle$, where list$\{pdiss'\}$ is the list of all the partial dissimilarities among the sequences with identifiers $idSeq_A$ and $idSeq_B$, with respect to the measure D. As output of the computation, the reduce function returns the pair $\langle (idSeq_A, idSeq_B, D),\ diss\rangle$, where $diss$ is the final value of D with respect to these two sequences.

3.2 A more effective Hadoop implementation of the basic paradigm

Our first implementation of the paradigm presented in Sect. 3.1 makes a straightforward use of the facilities provided by the Hadoop framework. A careful profiling of it allowed us to identify some of its performance bottlenecks (data not shown and available upon request). The first is that the execution of the map tasks run during the Indexing job is significantly slowed down by the number of output pairs so produced (one for each k-mer found). The second is related to the way the standard Hadoop input strategies can be used for processing very long sequences. The programmer is given two options: (a) to process a sequence by having different map tasks process different lines of a same sequence; (b) to have a single map task completely load a sequence into memory before indexing it. The first option does not allow to index k-mers spanning two or more lines, while the second does not work well or does not work at all with very long sequences, that is, letting one single map task (instead of many) process a long sequence prevents the possibility of exploiting the implicit parallelism of Hadoop. Moreover, very long sequences may be big enough to exceed the memory of a map task, thus causing its failure. We present the two following optimizations to alleviate these problems.

1. *Incremental in-mapper combining*
   The map tasks in the Indexing job of our paradigm now take advantage of a hash table data structure. It is used to both index and sum the frequencies of the k-mers counts while scanning an input sequence.
2. *Input split strategies*
   We propose an input strategy that can be used by map tasks to manage very long multi-line sequences during the Indexing job of our paradigm. It works by splitting an input sequence S, encoded as a FASTA file, in several records $\langle id_S,\ (S_i, S_{i+1})\rangle$, where id_S is an identifier for S, S_i is the ith row of S and S_{i+1} contains the first $k-1$ characters of the $(i+1)$th row of S (S_{i+1} is empty if i is the last row of S).

With respect to the straightforward implementation, the first optimization has two important advantages. The first is that the execution of a map task is not slowed down by the frequent outputs, as there will be only one single bulk output phase at the end of the task, including all the output pairs. The second is that by aggregating multiple

occurrences of the same k-mer, the number of output pairs of a map task is greatly reduced.

Such an optimization is also important methodologically. Indeed, one could use the Hadoop built-in `Combiner` module, since it allows a map task to buffer all of its output pairs and to summarize them, through the execution of a user-defined combiner function. However this solution has a major drawback with respect to our solution: the aggregation is not incremental but, generally, takes place at the end of the task. This implies that the map task has to keep in memory all of its output pairs during its execution. As a consequence, it would likely run out of memory when processing long sequences. To avoid this problem, we explicitly implemented this optimization by redefining the methods used by Hadoop to control the initialization and the finalization of the map task. It is interesting to note that in this bioinformatics context, as well as the context of image processing [5], there is need of explicitly managing and updating a persistent data structure (here, a hash table; a matrix, in [5]) during the execution of a map task. So, the solution we propose here, together with the one in [5], provides tools for incremental in-mapper combining.

The second optimization represents a more efficient alternative to the standard Hadoop input strategies available when processing very long sequences, as it is able to manage sequences of arbitrary length while exploiting the implicit parallelism available with Hadoop.

4 Experiments

4.1 Experimental setup

4.1.1 Datasets

It is somewhat unfortunate that there are no commonly accepted or standard benchmark datasets available in the literature for the scenario that now has become a standard for sequence analysis, due to the pervasiveness of high-throughput sequencing technologies: very large datasets that are composed of many relatively short genomes (e.g., megabytes in length). Indeed, methods that have been presented in the Literature hardly use the same datasets for their experimentation, e.g., [15,26].

Given the above state of the art, we set up two datasets, referred to as 1 and 2, for our experiments. The first comes from meta-genomic studies, e.g., [24], and includes all the sequenced microbial genomes (bacteria, archea, viruses) available in public databases. It consists of 40, 988 genomes, for a total of 172 GB. This scenario accounts also for the way in which the use of alignment-free methods is consolidated in the literature, e.g., [26], the main difference being the size of the dataset.

The complementary scenario of very large datasets composed of few very long genomes (e.g., gigabytes in length), due to computational constraint, has received very little attention in the literature and the biological relevance provided by alignment-free methods for datasets composed of, say, many mammalian genomes is still under study (see, e.g., [14]). Therefore, we consider this scenario for the sake of completeness. Again, there are no benchmark datasets available in the literature. Since the biologi-

cal relevance of those methods is still under investigation, our Dataset 2 is obtained artificially by generating at random 150 sequences on the DNA alphabet, each having a size of 1.6 GB, for a total of 240 GB.

4.1.2 Hardware

All the experiments have been conducted on a homogeneous cluster of 5 nodes, equipped each with 32 GB of RAM, 2 *AMD Opteron @ 2.10 GHz* processors, *CentOS 6* operating system, 1 TB disk drive and a Giga-Ethernet network card. The cluster includes four-slave node and a master node. The master node executes the `ResourceManager` and the `NameNode` services, while the slave nodes execute the `DataNode` and the `NodeManager` services. The Hadoop version is 2.7.1. On each slave node, up to eight concurrent map/reduce tasks are allowed. A HDFS replication factor set to two and a block size set to 128 MB have been used.

Experiments involving sequential algorithms have been conducted by executing a given algorithm on just one of the aforementioned slave nodes, once that the Hadoop support has been disabled.

4.2 Results and discussion

The aim of the experimentation reported here is to provide evidence that the paradigm discussed in Sect. 3.2 is indeed effective in extending the applicability, i.e., the range of problem sizes that alignment-free measures can handle with respect to the sequential setting. It is divided into three parts.

The first assesses that all the measures presented in Sect. 2.1 use essentially the same amount of time and memory, in the sequential setting. The results are reported in Fig. 2. Based on them, in what follows, we consider only the Squared Euclidean dissimilarity measure to assess the limits of the sequential implementations and as a term of comparison with the Hadoop paradigm.

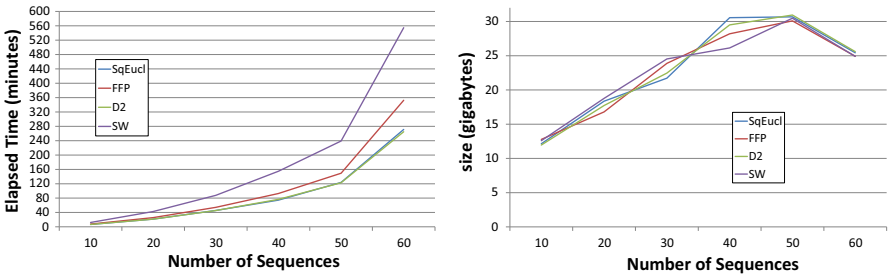

Fig. 2 An estimate of elapsed times (**a**) and overall memory (**b**) required for evaluating different dissimilarity measures among 60 sequences extracted from Dataset 1 with a total size of ≈600,000,000 characters, using the sequential implementation. Measures based on exact k-mers counts have been tested with $k = 12$ while the approach based on spaced-word frequencies (SW) has been tested using a pattern containing 12 care positions and 2 don't care positions

The second assesses the limits of the sequential algorithms with respect to memory size, since it is the key resource constraint limiting their execution.

The hardware and datasets available are used as follows: one slave processor of the cluster described in Sect. 4.1.2 and from the ones available, we extract datasets containing an increasing number of sequences. Moreover, we also vary the amount of memory available to the algorithm. The same tests are repeated with the distributed version of the algorithm, using 32 concurrent map/reduce tasks running on 4 slave nodes. In addition, we use a value of k that is representative of the values that are commonly used in the literature [8,16,18], i.e., $k = 12$. The results are reported in Fig. 3. It is to be noted that 150 sequences do not represent a limit to the execution of the distributed algorithm, but it is rather the setting where we concluded our experimentation due to its long execution times. Given that observation, if we consider the dataset coming from meta-genomic studies and take as a reference sequential machine a slave node equipped with 16 GB of RAM (that we take as a standard reference for desktop computers), we notice that the distributed algorithm, executed using four times the number of slave nodes, is able to solve problems that count at least five times the maximum number of sequences and at least four times the maximum size of sequences processable by the corresponding sequential algorithm. As far as the second dataset is concerned, the distributed algorithm, executed using four times the number of slave nodes, is able to solve problems that count at least ten times the maximum number and the maximum size of sequences processable by the corresponding sequential algorithm. For completeness, Table 1 summarizes the speedups obtained by the distributed algorithm proposed here with respect to the sequential implementation, using 16 GB of memory, when considering the maximum number of sequences from Dataset 1 and Dataset 2 processable by the sequential algorithm.

The third part of the experimentation assesses the scalability of the proposed distributed paradigm, compatible with the hardware available for this research, that is, how its time performance improves with respect to the degree of parallelism the algorithm is given on the cluster described in Sect. 4.1.2.

The hardware and datasets available are used as follows: four-slave nodes of the cluster described in Sect. 4.1.2 and from the first dataset available, we consider the 60 longest sequences for the experiment, for an overall size of ≈600 MB.

Concerning the second dataset, we extract only the 20 longest sequences, for an overall size of ≈32 GB. Such a choice is a good compromise since it allows for a comparison with the sequential algorithm while keeping experimentation time limited.

The first dataset so obtained has been processed with $k = 12$, while the second one has been processed with $k = 10$, since the sequential algorithm is unable to run in such a setting with $k = 12$, again preventing the possibility of performing a comparative analysis with the distributed algorithm. Moreover, we also vary the number of CPU cores that are reserved to the cluster and, consequently, the number of map/reduce tasks that are run concurrently. The same tests are repeated with the sequential algorithm. The overall results are reported in Figs. 4, 5 and 6, where we also indicate the time spent for steps 1 and 2, respectively (see basic paradigm in Sect. 3.1). For completeness, we also report in Table 2 the speedups obtained by the distributed algorithm in these two settings with respect to the sequential implementation, using 32 GB of memory.

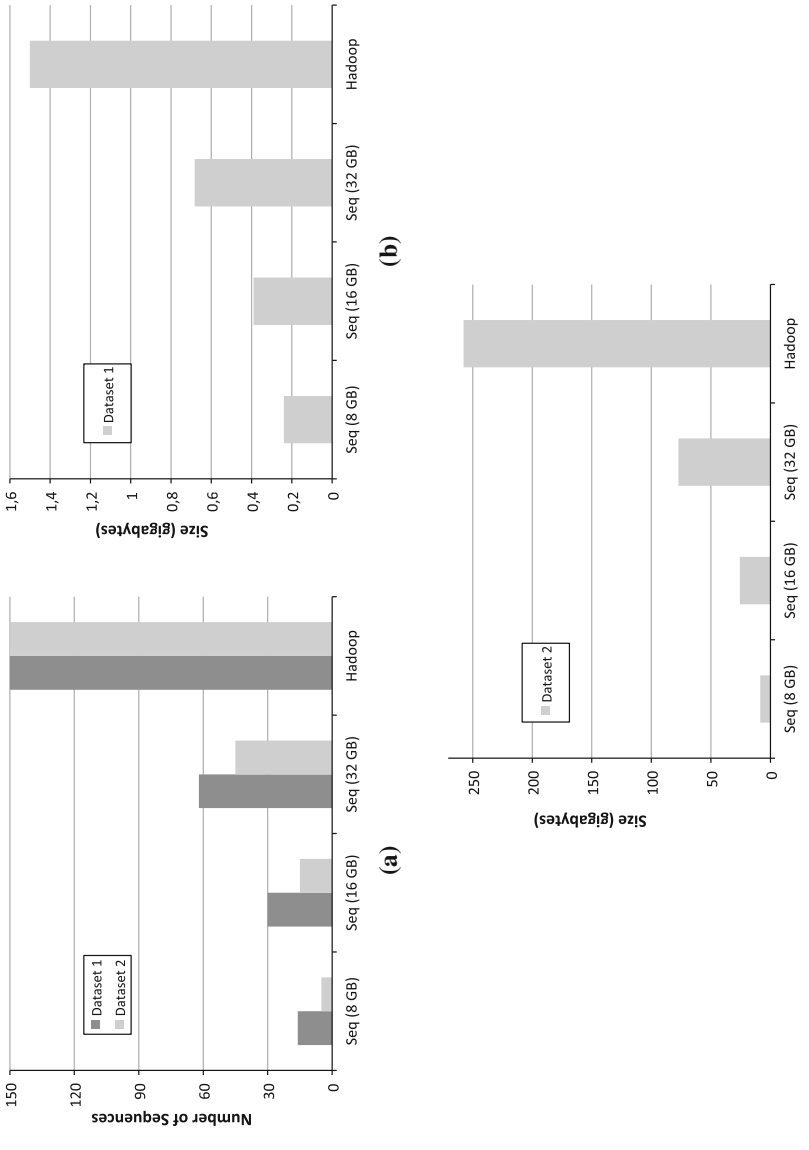

Fig. 3 The histograms provide the maximum number of sequences (**a**) that can be processed by the Squared Euclidean dissimilarity measure (sequential implementation), and their corresponding sizes (**b**, **c**), using an increasing amount of memory, by setting $k = 12$

Table 1 Speedup of the distributed implementation with respect to the sequential implementation, with 16 GB of memory, $k = 12$ and an increasing amount of concurrent map/reduce tasks

Total no. of concurrent map/reduce tasks	Speedup dataset 1	Speedup dataset 2
4	0.48	0.45
8	1.05	0.99
16	2.01	2.10
32	4.15	3.75

Speedup Dataset 1 refers to the experiment involving 30 sequences extracted from Dataset 1, with a total size of ≈370,000,000 characters, roughly corresponding to the maximum number of sequences from this dataset that can be processed by the Squared Euclidean dissimilarity measure (sequential implementation). Speedup Dataset 2 refers to the experiment involving 15 sequences extracted from Dataset 2, with a total size of ≈26,000,000,000 characters, roughly corresponding to the maximum number of sequences from this dataset that can be processed by the Squared Euclidean dissimilarity measure (sequential implementation)

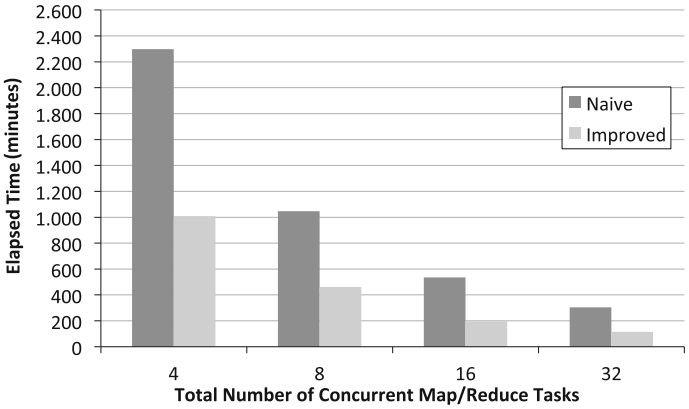

Fig. 4 Elapsed times for evaluating the Squared Euclidean dissimilarity measure among 60 sequences extracted from Dataset 1 with a total size of ≈600,000,000 characters, with $k = 12$ and an increasing number of concurrent map/reduce tasks, using both the naive and improved versions of our distributed paradigm

The following comments are in order. As expected, both the straightforward and the improved version of our distributed paradigm scale well with the number of containers. Moreover, it is clear that the improved version of the paradigm performs much better than the straightforward implementation, which would not be an improvement with respect to the sequential setting. Those facts give indication of the effectiveness of the splitting strategy and its refinement presented in Sect. 3. In addition, in settings where the improved version of the distributed algorithm can be compared to the sequential one, the distributed algorithm is obviously faster and much more so, when the sequences tend to be few and long.

In conclusion, based on the results and discussions outlined above, the distributed paradigm proposed here indeed extends the range of problem sizes that can be handled with respect to the sequential setting. Moreover, it also allows for a considerable speedup with respect to the sequential case. Finally, it is worth pointing out that the

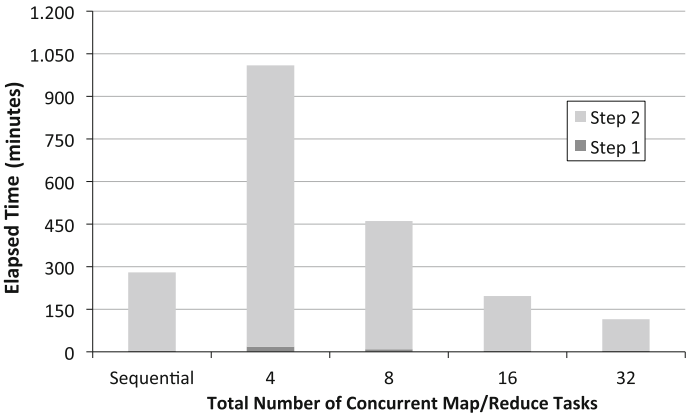

Fig. 5 Elapsed times for evaluating the Squared Euclidean dissimilarity measure among 60 sequences extracted from Dataset 1 with a total size of ≈600,000,000 characters, with $k = 12$ and an increasing number of concurrent map/reduce tasks. For completeness, we also report the time of the sequential algorithm

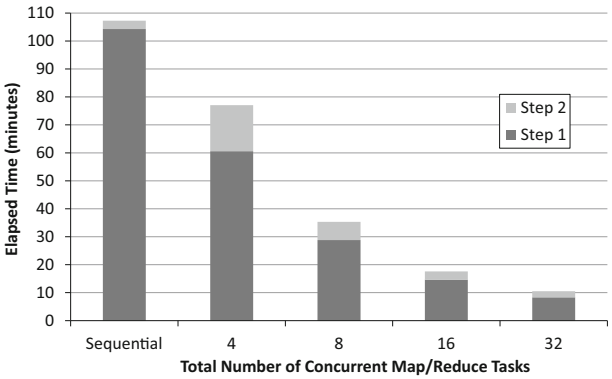

Fig. 6 Elapsed times for evaluating the Squared Euclidean dissimilarity measure among 20 sequences extracted from Dataset 2 of ≈1,600,000,000 characters each, with $k = 10$ and an increasing number of concurrent map/reduce tasks. For completeness, we also report the time of the sequential algorithm

straightforward implementation of our distributed paradigm, run using 32 containers, exhibits the same execution times of the sequential algorithm. This is a further evidence of the relevance of our improvement.

A final observation is in order. As shown in Figs. 5 and 6, the performance hotspot of the distributed algorithm changes from step 1 to step 2 when switching from $k = 10$ to $k = 12$, suggesting the following fact. When k is small, most of the execution time is spent for extracting k-mers, while when k increases this time becomes negligible with respect to the time spent for evaluating the dissimilarity function for each pair of input sequences. Such a switch has the following explanation, which points to possible future further improvement of the distributed algorithm proposed here. During step 2, a map task is run for each distinct k-mer found during step 1 and an output pair is created for each pair of sequences where it appears. This implies that, when increasing

Table 2 Speedup of the distributed implementation with respect to the sequential implementation, with 32 GB of memory and an increasing amount of concurrent map/reduce tasks

Total no. of concurrent map/reduce tasks	Speedup dataset 1	Speedup dataset 2
4	0.28	1.39
8	0.61	3.04
16	1.42	6.10
32	2.43	10.21

Speedup Dataset 1 refers to the experiment involving 60 sequences extracted from Dataset 1, with a total size of ≈ 600.000.000 characters, described in Fig. 5. Speedup Dataset 2 refers to the experiment involving 20 sequences extracted from Dataset 2, with a total size of ≈1,600,000,000 characters, described in Fig. 6

k, while the execution time of step 1 remains approximately the same, the execution time of step 2 increases with the number of distinct k-mers. Moreover, the performance deterioration of step 2 is exacerbated by the fact that, in Hadoop, the output of a map task has to be saved twice on disk. The first time, on the disk of the node executing the map task, the second time, on the disk of the node executing the following reduce task.

5 Conclusions and future work

We have presented the first k-mer-based alignment-free sequence comparison paradigm that makes good use of the Hadoop framework. As a result, we obtain an advancement of the state of the art in the important area of sequence analysis, since we significantly extend the range of problem sizes that can be solved by sequential alignment-free methods. Moreover, our solution is competitive with respect to them also in terms of time.

However, our results are only an initial step in assessing how informative, from a biological point of view, alignment-free methods can be on very large problem instances, in particular for higher eukaryotes and plant genomes since, as mentioned earlier, only partial results are available. Without the support of carefully engineered HPC alignment-free methods such an essential investigation cannot even be started.

As for future research directions, we also point out the following. Due to the large datasets now available, there are very important bioinformatics computational fundamental tasks that can take full advantage of Hadoop as, for instance, the determination of k-mer statistics. In fact, this group has work in progress dealing with this problem. Moreover, although energy consumption issues have not been extensively investigated in bioinformatics applications, they are certainly of relevance, e.g., [41]. Therefore, it would be of interest to design energy-aware sequence analysis algorithms.

Acknowledgements We would like to thank the Department of Statistical Sciences of University of Rome-La Sapienza for computing time on the TeraStat cluster and Nicola Segata for providing the meta-genomic dataset. We also would like to thank the referees for comments that helped in the presentation of our results.

Compliance with ethical standards

Funding MIUR PRIN Project: 2010RTFWBH_003 "Data-Centric Genomic Computing (GenData 2020)" and Unipa Progetto di Ateneo 2012-ATE-0298 "Metodi Formali ed Algoritmici per la Bioinformatica su Scala Genomica".

References

1. Allen F, Almasi G, Andreoni W, Beece D, Berne BJ, Bright A, Brunheroto J, Cascaval C, Castanos J, Coteus P et al (2001) Blue Gene: a vision for protein science using a petaflop supercomputer. IBM Syst J 40(2):310–327
2. Apostolico A, Giancarlo R (1998) Sequence alignment in molecular biology. J Comput Biol 5(2):173–196
3. Audano P, Vannberg F (2014) KAnalyze: a fast versatile pipelined k-mer toolkit. Bioinformatics 30(14):2070–2072
4. Boden M, Schöneich M, Horwege S, Lindner S, Leimeister C, Morgenstern B (2013) Alignment-free sequence comparison with spaced k-mers. OASIcs-OpenAccess Series in Informatics, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik 34:24–34
5. Cattaneo G, Roscigno G, Ferraro Petrillo U (2014) A scalable approach to source camera identification over Hadoop. In: 28th IEEE International Conference on Advanced Information Networking and Applications (AINA), IEEE, pp 366–373
6. Cattaneo G, Ferraro Petrillo U, Giancarlo R, Roscigno G (2015) Alignment-free sequence comparison over Hadoop for computational biology. In: 44rd International Conference on Parallel Processing Workshops (ICCPW 2015), IEEE, pp 1–9
7. Chan CX, Bernard G, Poirion O, Hogan JM, Ragan MA (2014) Inferring phylogenies of evolving sequences without multiple sequence alignment. Sci Reports 4:6504
8. Chor B, Horn D, Goldman N, Levy Y, Massingham T et al (2009) Genomic DNA k-mer spectra: models and modalities. Genome Biol 10(10):R108
9. Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. Operating Systems Design and Implementation (OSDI) pp 137–150
10. Deorowicz S, Kokot M, Grabowski S, Debudaj-Grabysz A (2015) KMC 2: fast and resource-frugal k-mer counting. Bioinformatics 31(10):1569–1576
11. Durbin R, Eddy S, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press, New York
12. Ekanayake J, Pallickara S, Fox G (2008) MapReduce for data intensive scientific analyses. In: 2008 IEEE Fourth International Conference on eScience, pp 277–284
13. Elsayed T, Lin J, Oard DW (2008) Pairwise document similarity in large collections with MapReduce. In: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies, pp 265–268
14. Fan H, Ives AR, Surget-Groba Y, Cannon CH (2015) An assembly and alignment-free method of phylogeny reconstruction from next-generation sequencing data. BMC Genom 16(1):1–18
15. Ferragina P, Giancarlo R, Greco V, Manzini G, Valiente G (2007) Compression-based classification of biological sequences and structures via the Universal Similarity Metric: experimental assessment. BMC Bioinform 8:252
16. Giancarlo R, Scaturro D, Utro F (2009) Textual data compression in computational biology: a synopsis. Bioinformatics 25(13):1575–1586
17. Giancarlo R, Lo Bosco G, Pinello L, Utro F (2013) A methodology to assess the intrinsic discriminative ability of a distance function and its interplay with clustering algorithms for microarray data analysis. BMC Bioinform 14(1):1–14
18. Giancarlo R, Rombo SE, Utro F (2014) Compressive biological sequence analysis and archival in the era of high-throughput sequencing technologies. Briefings Bioinform 15(3):390–406

19. Greco V, Giancarlo R (2007) Grid-K: A cometa VO service for compression-based classification of biological sequences and structures. Symposium GRID Open Days at the University of Palermo, Italy pp 87–93
20. Gunarathne T, Wu TL, Qiu J, Fox G (2010) MapReduce in the clouds for science. In: 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, pp 565–572
21. Gusfield D (1997) Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, New York
22. Haubold B (2014) Alignment-free phylogenetics and population genetics. Briefings Bioinform 15(3):407–418
23. Horwege S, Lindner S, Boden M, Hatje K, Kollmar M, Leimeister CA, Morgenstern B (2014) Spaced words and kmacs: fast alignment-free sequence comparison based on inexact word matches. Nucleic Acids Res 42(W1):7–11
24. Huang K, Brady A, Mahurkar A, White O, Gevers D, Huttenhower C, Segata N (2013) MetaRef: a pan-genomic database for comparative and community microbial genomics
25. Kullback S, Leibler RA (1951) On information and sufficiency. Ann Math Stat 22(1):79–86
26. Leimeister CA, Boden M, Horwege S, Lindner S, Morgenstern B (2014) Fast alignment-free sequence comparison using spaced-word frequencies. Bioinformatics 30(14):1991–1999
27. Li KB (2003) ClustalW-MPI: ClustalW analysis using distributed and parallel computing. Bioinformatics 19(12):1585–1586
28. Lloyd S, Snell Q (2011) Accelerated large-scale multiple sequence alignment. BMC Bioinform 12:466
29. Marçais G, Kingsford C (2011) A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. Bioinformatics 27(6):764–770
30. Myers EW, Sutton GG, Delcher AL, Dew IM, Fasulo DP, Flanigan MJ, Kravitz SA, Mobarry CM, Reinert KH, Remington KA et al (2000) A whole-genome assembly of drosophila. Science 287(5461):2196–2204
31. Nordberg H, Bhatia K, Wang K, Wang Z (2013) BioPig: a Hadoop-based analytic toolkit for large-scale sequence data. Bioinformatics 29(23):3014–3019
32. Schatz MC (2009) Cloudburst: highly sensitive read mapping with MapReduce. Bioinformatics 25(11):1363–1369
33. Shvachko K, Kuang H, Radia S, Chansler R (2010) The Hadoop distributed file system. In: IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), IEEE, pp 1–10
34. Sims GE, Kim SH (2011) Whole-genome phylogeny of *Escherichia coli*/Shigella group by feature frequency profiles (FFPs). Proceed Nat Acad Sci 108(20):8329–8334
35. Talia D, Trunfio P, Marozzo F (2015) Data analysis in the cloud: models, techniques and applications, 1st edn. Elsevier Science Publishers B. V, Amsterdam
36. Taylor RC (2010) An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. BMC Bioinform 11(Suppl 12):1–6
37. Torney DC, Burks C, Davison D, Sirotkin KM (1990) Computation of d2: a measure of sequence dissimilarity. In: Computers and DNA: the proceedings of the Interface between Computation Science and Nucleic Acid Sequencing Workshop, Redwood City, Calif.: Addison-Wesley Pub. Co
38. Vinga S (2014) Editorial: alignment-free methods in computational biology. Brief Bioinform 15(3):341–342
39. Vinga S, Almeida J (2003) Alignment-free sequence comparison-a review. Bioinformatics 19:513–523
40. Vouzis PD, Sahinidis NV (2010) GPU-BLAST: Using graphics processors to accelerate protein sequence alignment. Bioinformatics
41. Warnke J, Pawaskar S, Ali H (2012) An energy-aware Bioinformatics application for assembling short reads in high performance computing systems. In: 2012 International Conference onHigh Performance Computing and Simulation (HPCS), pp 154–160
42. Wong AK, You M (1985) Entropy and distance of random graphs with application to structural pattern recognition. IEEE Trans Patt Anal Mach Intel 7(5):599–609
43. Yang K, Zhang L (2008) Performance comparison between k-tuple distance and four model-based distances in phylogenetic tree reconstruction. Nucl Acids Res 36(5):1–9