SparkFlow: Towards High-Performance Data Analytics for Spark-based Genome Analysis

Rosa Filgueira
School of Computer Science
University of St. Andrews
St Andrews, UK
rf208@st-andrews.ac.uk

Feras M. Awaysheh
Institute of Computer Science, Delta center
University of Tartu
Tartu, Estonia
feras.awaysheh@ut.ee

Adam Carter and Darren J. White
Edinburgh Parallel Computing Centre, EPCC
University of Edinburgh
Edinburgh, UK
{a.carter, d.white}@epcc.ed.ac.uk

Omer Rana
Physical Sciences and Engineering College
Cardiff University, UK
Cardiff, UK
ranaof@cardiff.ac.uk

***Abstract*—The recent advances in DNA sequencing technology triggered next-generation sequencing (NGS) research in full scale. Big Data (BD) is becoming the main driver in analyzing these large-scale bioinformatic data. However, this complicated process has become the system bottleneck, requiring an amalgamation of scalable approaches to deliver the needed performance and hide the deployment complexity. Utilizing cutting-edge scientific workflows can robustly address these challenges. This paper presents a Spark-based alignment workflow called SparkFlow for massive NGS analysis over singularity containers. SparkFlow is highly scalable, reproducible, and capable of parallelizing computation by utilizing data-level parallelism and load balancing techniques in HPC and Cloud environments. The proposed workflow capitalizes on benchmarking two state-of-art NGS workflows, i.e., `BaseRecalibrator` and `ApplyBQSR`. SparkFlow realizes the ability to accelerate large-scale cancer genomic analysis by scaling vertically (`HyperThreading`) and horizontally (provisions on-demand). Our result demonstrates a trade-off inevitably between the targeted applications and processor architecture. SparkFlow achieves a decisive improvement in NGS computation performance, throughput, and scalability while maintaining deployment complexity. The paper's findings aim to pave the way for a wide range of revolutionary enhancements and future trends within the High-performance Data Analytics (HPDA) genome analysis realm.**

***Index Terms*—Big Data, High-performance Data Analytics, Apache Spark, Genome Analysis, HPC, Scientific Workflow**

I. Introduction

Given the projected increase in the use of genome analysis in medical practice, using traditional data analysis methods may not be sufficient and scalable enough to meet the modern requirements [4]. Big Data (BD) analytics must be developed for genomics to automatically handle the massive amount of data robustly, with speed and flexibility [1]. The Burrows-Wheeler aligner (BWA) alignment tool is the most accurate method for searching biological sequence reads to a large reference genome. Unfortunately, BWA stages are computationally demanding and require a series of quality control stages (pre-process) under the exponential growth of bio detests. There is an imperative necessity for integrative approaches of the BD realm to manage these large-scale genomics data. However, this demands the progression of robust processing workflows to ensure that the alignment process is of the highest quality, speed, and accuracy. In this context, a workflow is a sequence of operations (by different tools) that controls the process using a series of steps via software.

Selecting the most suitable collection of tools and technologies influences the NGS workflow performance significantly. This factor can be decisive in the widespread adoption and acceptance of such a workflow. BD pipelines aids in the automation of the genome analysis requirements using cutting-edge management and orchestration platforms. Such an effort aims to manage the execution of sequence aligners data by taking advantage of the BD parallel features with high performance and accuracy [5]. Apache Spark [11], the BD de-facto clustering engine, achieves boost performance by promoting in-memory data execution using its Resilient Distributed Datasets (RDDs) abstraction [43]. Its architecture offers easy-to-use APIs that are required when dealing with large-scale volumes of data. Several trending technologies have been investigated and examined to address this challenge. For instance, BD processing engines (i.e., Apache Spark), underlying resources (CPU threading, GPU accelerators, amount of memory, etc.), containerization technology (i.e., Singularity), and different deployment architectures support high-throughput infrastructures.

A new paradigm that compiles high-performance computing and big data analytics in bio-discovery is spearheaded known as High-performance Data Analytics (HPDA) [6], [7]. In this work, we have focused on improving the ease of use and scalability of the genomic cancer analysis by providing a new container-enabled workflow for fully automated

The work by Dr. Feras M. Awaysheh is funded by the European Regional Development Funds via the Mobilitas Plus programme (grant MOBTT75).

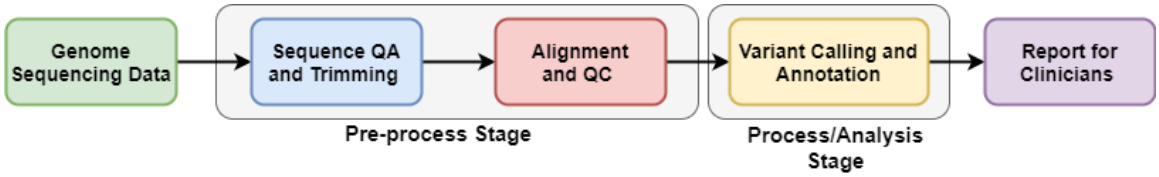


Fig. 1: Overview on the main stages of cancer analysis pipeline.

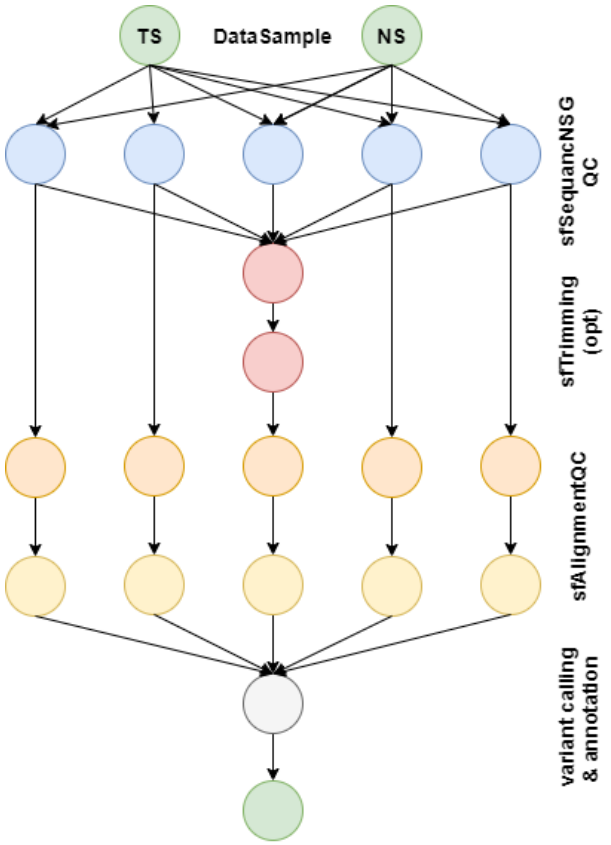


Fig. 2: SparkFlow genome analysis reference architecture.

HPDA in the 'alignment' step. This paper benchmarks two computing-intensive tools used in the workflow's recalibration step (Figure 1). Namely, `BaseRecalibrator` and `ApplyBQSR`, i.e., provided by GATK to measure their performance under different settings and computational infrastructures. `BaseRecalibrator` recalculates systematic errors made by the sequencer when it estimates the quality score of each base call by applying a machine learning algorithm. This algorithm uses the known sites file to help distinguish actual variants from false positives. Next, it computes a recalibrated table with the adjustments necessary for the whole sequence. Then, `ApplyBQSR` applies the recalibrated table to the initial sequence file and produces a new recalibrated aligned sequence file for further analysis. This way ensures that the alignment files produced are of the highest quality and several more to guarantee the variants are called correctly. Both stages compromise a series of intermediately computing and data-intensive steps that often are handcrafted by researchers and analysts.

SparkFlow provides a basis for building an interoperable genome data analysis scheme that includes all major steps and reflects specifics in the recalibration process. The contributions of this paper are four-fold:

- Introducing SparkFlow, a Spark-based workflow to scale-up and out cancer genomic analysis by providing a new alignment pipeline for fully automated high throughput micro-batch analysis in the 'alignment' stage.
- Improving the portability, parallelism, and agility of the GATK best practices by addressing the scalability bottlenecks and allowing full advantage of the HPDA capabilities, i.e., Spark and MPI using Singularity.
- Comparing the performance of SparkFlow with and without the `HyperThreading` technology for the benefit of cancer genomic analysis over different deployment architecture and providing insight into the trade-offs among these configurations.
- An in-depth investigation of two leading tools in the NGS recalibration quality control in the alignment stage, i.e., `BaseRecalibrator` and `ApplyBQSR`, to improve ease of use and scalability of the approach under container-based environment.

The remainder of this paper is organized as follows. Section II provides an overview of the background. Section III details the design of SparkFlow and introduces its implementation. The experiments and evaluation of SparkFlow are presented in Sections IV. Our reflection on the experiment results are discussed in section V and the related work are presented in Section VI. Finally, we conclude in Section VII.

II. Background

Typically in a cancer genomics analysis, both a tumor sample (TS) and a normal sample (NS) from the same individual are first sequenced using NGS systems [2] and compare using a series of quality control stages. These stages utilize sequencing technologies to generate massively distributed and heterogeneous datasets. These datasets are predominantly composed of billions of nucleotides stored as plain text files in gigabytes (GBs) over a distributed file system. Figure 2 demonstrates a general overview of the cancer analysis pipeline's main stages for high throughput sequencing data. The first control stage in a cancer genomics analysis, 'Sequence Quality Control,' is optional to check sequence quality and performs some trimming. While the second one, 'Alignment,' involves several phases, such as alignment, indexing, and recalibration, to ensure that

the alignment files produced are of the highest quality and several more to guarantee the variants are called correctly. Both stages compromise a series of intermediately computing and data-intensive steps that researchers and analysts often handcraft.

Cancer NGS complexity and characteristics are fairly distinct from genomics analysis quality control stages. For example, they often comprise multiple stages such as alignment, indexing, and recalibration to ensure that the alignment files produced are of the highest quality and several more to guarantee the variants are called correctly. These stages are range from I/O intensive to compute- and memory-intensive. The different genome analysis stages can be bound with different data access patterns, i.e., random and sequential of pre-processing and advanced analytics. In this way, the alignment process remains a bottleneck in bioinformatics pipelines. MapReduce was considered the primary data processing framework in this stage. However, the access model to data file systems that required saving the hard disk's metadata harmed the MapReduce-based solutions. Apache Spark RDD model has emerged as the new alignment data process trend [19].

A. Bioinformatics Workflow Systems

In the large-scale and complex scientific analysis era, handling large pipelines of different processing tools is called workflows. These workflows aim to orchestrate the pipeline tools automatically, efficiently, and in a time/cost-effective manner. Workflow management systems are designed to allow these workflows expressed formally and provide the required environment for setting up, executing, and monitoring the processes. Hence, scientists can share and reuse them and effectively verify results and compare the performance of different published works. These workflows can be classified based on their execution model, heterogeneous computing environments, and data access methods [24].

Many platforms and standards aim to improve the interoperability among them. A leading example is the common workflow language (CWL) initiative [13]. Cutting-edge workflow systems, such as COMPSs [14], Apache Taverna [47], and Galaxy [15], can also be described graphically, allowing communication with domain scientists and map the best practices over on-premise HPC or cloud-based infrastructure. In this context, the bioinformatic workflow is a formal representation of multi-sequential tasks of a repeatable pattern systematically designed to carry out bioinformatics analysis. A fundamental tenet of these bioinformatic workflows is to provide a common ground between the practitioners by sharing and reusing the workflow's components. Also, it provides an easy-to-use media for NGS pipeline deployment and tracks the provenance of the workflow execution results. Hence, it provides an interactive tool for scientists to execute their workflows, view, and share their results in real-time. As they represent a series of sequential stages, the outputs of one stage work as inputs to the next one. Moreover, bioinformatics workflow systems seek to facilitate the evaluation of different NGS computational and data execution series by comparing and contrasting their methodologies in terms of performance, scalability, cost, etc [12] [5].

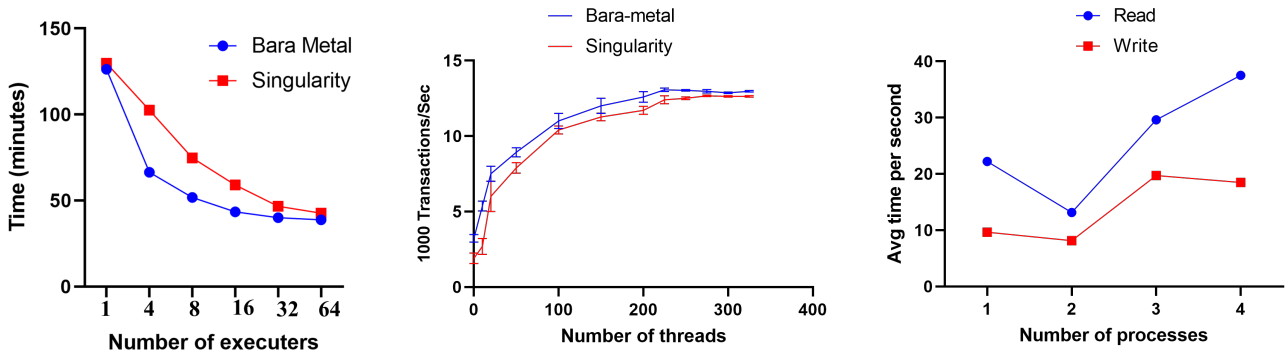
B. Containerization Technology

Targeting a native clustering approach was the primary goal of Kubernetes. It aims to coordinate clusters of nodes at scale in production with efficiency [28]. Nevertheless, Kubernetes and Docker expect root privileges to orchestrate processes. HPC is typically a shared runtime environment that requires robust isolation with a user ID and iterative access to the file system. These limitations were making them impractical for BD applications over HPC. Singularity, i.e., a native HPC container engine, was proposed by Gregory M. Kurtzer at LBNL [16] to overcome these limitations. Singularity emphasizes the mobility of computing and takes advantage of the high-end note capabilities. Also, Singularity gains automatic access to the host filesystem ($\$HOME$, $\$PWD$, and $/tmp$ are mounted automatically) and run as the current user-id and group id. Hence, Singularity does not require a daemon process, as the user context is always maintained at the container launch time (run in userspace). besides, it does not use layers to build their containers nor requires setting up the read-write layer. Singularity also has native support for HPC architectures for MPI, making containers' execution at scale over OpenMPI with near bare-metal performance [17]. Figure 4 shows a comparison of different OS startup times using Docker and Singularity.

Aiming to provide an empirical analysis of Singularity performance against a native (Bare-metal) in HPC environments, we conduct a series of evaluations in Figure 3. We used a cloud instance runs CentOS over a Lustre file system with up to 64 VCPU, 32GB memory, and 40GB total disk. We study Singularity performance against a bare-metal cluster using CPU-intensive and I/O intensive Intel HiBench suite benchmarks[1].

In Figure 3a we conduct a Spark TeraSort that includes three applications Teragen, a MapReduce program to generate input data, TeraSort to sort the input data, and TeraValidata, which can be used to check the output. We can see that the bare-metal accelerates the execution substantially, e.g., when using four processors/executors , the improvement of the bare-metal version over Singularity is bigger than a factor of 2×. However, when the number of processes/executors is increased, Singularity's improvement exhibited almost the same performance within the 64 processors/executors . Meanwhile, Figure 3b demonstrates a SparkSQL throughput (transactions/s) vs concurrency where throughput increases with load until the machine is saturated using different number of threads without using the `HyperThreading`. Singularity shows a similar performance compared to the bare-metal with asymptotically 2% peak difference. Finally, a read/write benchmark from the Lustre Object Storage Servers (OSS) in Figure 3c. Singularity uses a single image container which means that the container

[1]https://github.com/Intel-bigdata/HiBench



(a) Spark execution time

(b) SparkSQL throughput

(c) I/O from OSS (MB)

Fig. 3: Performance comparison of bare-metal vs. Singularity settings.

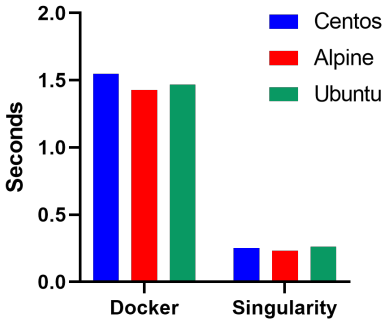


Fig. 4: Overview of different OS image startup times latency in containerization environments.

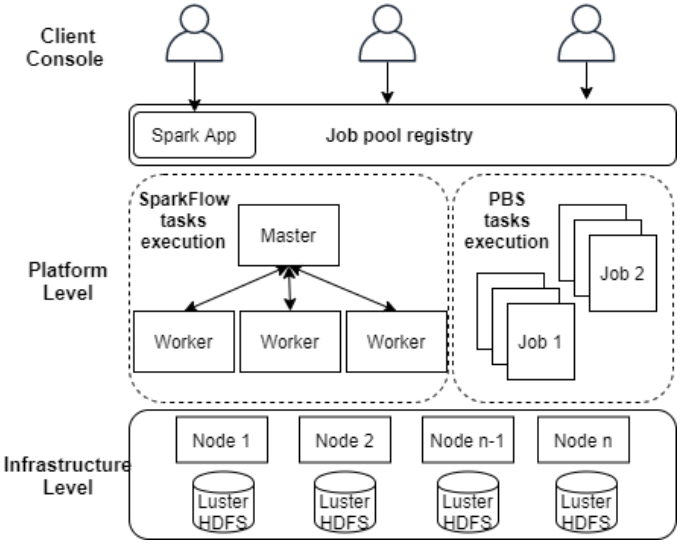


Fig. 5: A high-level overview of SparkFlow abstraction layers.

is wrapped in a single image file with minimum read/write overhead with a similar performance to the bare-metal.

III. SparkFlow: Architectural Design

This section introduces our novel Spark-based alignment workflow called SparkFlow. SparkFlow integrates high-performance data analytics environments into the Spark framework by utilizing containerization technology in a pilot-abstraction model over HPC infrastructure. SparkFlow design and methods are discussed, as well as its algorithms and components.

A. SparkFlow Methods and Design

Figure 5 illustrates the basic structure of SparkFlow that consists of three abstracted layers.

SparkFlow was designed with the ability to automatically handle the parallel placement of workloads across the cluster nodes. It performs data-level parallelism as it adapts well by elastically scaling with the sample requirement in a multi-node approach. SparkFlow, hence, supports job arrays; its workload manager copes with the boost increase in analyzing large datasets and the number of jobs —by provisioning the workflow with new containers in a PBS job. SparkFlow is isolated and lightweight by deploying a container-based (Docker) environment that leverages kernel namespaces to isolate the application. Hence, SparkFlow affords an easy way to implement existing genome analysis tools. Its container-enabled feature also eliminates the need for managing complex software delivery processes and dependency management.

SparkFlow was designed with the following objectives in mind:

1) **Integrity:** the workflow has to be compatible with the GATK tool and different BAW distributions without significant modification to their original design.
2) **Agility:** the workflow must be implemented over any underlying computation (infrastructure) to utilize standalone machines and clusters of commodity hardware and HPC systems over both on-premise and cloud deployment architectures.
3) **Scalability:** the workflow must keep high scalability and provisioning the computation elastically to meet the application requirements on-demand.
4) **Performance:** the workflow must enhance the overall

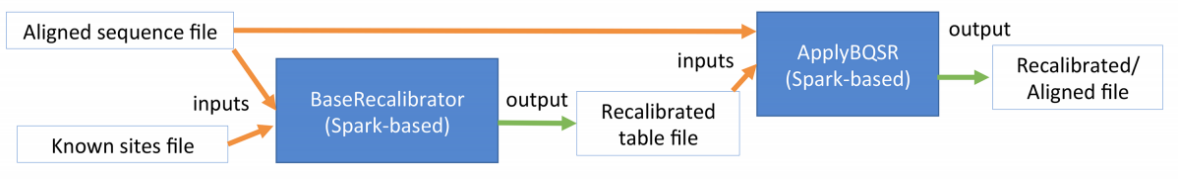


Fig. 6: Recalibration step of the Alignment workflow that uses two GATK spark-based tools in Table II.

Algorithm 1: Elastic resource provisioning on-demand of Spark-based environment over HPC using PBS job

Let M and W_s be a Master and a finite set of Worker nodes respectively.
Let SFc = SparkFlow cluster.
Configure Spark parameters and & URL.
Input: number of active nodes (α) and project budget β
Output: multinode Spark standalone cluster within a PBS-job
if $\alpha \neq 0 \parallel \beta \not\leq 0$ **then**
  for $\forall(\alpha) \in$ *Cirrus, SFc** $\rightarrow$ *resource allocation* **do**
    Lunch PBS job to start Spark M
    if *HOSTNAME* $\neq$ *hostmaster* $\cup$ *HOSTNAME* $\neq$ *hostdriver* **then**
      Started Ws on hostname;
      echo HOSTNAME $\rightarrow$ worker.log
    else
      Master or driver node do not launch Ws on HOSTNAME
    end
  end
  PBS job *BaseRecalibrator* and *ApplyBQSR*// SparkFlow pipeline in Algo 2
  return SparkFlow = true;
else
  return SparkFlow = failure;
  terminate;
end

Algorithm 2: SparkFlow pipeline process using Spark-based approach

Module load anaconda and mpt
Set Genome sample = RFF
Set genetic variation data base = DBSNP
for *Each recalibration stage in the pipeline* **do**
  export SPARK_HOME;
  export REF= gatk/GRCh37;
  export DBSNP= gatk/dbsnp-147;
  for *BaseRecalibrator* $\parallel$ *ApplyBQSR* **do**
    pass $\sum$ (executor-cores)
    pass $\sum$ (executor-memory)
    if $SFc* = true$ **then**
      Phase 1 κ = RDD Join { $\exists$ RDD unsorted};
      Phase 2 λ = RDD sort $\gg$ by key; // Basic sets and functions # 3 & 4
      Phase 3 μ = Apply_Merg;
    else
      return failure;
      terminate;
    end
    Write bins in the File System (FS)
  end
  Evaluate model performance // Basic sets and functions # 2
  Callculate the efficiency* [BQSR = $\kappa+\lambda+\mu$]
end

Basic Sets and Functions
1- $SFc*$ (SparkFlow cluster)
2- * efficiency = S_p/p, parallel speedup (S_p) divided by the parallelism (p). Where efficiency is better the closer it gets to 1.
3- RDD arranged as `<binID, content>` mappedInfo runMapper(fileName)
4- Return information. RDD is of type `<binID, numReads>`

performance and improve the throughput with minimal cost of deployment.

5) **Automation:** The workflow must be designed for automation and agonistic to content and infrastructure. This automation feature addresses many time-consuming and error-prone human efforts, which results in a fast process and more efficient as well.

In Figure 6 The `BaseRecalibrator` tool recalculates systematic errors made by the sequencer. It distinguishes true variants from false positives by applying a machine learning algorithm. The `BaseRecalibrator` tool also computes a recalibrated table with the adjustments necessary. In contrast, the `ApplyBQSR` tool produces a new adjusted/recalibrated aligned sequence file and applies the recalibrated table to the initial sequence file. It is worth mentioning that the whole Alignment workflow includes a stage before this diagram, using other tools like `BWA-Mem`, `Samblaster`, and `SAMtools` to perform the alignment itself. The recalibration process involves two key steps that can be summarised as follows.

For an aligned sequence data file, the `BaseRecalibrator` tool recalculates systematic errors

made by the sequencer when it estimates the quality score of each base call by applying a machine learning algorithm, which uses the known sites file to help distinguish true variants from false positives.

Moreover, it computes a recalibrated table with the adjustments necessary for the whole sequence. Then the `ApplyBQSR` tool applies the recalibrated table to the initial sequence file and produces a new adjusted/recalibrated aligned sequence file for further analysis.

With the GATK's latest release, both tools have been completely reengineered to support an open-source model that relies on Apache Spark to provide single-node multithreading and multi-node scaling capabilities to increase their speed and scalability. In prior versions, these tools were limited to running on a single-node shared-memory environment.

It is unnecessary to have a Spark cluster configured to run these new Spark-based tools in parallel using a single-node (multithreading). The GATK engine can still use Spark to create a single-node standalone cluster dynamically in place, using as many cores as available on a node. However, in our case, we were interested in testing the scalability of `BaseRecalibrator` and `ApplyBQSR` using a multi-node Spark Cluster.

B. Implementation

In general, SparkFlow provides a systematic way to dynamically provision the application demands with elastic resources over a shared infrastructure using PBS jobs. Algorithm 1 demonstrates the elastic resource provisioning mechanism that match-make the active cluster nodes with the required container instances. On the other hand, Algorithm 2 illustrates the Apache Spark pipeline that employees `BaseRecalibrato` and `ApplyBQSR` with the speedup technique.

The base recalibration process of our workflow consists of two tools, `BaseRecalibrator` and `ApplyBQSR`. The `BaseRecalibrator` read data whose base quality scores need to be assessed. It also provides a recalibration table by the reading group, all the optional covariates, and the quality score with a list of arguments to be aligned to the reference genome. The efficiency of `BaseRecalibrator`, however, is proportional to the number of base pairs provided.

`ApplyBQSR`, on the other hand, represents the second pass of the Base Quality Score Recalibration (BQSR). It has replaced the prior PrintReads in the GATK 3.x distribution in a two-stage process. At first, it recalibrates the base qualities of the input reads based on the recalibration table produced by the `BaseRecalibrator` tool. Next, recalibrated the outputs of a BAM or CRAM file.

The use of SparkFlow in launching a spark-based cluster within a PBS job is described in Figure 7 as follow:

1) A client initially submit a job in the job pool.
2) SparkFlow will request the available resources from the cluster resource manager on behalf of clients.
3) the cluster resource manager response by allow/denial of the operation.

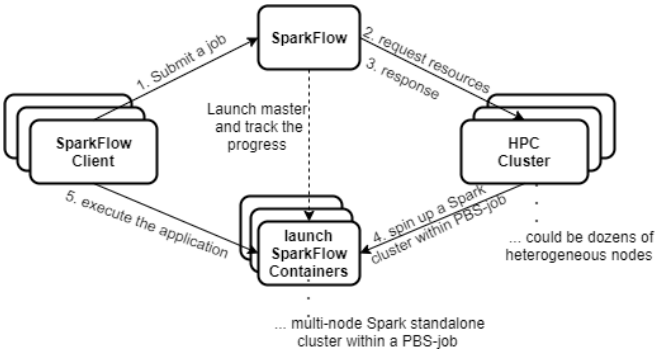


Fig. 7: A simplified diagram of the SparkFlow mechanisms.

4) Spin up an ad-hoc multi-node standalone Spark-based cluster within a PBS-job. This PBS-job provisions on-demand and for a specific time the desired Spark cluster by starting the master, workers, and registering all workers against master.
5) finally, the client application can access the allocated resources using the granted permission and run its application.

C. Data structure

Based on the previously mentioned problem statement and in light of the SparkFlow design primitives. We extend the problem formulation to be in line with the proposed solution in this work. Herein, we describe the data construction (preparation phase) centered on the Apache Spark approach. We then show how to utilize this approach on large-scale genome datasets. It is important to note that the literature lacks a reference model when designing a Spark-based NGS workflow to the best of the authors' knowledge. It is also worth noting that Spark supports different storage backends for reading and writing data. Typically, Spark's physical plan starts by *scanning* data sources for reading purposes. This process leads to different read latency requirements with various storage backends and with different file formats. In our experiments, we use the *Hadoop Distributed File System*(*HDFS*) backend over Luster. Typically, HDFS partitions the data into blocks. Each data partition on HDFS is equal to the default block size (128MB). HDFS also manages the replication of these partitioned blocks according to a configurable *replication factor(RF)* (we used the default one $RF = 3$). Our experiments also used the baseline partitioning technique that depends on the default HDFS partitioning of the cluster nodes' datasets.

The data construction formulation considers a collection C of g genomicsequences (reference genomes). Each of which is divided into chunks (data blocks) n of size s that overlap by L_1 base-pairs. As in a default Spark runtime environment, the anticipated read length of similar size s = 128MB for data as default. For each chunk, a sketch is calculated using MinHash (i.e., estimating how two sets are similar). A strand-neutral canonical representation of that sketch contains a x_1 hash function with s as the smallest L-mers. Therefore, the sketching procedure selects only a subset of L-mers inserted

TABLE I: SparkFlow validation computing environments.

Cluster name	Nodes Number	Cores per node	Memory	Storage	File System	Resource Manager	`HyperThreading`
Cray Urika-GX	12	36 Broadwall	256 GB	60 TB	HDFS & Lustre	Apache Mesos	*X*
Cirrus	280	36 Intell Xeon	256 GB	406 TB	Lustre	PBS	✓
Common criteria	1 Gbps network connection, Ubuntu 16.04 LTS OS						
In Cirrus each node has 72 virtual cores using the `HyperThreading` technology.							
Urika-GX has pre-integrated stack of popular analytics packages: Apache Spark, Apache Hadoop, Jupyter Notebooks, etc.							

TABLE II: SparkFlow aligners scalability evaluation.

Tool	Input files/size	Output files/size
`BaseRecalibrator`	Aligned file 145GB, Known sites files 3MB	Recalibrated table 260KB
`ApplyBQSR`	Aligned file 145GB, Recalibrated table 260kb	Recalibrated aligned file 230GB

into the HDFS (can be later stored in Avro and Parquet formats for further analysis queries) used for similarity computation. The subsampling factor can be determined as $M = \frac{sL+1}{m}$ (assuming unique L-mers). Where m, L, and s are 8, 16, and 128, respectively, and data reduction magnitude (M = 14.125). Also, MinHash shows a desirable mathematical feature when matching two sketches and provides data reduction. The relative intersection ratio between two sketched chunks approximates the accurate Jaccard index evaluated on the whole L-mer space.

IV. Evaluation and Results

Herein, SparkFlow is evaluated regarding its performance and scalability. A complete description of the experimental setup is provided. Following, we analyze SparkFlow results in detail.

A. Experiment setup

SparkFlow was tested using data from the BioExcel Genomes Project. To demonstrate SparkFlow abilities, we carry out the experiments in two different HPC environments, i.e., Cray Urika-GX [2] system and Cirrus [3]. The validation computing environment and experiment setup are detailed in Table I.

The Urika-GX system is a high-performance analytics cluster with a pre-integrated stack of popular analytics packages. Little work was required to run the recalibration step of the Alignment workflow in Urika-GX using several nodes. The Urika-GX software stack includes a fault-tolerant Spark cluster configured and deployed to run under Apache Mesos [41], which acts as the cluster manager. Therefore, we just needed to adjust four Spark parameters. Namely, Spark master with the URL of the Mesos master node; the number of executors (worker nodes' processes in charge of running individual Spark tasks); the number of cores per executor (up to 36 cores); and the amount of memory to be allocated to each executor (up to 250GB).

[2]ATI HPC ATI cluster: https://www.cray.com/products/analytics/

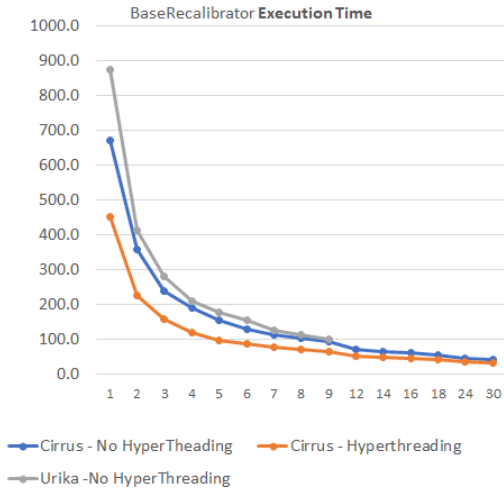
[3]EPCC HPC cluster: https://www.epcc.ed.ac.uk/facilities/demand-computing/cirrus

On the other hand, to run the same recalibration step in Cirrus, we had first to spin up an ad-hoc multi-node standalone Spark cluster within a PBS-job. This PBS-job provisions on-demand and the desired Spark cluster for a specific period by starting the master, workers, and registering all workers against master. We decided to configure the Spark cluster with one node as the master and 30 nodes as workers. Each node in Cirrus has 36 physical cores (or 72 virtual cores using `HyperThreading`). Therefore we decided to test the performance of GATK Spark-based tools in Cirrus configuring workers with and without `HyperThreading`, which means that each worker was configured with 250GB of memory and either with 36 cores or 72 virtual cores. Therefore, we had 1080 cores (or 2160 virtual cores) available for our experiments. After setting up the Spark cluster in Cirrus, the same parameters (spark master URL, number of executors, number of cores per executors, memory) needed to be configured for running both Spark-based tools within the workflow. The main characteristics of the SparkFlow aligners scalability input datasets are shown in Table II.

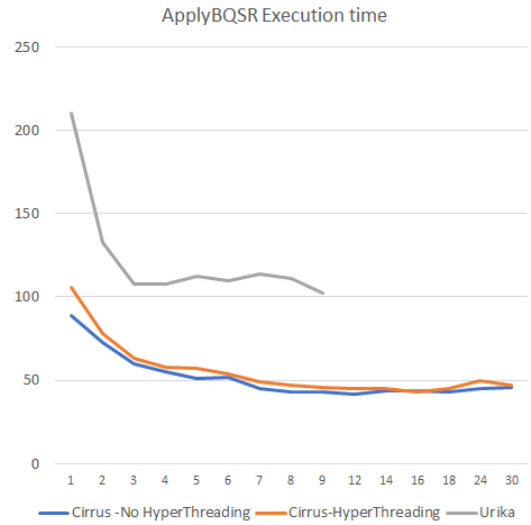
It is worth mentioning that for both computing environments, each Spark-executor runs in a different worker node. Moreover, each executor can be configured with 36 cores (in Urika and Cirrus without `HyperThreading`) or with 72 cores (in Cirrus with `HyperThreading`).

B. Performance Evaluation

For our evaluations, we configured the recalibration step (In Table II) to use an initial sequence file (145GB) and known site files (3MB), and it produced a recalibrated table (260KB) and a new recalibrated file (230GB) as output files. We performed several recalibration runs to capture each execution time by varying the total-executor-cores flag in our Spark-jobs. This flag represents the total amount of cores (of all executors) assigned to a Spark application. So increasing the total-executor-cores automatically increases the number of executors for our runs. For example, if we set up in our Spark-job the total-executor-cores to 144, it will automatically use two executors in Cirrus if `HyperThreading` is enabled (144/72) while it will use four executors in Urika-GX and Cirrus if `HyperThreading` is not enabled (144/36).

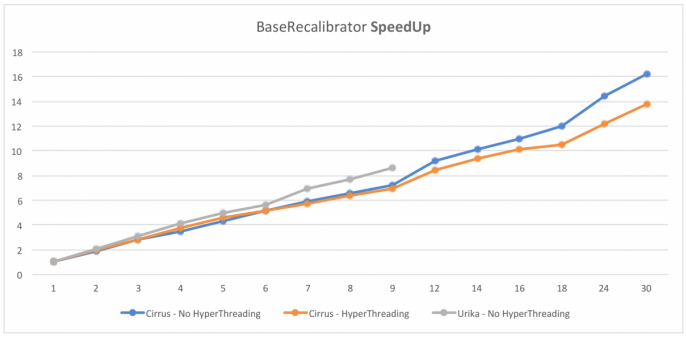


(a) `BaseRecalibrator` execution time

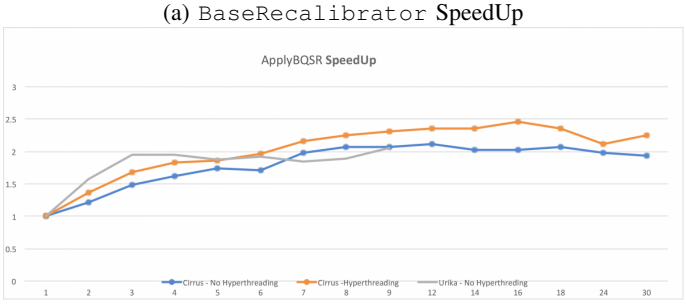


(b) ApplyBQRSpark execution time

Fig. 8: Graph of the relationship between the execution time of the two benchmarks with varying number of nodes used



(a) `BaseRecalibrator` SpeedUp



(b) ApplyBQRSpark SpeedUp

Fig. 9: Graph of the speedup in execution time

Figure 8 shows the execution times in minutes (axis Y) of `GATK` Spark-based tools on Cirrus with `HyperThreading` (orange), without `HyperThreading` (blue), and Urika-GX (grey) modifying the number of executors (axis X). The number of executors used is calculated by dividing the total-executor-cores by the number of cores (72 in Cirrus with `HyperThreading` and 36 in Urika-GX and Cirrus without `HyperThreading`) available per node.

Figure 9 shows the speedup (axis Y) is calculated dividing the execution time using one executor (T1) between the execution time using P executors (Tp). Axis X represents the number of executors.

$$S = T1/Tp \tag{1}$$

V. Discussion

Genome analysis in medical practice and biomedical genomics research is projected to increase exponentially. Using high-performance data analytic frameworks and computing environments is vital to handle this vast amount of data robustly. This work has focused on improving the ease of use and scalability of the genomic cancer analysis by providing a new Alignment workflow for fully automated high throughput analysis in the 'alignment' stage. This work resulted in benchmarking two computing-intensive tools used in the recalibration step (Figure 6) of the workflow, `BaseRecalibrator`, and `ApplyBQSR` (provided by `GATK` tool), to measure their performance under different settings and computational infrastructures.

This paper represents an in-depth investigation of Spark-based workflow, architectural components associated with the deployment of biomedical genomics research, and scalability and performance benchmarking. SparkFlow also addresses many technical aspects that optimize the context-specific design in the bioinformatics pipelines. Thus, SparkFlow provides a basis for building an interoperable large-scale genome data analysis scheme that includes all significant stages. It also reflects specifics in providing a new alignment workflow for fully automated high throughput analysis in the 'alignment' stage. This work resulted in benchmarking two computing-intensive tools used in the recalibration step of the workflow. The `BaseRecalibrator` and `ApplyBQSR` (provided by GATK [20]) measure their performance under different settings and computational infrastructures in HPC and Cloud environments.

In this work, we have focused on improving the ease of use and scalability of the genomic cancer analysis by providing a new Alignment workflow for fully automated high throughput analysis in the 'alignment' step. SparkFlow was designed to handle a large amount of data automatically robustly, with speed, scalability, and flexibility for biomedical genomics research. Its core philosophy meets three main requirements of next-generation sequencing. First, SparkFlow should be compatible with the GATK tools and keep agnostic to BWA versions to meet compliance and governance demands. Second, efficiently hide the workflow complexity to perform sequence alignments with both performance and scalability. The final requirement is related to improved portability, security, and cost of deployment (with a better return on infrastructure investment).

This study concludes that genome analysis in medical practice is characterized by:

1) elastically paralyzing NGS data execution at scale,
2) accelerated computation provisioning on-demand,
3) flexibility for adapting to the application requirements
4) compatibility with the different BWA functionalities and algorithms without modifications of its source code,
5) efficiently perform sequence alignments with ease of use by hidden the deployment complexity,

Since we used the standalone mode (in cirrus), the master is used for scheduling resources and distributing data across the Spark cluster, making it lightweight (without a scheduler like Yarn or Mesos), but on the other hand, creating a single point of failure. We also ran several tests in both environments modifying the memory available per executor, but a slight improvement was achieved by using values higher than 20GB.

VI. Related Work

Recently, there have been approaches to tackle the scalability problem of the DNA analysis pipeline using BD techniques and frameworks. Many of these solutions are Apache Hadoop based like [30] [31]. However, the recent advancement in BD science has focused on improving the ease of use and scalability using the Apache Spark platform. One such example is SparkRA, a scale-up pipeline for the GATK RNA-seq variant to efficiently analyze multiple cores. SparkRA divides the data files into chunks and processes them in parallel based on the Apache Spark platform [18]. Another example is SparkGA [21] that employs static and dynamic load balancing to spread the input data more uniformly across the cluster nodes aiming for greater scalability. However, SparkGA creates too many files in the Hadoop distributed file system after the mapping phase, affecting overall performance. To address this drawback, SparkGA2 [22] aims at reducing the copied data in memory and reducing the memory footprint. SparKGA adapts the amount of generated files from the mapping phase in the cluster by analyzing the available memory.

Other frameworks sharpened the computational load required in each chromosome by keeping data active in the memory between the map and reduce rounds [23]. This way, optimize the performance using runtime statistics of the active workloads that dynamically balance the load. A Spark-based implementation of the most widely adopted aligner, the burrows-wheeler aligner, was proposed in SparkBWA [25]. SparkBAW aims to boost the process of the alignment phase in the DNA sequence analysis by targeting the short-read mapping. Another multiple sequence alignment Spark-based implementation are PASTASpark [26] and [27] with a with supervised learning approach.Also, utilizing in-memory data analytics applications that process columnar data as for ArrowSAM [29] that employes Apache Arrow reported in the literature. In PipeMEM [33], a pipeline parallel pattern that ensures no local disk access, the authors optimized the computation phase by employing standard stream and PipeRDD. In Spark-DNAligning [38], the authors proposed a short reads alignment problem for the NGS.

The genome analysis community has made great efforts in the literature to use a wide variety of hardware-accelerated methods, such as GPU and FPGA [34] [37] implementations that efficiently proved to manage GATK workloads. FPGA-based solutions are accelerating the DNA alignment analysis by affording extensive parallelism and high performance. Besides, they return with better energy efficiency and reconfigurability [36]. In [35], the authors address the variant calling tool, i.e., HaplotypeCaller performance, by analyzing the PairHMM and propose an FPGA-based architecture. GASAL2 is a high-throughput NGS data using GPU library that expedites sequence alignment [39], while the work by Ren, Shanshan, et al., [40] aims at retrieving the optimal alignment accurately on a GPU environment. In contrast, the role of the control loop mechanism in the industrial control systems of distributed workflows was discussed in [46].

Against the previous literature, SparkFlow provides an in-depth investigation into the impact of containerization and `HyperThreading` technology on genome analysis. Also, SparkFlow supports dynamic resource allocation over heterogeneous compute architecture by employing Singularity containers in PBS jobs. We keep SparkFlow in an open source repostory [42]. This repository describes all the steps necesaries to create a multinode Spark standalone cluster within a PBS-job.

VII. Conclusion

The ever-increasing amount of generated data required advances in data distribution schemes, processing engines, and communication patterns to meet medical discoveries demands. Employing high-throughput sequencing frameworks across High-performance infrastructures has a significant interest in using advanced Big Data technologies. This trend has sparked several approaches to deal with large-scale complex genome datasets using Spark-based workflows. This paper focuses on providing a new Alignment workflow for fully automated high throughput analysis in the alignment stage. This paper introduced SparkFlow, a parallelized method for pre-processing genome sequence data based on High-performance Data Analytics. SparkFlow reduces the analysis time by utilizing big data (Apache Spark), containerization technology, and the

high-performance computing cluster (PBS-job abstraction). Relying on Singularity, a native HPC-enabled container-based cluster, SparkFLow can scale out elastically towards vast amounts of datasets, making it suitable for processing broad-spectrum of NGS applications (Oncology, cancer research, etc.). Also, this paper represents a showcase of Spark-based workflow and the architectural components associated with the deployment of biomedical genomics research.

REFERENCES

[1] Baker, Qanita Bani, et al. "Comprehensive comparison of cloud-based NGS data analysis and alignment tools." Informatics in Medicine Unlocked 18 (2020): 100296.

[2] Pleasance, Erin D., et al. "A comprehensive catalogue of somatic mutations from a human cancer genome." Nature 463.7278 (2010).

[3] Lander, Eric S., et al. "Initial sequencing and analysis of the human genome." (2001).

[4] Al Kawam, Ahmad, Sunil Khatri, and Aniruddha Datta. "A survey of software and hardware approaches to performing read alignment in next generation sequencing." IEEE/ACM transactions on computational biology and bioinformatics 14.6 (2016): 1202-1213.

[5] Bharathi, Shishir, et al. "Characterization of scientific workflows." 2008 third workshop on workflows in support of large-scale science. IEEE, 2008.

[6] Chen, S., He, Z., Han, X., He, X., Li, R., Zhu, H., and Niu, B. (2019). How big data and high-performance computing drive brain science. Genomics, proteomics bioinformatics, 17(4), 381-392.

[7] Schmidt, B., and Hildebrandt, A. (2017). Next-generation sequencing: big data meets high performance computing. Drug discovery today, 22(4), 712-717.

[8] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[9] Cutting, Doug and Cafarella, Mike, http://hadoop.apache.org, note = Accessed: 2020-06-02 year = 2006,

[10] Alnasir, Jamie J., and Hugh P. Shanahan. "The application of hadoop in structural bioinformatics." Briefings in bioinformatics 21.1 (2020).

[11] Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). 2012.

[12] Deelman, Ewa, et al. "The future of scientific workflows." The International Journal of High Performance Computing Applications 32.1 (2018): 159-175.

[13] Peter Amstutz, Michael R. Crusoe, Nebojša Tijanić (editors), Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Hervé Ménager, Maya Nedeljkovich, Matt Scales, Stian Soiland-Reyes, Luka Stojanovic (2016): Common Workflow Language, v1.0. Specification, Common Workflow Language working group. https://w3id.org/cwl/v1.0/

[14] Badia, Rosa M., et al. "Comp superscalar, an interoperable programming framework." SoftwareX 3 (2015): 32-36.

[15] Goecks, Jeremy, et al. "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences." Genome biology 11.8. 2010

[16] Kurtzer, Gregory M., Vanessa Sochat, and Michael W. Bauer. "Singularity: Scientific containers for mobility of compute." PloS one 12.5 (2017): e0177459.

[17] Zhang, Jie, Xiaoyi Lu, and Dhabaleswar K. Panda. "Is singularity-based container technology ready for running MPI applications on HPC clouds?." Proceedings of the10th International Conference on Utility and Cloud Computing. 2017.

[18] Al-Ars, Zaid, Saiyi Wang, and Hamid Mushtaq. SparkRA: Enabling Big Data Scalability for the GATK RNA-seq Pipeline with Apache Spark. Genes 11.1 (2020): 53.e

[19] Guo, Runxin, et al. "Bioinformatics applications on apache spark." GigaScience 7.8 (2018): giy098.

[20] McKenna, Aaron, et al. "The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data." Genome research 20.9 (2010): 1297-1303.

[21] Mushtaq, Hamid and Liu, Frank and Costa, Carlos and Liu, Gang, SparkGA: A spark framework for cost effective, fast and accurate dna analysis at scale, Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, 148–157, 2017.

[22] Mushtaq, Hamid, Nauman Ahmed, and Zaid Al-Ars. "SparkGA2: Production-quality memory-efficient Apache Spark based genome analysis framework." PloS one 14.12 (2019).

[23] Mushtaq, Hamid, and Zaid Al-Ars. "Cluster-based Apache Spark implementation of the GATK DNA analysis pipeline." 2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). IEEE, 2015.

[24] da Silva, Rafael Ferreira, et al. "A characterization of workflow management systems for extreme-scale applications." Future Generation Computer Systems 75 (2017): 228-238.

[25] Abuín, José M., et al. "SparkBWA: speeding up the alignment of high-throughput DNA sequencing data." PloS one 11.5 (2016).

[26] Abuín, José M., Tomás F. Pena, and Juan C. Pichel. "PASTASpark: multiple sequence alignment meets Big Data." Bioinformatics 33.18 (2017): 2948-2950.

[27] Vineetha, V., C. L. Biji, and Achuthsankar S. Nair. "SPARK-MSNA: Efficient algorithm on Apache Spark for aligning multiple similar DNA/RNA sequences with supervised learning." Scientific reports 9.1 (2019): 1-11.

[28] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J. (2016). Borg, omega, and kubernetes. Communications of the ACM, 59(5), 50-57.

[29] Ahmad, Tanveer, et al. "ArrowSAM: In-Memory Genomics Data Processing Using Apache Arrow. "2020 3rd International Conference on Computer Applications & Information Security (ICCAIS). IEEE, 2020.

[30] Abuín, José M., et al. "BigBWA: approaching the Burrows–Wheeler aligner to Big Data technologies." Bioinformatics 31.24 (2015): 4003-4005.

[31] Decap, Dries, et al. "Halvade: scalable sequence analysis with MapReduce." Bioinformatics 31.15 (2015): 2482-2488.

[32] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: Simplified data processing on large clusters." (2004).

[33] Zhang, Lingqi, Cheng Liu, and Shoubin Dong. "PipeMEM: A Framework to Speed Up BWA-MEM in Spark with Low Overhead." Genes 10.11 (2019): 886.

[34] Hasan, Laiq, and Zaid Al-Ars. "An overview of hardware-based acceleration of biological sequence alignment." Computational Biology and Applied Bioinformatics (2011): 187-202.

[35] Sampietro, Davide, et al. "Fpga-based pairhmm forward algorithm for DNA variant calling." 2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP). IEEE, 2018.

[36] Salamat, Sahand, and Tajana Rosing. "FPGA Acceleration of Sequence Alignment: A Survey." arXiv preprint arXiv:2002.02394 (2020).

[37] Waidyasooriya, Hasitha Muthumala, and Masanori Hariyama. "Hardware-acceleration of short-read alignment based on the burrows-wheeler transform." IEEE Transactions on Parallel and Distributed Systems 27.5 (2015): 1358-1372.

[38] AlJame, Maryam, and Imtiaz Ahmad. "DNA short read alignment on apache spark." Applied Computing and Informatics (2019).

[39] Ahmed, Nauman, et al. "GASAL2: a GPU accelerated sequence alignment library for high-throughput NGS data." BMC bioinformatics 20.1 (2019): 520.

[40] Ren, Shanshan, et al. "GPU accelerated sequence alignment with traceback for GATK HaplotypeCaller." BMC genomics 20.2 (2019): 184.

[41] Hindman, Benjamin, et al. "Mesos: A platform for fine-grained resource sharing in the data center." NSDI. Vol. 11. No. 2011. 2011.

[42] Spark on HPC cluster. https://github.com/rosafilgueira/Spark_on_HPC_clusterGitHub repository. Last access on December 13 2021.

[43] Awaysheh, F. M., Alazab, M., Garg, S., Niyato, D., and Verikoukis, C. (2021). Big data resource management & networks: Taxonomy, survey, and future directions. IEEE Communications Surveys & Tutorials.

[44] Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." Linux journal 2014.239 (2014): 2.

[45] Bernstein, David. "Containers and cloud: From lxc to docker to kubernetes." IEEE Cloud Computing 1.3 (2014): 81-84.

[46] da Silva, Rafael Ferreira, et al. "Using simple pid-inspired controllers for online resilient resource management of distributed scientific workflows." Future Generation Computer Systems 95 (2019): 615-628.

[47] Hull, Duncan, et al. Taverna: a tool for building and running workflows of services. Nucleic acids research 34.suppl_2. 2006