Received July 23, 2020, accepted August 2, 2020, date of publication August 7, 2020, date of current version August 20, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3015016

SeQual: Big Data Tool to Perform Quality Control and Data Preprocessing of Large NGS Datasets

ROBERTO R. EXPÓSITO, ROI GALEGO-TORREIRO, AND JORGE GONZÁLEZ-DOMÍNGUEZ
Universidade da Coruña, CITIC, Computer Architecture Group, 15071 A Coruña, Spain

Corresponding author: Roberto R. Expósito (roberto.rey.exposito@udc.es)

This work was supported in part by the Ministry of Science and Innovation of Spain under Grant TIN2016-75845-P and Grant PID2019-104184RB-I00, in part by AEI/FEDER/EU under Grant 10.13039/501100011033, and in part by the Xunta de Galicia and FEDER funds (Centro de Investigación de Galicia accreditation 2019–2022 and the Consolidation Program of Competitive Reference Groups) under Grant ED431G 2019/01 and Grant ED431C 2017/04.

ABSTRACT This paper presents SeQual, a scalable tool to efficiently perform quality control of large genomic datasets. Our tool currently supports more than 30 different operations (e.g., filtering, trimming, formatting) that can be applied to DNA/RNA reads in FASTQ/FASTA formats to improve subsequent downstream analyses, while providing a simple and user-friendly graphical interface for non-expert users. Furthermore, SeQual takes full advantage of Big Data technologies to process massive datasets on distributed-memory systems such as clusters by relying on the open-source Apache Spark cluster computing framework. Our scalable Spark-based implementation allows to reduce the runtime from more than three hours to less than 20 minutes when processing a paired-end dataset with 251 million reads per input file on an 8-node multi-core cluster.

INDEX TERMS Big data, next-generation sequencing (NGS), bioinformatics, quality control, apache spark.

I. INTRODUCTION

The development of Next-Generation Sequencing (NGS) technologies [1], [2] has revolutionized biological research over the last decade by drastically decreasing the cost of DNA/RNA sequencing and significantly increasing the throughput of generated data. The quality of NGS data is considered very important for various downstream analyses such as gene expression studies and genome sequence assembly [3]. However, NGS platforms introduce, as a downside, different kinds of artefacts in the raw sequence fragments (the so-called ''reads'') such as duplicates, poor-quality reads and insertions/deletions, which can lead to serious negative impact on downstream analyses. Therefore, most bioinformatics pipelines start by applying a quality control over the input datasets in order to increase the accuracy of subsequent processing. Some examples of these operations are the removal of duplicate reads, the deletion of reads with low average quality, or their transformation to maintain only the fragments with high quality (trimming). Moreover, during this preprocessing step the datasets sometimes must be transformed in order to adapt them to the requirements of the pipeline. For instance, transforming the input data from FASTQ to FASTA format may be necessary if any bioinformatics application can only work with data stored in the latter format. Currently, there are several tools to perform quality control and preprocessing of raw NGS data in order to ensure the necessary quality for further processing [4], [5].

However, state-of-the-art tools still require excessive time to process the increasingly large datasets generated through mainstream NGS platforms. Although there are some parallel tools that allow to accelerate their computations on shared-memory systems thanks to including efficient multithreading support, this is not enough to complete the quality control of current large datasets in reasonable time since their scalability is limited to the resources of a single machine. In this context, the exploitation of Big Data technologies seems an adequate approach in order to accelerate those calculations on distributed-memory systems such as clusters and cloud platforms, as extensively demonstrated by the existing literature [6]–[8]. In this paper we introduce SeQual[1], a scalable tool for quality control and preprocessing of raw sequencing data implemented upon the most popular open-source distributed framework for Big Data processing:

The associate editor coordinating the review of this manuscript and approving it for publication was Juan Wang.

[1]Source code available at https://github.com/roigalegot/SeQual.

Apache Spark [9]. SeQual is mainly inspired by PRINSEQ [10], one of the most popular tools for quality control which has been widely used in many recent biological studies [11], [12]. The main advantages of PRINSEQ over alternative tools are its simplicity and great functionality, providing support not only for a wide range of quality control operations (such as filtering and trimming), but also for data formatting. Our tool also provides all this functionality (and even more) but in a significantly lower runtime by fully exploiting the parallel processing capabilities of Spark. Although there are a few parallel tools to remove duplicate DNA/RNA sequences (one specific operation that can be used for quality control) on distributed-memory systems [13], [14], up to our knowledge, SeQual is the first publicly available tool intended for this type of parallel systems that provides full functionality (more than 30 operations) instead of only allowing to remove duplicate reads. Furthermore, SeQual includes a graphical user interface intended for simplifying its usage.

The remainder of the paper is organized as follows. Section II discusses the related work. Section III describes the overall functionality provided by SeQual. Section IV describes our parallel approach. The performance of SeQual is evaluated and compared to state-of-the-art quality control tools in Section V. Finally, Section VI concludes the paper and proposes future work.

II. RELATED WORK

To address the sequencing quality problem, besides the quality control pipeline supplied by some sequencing platform manufacturers, several standalone tools have been proposed in the literature. A representative list includes tools such as FASTX-Toolkit [15], FastQC [16], PRINSEQ [10], NGS-QC [17], QC-Chain [18], FaQCs [19], Trimmomatic [20], PEAT [21], AfterQC [22], FastProNGS [23] and PRINSEQ++ [24]. With the expected increase in total generated data and decrease in costs associated with NGS technologies, one important concern is their processing speed. Some tools do not provide parallel implementations (FASTX-Toolkit, PRINSEQ), whereas others (FastQC) handle parallelism only at the file level, so they cannot accelerate the processing of a very large single dataset. The remaining tools do provide some kind of parallel support but all of them are based on multithreading, so their overall speed is limited to the computational resources of a single machine.

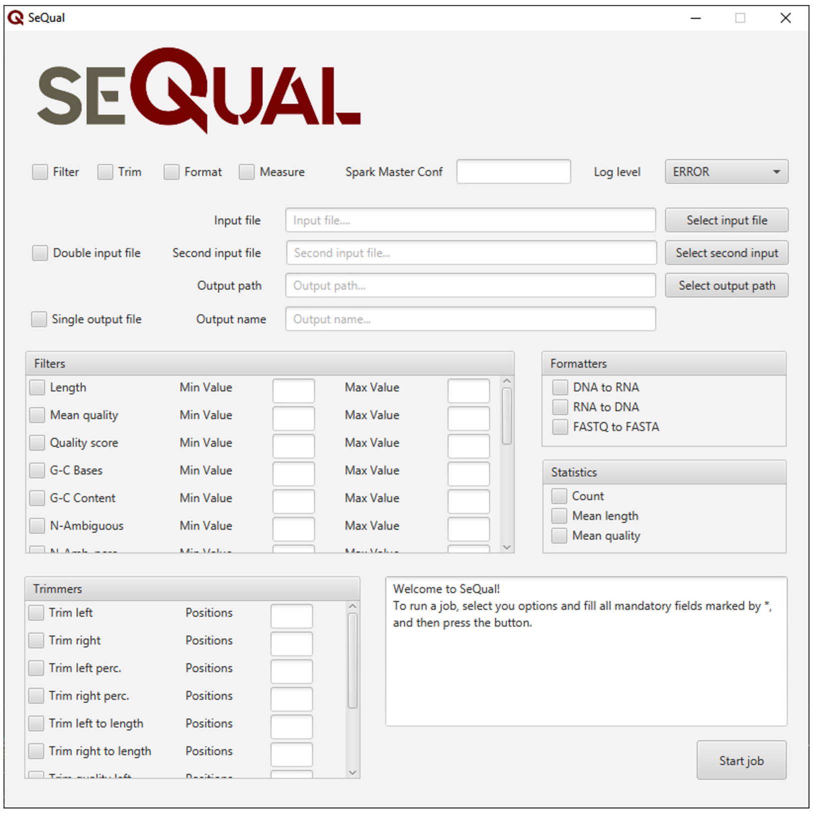
In terms of functionality, FastQC does not have trimming and filtering features, whereas Trimmomatic is focused on just one operation type (trimming), and PEAT provides very few filter options to the users. FASTX-Toolkit does not even support paired-end datasets, requiring further postprocessing to link paired reads. Other tools (FaQCs, FastProNGS) do not support FASTA as input format, while also provide basic user interfaces only limited to command-line interaction. Moreover, there are tools that just seem to be currently unavailable as their websites do not longer work (NGS-QC, QC-Chain). Among all of them, PRINSEQ is by far the solution that provides the widest functionality supporting different quality-control and preprocessing operations together with a nice web-based graphical user interface. This is the main reason why the functionality of SeQual has been based on PRINSEQ, even extending it. However, the sequential implementation of PRINSEQ using Perl clearly hinders its performance for large datasets, whereas its multithreaded C++version (PRINSEQ++) is much faster but provides less functionality than the original tool, while its scalability is still limited to a single machine.

SeQual tries to combine the functionality and usability of PRINSEQ together with the performance of PRINSEQ++ but in a distributed manner relying on Big Data technologies. In fact, the exploitation of Big Data clusters to accelerate the storage, processing and visualization of large NGS datasets has been recently explored in multiple previous works. For instance, many bioinformatics tools implemented on top of Big Data processing frameworks such as Hadoop [25] and Spark [9] have emerged in recent years, from error correction [26], [27], duplicate read removal [13] and sequence alignment [28]–[31], to variant calling [32], *de novo* genome assembly [33], [34] and protein structure prediction [35]–[37], among many others. Most of these tools are executed within a bioinformatics pipeline (or scientific workflow engines such as SAASFEE [38] or Pegasus [39]) that usually starts with a quality control of the input FASTA/FASTQ datasets. Therefore, they will benefit from SeQual in order to accelerate this first step of the pipeline, which reinforces the need of our proposal in the context of quality control and preprocessing.

III. OVERVIEW OF SeQual

SeQual is a parallel tool implemented in Java that currently provides a full set of 33 operations for performing quality control and preprocessing on raw NGS datasets. It can receive as input either single-end or paired-end DNA/RNA sequences, which can be stored either in FASTA or FASTQ files, as these are the most popular unaligned sequence formats. The operations provided by SeQual can be divided into the following four main functionalities:

1) Filters. These operations discard those input reads that do not fulfill a certain criteria specified by the user. Filters are divided into two categories, depending on the number of sequences involved in the filter rule:
   - Single filters, which evaluate reads one-by-one. SeQual includes 12 single filters. For instance, sequences can be filtered according to their length, quality or the absence/presence of a certain pattern in their bases.
   - Group filters, which compare reads by pairs and discard those that are equal (keeping the one with the highest quality score when possible). SeQual contains 5 group filters that allow, for instance, to compare the sequences as complement or reverse-complement. The user can also specify a certain number of allowed mismatches to discard those sequences that are almost equal.



Log level ERROR

Input file...

Second input file...

Output path...

Output name...

Welcome to SeQual!
To run a job, select you options and fill all mandatory fields marked by *, and then press the button.

FIGURE 1. **Graphical user interface included with SeQual.**

2) Trimmers. SeQual includes 10 operations in order to trim the beginning or ending of the sequences by removing those bases that are not interesting for the user. The user can specify the number of bases that must remain, or the quality required for the trimmed sequences.
3) Data formatters. Three functions to convert from DNA to RNA reads (and vice versa) or from FASTQ to FASTA formats are also provided by our tool.
4) Statistical operations. Finally, SeQual provides three additional functions to obtain some statistics about the initial and/or final data. For instance, these operations can be used to count the number of input sequences, or to calculate their average length/quality.

Regarding to the usage of the tool, SeQual provides two execution modes:

- Through the command-line interface by specifying: (1) the path to the dataset(s) as input arguments; (2) the operations to be performed on these datasets using a Java Properties file.
- Through a graphical interface provided by SeQual in order to simplify its usage to non-computer science experts (see Fig. 1). This graphical interface has been implemented upon the open-source JavaFX project [40], which allows built-in separation between the application logic and the visual part of SeQual.

It is worth noting that the user can apply multiple operations to the same input dataset in a single execution (see the available check boxes in Fig. 1). In this scenario, SeQual implements a priority-based strategy for all filters and trimmers to improve overall performance when multiple ones are selected by the user. Based on their priority, SeQual automatically sorts them to apply first those filters that can potentially discard more reads and those trimmers that can reduce more their length. This strategy aims to reduce overall runtime as subsequent operations can be accelerated taking advantage of this approach.

For more details about all the available operations, compilation and execution instructions, as well as a brief overview of the graphical interface, refer to the detailed README file available at the SeQual's website.

IV. IMPLEMENTATION

At the highest level of abstraction, the overall workflow of SeQual is divided into the following three main stages:

1) Reading of the input dataset(s) specified by the user, consisting of one or two FASTQ/FASTA text-based sequence files when working in single- or paired-end mode, respectively.
2) Processing of the input files according to the quality-control operations selected by the user in the graphical interface or, otherwise, specified in a Properties file when using the command-line interface.
3) Writing of the processed dataset(s) to their corresponding output text files as a result of the computations previously performed.

In order to understand how these stages have been implemented on top of Spark (Sections IV-B and IV-C), some basic

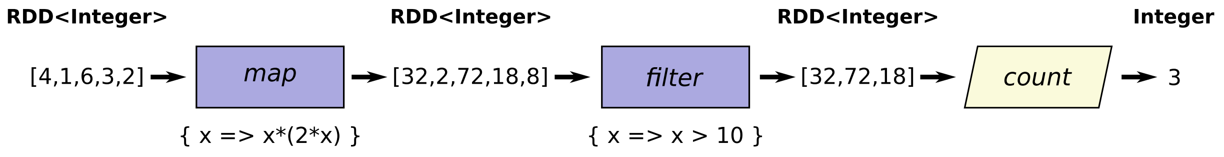


FIGURE 2. Spark example of combining *map/filter* transformations and *count* action over an RDD of type *Integer*.

```
@SRR921890.500 HAL:1217:C10NCACXX:2:2310:5797:41908 length=100
TGTTAGATTTATTTGTTTTATAATTTTTGTTAGGTTTATTGTGTTTTTGTTTAGTTTTTGTTTTTATGATTTGTATGTTGGTGAAAGTGGTGTGTTGAAG
+SRR921890.500 HAL:1217:C10NCACXX:2:2310:5797:41908 length=100
BC@FFFDFHHHHGJJEGIIJJJJJJJJJJHIIJJ?FHGIIJHHDHIJJJIJJJJFHIJJJHIJJJIIJHIJJJHFAEHEHFDFF@EECCDEBDBBDDDCC
@SRR921890.501 HAL:1217:C10NCACXX:2:2310:5850:41919 length=100
ATTTGAGTTTTGTGGTTTATTTTTTTAGGTAGTAAGAGAAAAGTTATTTTGAGAAATGTTTTTTGATTTTTATTTATGTGTTGTGACGTGTGGTTTGAGT
+SRR921890.501 HAL:1217:C10NCACXX:2:2310:5850:41919 length=100
@@@FFDD;DFACCFGHHGHHIJIIGIGII:BF0:DGGH99BDDFFIIJJIJGG@@3=C=DHGGHFFFEEEE9@>CDC@CDFDDDDDDDDDCDDDDDD>@C
```

FIGURE 3. Example of two DNA reads in FASTQ format (100 base pairs).

concepts about the programming model provided by this Big Data framework need first to be introduced (Section IV-A).

A. APACHE SPARK

Spark [9] is a popular Big Data processing framework that supports efficient in-memory computations by relying on a novel, distributed data abstraction known as Resilient Distributed Dataset (RDD) [41]. Basically, an RDD is a partitioned collection of data elements that can be distributed across the nodes of a commodity cluster. One important feature of RDDs is that their partitions can be operated in parallel and cached in memory to be reused in subsequent MapReduce-like operations [42]. A Spark programmer can create an RDD in two different ways: either by parallelizing an existing collection of objects (e.g., a list); or by loading an external dataset from a supported file system. In order to allow data processing in a distributed manner, Spark provides support for the Hadoop Distributed File System (HDFS) [43] so that RDDs can be created and efficiently processed from datasets stored in it. Nowadays, HDFS is considered the most popular open-source distributed file system for Big Data processing, providing the fundamental storage layer within the Hadoop ecosystem [25].

The RDD programming API provided by Spark supports a wide range of data-parallel operations that can be performed over an RDD. Those operations can be divided into transformations and actions. On the one hand, transformations (e.g., *map*, *filter*, *join*) create a new RDD from an existing one. For instance, a *map* transformation processes each RDD element through a user-defined function, returning a new RDD as result. Another example is *filter*, which returns a new RDD formed by selecting only those elements of the source RDD on which a user-defined function returns true. Note that transformations are lazily evaluated in Spark, so they do not compute anything until an action that requires the result from them is triggered. On the other hand, actions return non-RDD values, converting the laziness of transformations into actual computation. Actions can be used to either return a result to the main Spark program (e.g., *reduce*, *collect*, *count*), or to store an RDD in external storage after running a certain computation (e.g., *saveAsTextFile*, *saveAsObjectFile*). For instance, the *reduce* action aggregates all the RDD elements according to a user-defined function and returns the final result to the main program. As an illustrative example, Fig. 2 shows the chaining of a *map* and *filter* transformations together with a *count* action over an RDD of type *Integer*. Note that the user-defined functions executed over the input RDD are shown below the corresponding boxes for *map* and *filter* transformations.

Finally, another interesting feature of Spark is that it allows to explicitly cache or persist the RDD elements in memory, thus providing much faster access to them the next time they are queried. This is extremely useful for implementing efficient iterative algorithms [44].

B. RDD MANAGEMENT IN SeQual

All the RDD objects managed by SeQual are created from the input datasets stored in HDFS, which represents the first stage of the overall workflow previously described. The most straightforward way to create an RDD from an input text file stored in HDFS would be using the *textFile* method provided by Spark. Unfortunately, this method is not able to handle properly the specific structure of the FASTQ/FASTA text-based file formats, as both involve multiple lines per sequence (e.g., four lines for FASTQ, as shown in the example of Fig. 3). This Spark method relies by default on newline characters to identify the individual records in the input file (i.e., it creates one input record per line). Although it is possible to change the default delimiter to separate individual records according to the sequence format (e.g., FASTQ reads begin with character '@'), this solution would not work since such character can also occur in the string that represents the quality scores associated with each base (qualities are stored in the fourth line of each FASTQ read, as shown in Fig. 3).

To overcome such issues, other previous bioinformatics tools implemented using Big Data technologies [28], [45] generally perform a preprocessing of the input files to convert them into the required line-by-line format (i.e., one read per line). Next, the converted files are copied to HDFS to be processed. In the specific case of Spark, another solution is to create the RDD using the previous *textFile* method

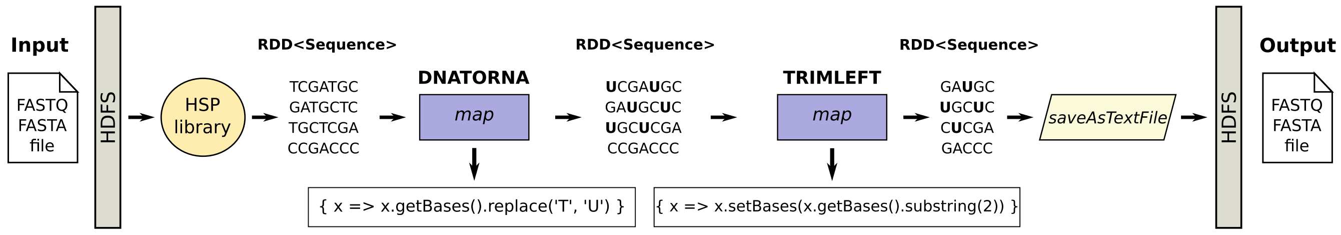


FIGURE 4. SeQual example of combining DNATORNA and TRIMLEFT operations.

and then operate over it with additional transformations and actions to obtain the desired format [29]. However, those approaches incur additional disk/memory overheads, degrading the overall performance. Instead, SeQual relies on the Hadoop Sequence Parser (HSP) library [46] to create the input RDDs in order to avoid any additional preprocessing/transformation of the input files. HSP is a Java-based library that provides specific and optimized routines to parse FASTQ/FASTA files directly from HDFS, and it is currently compatible with Hadoop, Spark and Flink [47] data processing frameworks.

Once the input RDDs are created using the HSP library (first stage), the transformations and actions provided by the Spark's API can process their partitions during the second stage according to the quality-control operations specified by the user, as will be explained in the next subsection. Finally, the RDDs resulting from performing those operations are written back to HDFS by SeQual to create the output files (third stage). In this case, Spark provides a suitable RDD action (*saveAsTextFile*) to do so straightforwardly.

C. SPARK-BASED QUALITY CONTROL AND PREPROCESSING

To efficiently implement all the functionality provided by SeQual (see Section III), each supported quality operation must be translated into the appropriate combination of transformations/actions to be performed over the input RDDs which have been previously created using the HSP library.

Regarding to single filters, these operations were implemented using an RDD *filter* transformation, as they evaluate input reads one-by-one. As mentioned before, this transformation returns a new RDD that contains only those elements of the input RDD on which a user-defined function returns true. So, the implementation of each single filter provides two functions for single- and paired-end mode, and their specific logic depends on the rule used to filter out sequences. For instance, the LENGTH filter compares the length of each read (i.e., the number of bases) with a minimum or maximum threshold specified by the user, returning false when the read must be filtered out from the resulting RDD and true otherwise.

Group filters represent a much more complex computation as input reads are compared by pairs. For instance, the DISTINCT filter requires to check all read pairs in order to remove duplicated sequences. These group filters first generate a PairRDD, which is an RDD consisting of key/value pairs as elements. To do so, these operations apply a *mapToPair* transformation to the input RDD, which is similar to *map* but it allows returning a PairRDD. The function executed by *mapToPair* outputs as key a string that represents the bases of each read for the DISTINCT filter (or the reverse, complementary or reverse complementary if the filter requires so). As value, the function outputs the sequence object itself, which contains not only the bases but also the sequence identifier and the qualities (if available). Once this PairRDD<String, Sequence> is created, a *reduceByKey* action is applied over it so that all the values (i.e., sequences) for each key are aggregated and then reduced based on a given user-defined function. The reduce function simply discards one of these similar sequences, keeping the one with the highest quality score (if available). Note that the group filters are considered network-intensive operations as the *reduceByKey* action requires to shuffle data over the network in order to aggregate all the values for the same key.

The implementation of trimmers and data formatters both rely on applying a single *map* transformation over the input RDD, performing the appropriate modifications to each read depending on the specific operation. For instance, the function executed by the *map* transformation in the case of TRIMLEFT (operation that removes a number of bases specified by the user starting from the left) modifies the string that represents the bases for each read using the *substring* Java method. Such modifications must also be performed on the string that represent the quality scores when available. An example of a data formatter is DNATORNA, whose function executed by *map* replaces each thymine base from the input DNA reads (represented by a 'T' character) by its corresponding uracil counterpart (a 'U' character) in the output RNA reads, using the *replace* method provided by Java. As a representative example, Fig. 4 shows the combination of both operations (DNATORNA and TRIMLEFT) over an input RDD containing four DNA reads.

Finally, the implementation of the different statistical operations differ greatly. The COUNT operation was straightforward to implement as it takes advantage of the *count* action provided by Spark that returns the number of RDD elements (i.e., sequences) in the dataset. However, the remaining two operations (MEANLENGTH and MEANQUALITY) require a more complex approach, being very similar for both of them. To implement those functions, the *aggregate* action was selected. This action allows operating an RDD to generate a single final result that can be of a different type than that

TABLE 1. Cluster node characteristics.

Hardware configuration	
CPU	2 × Intel Xeon E5-2660 Sandy Bridge-EP
CPU Speed/Turbo	2.20 GHz/3 GHz
#Cores	16
Cache L1/L2/L3	32 KiB/ 256 KiB / 20 MiB
Memory	64 GiB DDR3 1600 MHz
Disk	1 × 1 TiB HDD SATA3
Networks	InfiniBand FDR & Gigabit Ethernet
Software configuration	
OS version	CentOS release 7.7.1908
Kernel	3.10.0-1062.4.1.el7.x86_64
Java	Oracle JRE 1.8.0_241
Spark	2.4.4
Hadoop	2.9.2
PRINSEQ	0.20.4
PRINSEQ++	1.2
GNU compiler	8.3.0
Perl	5.16.3

of the input RDD. To do so, the *aggregate* action takes two user-defined functions as arguments. The first one operates once for each RDD element in a partition, so it is used to accumulate the results for each RDD. The second function combines all the intermediate results (one result per RDD partition) to produce the final result that is finally returned to the main program. For instance, the first function for MEANQUALITY computes the number of reads in each partition and the accumulated quality for all of them, while the second function combines all the accumulated qualities and number of reads for all the partitions. Next, the final result (i.e., the mean quality) is simply obtained by dividing the total quality score by the total number of reads.

V. PERFORMANCE EVALUATION

The correctness of the results provided by SeQual has been assessed by checking that it provides the same outputs as PRINSEQ (a widely used and tested tool) when applying identical operations over the same input datasets. Therefore, the experimental evaluation has only focused on execution time. In order to check the correctness of the statistics (not available in the state-of-the-art tools), we have compared the outputs of SeQual to the statistics provided by some text editors about the total number of lines and characters in the output files.

To evaluate the performance of SeQual, an eight-node multi-core cluster has been used for the experimental evaluation. Table 1 shows the main hardware and software characteristics of each cluster node, which mainly consists of two Intel Xeon E5-2660 octa-core Sandy Bridge-EP processors at 2.2 GHz (i.e., 16 physical cores per node), 64 GiB of memory and one local disk intended to be used for both HDFS and intermediate data storage during the execution of the experiments. The cluster nodes are interconnected through Gigabit Ethernet (1 Gbps) and InfiniBand FDR (56 Gbps). The system runs Linux CentOS release 7.7.1908 with kernel 3.10.0-1062 and the Java version

TABLE 2. Main configuration parameters of Spark and HDFS.

Spark		HDFS	
Executors per node	1	Block size	256 MiB
Executor heap size	55 GiB	Replication factor	3
Executor cores	16		

is Oracle JRE 1.8.0_241. According to these characteristics, Apache Spark version 2.4.4 was configured as shown in Table 2, which also contains the main relevant configuration parameters for HDFS (i.e., block size and replication factor). The version of Hadoop deployed in the cluster to store the input datasets in HDFS was 2.9.2. We have compared SeQual with PRINSEQ [10], one of the most popular quality control tools (see Section II), together with its multithreaded counterpart PRINSEQ++ [24], using the latest available version of both tools. PRINSEQ was executed with Perl v5.16.3, whereas PRINSEQ++ was compiled with GNU GCC v8.3.0 using the -O3 optimization flag.

Two publicly available datasets in FASTQ format obtained from the Sequence Read Archive (SRA) [48], [49] of the National Center for Biotechnology Information (NCBI) [50], [51] were used for the performance evaluation: SRR534301 and SRR567455. Table 3 shows their main characteristics. The number of reads (fourth column in the table) refers to the number of sequences per input file contained in the dataset, whereas the read length (fifth column) is expressed in terms of the number of base pairs (bp) per sequence. We have selected these datasets as they represent two different scenarios in terms of size and read lengths.

Table 4 shows the runtimes of PRINSEQ, PRINSEQ++ and SeQual when processing those datasets both in single- and paired-end modes (i.e., processing one or two input files, respectively) for the following six representative operations:

- NONIUPAC: single filter to remove those reads with one or more Non-IUPAC bases (any base other than 'A', 'T', 'G', 'C' or 'N').
- GCCONTENT: single filter to remove those reads with a percentage of Guanine ('G') and Cytosine ('C') lower or higher than a threshold specified by the user.
- DISTINCT: group filter to remove duplicate reads maintaining the ones with the highest quality.
- DNATORNA: data formatter to convert from DNA to RNA reads.
- COUNT: statistical operation to count the total number of reads in the dataset before and after performing any other operation over it.
- MEANQUALITY: statistical operation to compute the average quality of all the sequences available in the input dataset.

We have not assessed the performance of complex jobs that combine several operations in order to keep this section easy to read. Nevertheless, the improvement of SeQual over PRINSEQ and PRINSEQ++ in this type of jobs would be at least the addition of the performance improvement in the individual operations. Note also that Table 4 shows

TABLE 3. Public datasets used in the experimental evaluation.

Dataset	Tag	Organism	#Reads	Read length	Size
SRR534301	SRR53	Homo sapiens	2 × 108.7 M	101 bp	2 × 24 GiB
SRR567455	SRR56	Homo sapiens	2 × 251.9 M	76 bp	2 × 45 GiB

TABLE 4. Runtimes (in seconds) for PRINSEQ (using one core), PRINSEQ++ (using one whole node, 16 cores) and SeQual (using 16 cores in one node and 128 cores in eight nodes) when performing different operations on two different datasets in single- and paired-end modes. Operations not available in PRINSEQ and PRINSEQ++ are indicated with '–'.

Operation	Dataset	Mode	PRINSEQ	PRINSEQ++	SeQual	
					1 node	8 nodes
NONIUPAC	SRR53	Single	2,454	1,050	767	109
		Paired	5,694	2,201	2,494	410
	SRR56	Single	5,439	2,458	1,602	214
		Paired	12,321	5,262	5,409	672
GCCONTENT	SRR53	Single	2,689	588	484	104
		Paired	5,886	2,146	1,497	255
	SRR56	Single	5,391	1,211	917	130
		Paired	12,689	4,891	4,963	396
DISTINCT	SRR53	Single	2,997	1,096	764	126
		Paired	6,546	2,145	4,834	485
	SRR56	Single	6,680	1,953	1,702	171
		Paired	14,226	4,192	6,893	891
DNATORNA	SRR53	Single	2,370	-	790	86
		Paired	5,472	-	2,456	355
	SRR56	Single	5,422	-	1,473	219
		Paired	11,836	-	4,969	801
COUNT	SRR53	Single	-	-	743	177
		Paired	-	-	2,855	610
	SRR56	Single	-	-	1,762	365
		Paired	-	-	10,375	1,229
MEANQUALITY	SRR53	Single	-	-	861	189
		Paired	-	-	2,557	664
	SRR56	Single	-	-	1,524	395
		Paired	-	-	12,871	1,012

two runtime results for SeQual: using one whole node (i.e., 16 cores) and the eight nodes of the cluster (128 cores in total). PRINSEQ++ was executed on the 16 cores of one whole node, while PRINSEQ only used one core, as it is a sequential tool. Statistical operations could not be compared as they are not available neither in PRINSEQ nor in PRINSEQ++. Moreover, PRINSEQ++ does not provide the DNATORNA formatter.

As can be observed, SeQual is significantly faster than the original tool PRINSEQ in all the scenarios even using only one node. When comparing SeQual with the multithreaded version (i.e., PRINSEQ++) using the same amount of hardware resources (i.e., one whole node), SeQual is faster for half of the scenarios (it depends on the dataset and/or the operation). For instance, SeQual is faster than PRINSEQ++ for all the single-end experiments. Nevertheless, the main benefit of implementing SeQual upon a cluster computing framework such as Spark is the possibility of exploiting the performance of multiple nodes in order to reduce even more the execution time. When exploiting the whole cluster (8 nodes), SeQual is significantly faster than PRINSEQ++ for all the scenarios. More specifically, our tool is on average around 23.6 and 8.3 times faster than PRINSEQ and PRINSEQ++, respectively, providing significant speedups of up to 41.5x and 12.4x (both results achieved for the GCCONTENT filter operation when processing the SRR56 dataset). It is worth noting that the performance comparison has been limited to PRINSEQ and PRINSEQ++ as, up to our knowledge, these are the tools of the current state of the art with the widest functionality (although, as can be seen in Table 4, SeQual provides even more operations). We have not compared to other tools such as Trimmomatic [20] as the number of operations that they offer is quite limited, and therefore in our opinion their functionality is not comparable to that of SeQual or even PRINSEQ. For instance, none of the operations that have been assessed in this experimental evaluation are available in Trimmomatic.

In order to measure the scalability provided by the Spark-based implementation included in SeQual, Fig. 5 reports the speedups obtained when varying the number of nodes from one to eight. The baseline is the execution time of SeQual for each operation when using one whole node, i.e., the speedups show the acceleration obtained thanks to exploiting multiple nodes compared to using only one. As can

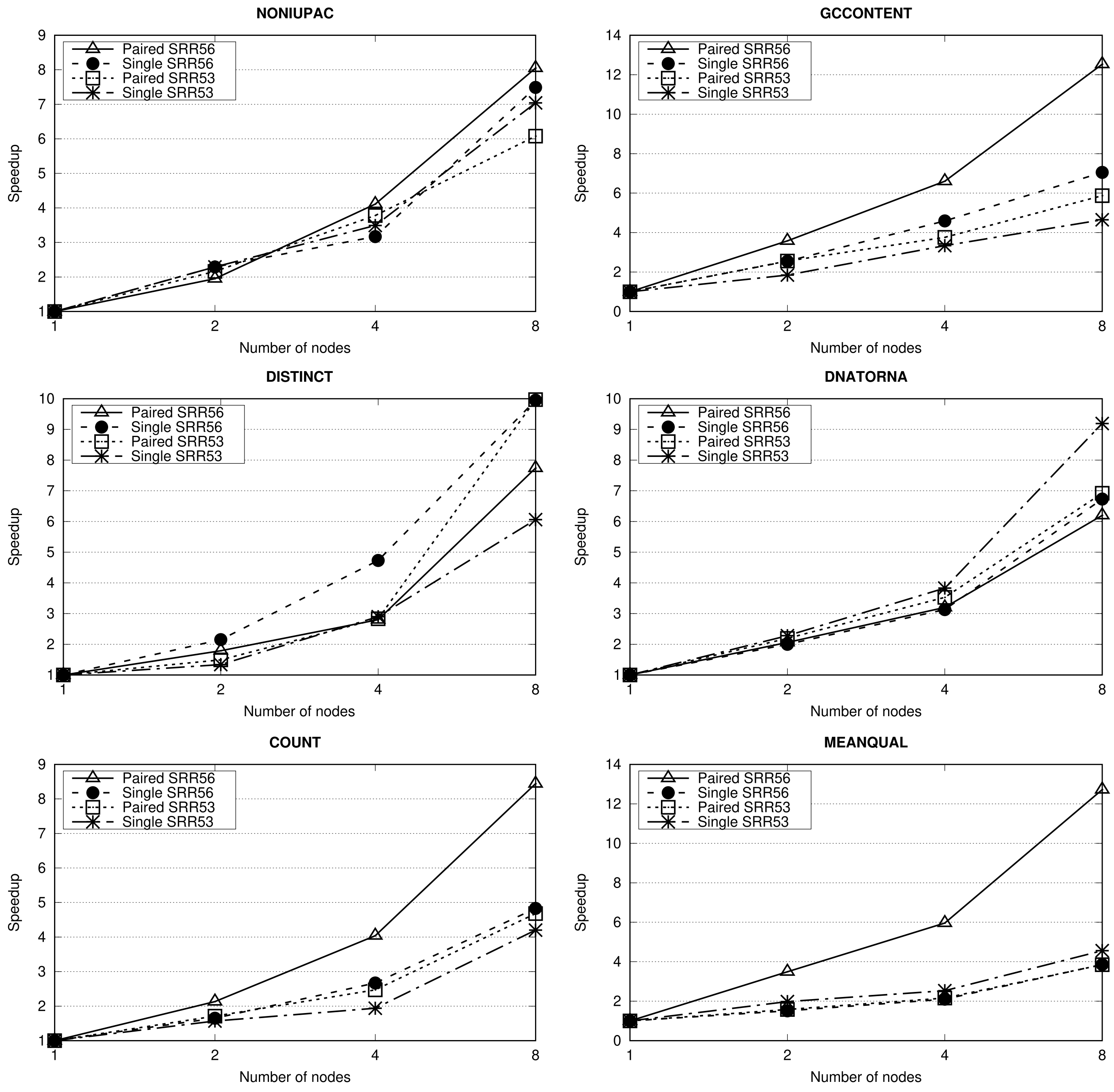


FIGURE 5. Speedups of SeQual for six different quality-control operations when varying the number of nodes using as baseline the runtime on one node.

be observed, the scalability of the tool is generally high, especially for those scenarios with a heavy computational demand (e.g., paired-end processing). In fact, the scalability of SeQual is even superlinear in some specific cases (i.e., speedups higher than 8x using eight nodes). Note that not only more cores and memory can be exploited when using more nodes but also a higher aggregated I/O bandwidth is provided by HDFS, which benefits the superlinear behaviour as the disk is the major performance bottleneck when using a single node. As a summary, the average speedups provided by SeQual for these operations when using 8 nodes are: 7.2x, 7.5x, 8.4x, 7.3x, 5.5x and 6.3x, respectively, according to their order in Fig. 5 (from top to bottom and left to right).

VI. CONCLUSION

The massive amount of data produced by modern NGS technologies reinforces the need for scalable tools with the ability to perform parallel computations by taking advantage of distributed-memory systems such as clusters and clouds. In this paper we have presented SeQual, a Big Data tool that fully exploits the features of Apache Spark to reduce the runtime needed for the quality control and preprocessing of DNA/RNA sequences. Our tool is based in PRINSEQ, one of the most widely used counterparts, and, up to our knowledge, the tool with the most extensive functionality in the state of the art. SeQual not only improves significantly the performance of PRINSEQ thanks to its Spark-based parallel implementation, but also extends its functionality with more operations. Moreover, SeQual is especially intended for clusters based on commodity processing nodes, as it does not require any specific hardware device or feature.

The performance of SeQual has been evaluated on a eight-node multi-core cluster using two publicly available datasets, both of them processed in single- and paired-end modes. The experimental results have shown that our tool provides average performance improvements of 23.6x and 8.3x over the sequential PRINSEQ tool and its multithreaded counterpart PRINSEQ++, respectively.

SeQual is distributed as free open-source software released under the GNU AGPLv3 license and is publicly available to download at https://github.com/roigalegot/SeQual. As future work, we aim to explore the streaming processing capabilities of Spark to allow SeQual to perform quality control and preprocessing operations while the input datasets are being downloaded and/or stored in HDFS. Moreover, we will try to integrate our tool with other Hadoop/Spark-based bioinformatics tools that require quality control.

REFERENCES

[1] K. A. Phillips, "Assessing the value of next-generation sequencing technologies: An introduction," *Value Health*, vol. 21, no. 9, pp. 1031–1032, Sep. 2018.

[2] W. R. McCombie, J. D. McPherson, and E. R. Mardis, "Next-generation sequencing technologies," *Cold Spring Harbor Perspect. Med.*, vol. 9, no. 11, p. a036798, 2019.

[3] C. Alkan, S. Sajjadian, and E. E. Eichler, "Limitations of next-generation genome sequence assembly," *Nature Methods*, vol. 8, no. 1, pp. 61–65, Jan. 2011.

[4] R. Bao, "Review of current methods, applications, and data management for the bioinformatics analysis of whole Exome sequencing," *Cancer Informat.*, vol. 13, no. 2, pp. 67–82, 2014.

[5] S. Pabinger, "A survey of tools for variant analysis of next-generation genome sequencing data," *Briefings Bioinf.*, vol. 15, no. 2, pp. 256–278, 2013.

[6] A. O'Driscoll, J. Daugelaite, and R. D. Sleator, "'Big Data', Hadoop and Cloud Computing in Genomics," *J. Biomed. Informat.*, vol. 46, no. 5, pp. 774–781, 2013.

[7] J. Luo, M. Wu, D. Gopukumar, and Y. Zhao, "Big data application in biomedical research and health care: A literature review," *Biomed Inf. Insights*, vol. 8, pp. 1–10, Jan. 2016,

[8] C. Smowton, A. Balla, D. Antoniades, C. Miller, G. Pallis, M. D. Dikaiakos, and W. Xing, "A cost-effective approach to improving performance of big genomic data analyses in clouds," *Future Gener. Comput. Syst.*, vol. 67, pp. 368–381, Feb. 2017.

[9] M. Zaharia, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[10] R. Schmieder and R. Edwards, "Quality control and preprocessing of metagenomic datasets," *Bioinformatics*, vol. 27, no. 6, pp. 863–864, Mar. 2011.

[11] Q. Li, "Developmental heterogeneity of microglia and brain myeloid cells revealed by deep single-cell RNA sequencing," *Neuron*, vol. 101, no. 2, pp. 207–223, 2019.

[12] M. C. de Goffau, S. Lager, U. Sovio, F. Gaccioli, E. Cook, S. J. Peacock, J. Parkhill, D. S. Charnock-Jones, and G. C. S. Smith, "Human placenta has no microbiome but can contain potential pathogens," *Nature*, vol. 572, no. 7769, pp. 329–334, Aug. 2019.

[13] R. R. Expósito, J. Veiga, J. González-Domínguez, and J. Touriño, "MarDRe: Efficient MapReduce-based removal of duplicate DNA reads in the cloud," *Bioinformatics*, vol. 33, no. 17, pp. 2762–2764, Sep. 2017.

[14] J. González-Domínguez and B. Schmidt, "ParDRe: Faster parallel duplicated reads removal tool for sequencing studies," *Bioinformatics*, vol. 32, no. 10, pp. 1562–1564, May 2016.

[15] A. Gordon and G. J. Hannon. (2010). *FASTX-Toolkit: FASTQ/A Short-Reads Pre-Processing Tools*. [Online]. Available: http://hannonlab.cshl.edu/fastx_toolkit

[16] S. Andrews. (2010). *FastQC: A Quality Control Tool for High Throughput Sequence Data*. [Online]. Available: http://www.bioinformatics.babraham.ac.uk/projects/fastqc

[17] R. K. Patel and M. Jain, "NGS QC toolkit: A toolkit for quality control of next generation sequencing data," *PLoS ONE*, vol. 7, no. 2, Feb. 2012, Art. no. e30619.

[18] Q. Zhou, X. Su, A. Wang, J. Xu, and K. Ning, "QC-chain: Fast and holistic quality control method for next-generation sequencing data," *PLoS ONE*, vol. 8, no. 4, 2013, Art. no. e60234.

[19] C.-C. Lo and P. S. G. Chain, "Rapid evaluation and quality control of next generation sequencing data with FaQCs," *BMC Bioinf.*, vol. 15, no. 1, p. 366, Dec. 2014.

[20] A. M. Bolger, M. Lohse, and B. Usadel, "Trimmomatic: A flexible trimmer for illumina sequence data," *Bioinformatics*, vol. 30, no. 15, pp. 2114–2120, Aug. 2014.

[21] Y.-L. Li, J.-C. Weng, C.-C. Hsiao, M.-T. Chou, C.-W. Tseng, and J.-H. Hung, "PEAT: An intelligent and efficient paired-end sequencing adapter trimming algorithm," *BMC Bioinf.*, vol. 16, no. 1, p. 2, 2015.

[22] S. Chen, T. Huang, Y. Zhou, Y. Han, M. Xu, and J. Gu, "AfterQC: Automatic filtering, trimming, error removing and quality control for fastq data," *BMC Bioinf.*, vol. 18, no. S3, pp. 91–100, Mar. 2017.

[23] X. Liu, Z. Yan, C. Wu, Y. Yang, X. Li, and G. Zhang, "FastProNGS: Fast preprocessing of next-generation sequencing reads," *BMC Bioinf.*, vol. 20, no. 1, p. 345, Dec. 2019.

[24] V. A. Cantu, J. Sadural, and R. Edwards, "PRINSEQ++: A multi-threaded tool for fast and efficient quality control and preprocessing of sequencing datasets," *PeerJ Preprints*, vol. 7, Feb. 2019, Art. no. e27553v1.

[25] The Apache Software Foundation. *Apache Hadoop*. Accessed: Jul. 2020. [Online]. Available: http://hadoop.apache.org

[26] W.-C. Chung, J.-M. Ho, C.-Y. Lin, and D. T. Lee, "CloudEC: A MapReduce-based algorithm for correcting errors in next-generation sequencing big data," in *Proc. IEEE Int. Conf. Big Data*, Boston, MA, USA, Dec. 2017, pp. 2836–2842.

[27] R. R. Expósito, J. González-Domínguez, and J. Touriño, "SMusket: Spark-based DNA error correction on distributed-memory systems," *Future Gener. Comput. Syst.*, vol. 111, pp. 698–713, Oct. 2020.

[28] J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo, "BigBWA: Approaching the Burrows-Wheeler Aligner to Big Data Technologies," *Bioinformatics*, vol. 31, no. 24, pp. 4003–4005, 2015.

[29] J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo, "SparkBWA: Speeding up the alignment of high-throughput DNA sequencing data," *PLoS ONE*, vol. 11, no. 5, May 2016, Art. no. e0155461.

[30] A. Nellore, "Rail-RNA: Scalable analysis of RNA-seq aplicing and coverage," *Bioinformatics*, vol. 33, no. 24, pp. 4033–4040, 2017.

[31] R. R. Expósito, J. González-Domínguez, and J. Touriño, "HSRA: Hadoop-based spliced read aligner for RNA sequencing data," *PLoS ONE*, vol. 13, no. 7, Jul. 2018, Art. no. e0201483.

[32] D. Decap, J. Reumers, C. Herzeel, P. Costanza, and J. Fostier, "Halvade-RNA: Parallel variant calling from transcriptomic data using MapReduce," *PLoS ONE*, vol. 12, no. 3, Mar. 2017, Art. no. e0174575.

[33] Y.-J. Chang, C.-C. Chen, C.-L. Chen, and J.-M. Ho, "A de novo next generation genomic sequence assembler based on string graph and MapReduce cloud computing framework," *BMC Genomics*, vol. 13, no. 7, p. 28, Dec. 2012.

[34] A. Abu-Doleh and U. V. Catalyurek, "Spaler: Spark and GraphX based de novo genome assembler," in *Proc. IEEE Int. Conf. Big Data*, Santa Clara, CA, USA, Oct. 2015, pp. 1013–1018.

[35] X. L. Dencelin and T. Ramkumar, "Analysis of multilayer perceptron machine learning approach in classifying protein secondary structures," *Biomed. Res.*, vol. 15, pp. 166–173, Feb. 2016.

[36] X. L. Dencelin and T. Ramkumar, "A distributed tree-based ensemble learning approach for efficient structure prediction of protein," *Int. J. Intell. Eng. Syst.*, vol. 10, no. 3, pp. 226–234, 2017.

[37] X. L. Dencelin and T. Ramkumar, "An approach to predict protein secondary structure using deep learning in spark based big data computing framework," in *Proc. 4th Int. Conf. Appl. Theor. Comput. Commun. Technol. (iCATccT)*, Mangalore, India Sep. 2018, pp. 25–30.

[38] M. Bux, J. Brandt, C. Lipka, K. Hakimzadeh, J. Dowling, and U. Leser, "SAASFEE: Scalable scientific workflow execution engine," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1892–1895, Aug. 2015.

[39] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, ''Pegasus, a workflow management system for science automation,'' *Future Gener. Comput. Syst.*, vol. 46, pp. 17–35, May 2015.
[40] OpenJFX. *The JavaFX Project*. Accessed: Jul. 2020. [Online]. Available: https://openjfx.io
[41] M. Zaharia, ''Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing,'' in *Proc. 9th USENIX Symp. Networked Syst. Des. Implement.*, San Jose, CA, USA, 2012, pp. 15–28.
[42] J. Dean and S. Ghemawat, ''MapReduce: Simplified data processing on large clusters,'' *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[43] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, ''The Hadoop distributed file system,'' in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, Incline Village, NV, USA, 2010, pp. 1–10.
[44] J. Shi, Y. Qiu, U. F. Minhas, L. Jiao, C. Wang, B. Reinwald, and F. Özcan, ''Clash of the titans: MapReduce vs. Spark for large scale data analytics,'' *Proc. VLDB Endowment*, vol. 8, no. 13, pp. 2110–2121, Sep. 2015.
[45] R. V. Pandey and C. Schlötterer, ''DistMap: A toolkit for distributed short read mapping on a Hadoop cluster,'' *PLoS ONE*, vol. 8, no. 8, Aug. 2013, Art. no. e72614.
[46] R. R. Expósito, L. L. Mosquera, and J. González-Domínguez. *Hadoop Sequence Parser (HSP) Library for FASTQ/FASTA Datasets*. Accessed: Jul. 2020. [Online]. Available: https://github.com/rreye/hsp
[47] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, ''Apache flink: Stream and batch processing in a single engine,'' *IEEE Data Eng. Bull.*, vol. 36, no. 4, pp. 28–38, Dec. 2015.
[48] INSDC. *The Sequence Read Archive (SRA)*. Accessed: Jul. 2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/sra
[49] Y. Kodama, M. Shumway, R. Leinonen, and O. Behalf of the International Nucleotide Sequence Database Collaboration, ''The sequence read archive: Explosive growth of sequencing data,'' *Nucleic Acids Res.*, vol. 40, no. 1, pp. 54–56, Jan. 2012.
[50] NCBI. *National Center for Biotechnology Information*. Accessed: Jul. 2020. [Online]. Available: https://www.ncbi.nlm.nih.gov
[51] D. L. Wheeler, ''Database resources of the national center for biotechnology information,'' *Nucleic Acids Res.*, vol. 33, no. 1, pp. 39–45, Dec. 2004.

ROBERTO R. EXPÓSITO received the B.S., M.S., and Ph.D. degrees in computer science from the Universidade da Coruña (UDC), Spain, in 2010, 2011, and 2014, respectively. He is currently an Assistant Professor with the Department of Computer Engineering, UDC. His main research interests include HPC and big data computing, focused on the performance optimization of distributed processing models in cluster and cloud environments, and the parallelization of bioinformatics and data mining applications.

ROI GALEGO-TORREIRO received the B.S. degree in computer science from the Universidade da Coruña (UDC), Spain, in 2019. His B.Sc. Thesis was focused on working with big data technologies within the bioinformatics field.

JORGE GONZÁLEZ-DOMÍNGUEZ received the B.S., M.S., and Ph.D. degrees in computer science from the Universidade da Coruña (UDC), Spain, in 2008, 2009, and 2013, respectively. He is currently an Assistant Professor with the Department of Computer Engineering, UDC. His main research interests include development of parallel applications on multiple fields, such as bioinformatics, data mining, and machine learning, focused on different architectures (multicore systems, GPUs, clusters, and so on).