MarDRe: efficient MapReduce-based removal of duplicate DNA reads in the cloud

Roberto R. Expósito, Jorge Veiga, Jorge González-Domínguez, Juan Touriño

Grupo de Arquitectura de Computadores, Universidade da Coruña
Campus de A Coruña, 15071 A Coruña, Spain

***Abstract*—This paper presents *MarDRe*, a *de novo* cloud-ready duplicate and near-duplicate removal tool that can process single-end and paired-end reads from FASTQ/FASTA datasets. *MarDRe* takes advantage of the widely adopted MapReduce programming model to fully exploit Big Data technologies on cloud-based infrastructures. Written in Java to maximize cross-platform compatibility, *MarDRe* is built upon the open-source Apache Hadoop project, the most popular distributed computing framework for scalable Big Data processing. On a 16-node cluster deployed on the Amazon EC2 cloud platform, *MarDRe* is up to 8.52 times faster than a representative state-of-the-art tool. Source code in Java and Hadoop as well as a user's guide are freely available under the GNU GPLv3 license at http://mardre.des.udc.es.**

I. Introduction

The unprecedented deluge of data produced by Next Generation Sequencing (NGS) platforms cannot be coped with traditional data processing systems, which has spurred the use of Big Data and cloud computing technologies [1]. On the one hand, MapReduce [2] is Google's solution for scalable Big Data analysis on commodity hardware, being Hadoop the most popular open-source implementation. Hadoop uses its own distributed file system (HDFS) to store large datasets across the locally attached disks of the computing nodes. The Hadoop scheduler tries its best to co-locate computing tasks on the nodes where the input data reside, improving data locality while minimizing data movements across the network. This data-parallel model differs widely from that of traditional High Performance Computing (HPC), which generally relies on the Message Passing Interface (MPI) and the availability of network/parallel file systems (e.g., Lustre), where data are distributed from dedicated storage nodes to computing nodes over the network. Furthermore, Hadoop provides built-in fault-tolerance capabilities, while MPI cannot deal with node failures. On the other hand, cloud computing allows users to hire infrastructure over the Internet on a pay-as-you-go basis, thereby avoiding huge capital investments and maintenance costs. Public cloud providers are proving very popular for Big Data analysis by offering easy-to-use cloud services that enable to set up elastic virtual clusters to exploit supercomputing-level power. Deployment of a Hadoop cluster in the cloud has gained increasing attention in recent years as a convenient, cost-effective and scalable way to store and analyze biological data [3].

Given the rapidly increasing size of NGS datasets, preprocessing is often required to either reduce their sizes or ensure the necessary data quality for further analysis. One preprocessing step is the removal of duplicate DNA reads that are introduced, for instance, due to PCR amplification [4], being the *de novo* strategy the preferred one when a complete reference genome is not available for mapping-based tools [5]. However, existing *de novo* removal tools that can be deployed on distributed systems do not fully exploit Big Data and cloud computing technologies. *ParDRe* [6] is a hybrid MPI/multithreaded tool intended for HPC systems. Nevertheless, its performance on cloud platforms is heavily limited by its poor data access efficiency, caused by the limited network bandwidth and the unavailability of high-performance file systems in the cloud. *Fulcrum* [7] is a Python-based tool that provides two distributed modes: (1) local-network mode, which uses the parallel Python library; and (2) MapReduce mode, which uses HiveQL and Python over Hadoop streaming. However, the MapReduce mode is not publicly available in the bundle distribution, which lacks the required source files. Furthermore, Fulcrum only supports FASTQ datasets, and its performace has proved to be significantly worse than *ParDRe* according to the experimental evaluation of [6], mainly due to its inefficient sequential way of grouping similar sequences.

This paper presents *MarDRe*, a *de novo* MapReduce-based parallel tool to remove duplicate and near-duplicate reads through the clustering of single-end and paired-end sequences from FASTQ/FASTA datasets. *MarDRe* can be considered the Big Data counterpart of *ParDRe*, but significantly improving its performance on distributed systems, especially on cloud-based infrastructures.

II. Implementation

MarDRe is a Java-based tool that implements a prefix-suffix approach [7] which considers as potentially duplicate reads those with an identical prefix. Once the reads have been clustered according to their prefixes, their suffixes are compared. This prefix-clustering approach is conceptually well suited for MapReduce-style chunk processing, as each cluster can be generated in parallel during the map phase, while those reads in the same cluster can be compared during the reduce phase.

TABLE I
RUNTIME AND ACCURACY RESULTS FOR *ParDRe* AND *MarDRe* REMOVING NEAR-DUPLICATE READS ON A 16-NODE AMAZON EC2 CLUSTER. SPEEDUPS SHOWN ARE THE *MarDRe* RUNTIMES OVER THE *ParDRe* ONES.

Prefix Length	#Mismatches	Single-end: SRR377645 (213 million 100-bp reads)					Paired-end: SRR948355 (69 million 202-bp reads)				
		ParDRe		MarDRe		Speedup	ParDRe		MarDRe		Speedup
		Runtime	%Removed	Runtime	%Removed		Runtime	%Removed	Runtime	%Removed	
15	1	2320 sec	8.35%	440 sec	8.35%	5.27	1300 sec	8.55%	157 sec	8.44%	8.28
15	3	2508 sec	11.61%	511 sec	11.60%	4.91	1283 sec	10.37%	155 sec	10.28%	8.28
25	1	2160 sec	8.15%	272 sec	8.14%	7.94	1301 sec	8.49%	153 sec	8.38%	8.50
25	3	2097 sec	10.90%	292 sec	10.90%	7.18	1286 sec	10.25%	151 sec	10.16%	8.52

MapReduce jobs typically process data chunks in line-based text formats, where identifying individual records is simple as line boundaries are denoted by newline characters. However, FASTQ/FASTA are text-based formats that involve multiple lines per sequence. Therefore, *MarDRe* implements custom Hadoop input formats and record readers in order to properly parse the reads from those widely adopted sequence formats.

A. Single-end mode

This mode has been implemented using one MapReduce job followed by a copy-merge operation to provide a single output file. The input dataset is first partitioned into a number of HDFS blocks, with each map task operating on a single block at a time. During the map phase, mappers process in parallel their corresponding input blocks and emit key-value pairs where the value is the parsed read and the key is generated by the first l encoded bases, being l the prefix length specified by the user in the command line. The clustering itself is naturally performed by the underlying grouping-by-key operation of the MapReduce pipeline, where the key-value pairs are first partitioned across the available reducers according to their keys (i.e., their prefixes), and then they are sorted by key within each partition (i.e., within each cluster). In the reduce phase, each reducer is in charge of computing different clusters according to the default Hadoop hash-based partitioner. For each cluster, reducers take the first read as a seed and compare its suffix with that of the other reads, computing the number of mismatches (i.e., the distance) for each one. Next, only those reads whose distance difference is less or equal than m are actually compared, being m the number of allowed mismatches specified by the user. This comparison step has been optimized by using a 4-bit encoding for the suffix bases and a bitwise XOR operation to avoid base-per-base comparisons, just as done in *ParDRe*. Finally, non-duplicate reads are written to HDFS (one output file per reducer).

After the MapReduce job has finished, an HDFS-level operation is performed to merge all the intermediate output files into the final output. This step can be disabled via a configurable option, which can be useful for subsequent data processing (e.g., sequence alignment) on HDFS.

B. Paired-end mode

This mode requires two input datasets with a one-to-one mapping between the forward (or "left") and reverse ("right") reads of each sequence. Before paired-end reads can be clustered, both reads must first be joined, which involves chaining two MapReduce jobs. The first job performs this join-like operation by parsing both input files as separate single-end datasets. Thus, mappers emit each forward/reverse read as value and its starting position in the input file (i.e., the offset) as key. In this way, the two reads of each sequence are sent to the same reducer, which outputs key-value pairs to HDFS, where the value consists of both reads and the key is the prefix of the "left" read.

The second job carries out the duplicate removal, being similar to that of the single-end mode but taking as input the output files of the former job. First, paired-end reads are parsed from HDFS during the map phase, emitting the prefix of the "left" read as key to perform the clustering. Next, reducers compare the paired-end reads that belong to the same cluster in a similar way as before, but taking into account both ends of each read.

III. PERFORMANCE EVALUATION

The experiments have been carried out on the Amazon EC2 cloud using a 16-node virtual cluster based on the c3.8xlarge instance type. Each instance provides 32 cores, 60 GB of memory, 2 local SSD disks and 10 Gigabit Ethernet network. The Linux distribution selected for the performance evaluation was Amazon Linux 2016.09 with kernel 4.4.51. *ParDRe* v1.3.5 was compiled with GNU v4.8.3 (-O3 flag) and Open MPI v1.10.5, using the hybrid MPI/multithreaded mode with the best combination of processes and threads. The network file system for *ParDRe* was Amazon Elastic File System (EFS), which provides EC2 instances with shared, low-latency access to an NFSv4-like storage system. Regarding *MarDRe*, Hadoop v2.7.3 was used. The number of map and reduce slots on each node was set to the number of cores, which is a common setting for Hadoop clusters. The HDFS block size was set to 512 MB while the Java environment used was OpenJDK v1.8.0_121.

Two different sets of experiments have been conducted on the Amazon EC2 cloud. On the one hand, the runtime and accuracy (i.e., the percentage of removed reads) of the tools have been analyzed on the 16-node EC2 cluster (Section III-A). These experiments have evaluated both single-end and paired-end modes using FASTQ datasets while varying the prefix length (l) and the number of allowed mismatches (m). On the other hand, the second set of experiments consists of analyzing the strong scalability of the tools (Section III-B). For doing so, the paired-end mode has been evaluated using 1, 2, 4,

TABLE II
STRONG SCALABILITY RESULTS FOR *ParDRe* AND *MarDRe* REMOVING NEAR-DUPLICATE READS OF A PAIRED-END DATASET AND ALLOWING ONE MISMATCH ON AMAZON EC2. SPEEDUPS SHOWN ARE THE *MarDRe* RUNTIMES OVER THE *ParDRe* ONES.

#Nodes	#Cores	Paired-end: SRR948355 (69 million 202-bp reads)								
		Prefix Length = 10			Prefix Length = 15			Prefix Length = 25		
		ParDRe	MarDRe	Speedup	ParDRe	MarDRe	Speedup	ParDRe	MarDRe	Speedup
1	32	1014 sec	1435 sec	0.71	1015 sec	1179 sec	0.86	1008 sec	1041 sec	0.97
2	64	840 sec	782 sec	1.07	845 sec	686 sec	1.23	831 sec	653 sec	1.27
4	128	901 sec	511 sec	1.76	911 sec	524 sec	1.74	915 sec	478 sec	1.91
8	256	1126 sec	267 sec	4.22	1120 sec	234 sec	4.79	1115 sec	233 sec	4.79
16	512	1306 sec	169 sec	7.73	1300 sec	157 sec	8.28	1301 sec	153 sec	8.50

8 and 16 nodes (i.e., from 32 up to 512 cores). Scalability results are shown for three different prefix length values while allowing only one mismatch. Finally, the reported runtimes for both tools are the mean value of 10 executions for each experiment.

A. Runtime and accuracy results

Table I summarizes the runtime to remove near-duplicate reads for single-end and paired-end modes using four different configurations, also reporting the percentage of removed reads. As can be observed, there are negligible differences in levels of removed duplicates, caused by the different order when comparing the reads within the cluster, as Hadoop always sorts key-value pairs by key after the map phase. Regarding execution times, the results show that *MarDRe* clearly outperforms *ParDRe*, being the average speedups 6.33 and 8.40 for single-end and paired-end modes, respectively. As mentioned in Section I, the main reason is the poor I/O efficiency of *ParDRe* due to the limited network bandwidth in a virtualized cloud environment. In *ParDRe*, all processes read the input files completely, discarding those reads that do not belong to their corresponding clusters. This causes high network overhead and heavy EFS contention. In *MarDRe*, mappers only parse their corresponding HDFS blocks that are generally stored on local disks, which provides better data locality and avoids contention for shared storage resources.

B. Scalability results

Table II shows the strong scalability results removing near-duplicate reads of a paired-end dataset using three different prefix length values and allowing one mismatch. As can be seen, *ParDRe* is the fastest tool on 1 node, especially for the lowest prefix length value, while *MarDRe* outperforms *ParDRe* from 2 nodes on. In fact, *ParDRe* is not able to scale from this point, obtaining roughly similar runtimes on two nodes (64 cores) as on four (128 cores). Moreover, *ParDRe* runtimes are worse on eight and sixteen nodes than on one node. However, *MarDRe* is able to scale reasonably well using all the available cores. More specifically, the average speedup for *MarDRe* when using 16 nodes vs 1 is around 7.60. Furthermore, *MarDRe* runtimes using 16 nodes significantly outperform the best results for *ParDRe* (i.e., using 2 nodes). In this case, the average speedup (i.e., *MarDRe*-16 nodes vs *ParDRe*-2 nodes) is 5.26. The main conclusion that can be drawn from these results is that while *ParDRe* can be considered the fastest tool for multicore systems, *MarDRe* would be the preferred choice on distributed ones.

ACKNOWLEDGMENT

This work was supported by the Ministry of Economy and Competitiveness of Spain [TIN2016-75845-P (AEI/FEDER, UE)]; and by the FPU Program of the Ministry of Education of Spain [FPU014/02805].

REFERENCES

[1] A. O'Driscoll, J. Daugelaite, and R. D. Sleator, "'Big data', Hadoop and cloud computing in genomics," *Journal of Biomedical Informatics*, vol. 46, no. 5, pp. 774–781, 2013.

[2] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[3] Q. Zou *et al.*, "Survey of MapReduce frame operation in bioinformatics," *Briefings in Bioinformatics*, vol. 15, no. 4, pp. 637–647, 2013.

[4] M. T. Ebbert *et al.*, "Evaluating the necessity of PCR duplicate removal from next-generation sequencing data and a comparison of approaches," *BMC Bioinformatics*, vol. 17, no. 7, p. 239, 2016.

[5] L. Pireddu, S. Leo, and G. Zanetti, "SEAL: a distributed short read mapping and duplicate removal tool," *Bioinformatics*, vol. 27, no. 15, pp. 2159–2160, 2011.

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

[7] M. S. Burriesci, E. M. Lehnert, and J. R. Pringle, "Fulcrum: condensing redundant reads from high-throughput sequencing studies," *Bioinformatics*, vol. 28, no. 10, pp. 1324–1327, 2012.