Bioinformatics, 34(11), 2018, 1826–1833
doi: 10.1093/bioinformatics/bty018
Advance Access Publication Date: 12 January 2018
Original Paper

Genome analysis

Informational and linguistic analysis of large genomic sequence collections via efficient Hadoop cluster algorithms

Umberto Ferraro Petrillo[1,*,†], **Gianluca Roscigno**[2], **Giuseppe Cattaneo**[2] **and Raffaele Giancarlo**[3]

[1]Dipartimento di Scienze Statistiche, Università di Roma – La Sapienza, Rome 00185, Italy, [2]Dipartimento di Informatica, Università di Salerno, Fisciano, SA 84084, Italy and [3]Dipartimento di Matematica ed Informatica, Università di Palermo, Palermo 90133, Italy

*To whom correspondence should be addressed.
†Part of this work has been done while on a visit to College of Computing, Georgia Institute of Technology.
Associate Editor: Alfonso Valencia

Received on August 2, 2017; revised on December 12, 2017; editorial decision on January 8, 2018; accepted on January 9, 2018

Abstract

Motivation: Information theoretic and compositional/linguistic analysis of genomes have a central role in bioinformatics, even more so since the associated methodologies are becoming very valuable also for epigenomic and meta-genomic studies. The kernel of those methods is based on the collection of *k*-mer statistics, i.e. how many times each *k*-mer in $\{A, C, G, T\}^k$ occurs in a DNA sequence. Although this problem is computationally very simple and efficiently solvable on a conventional computer, the sheer amount of data available now in applications demands to resort to parallel and distributed computing. Indeed, those type of algorithms have been developed to collect *k*-mer statistics in the realm of genome assembly. However, they are so specialized to this domain that they do not extend easily to the computation of informational and linguistic indices, *concurrently* on sets of genomes.
Results: Following the well-established approach in many disciplines, and with a growing success also in bioinformatics, to resort to *MapReduce* and Hadoop to deal with 'Big Data' problems, we present `KCH`, the first set of *MapReduce* algorithms able to perform *concurrently* informational and linguistic analysis of large collections of genomic sequences on a Hadoop cluster. The benchmarking of `KCH` that we provide indicates that it is quite effective and versatile. It is also competitive with respect to the parallel and distributed algorithms highly specialized to *k*-mer statistics collection for genome assembly problems. In conclusion, `KCH` is a much needed addition to the growing number of algorithms and tools that use *MapReduce* for bioinformatics core applications.
Availability and implementation: The software, including instructions for running it over Amazon AWS, as well as the datasets are available at http://www.di-srv.unisa.it/KCH.
Contact: umberto.ferraro@uniroma1.it
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Linguistic and informational analysis of biological sequences is one of the mainstays of alignment-free genomic and proteomic sequence analysis, with further extensions and applications to epigenomic and metagenomic studies, e.g. (Benoit *et al.*, 2016; Giancarlo *et al.*, 2009, 2015, 2015; Lo Bosco, 2016; Nordstrom *et al.*, 2013; Pinello *et al.*, 2011; Utro *et al.*, 2016). At their heart, many of those methods have the collection of *k*-mer statistics, i.e. how many times each sequence of length k over a finite alphabet appears in a biological sequence. Once that such a statistics is available, one can use it to compute many informational and linguistic indices (Giancarlo *et al.*, 2009, 2014), the ones considered in this study being defined in Section 2.2. Although the computation of those statistics and indices are quite simple from the algorithmic point of view, the quantity of data on which they have to be computed makes the problem no longer solvable with a conventional computer. Here we propose KCH, the first suite of linguistic and informational analysis algorithms designed under the *MapReduce* paradigm (Dean and Ghemawat, 2004) and implemented and tested on a Hadoop cluster (White, 2015). Indeed, following other areas of science, *MapReduce*, Hadoop and Spark (Zaharia *et al.*, 2010) are rapidly becoming a standard in dealing with 'Big Data' problems in bioinformatics (see Cattaneo *et al.*, 2017a,b; Ferraro Petrillo *et al.*, 2017a,b; Zhou *et al.*, 2017). In order to properly place our contributions in the context of the State of the Art, we need a formal statement of the kinds of k-mer statistics one is interested in, together with the types of algorithms relevant for this study.

1.1 Exact *k*-mer statistics for sets of collections of genomic sequences: formal statement and algorithmic specification

We define two versions of the statistics of interest. Let Σ be an alphabet and let S be a finite set of collections of sequences in Σ^*. A local k-mer statistics (LS, for short) consists of computing how many times each of the k-mers in Σ^k appears in each of the collections in S, separately. Another statistics, referred to as cumulative statistics (CS for short), collects how many times each of the k-mers in Σ^k appears in the collections of sequences in S, cumulatively. We are interested in two types of algorithms, the first takes as input a collection of sequences and provides as output LS, while the second type provides CS as output. It is clear that algorithms computing LS can compute CS *in one execution*: pad each sequence in each collection in S with an extra symbol not in Σ and then concatenate the resulting sequences from all collections to obtain the input to LS, which will consist of a single collection with a single sequence in it. On the other hand, algorithms computing CS cannot compute LS, although they can be used as subroutines, i.e. m calls to an algorithm computing CS, where m is the number of collections in S.

1.2 State of the art

Motivated by genome assembly applications, several algorithms have been designed and tested that make use of parallel computing (HPC for short), both shared-memory multi-thread or distributed, for the collection of k-mer statistics. Technically, they address the computation of CS and not even for arbitrary sequence collections (most of them work for NGS data). For the convenience of the reader, they are reported in Section 2.1.2 of the Supplementary Material.

In regard to this study, the most serious drawback that they exhibit is that, although they can be used as subroutines to solve LS, it is open how that would affect their performance and scalability, in particular in reference to the downstream computation of informational and linguistic indices. For instance, the mentioned HPC programs count only k-mers, so their output must be stored on disk before it is passed on to another process that computes informational and linguistic indices on collections of sequences on which LS has been computed.

1.3 Our contributions

1.3.1 The first distributed suite of algorithms for both LS and CS

KCH is the first suite of Distributed Algorithms that can efficiently solve *both* LS and CS and provide the distributed computation of informational indices. Where KCH can be compared with the state of the art, i.e. the computation of CS, it is at least a factor of 30 faster with respect to the other *MapReduce* solution available, i.e. BioPig (Bhatia and Wang, 2011; Nordberg *et al.*, 2013). Although somewhat methodologically unfair to compare algorithms designed on different architectural principles, KCH is also very competitive with respect to shared-memory multi-thread algorithms. For the comparison, we have chosen a representative set (criteria detailed in Section 2.1.2 of the Supplementary Material): *KAnalyze* (Audano and Vannberg, 2014), *Jellyfish2*, an evolution of Jellyfish (Marçais and Kingsford, 2011), *DSK* (Rizk *et al.*, 2013) and *KMC3* (Kokot *et al.*, 2017), this latter being the fastest algorithm for CS in this class to date.

1.3.2 Methodology: architectures and programming for big data in bioinformatics

Our experiments show the following. (i) Via a rigorous comparison between KCH and state of the art shared-memory multi-thread algorithms, *MapReduce*, Hadoop and distributed computing are quite competitive for Bioinformatics applications, reinforcing the findings of the only previous quantitative study available in the Literature (Siretskiy *et al.*, 2015). (ii) Although scripting software platforms such as BioPig are quite useful for fast prototyping and software development, the design and engineering of efficient bioinformatics algorithms for fundamental tasks cannot be eluded even under the apparently highly scalable *MapReduce* paradigm.

1.3.3 Further insights into the linguistic and informational nature of genomes

We have performed two sets of experiments. The first concerns the origin of nullomers (Hampikian and Andersen, 2007), i.e. short sequences that do not appear in a given genome, and it provides further experimental support to the point of view that nullomers in genomes are accountable by means of a random process (Aurell *et al.*, 2016), i.e. they have a clear combinatorial nature although that does not rule out other biology-related reasons for the absence of some of them in genomes, e.g, mutational pressure (Acquisti *et al.*, 2007) or unfavorable structural conformation (Vergni *et al.*, 2016). The second concerns the first analysis of three large plant genomes that highlights the differences between their linguistic and informational content and their taxonomic classification.

2 Materials and methods

2.1 Hadoop, the *MapReduce* paradigm and algorithms: key factors influencing efficiency

It is well known that a distributed system is composed of a set of independent computing nodes, each consisting of several processing cores, main and external memory. Nodes must cooperate via communication in order to solve a specific problem (Ben-Ari, 2006). Key to the efficiency of the system is the middleware, in our case, Hadoop.

The algorithms that it supports have to be designed according to the *MapReduce* paradigm, which consists of splitting the input into several parts, each one processed by a dedicated *map task* in a parallel fashion. The final output is obtained by executing a set of independent *reduce tasks*, that are in charge of aggregating the results produced by the map tasks. We highlight next some of the key parameters that influence the efficiency of an algorithm executed under Hadoop. Additional aspects relating to Hadoop as well as the lifetime of an Hadoop application are presented in Cattaneo *et al.* (2017b).

With reference to Figure 1, a cluster of computers supervised by Hadoop includes at least a *master* node, mainly responsible for the global assignment of computational resources, and a set of *slave* nodes, where each node is part of the *Hadoop Distributed File System* (HDFS) (Shvachko *et al.*, 2010) and it is able to run one or more *worker* processes, which are software agents that allow to efficiently perform either map or reduce tasks. They play a key role.

Indeed, a single slave node can execute a variable number of workers, thus allowing to fully exploit the parallelism of multi-core nodes. In order to obtain an efficient distributed algorithm under Hadoop, both the number of workers per node and the number of tasks that each of them can handle are essential parameters since they affect the granularity of the parallelism and the trade-off overhead/useful computation. Likewise, the trade-off between the number of workers running on a slave node and the amount of physical memory available on that node may heavily affect the execution time due to virtual memory thrashing and I/O bus congestions (Denning, 1970).

A less obvious key role is played by HDFS, that includes efficient mechanisms to transparently and automatically split the input among the slave nodes. This approach favors data local computation, i.e. each slave operates on the data that are within it, and it seems the most suitable for k-mer statistics. For such a mechanism, one essential parameter of HDFS is the *block size*, which determines the number of blocks in a partition of each HDFS file and, implicitly, the amount of data that each map task has to process. In turn, the choice of a block size has impact on both the total number of map tasks necessary to execute an algorithm with a given input and the duration of each task. For instance, a small block size may result in an inefficient use of the system since the time spent by the middleware for the management of the given task may be larger than the one required for the execution of the task itself.

Summarizing the above discussion, in order to achieve a full use of the available resources one must account for: (a) the number of nodes, and for each of them, the number of cores and the main memory in it; (b) the number of blocks in which the input is partitioned. In terms of Hadoop configuration parameters, (a) and (b) can be controlled by a proper setting of (1) the maximum number of workers that can be simultaneously executed on each slave; (2) the maximum main memory available to each worker and (3) the HDFS block size.

2.2 A selection of informational and linguistic indices for sequence analysis

The indices that have been selected for inclusion in KCH are defined next, for a single sequence $X \in S$ and for a fixed value of k.

- *Estimating k-mer Probability Distributions from Counts* (see Giancarlo *et al.*, 2015). We provide both Maximum Likelihood and Bayesian estimates with the use of pseudo-counts, those latter reported in Section 1.1 of the Supplementary Material for the convenience of the reader. In its most basic version, it is obtained from LS, by dividing the frequencies of the k-mers occurring in X by the overall number of k-mers in X.
- *Empirical Entropy* (see Giancarlo *et al.*, 2009). The empirical entropy of a sequence X is defined as $H(X) = -\sum_{x\in\Sigma^k} p(x)log_2(p(x))$, where $p(x)$ denotes the empirical probability of the k-mer x occurring in X.

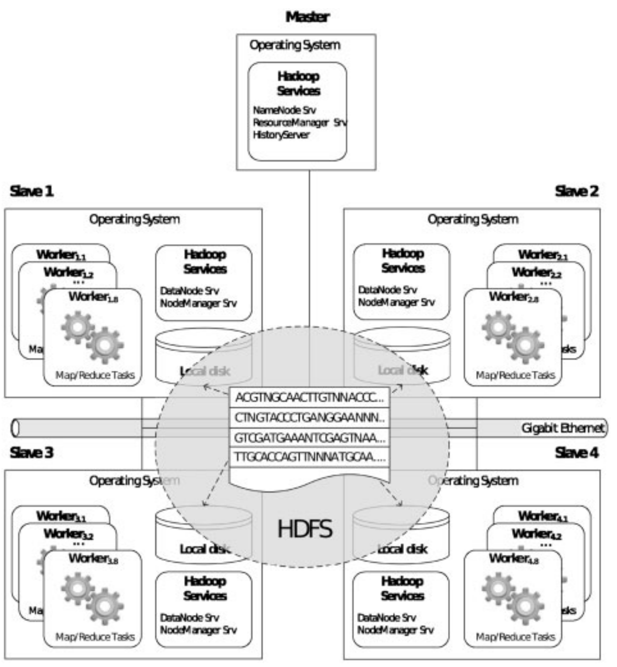

Fig. 1. A schematic representation of a cluster of processors configured to run an algorithm designed for Hadoop. Each component is as described in Section 2.1. The communication among nodes uses the Gigabit Ethernet, while the storage of input and output files is distributed among the local disks present in each slave node

- *k-mers Spectrum* (see Chor *et al.*, 2009). It is a plot of an histogram in which the abscissa denotes the potential number of occurrences of k-mers in X and the ordinate gives how many of the k-mers in Σ^k appear exactly that number of times.
- *Linguistic Complexity* (see Utro *et al.*, 2016). Assume that $k \le |X|$. $LCS(X, i)$ is defined as the number of distinct i-mers that are present in X, normalized by $|X|$. Then, the linguistic complexity $LC(S,k) = \sum_{i=1}^{k} LCS(S,i)$.

2.3 KCH—the core component

We present how to compute k-mer statistics for CS. How to further extend KCH to perform the same task for LS and to how compute the indices defined in Section 2.2 is given in Section 1.3 of the Supplementary Material. Moreover, from now on, when we mention KCH, we refer to the computation of either LS or CS, unless otherwise stated.

2.3.1 High level description

(1) Data Input Input sequences are read from HDFS nodes using the Fastdoop library (Ferraro Petrillo *et al.*, 2017a) (Fig. 2a).

(2) Map Phase and Network Shuffle. Each map task creates a set of r local hash tables Hts to maintain the frequency counts of the k-mers found while scanning its own input sequence(s) (Fig. 2b). These hash tables represent an implicit binning/partitioning of Σ^k and they are used to perform a first level of counting aggregation. Then, each map task extracts the k-mers from its input sequence(s) and updates its hash tables accordingly. At the end of the map phase, all hash tables related to the same partition of Σ^k are sent for aggregation to the same distinct reduce task by means of a network shuffle (Fig. 2c). If an hash table gets full while still scanning the input sequence(s), it is immediately sent to the reduce phase and replaced by a new empty hash table.

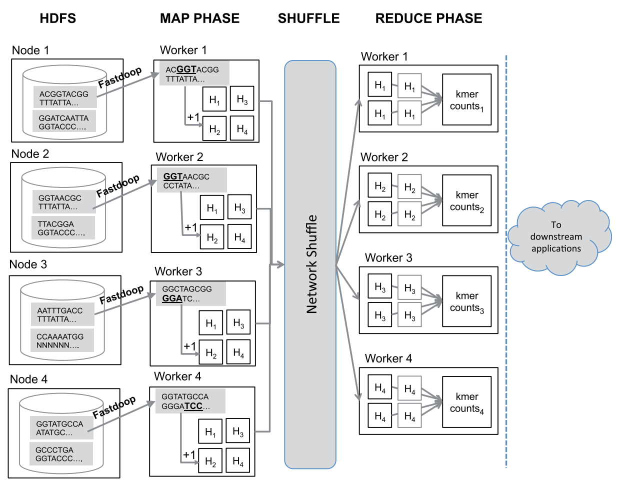

Fig. 2. A snapshot of the KCH architecture outlining how data is distributed, processed and recombined during the workflow. (**a**) **HDFS**. Input sequences are initially stored on an instance of the HDFS distributed filesystem and loaded in memory using the Fastdoop library. (**b**) **Map Phase**. Each map task creates a set of local hash tables that are used to evaluate and to maintain the *k*-mers frequencies of its own sequence(s). (**c**) **Network Shuffle**. All the hash tables coming from different map tasks and related to the same partition of Σ^k are sent for aggregation to the same distinct reduce task. (**d**) **Reduce Phase**. Each reduce task aggregates all the hash tables related to a same partition and makes available the computed *k*-mers counts to downstream applications

(3) **Reduce Phase and Downstream Applications.** In this phase, a reduce function receives as input all the hash tables of the same partition to compute their aggregated statistics. They are retrieved from the network shuffle and processed one at time, to reduce memory requirements. The *k*-mers counts contained in each input hash table are aggregated in a target hash table. After the aggregation, the reduce function will make available to downstream applications the counts of all *k*-mers found in its corresponding bin (Fig. 2d).

2.3.2 A second level of details

The KCH counting algorithm is composed of the following functions.

a. **Function** *setup*. A map task creates r hash tables with C_{map} entries.
b. **Function** *map*. The input of a map function is the identifier of the sequence *idSeq* and the sequence (or part of it in case of very long sequences) *Seq*. Our algorithm uses the standard binary encoding of a letter of the alphabet $\{A, C, G, T\}$ to pack a *k*-mer into an integer. Since now each character of a *k*-mer needs two bits rather than eight, we have a saving in memory usage but also an additional one in the transmission of partial statistics from the workers performing map to the ones performing reduce. Moreover, it provides a very effective implementation of the scanning strategy used to extract *k*-mers from an input sequence, described next. Initially, a new *k*-mer $kmer'$ is extracted by looking at the first k characters of the input sequence and packed into a single integer. The same is done for its canonical representation. From that point on, new *k*-mers are extracted by processing the last *k*-mer found by means of binary *SHIFT* and *AND* operations. The process goes on until the end of the sequence is reached or an N character is found. i.e. a character indicating that no letter from the DNA alphabet could be assigned to that genomic position. Accordingly, the algorithm skips all the sequences of length k containing the N character and starts over the scanning strategy. In addition, when a *newline* character is found, it is ignored. Whenever a new *k*-mer $kmer'$ is found, its local partial frequency count is updated accordingly, but no output is provided. In particular, the hash table for which one needs to increment the counter or start a new one is identified as follows: $kmer'$ is placed in the hash table having the id obtained by taking the numerical representation of $kmer'$ mod r. Once that its input has been scanned, the map task proceeds by emitting as output the copy of each hash table, together with an identifier, by executing the *endupFlush* function.
c. **Function** *intermediateFlush*. Once fixed an initial size for the hash tables, some of them may need to be expanded at run-time and that, in turn, may cause a map task to run out of memory. In order for that to be avoided, the algorithm follows a *flushing strategy* by means of which an hash table can be output even if the task has not completed yet its execution. In fact, the function *intermediateFlush* is executed to check if the number of elements in a hash table ht exceeds a certain threshold t. If this is true, the algorithm emits the id of the table as key, i.e. $idHt$, and its binary copy as value, i.e. ht. Then, this local table is replaced with an empty one.
d. **Function** *endupFlush*. The algorithm emits the id of the target table as key, i.e. $idHt$, and its binary copy as value, i.e. ht. Then, the table is discarded.

Algorithms 1 and 2, reported in Section S1.2 of the Supplementary Material, provide further details about KCH, including the critical choice of the hash function. We also point out that we have devised a methodology, presented in Section S1.4 of the Supplementary Material, to engineer KCH on a specific Cluster for taking full advantage of the hardware available.

3 Results

3.1 Experimental methodology

In analogy with (Rizk *et al.*, 2013), we use synthetic datasets derived from the Illumina human genome dataset (Available at: ftp://ftp.ddbj.nig.ac.jp/ddbj_database/dra/fastq/SRA010/SRA010896/SRX016231/). We also use a set of 12 genomes, namely *Bacillus subtilis, Escherichia coli, Helicobacter hepaticus, Legionella pneumophila, Neisseria meningitidis, Caenorhabditis elegans, Drosophila melanogaster, Homo sapiens, Saccharomyces cerevisiae*, all downloaded from GenBank (Benson *et al.*, 2013), and *Picea glauca* (Birol *et al.*, 2013) (PG for short), *Picea abies* (Nystedt *et al.*, 2013) (PA for short) and *Pinus taeda* (Zimin *et al.*, 2014) (PT for short) for a total of 60 GB. Additional information about the datasets are available in Section 2.1.1 of the Supplementary Material. As for hardware, we have used a cluster, whose configuration and Hadoop parameter setting are detailed in Section S2.2 of the Supplementary Material.

3.2 Scalability of KCH

One of the key factors in judging a *MapReduce* program is its scalability, i.e. its ability to take full advantage of the processing power made available to it. Here, we measure such an ability for KCH. We discuss first LS.

Experiments are performed with varying dataset sizes and values of k. For the former, we have used 2, 8, 32 and 128 GB, since that range of sizes well represents possible input sizes of datasets coming from genomic and metagenomic studies. Likewise, the chosen values of k, i.e. 3, 7 and 15, are representative of the ones that are expected to be used in applications such as alignment-free sequence comparison, informational, linguistic and compositional analysis of biological sequences, e.g. Chor *et al.*, (2009) and Giancarlo *et al.*, (2009, 2014), where values of k substantially above 10 are hardly found. For each size, we generate a dataset as outlined in Section S2.1.1 of the Supplementary Material.

As it is evident from the results reported in Figure 3, the advantage of using more and more workers, i.e. the scalability of the algorithm, becomes more and more evident as the dataset size increases. It is also evident that the beneficial effect of increasing the number of workers per slave fades out. This happens because having many workers running on the same node may produce performance bottlenecks due to several processes trying to access the same disk or the same network connection at the same time. Finally, it has to be pointed out that the behavior of our algorithm with respect to scalability has the same trend for all the tested values of k.

As for CS, we can draw the same conclusions as LS based on the experiments reported in Section 2.3 of the Supplementary Material.

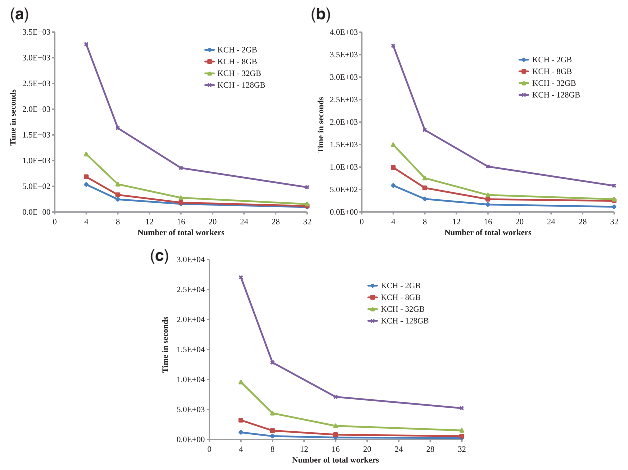

Fig. 3. Local Statistics. (**a**) Scalability of KCH, for the case $k = 3$, for all datasets, which are indicated in the figure according to the legend to the right. The abscissa gives the number of workers used, while the ordinate gives the corresponding time. (**b**)–(**c**) As in (a), but for $k = 7$ and $k = 15$

3.3 KCH in application domains: useful, versatile and competitive

In order to show the flexibility and competitiveness of KCH, we present in Section 3.3.1 results relating to the presence of nullomers in biological sequences (Hampikian and Andersen, 2007) versus the estimation of the informational content of a genome. We also provide an outline of the first linguistic and informational analysis of the three plant genomes with the use of the indices mentioned in Section 2.2. Additional details are given in Section S2.4 of the Supplementary Material, due to space limitations. Finally, Section 3.3.3 reports a comparison between KCH and the analogous shared-memory multi-thread software programs available and that are motivated and useful for genome assembly only.

3.3.1 Linguistic and informational analysis of sets of genomes

For each of the genomes included in this study, we first establish the largest value of k that we can use in our analyses involving k-mers. To this end, we resort to the technique proposed in (Giancarlo *et al.*, 2015), which identifies the maximum k such that it is still possible to estimate reliably the information content of a given genome via its empirical k-mer probability distribution. That is, the maximum value of k for which one can estimate the kth order entropy of the genome within a given error bound, expressed in percentage and given as a parameter to the method. For the interested reader, the technical details are in Section S1.5 of the Supplementary Material. We use two threshold values, one very conservative while the other standard in experimental analysis, i.e. 1% and 5%, respectively. For each of the genomes, the corresponding maximal values of k we can use for our analysis are reported in Table 1.

3.3.2 Nullomers versus the reliable estimation informational content of a genome

In order to place our findings in the proper context, it is worth recalling that the latest research on the origin of nullomers indicates that: (i) there may be structural properties of DNA involved in their absence (Vergni *et al.*, 2016); (ii) relatively short nullomers may be seen as the result of sampling from a random sequence (Aurell *et al.*, 2016); (iii) while very long ones have a biological origin (Aurell *et al.*, 2016). We contribute to give further support to (ii) by highlight the combinatorial nature of the origin of nullomers in terms of their relation with the well known finite sampling effect (Aurell *et al.*, 2016; Giancarlo *et al.*, 2009), i.e. if the genome one is analyzing has a short length with respect to 4^k, not all k-mers will occur in that genome, making a reliable estimation of the kth order entropy of that genome impossible.

For a given genome, let $k_{min,g}$ denote the minimum taken over all nullomer lengths in genome g. With reference to Table 1, there is a self-evident correlation between the second and third columns in that table. it is worth of mention that the Spearman rank correlation value is 0, 9734 with a P-value of 9, 9E–8. That is, as the genome length grows, so does $k_{min,g}$.

Now, let $k_{max,g,t}$ denote the maximum value of k for which one can estimate the kth order entropy of genome g with an error of t%. Then, for each genome in this study, we have that $k_{min,g}$ is within the interval $[k_{max,g,1}, k_{max,g,5}]$. Moreover, the values of $k_{min,g}$ and $k_{max,g,5}$ coincide for 8 of the 12 genomes in this study, while for two of them the same happens for $t = 1$. For completeness, the Spearman rank correlations between the k_{min} sequence and the sequence of left and right interval points is 0, 9847 (P-value 1.5E–8) and 0.9982 (P-value 1, 6E–13).

In conclusion, our experiments indicate that nullomers start to occur in proximity of where the finite sample effect starts to play a role in the reliable estimation (error at 5% or below) of the informational content of a genome. For completeness, we also report in Tables 2–6, the percentage of nullomers that the genomes in this study have in common, up to their k_{max} values at error rate of 5%. The results in the tables clearly indicate that the considered nullomer sets are highly species-specific, a result in agreement with the Literature. Indeed, nullomer sets, in particular nullomers of minimal length, retain enough species-specificity to be of use in Phylogeny, e.g. (Rahman *et al.*, 2016).

Taxonomy versus Compositional and Informational Content of Three Large Plant Genomes. Given the mentioned plant genomes, we have computed the informational and linguistic indices mentioned in Section 2.2, for each of them, up to $k = 14$ (the maximum value for which the informational content of the three genomes is preserved up to an error of 5%). A complete analysis is provided in Section S2.4 of the Supplementary Material, due to space limitations. Here we limit ourselves to highlight a finding that is perceived as methodologically significant: with reference to Supplementary Figures S2–S5, both in terms of informational and linguistic content, PT and PG are closer to each other than to PA, although PA and PG are taxonomically closer to each other than to PT (see ITIS Partnership, 2010).

Table 1. The first two columns are self-explanatory

Genome	Length (bytes)	k_{min}	k_{max}	
			1%	5%
Helicobacter hepaticus	1 821 649	7	5	7
Neisseria meningitidis	2 300 775	8	6	8
Legionella pneumophila	3 440 237	8	6	8
Bacillus subtilis	4 268 311	8	6	8
Escherichia coli	4 699 685	8	6	8
Saccharomyces cerevisiae	12 309 083	9	7	9
Caenorhabditis elegans	101 539 997	10	9	10
Drosophila melanogaster	145 522 589	11	8	11
Homo sapiens	3 294 586 009	11	11	13
Picea abies	12 666 745 333	12	12	14
Pinus taeda	22 353 200 238	12	13	14
Picea glauca	25 194 253 706	12	13	14

Note: The third gives the minimum value of k for which there are nullomers in each genome, while the remaining columns give the maximum value of k one can use in order to estimate within a given percentage of error the entropy of each genome via its k-order empirical probability distribution.

Table 2. Cardinality of the intersection of nullomer sets in BS, EC, LP and NM genomes, given in percentage w.r.t. to the size of the smaller set involved in the intersection, for $k=8$

	BS	EC	LP	NM
BS	–	2E–5	0	0
EC	2E–5	–	1E–4	1E–3
LP	0	1E–4	–	8E–5
NM	0	1E–3	8E–5	–

Note: Each species is indicated by the initials of its Latin name, e.g. *Bacillus subtilis* is indicated as BS.

Table 3. The table legend is as in Table 2, but for HS and DM and $k=11$

	DM
HS	3E–8

Table 4. The table legend is as in Table 2, but for HS, PT, PA and PG and $k=12$

	HS	PA	PG	PT
HS	–	6E–6	2E–6	1E–6
PA	6E–6	–	2E–7	5E–8
PG	2E–6	2E–7	–	9E–8
PT	1E–6	5E–8	9E–8	–

Table 5. The table legend is as in Table 2, but for HS, PT, PA and PG and $k=13$

	HS	PA	PG	PT
HS	–	6E–3	3E–3	2E–3
PA	6E–3	–	5E–4	3E–4
PG	3E–3	5E–4	–	2E–4
PT	2E–3	3E–4	2E–4	–

Table 6. The table legend is as in Table 2, but for PT, PA and PG and $k=14$

	PA	PG	PT
PA	–	7E–2	4E–2
PG	7E–2	–	2E–2
PT	4E–2	2E–2	–

3.3.3 Cumulative statistics for genome assembly

For the experiments conducted in this section, we use the same datasets as in Section 3.2. In addition to the values of k indicated in that section, we have also experimented with $k=31$, which is a value that finds use in k-mer statistics for genome assembly (Compeau *et al.*, 2011). Moreover, since we are dealing with reads that have yet to be assembled, we consider canonical k-mer counts (i.e. for each k-mer k mer' found, we consider the lexicographically smaller among $kmer'$ and its reverse complement). CS is the relevant statistic here.

Comparison of `KCH` *with* `BioPig`. It computes CS with the use of the 'kmerCount.pig' routine (Nordberg *et al.*, 2013). We have used it with the same parameter settings as `KCH`. We omit the results obtained when using the 2GB dataset as in this case we would have only 8 worker nodes used over 32 because of the HDFS block size set to 256 MB. Unfortunately, we were unable to test large values of k, e.g. 15 and 31, for the two largest datasets, due to the inefficiencies of `BioPig`. Therefore, in Figure 4, we limit ourselves to report as indication the results of the comparison of `KCH` with `BioPig` for the computation of CS on the 32GB and 128GB datasets, using $k=3, 7$. It is to be remarked that the average speed-up of `KCH` with respect to `BioPig` is about $\geq 30\times$ in those settings.

Comparison of `KCH` *with Shared-Memory Multi-Thread Algorithms*. Although it is difficult to compare a multi-thread algorithm with a distributed one, we proceed as follows in order to obtain results as fair as possible. A thread is considered as a worker in our framework and each of the multi-thread algorithms are executed on a single slave node of the available cluster. We take as measure of performance the speed and the scalability of the various algorithms. Of the chosen algorithms to be included in this study (see Introduction), here we discuss `KMC3` only: it is consistently the best performing algorithm among the shared-memory multi-thread ones, to date. For completeness, the experiments with the other algorithms selected for this study are reported in Section S2.5 of the Supplementary Material. Figure 5 provides a comparison between `KCH` and `KMC3`. It is worth recalling that `KMC3` has been specifically designed for NGS data and highly engineered for that type of data. Indeed, it works only with short reads as input. It is clear that the advantage granted by the specialization of `KMC3` to short reads versus the generality of `KCH` disappears as the degree of parallelism and the dataset sizes increase. It is also evident that `KMC3` hardly scales with the number of threads as its execution time decreases only slightly when increasing the number of threads.

4 Conclusions

We have presented `KCH`, the first suite of *MapReduce* algorithms specifically designed for the linguistic and informational analysis of large collections of biological sequences. Its implementation on Hadoop shows that it is useful, versatile and competitive even with respect to methods that use shared-memory multi-thread architectures to collect k-mer statistics only for applications related to genome assembly. By virtue of the growing interest that the bioinformatics community is

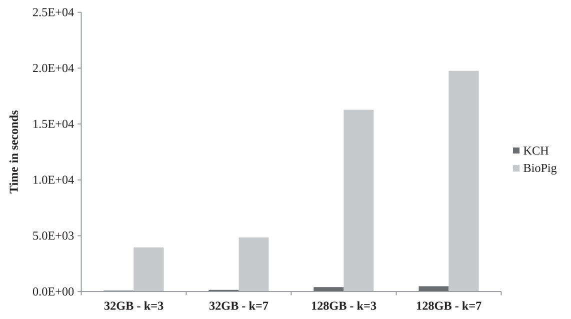

Fig. 4. Cumulative Statistics. Results of `KCH` with respect to BioPig, for the case $k = 3$, 7 (non-canonical *k*-mers), for 32 and 128 GB datasets, which are indicated in the figure according to the legend to the right

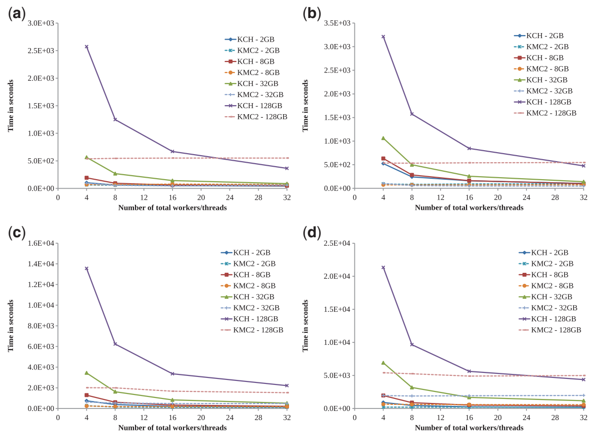

Fig. 5. Cumulative Statistics. (**a**) Scalability of `KCH` with respect to KMC3, for the case $k = 3$, for all datasets, which are indicated in the figure according to the legend to the right. The abscissa gives the number of workers/threads used, while the ordinate gives the corresponding time in seconds. (**b**)–(**d**) As in (a), but for $k = 7$, $k = 15$ and $k = 31$

giving to 'Big Data' techniques such as *MapReduce* and the centrality of the tools proposed here for alignment-free sequence analysis, `KCH` represents a much needed addition to the current set of *MapReduce* bioinformatics algorithms and tools.

Based on this success, for the future, one can envision the design of *MapReduce* and Hadoop algorithms for alignment-free sequence comparison, based on more sophisticated notions of '*k*-mers', such as spaced words and seeds [see Horwege *et al.* (2014) and Leimeister *et al.* (2017) and references therein]. Technically, the two-level aggregation strategy, with the use of bins partitioning Σ^k, characterizing `KCH` has been instrumental in its superiority with respect to `BioPig`, which follows a rather straightforward map and reduce scheme. Moreover, since there is no guarantee of balance among the various bins, the flushing strategy that has been adopted is also a key factor. Although it is not to be expected that the techniques produced here can be applied verbatim to the more general contexts outlined earlier, they provide a non-trivial path for the development of successful *MapReduce* alignment-free sequence comparison algorithms. Finally, given the growing level of attention that Spark is receiving in Bioinformatics, the further extensions of this work will have to account also for the proper choice of the middle-ware. At this stage, no obvious benefit in efficiency is seen with a Spark implementation of `KCH`.

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 for other computing resources, and the GARR Consortium for having made available a cutting edge OpenStack Virtual Datacenter for this research.

Funding

Ministry of Research PRIN Project: 2010RTFWBH_003 'Data-Centric Genomic Computing (GenData 2020)' to Giuseppe Cattaneo.
INdAM – GNCS Project 2017 'Algoritmi e tecniche efficienti per l'organizzazione, la gestione e l'analisi di Big Data in ambito biologico' to Giuseppe Cattaneo, Umberto Ferraro Petrillo, Raffaele Giancarlo, Gianluca Roscigno.

Conflict of Interest: none declared.

References

Acquisti,C. *et al.* (2007) Nullomers: really a matter of natural selection? *Plos One*, **2**, e1022.

Audano,P. and Vannberg,F. (2014) KAnalyze: a fast versatile pipelined k-mer toolkit. *Bioinformatics*, **30**, 2070–2072.

Aurell,E. *et al.* (2016) The bulk and the tail of minimal absent words in genome sequences. *Phys. Biol.*, **13**, 026004–026010.

Ben-Ari,M. (2006) *Principles of Concurrent and Distributed Programming*. Pearson Education, New Jersey, NJ, USA.

Benoit,G. *et al.* (2016) Multiple comparative metagenomics using multiset k-mer counting. *PeerJ Comput. Sci.*, **2**, e94.

Benson,D.A., *et al.* (2013) Genbank. *Nucleic Acids Res.*, **41**, D36–D42.

Bhatia,K. and Wang,Z. (2011) BioPig: developing cloud computing applications for next-generation sequence analysis. Technical report, Ernest Orlando Lawrence Berkeley National Laboratory, Berkeley, CA (US).

Birol,I. *et al.* (2013) Assembling the 20 Gb white spruce (picea glauca) genome from whole-genome shotgun sequencing data. *Bioinformatics*, **29**, 1492–1497.

Cattaneo,G. *et al.* (2017a) An effective extension of the applicability of alignment-free biological sequence comparison algorithms with Hadoop. *J. Supercomput.*, **73**, 1467–1483.

Cattaneo,G. *et al.* (2017b) Mapreduce in computational biology – a synopsis. In: Rossi, F. *et al.* (eds.) *Advances in Artificial Life, Evolutionary Computation, and Systems Chemistry: 11th Italian Workshop, WIVACE 2016, Fisciano, Italy, October 4-6, 2016, Revised Selected Papers*. Springer International Publishing, Berlin, Heidelberg, pp. 53–64.

Chor,B. *et al.* (2009) Genomic DNA k-mer spectra: models and modalities. *Genome Biol.*, **10**, R108.

Compeau,P. *et al.* (2011) How to apply de Bruijn graphs to genome assembly. *Nat. Biotechnol.*, **29**, 987–991.

Dean,J. and Ghemawat,S. (2004) MapReduce: simplified data processing on large clusters. *6th Symposium on Operating System Design and Implementation (OSDI)*. San Francisco, CA, USA, 137–150.

Denning,P.J. (1970) Virtual memory. *ACM Comput. Surv. (CSUR)*, **2**, 153–189.

Ferraro Petrillo,U. *et al.* (2017a) FASTdoop: a versatile and efficient library for the input of FASTA and FASTQ files for MapReduce Hadoop bioinformatics applications. *Bioinformatics (Oxford, England)*, **33**, 1575–1577.

Ferraro Petrillo,U. *et al.* (2017b) A new distributed alignment-free approach to compare whole proteomes. *Theoretical Computer Science.*, **698**, 100–112.

Giancarlo,R. *et al.* (2009) Textual data compression in computational biology: a synopsis. *Bioinformatics*, **25**, 1575–1586.

Giancarlo,R. *et al.* (2014) Compressive biological sequence analysis and archival in the era of high-throughput sequencing technologies. *Brief. Bioinf.*, **15**, 390–406.

Giancarlo,R. *et al.* (2015) Epigenomic k-mer dictionaries: shedding light on how sequence composition influences nucleosome positioning *in vivo*. *Bioinformatics*, **31**, 2939–2946.

Hampikian,G. and Andersen,T. (2007) Absent sequences: nullomers and primes. In: *Pac Symp Biocomput*. World Scientific, Hackensack, NJ, USA, pp. 355–366.

Horwege,S. *et al.* (2014) Spaced words and kmacs: fast alignment-free sequence comparison based on inexact word matches. *Nucleic Acids Res.*, **42**, W7–W11.

ITIS Partnership (2010) *Integrated Taxonomic Information System On-line Database*. Smithsonian Institution, Washington, DC.

Kokot,M. *et al.* (2017) KMC 3: counting and manipulating k-mer statistics. *Phys. Biol.*, **33**, 2759–2761.

Leimeister,C.-A. *et al.* (2017) Fast and accurate phylogeny reconstruction using filtered spaced-word matches. *Bioinformatics*, **33**, 971–979.

Lo Bosco,G. (2016) *Alignment Free Dissimilarities for Nucleosome Classification*. Springer International Publishing, Cham, pp. 114–128.

Marçais,G. and Kingsford,C. (2011) A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, **27**, 764–770.

Nordberg,H. *et al.* (2013) BioPig: a Hadoop-based analytic toolkit for large-scale sequence data. *Bioinformatics*, **29**, 3014–3019.

Nordstrom,K.J.V. *et al.* (2013) Mutation identification by direct comparison of whole-genome sequencing data from mutant and wild-type individuals using k-mers. *Nat. Biotechnol.*, **31**, 325–330.

Nystedt,B. *et al.* (2013) The norway spruce genome sequence and conifer genome evolution. *Nature*, **497**, 579–584.

Pinello,L. *et al.* (2011) A motif-independent metric for DNA sequence specificity. *BMC Bioinformatics*, **12**, 408.

Rahman,M.S. *et al.* (2016) Absent words and the (dis)similarity analysis of dna sequences: an experimental study. *BMC Res. Notes*, **9**, 1756.

Rizk,G. *et al.* (2013) DSK: k-mer counting with very low memory usage. *Bioinformatics*, **29**, 652–653.

Shvachko,K. *et al.* (2010) The Hadoop distributed file system. In: *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, Washington, DC, USA. IEEE Computer Society, pp. 1–10.

Siretskiy,A. *et al.* (2015) A quantitative assessment of the hadoop framework for analyzing massively parallel dna sequencing data. *GigaScience*, **4**, 26.

Utro,F. *et al.* (2016) The intrinsic combinatorial organization and information theoretic content of a sequence are correlated to the DNA encoded nucleosome organization of eukaryotic genomes. *Bioinformatics*, **32**, 835–842.

Vergni,D. *et al.* (2016) Nullomers and high order nullomers in genomic sequences. *Plos One*, **11**, e0164540.

White,T. (2015) *Hadoop: The Definitive Guide*, 4th edn. O'Reilly, Beijing.

Zaharia,M. *et al.* (2010) Spark: cluster computing with working sets. *HotCloud*, **10**, 95.

Zhou,W. *et al.* (2017) Metaspark: a spark-based distributed processing tool to recruit metagenomic reads to reference genomes. *Bioinformatics*, **33**, 1090–1092.

Zimin,A. *et al.* (2014) Sequencing and assembly of the 22-Gb loblolly pine genome. *Genetics*, **196**, 875–890.