UNIVERSIDADE DA CORUÑA

COMPUTER ARCHITECTURE GROUP
UNIVERSITY OF A CORUÑA
GAC.UDC.ES

# MarDRe: User's Guide

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

January 23, 2019

# Contents

**1 Introduction** **3**

1.1 Citation . . . 3

**2 Prerequisites** **3**

**3 Execution** **4**

3.1 Command-line arguments . . . 4

3.2 Examples . . . 5

3.3 Compression support . . . 5

3.4 Configuration . . . 6

**4 Compilation** **7**

**5 Contact** **7**

# 1 Introduction

MarDRe [1, 2] is a *de novo* MapReduce-based parallel tool to remove duplicate and near-duplicate DNA reads through the clustering of single-end or paired-end sequences from FASTQ/FASTA datasets. Duplicate reads can be seen as identical or nearly identical sequences with some mismatches. Depending on the application scenario, duplicate or near-duplicate reads do not provide any interesting biological information but can increase memory requirements and computational time of downstream analysis. This tool allows researchers and bioinformatics to avoid the analysis of not necessary reads, reducing the time of subsequent procedures with the dataset (e.g., assemblies, mappings, etc.).

MarDRe is the Big Data counterpart of ParDRe [3, 4], which employs HPC technologies (i.e., hybrid MPI/multithreading) to reduce runtime on multicore systems. Instead, MarDRe takes advantage of the MapReduce programming model originally developed by Google [5] to significantly improve ParDRe performance on distributed systems, specially on cloud-based infrastructures. Written in pure Java to maximize cross-platform compatibility, MarDRe is built upon the open-source Apache Hadoop project [6], the most popular distributed computing framework for scalable Big Data processing.

This tool is distributed as free software and is publicly available at [2] under the GPLv3 license [7].

## 1.1 Citation

If you have used MarDRe in your research, please cite our work using reference [1].

# 2 Prerequisites

In order to use MarDRe, the prerequisites are:

1. Make sure you have Java Runtime Environment (JRE) version 1.7 or above
2. Make sure you have a working Hadoop distribution version 2.2 or above [8]
   - HADOOP_HOME environmental variable must be set accordingly
3. Untar the downloaded MarDRe distribution
   - On Linux/Mac OS, just follow the instructions below:

   ```
   [user@host ~]$ tar xzf MarDRe-v1.4.tar.gz
   ```

   - Alternatively use your preferred archive extraction tool
4. Set MARDRE_HOME and PATH environmental variables
   - On Linux/Mac OS you can set them in your profile or your shell configuration files (e.g., .bashrc). Follow the instructions below:

   ```
   [user@host ~]$ export MARDRE_HOME=/path/to/mardre
   [user@host ~]$ export PATH=$MARDRE_HOME/bin:$PATH
   ```

# 3 Execution

MarDRe can be executed by using the provided *mardrerun* command, which launches the MapReduce jobs to the Hadoop cluster. The only compulsory argument is the input dataset for single-end executions, or two datasets in case of paired-end executions.

The input FASTQ/FASTA datasets must be stored in the Hadoop Distributed File System (HDFS) [9]. MarDRe versions greater than 1.1 support compressed datasets (see Section 3.3 for more information about this topic). Versions greater than 1.2 use the Hadoop Sequence Parser (HSP) library [10] to read the input datasets for improved overall performance.

## 3.1 Command-line arguments

The available command-line arguments are:

- -i. Compulsory both in single-end and paired-end scenarios. String with the HDFS path to the input sequence file in FASTA/FASTQ format.

- -p. Compulsory only in paired-end scenarios. String with the HDFS path to the second input sequence file in FASTA/FASTQ format.

- -o. Optional. String with the file name for the first output file in FASTA/FASTQ format. The default value is the same as the first input file name followed by `.NonDup`.

- -r. Optional. String with the file name for the second output file in FASTA/FASTQ format in paired-end scenarios. The default value is the same as the second input file name followed by `.NonDup`.

- -q. Specify that input sequence files are in FASTQ format. By default, MarDRe tries to autodetect the input format, but if input files are compressed the user must specify the appropriate argument.

- -f. Specify that input sequence files are in FASTA format. By default, MarDRe tries to autodetect the input format, but if input files are compressed the user must specify the appropriate argument.

- -m. Optional. Integer with the number of allowed mismatches to identify two reads as equivalent. The default value is 0.

- -l. Optional. Integer with the prefix length. During the mapping phase, MarDRe emits as key the first $l$ encoded bases of each read. Then, reads of the same key are compared in the reducing phase. Higher prefix-length usually leads to shorter computation but can miss some duplicates. The loss of accuracy is observed when removing near-duplicate reads. Reads in the same group have exactly the same prefix (i.e., mismatches are not allowed in the prefix). The longer the prefix, more near-duplicate can be missed. Lets use as example an scenario where we try to compress near-duplicate reads with up to one mismatch. If we use a prefix of length 20 to compare two reads that only have one mismatch in position 16, MarDRe stores them in different groups, they are never compared, and both reads will be in the output. Otherwise, with a prefix of length 15 MarDRe compares them and, as only one base is different, discards one of them. The default value is 20.

- -c. Optional. Integer with the number of bases to compare for each read (starting from the beginning of the sequence). It must be equal or greater than the prefix length and equal or less than the sequence length. The default value is equal to the sequence length (i.e., all bases are compared).

- -nr. Optional. Integer with the number of reducers. The default value is 1.

- -v. Print out the version of the program and exit.

- -h. Print out the usage of the program and exit.

## 3.2 Examples

The following command removes the duplicates of identical reads (no mismatches) of a single-end dataset using a prefix length of 15 and 8 reducers:

```
[user@host ~]$ mardrerun -i dataset.fastq -l 15 -nr 8
```

The following command shows a similar example but for paired-end execution, allowing two mismatches and using the default prefix length (20) and 8 reducers:

```
[user@host ~]$ mardrerun -i dataset1.fastq -p dataset2.fastq -m 2
    -nr 8
```

## 3.3 Compression support

MarDRe supports the processing of input datasets compressed with Gzip (i.e., .gz extension) and BZip2 (i.e., .bz2 extension) codecs. However, when considering compressed data that will be processed by Hadoop, it is important to understand whether the underlying compression format supports splitting, as many codecs need the whole input stream to uncompress successfully.

On the one hand, it is impossible to start reading at an arbitrary point in a Gzip file and therefore impossible for a map task to read its input split independently of the others. For this reason, Gzip does not support splitting and Hadoop will not split the gzipped input dataset. This will work, but at the expense of performance: a single map will parse the whole input dataset, which prevents parallelism during the parsing of the reads during the map phase, while the duplicate removal step can still be performed in parallel during the reduce phase. In terms of performance, it would be probably better to first uncompress the input dataset before storing it in HDFS. On the other hand, BZip2 does compression on blocks of data and later these compressed blocks can be decompressed independent of each other, so it does support splitting. Therefore, BZip2 is the recommended codec to use with Hadoop for best performance and parallelism. Note that if you are using compressed input datasets, MarDRe will also compress the output datasets using the same codec.

Finally, even if you are using uncompressed input datasets, Hadoop may benefit from compressing the intermediate output of the map phase in both single-end and paired-end modes. Since the map output is written to disk and transferred across the network to the reducer nodes, by using a fast compressor such as snappy, you may get performance gains simply because the volume of network data to transfer is reduced. Moreover, the output files of the intermediate MapReduce jobs needed in paired-end mode when using a reduce-side join (see next section) can also be compressed, thus reducing disk I/O overhead between jobs. MarDRe supports both types of intermediate compression using the snappy codec, which can be configured by setting COMPRESS_MAP_OUTPUT and COMPRESS_INTERMEDIATE_OUTPUT options, as shown next.

## 3.4 Configuration

MarDRe can be configured by means of the *mardre.conf* file located at the *etc* directory. The available parameters are:

- MERGE_OUTPUT (boolean). Merge output files into a single file stored in HDFS. The default value is false.
- DELETE_TEMP (boolean). Delete intermediate files created by Hadoop (if any). The default value is true.
- PAIRED_END_MAP_JOIN (boolean). Enable the use of a map-side join in paired-end scenarios when set to true, otherwise a reduce-side join is performed. The default value is true, which usually leads to a better performance.
- COMPRESS_MAP_OUTPUT (boolean). Enable the compression of the intermediate map output using the snappy codec. It requires the snappy library installed on the system and the Hadoop native library (i.e., libhadoop) compiled with snappy support. The default value is false.
- COMPRESS_INTERMEDIATE_OUTPUT (boolean). Enable the snappy compression of the intermediate output files in paired-end scenarios using a reduce-side join (the same requirements apply as before). The default value is false.
- INPUT_BUFFER_SIZE (integer). The buffer size in bytes that is used for input read operations. It should probably be a multiple of the hardware page size (e.g., 4096). The default value is 65536 bytes.
- IN_MAPPER_COMBINER (boolean). Enable the use of the in-mapper combiner when possible. The default value is false.
- IN_MAPPER_COMBINER_CACHE_SIZE (integer). Maximum number of entries cached by the in-mapper combiner. The default value is 65536 entries.
- CLUSTER_LIST_INITIAL_CAPACITY (integer). Initial capacity of the ArrayList used to store reads during duplicate removal. The default value is 65536 entries.
- HDFS_BASE_PATH (string). Base path on HDFS where MarDRe stores the output files as well as temporary intermediate files. The user running MarDRe must have write permissions on this path. The default value is blank, which means to use the HDFS user's home directory.
- HDFS_BLOCK_REPLICATION (short). HDFS block replication factor for output sequence files. The default value is 1.

# 4 Compilation

In case you need to recompile the MarDRe distribution, the prerequisites are:

1. Make sure you have Java Development Kit (JDK) version 1.7 or above
2. Make sure you have a working Apache Maven distribution version 3 or above
    - https://maven.apache.org/install.html

In order to build the JAR distribution, just execute the following Maven command from within the MarDRe root directory:

```
[user@host mardre]$ mvn package
```

The first time you execute this command, Maven will download all the plugins and related dependencies it needs to fulfill the command. From a clean installation of Maven, this can take quite a while. If you execute the command again, Maven will now have what it needs, so it will be able to execute the command much more quickly.

# 5 Contact

MarDRe has been developed in the Computer Architecture Group [11] at the University of A Coruña [12] by the following authors:

- Roberto R. Expósito: http://gac.udc.es/~rreye
- Jorge Veiga: http://gac.udc.es/~jveiga
- Jorge González-Domínguez: http://gac.udc.es/~jgonzalezd
- Juan Touriño: http://gac.udc.es/~juan

To report any question, bug, requirement or information about MarDRe, feel free to contact us at [2].

# References

[1] Roberto R. Expósito, Jorge Veiga, Jorge González-Domínguez, and Juan Touriño. MarDRe: efficient MapReduce-based removal of duplicate DNA reads in the cloud. Bioinformatics. *Bioinformatics*, 33(17):2762–2764, 2017.

[2] MarDRe webpage. http://mardre.des.udc.es.

[3] Jorge González-Domínguez and Bertil Schmidt. ParDRe: faster parallel duplicated reads removal tool for sequencing studies. *Bioinformatics*, 32(10):1562–1564, 2016.

[4] ParDRe webpage. https://sourceforge.net/projects/pardre.

[5] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[6] Apache Hadoop. http://hadoop.apache.org.

[7] GNU General Public License version 3 (GPLv3). https://www.gnu.org/licenses/gpl-3.0.en.html.

[8] Apache Hadoop Cluster Setup. http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ClusterSetup.html.

[9] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop distributed file system. In *Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST'10)*, pages 1–10, Incline Village, NV, USA, 2010.

[10] HSP: Hadoop Sequence Parser library for FASTQ/FASTA datasets. https://github.com/rreye/hsp.

[11] Computer Architecture Group. http://gac.udc.es/english.

[12] University of A Coruña. http://www.udc.gal/index.html?language=en.