Secure Genome Processing in Public Cloud and HPC Environments

André Brinkmann*, Jürgen Kaiser*, Martin Löwer†, Lars Nagel*, Ugur Sahin†, Tim Süß*
*Zentrum für Datenverarbeitung at JGU Mainz, Germany
Email: {brinkman, kaiserj, nagell, suesst}@uni-mainz.de
†Translational Oncology at JGU Mainz (TRON gGmbH), Germany
Email: {martin.loewer, ugur.sahin}@tron-mainz.de

***Abstract*—Aligning next generation sequencing data requires significant compute resources. HPC and cloud systems can provide sufficient compute capacity, but do not offer the required data security guarantees. HPC environments are typically designed for many groups of trusted users and often only include minimal security enforcement, while Cloud environments are mostly under the control of untrusted entities and companies.**

In this work we present a scalable pipeline approach that enables the use of public Cloud and HPC environments, while improving the patients' privacy. The applied techniques include adding noisy data, cryptography, and a *MapReduce* program for the parallel processing of data.

***Keywords*—genome sequencing, data security, MapReduce, Hadoop, pipeline architecture**

I. Introduction

High-throughput sequencing, also known as *next-generation sequencing* (NGS), is a revolutionary technology that enables the sequencing of entire genomes in a matter of hours. Common techniques like *sequencing-by-synthesis* require a lot of computational power for post-processing the genome data, that is, for aligning them to a reference sequence / genome and assembling them according to this mapping.

NGS techniques produce large amounts of genome data in the form of short reads (about 50 to 300 bases) which usually need to be aligned to a reference genome [20]. NGS has not only revolutionized biological and medical research [19], but also offers opportunities for the treatment of diseases. The implementation of personalized medicine [10], for example, requires the analysis of human genomes at a large scale and involves sequencing genome data from thousands of individuals per year at one facility [7]. The alignment of such an amount of data is only possible by utilizing the processing power of large computing centers.

Current solutions typically do not involve public cloud computing or academic high-performance computing (HPC) systems, but rely on expensive in-house facilities to ensure the privacy of the data. The reason is that the patients' data could be hijacked in public computing environments. HPC systems are shared by many groups of trusted users, and data security has a low priority. Cloud environments, on the other hand, may provide a slightly better data security, but they are mostly under the control of untrusted entities and companies. Hence, to exploit external computing clusters, it is necessary to make the transfer of sensitive genome data *and* their processing secure – without impacting performance too much.

In this paper we present a solution to this problem which outsources the computations to public multi-user HPC or cloud facilities while ensuring data security and fast processing. The techniques applied include adding noise, cryptography, and a novel method for the parallel processing of genome data of many patients. We argue that the system is not only inexpensive, but also secure and show that the system's performance is only slightly degraded by our security measures.

The remainder of the paper is structured as follows: In Section II we describe the scenario in more detail and derive five requirements. In Section III we discuss tools and solutions from the literature. In Section IV we outline our approach before we give a more detailed description of our pipeline architecture in Section V. In the evaluation in Section VI we check whether the requirements are fulfilled. Finally, we conclude the paper in Section VII.

II. Scenario & Requirements

Every day in biological and medical research, large amounts of genomes are processed to determine their nucleotide sequences. Besides research, genome sequencing is vital for the personalized medicine of the present and future [10], for example, for individualized immunotherapies which were considered in the CI3 project[1]. The techniques of high-throughput sequencing or next-generation sequencing have accelerated the process and made it possible to sequence the entire human genome in one day for less than $1000 [8][6].

An important part of the process is performed on high-performance computers. Since sequencers produce their (digitalized) output in the form of short *reads* (*i.e.*, sequences), the reads need to be arranged and interpreted. This is done by aligning the reads to a reference genome which is a very costly operation requiring a lot of computational power. At the same time, the data are highly confidential and must not be vulnerable to attackers. It is therefore not possible to simply transmit the data to a public cloud or high-performance computing environment. Yet, on the other hand, it is very / too costly for many institutes to abandon such cheap solutions and install a high-performance computing (HPC) facility in-house.

[1] http://www.ci-3.de/en

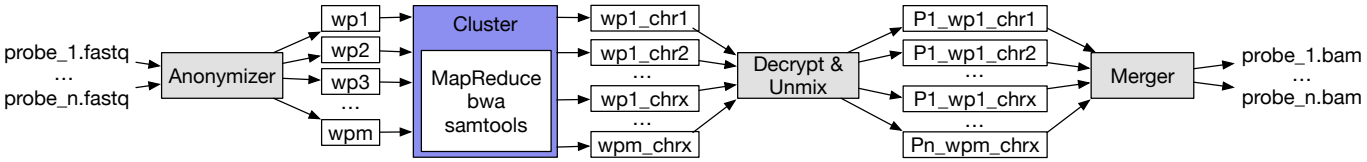

Fig. 1: Overview of the pipeline.

For this reason, we consider the scenario in which a public HPC cluster is used, but under the condition that the data are secure at any time in that an attacker might steal information, but is not able to identify the person that these data belong to. We assume that the sequencing has already been performed and that the short reads are provided as files. We regard the environment where the data originate (*e.g.*, some institute) as a secure environment with strict access rights enforcement. Thus, we only consider the transmission to and the processing at the HPC cluster as vulnerable. It can be accessed by potentially malicious users, and no data being processed or stored there can be deemed secure. So, malicious users may read the data and try to identify patients, either by reading the identifiers of the sequences or by aligning the data to a (not necessarily complete) genome of the patient that they obtained prior to the attack.

Altogether we have the following requirements:

1) The computationally expensive sequence alignment and assembly must be performed on a shared, potentially vulnerable HPC cluster.
2) The data should not be stored on disk in the cluster and must be completely deleted once the computation is finished.
3) Wherever possible, the data must be secured using strong encryption algorithms.
4) While it cannot be guaranteed that data are intercepted during computation in a public environment, it must not be possible to identify persons or probes based on the data.
5) There must be no significant degradation of the computational performance, for example, by using encryption or other means to provide data security.

In our use case we considered a particular system where the data are generated by Illumina sequencers which produce reads of roughly 50 to 300 nucleotides. The design and implementation of the system were influenced by this use case, but most of the descriptions and results are valid for other types of sequencers as well.

III. Related Work

For processing big data, *Hadoop* [26] is one of the best known MapReduce frameworks. Embarrassingly parallel jobs can be easily mapped onto this framework. In a MapReduce framework tasks are *mapped* onto nodes for processing, and the intermediate results are then *reduced* to final results [5]. Hadoop consists of three main components: the MapReduce framework itself, the resource manager *YARN* [25], and the distrubuted file system *HDFS* [22]. The HDFS file system is distributed over the complete Hadoop cluster and stores the job-related data. The resource manager YARN allows to execute applications distributed on a Hadoop cluster and to reserve the required resources.

There are many alternatives to Hadoop. In HPC systems, one usually uses *MPI* for job parallelization [24]. Big vendors like *Microsoft*, *Amazon* or *Google* offer infrastructures for processing big data in the cloud [21][23][11].

In recent years many tools for short-read alignment have been developed. The first ones focused on reads that consist of at most 100 nucleotide bases. Due to the small read size it was assumed that the quality / correctness of the sequences was high. Examples for these first-generation tools are *Bowtie* [13], *Burrows-Wheeler Aligner* [14] and *CUSHAW* [18]. In our environment we use the Burrows-Wheeler Aligner in combination with *SAMtools* [16].

New NGS sequencers are able to generate reads with longer base sequences. One drawback of these reads is that they contain more errors which the alignment tool must compensate. Aligners that follow the *seed-and-extend paradigm*, can deal with an increased number of incorrect bases. Examples are *BWA-SW* [15], *Bowtie2* [12] or *CUSHAW2* [17].

There are some approaches that use big data techniques for their short-read alignments. Similar to our approach, Abuin *et al.* use the MapReduce framework *Apache Spark* [27] for parallelizing the Burrows-Wheeler Aligner [2]. However, they do not secure the processed data what disallows the usage of unsecure shared environments. Chen *et al.* presented another approach based on the seed-and-extend paradigm that uses public and private computing clouds for their computations [3]. The public cloud is used to reduce the number of potential alignment positions while the reads' *final* positions are determined in the private cloud. In contrast to our approach, this method only aligns the reads of a single patient, whereas our approach processes the data of many patients in parallel.

IV. Approach

In this section we shortly describe the main ideas and techniques of our approach before we give a more detailed description of our architecture in the next section. As we stated in the requirements, the new system must guarantee data security and perform fast parallel alignment of sequencing data.

We use a pipeline architecture which we divide into four phases. The pre- and postprocessing Phases I, III and IV in the secure environment package and sort the data, but they

mostly serve data security in Phase II. Phase II is the critical one in which the data are sent to and processed on the shared HPC cluster. During this complete phase the patient / probe identifiers of the reads are encrypted which is possible as they are not needed for the alignment. The rest of the data is only encrypted during transmission, but vulnerable when it is processed.

Since it is possible to identify patients based on parts of their DNA even if the respective genome sequences are in a larger pool of reads [3][9], we additionally obfuscate potential attackers by mixing large amounts of reads, salting the mix with additional fake ones and randomizing the order of all reads. If the noisy sequences are chosen cleverly, it is much more difficult for attackers to identify patients by applying statistical methods. In our pipeline, fake data is randomly picked from a large, fixed set of genome data.

For data security, we finally ensure that data are never stored in persistent memory in the cluster to reduce the number of side channels that could be used by attackers.

With respect to performance, we exploit the fact that sequences can be aligned independent of each other and use a parallel *MapReduce* program. The principle of MapReduce fits very well to our scenario [5]: the "Map" step aligns the sequences and sorts the aligned reads, the "Reduce" step collects and outputs the data chromosome-wise. We chose Hadoop (with YARN) as MapReduce framework. For the alignment we use the *Burrows-Wheeler Aligner* (bwa) [14] and for sorting the *SAMtools* tool suite [16].

The file formats that we use are a result of the design / software decisions. *FASTQ* is the format of the files produced by Illumina sequencers. The Sequence Alignment/Map (SAM) format "is a generic alignment format for storing read alignments against reference sequences" and the Binary Alignment/Map (BAM) format is the binary representation of it [16]. Both formats are supported by the Burrows-Wheeler Aligner and the SAMtools.

V. Pipeline Architecture

This section describes our pipeline architecture which we restrict to the computational part; *i.e.*, we ignore the data acquisition by the sequencers and only consider the process between the point at which the input data in the form of short reads are gathered on computers in the safe environment and the point at which the aligned reads are written back to the safe environment. We assume that the short reads are contained in FASTQ files and that the results are returned in BAM format. Hence, our solution consists of a pipeline that takes FASTQ files as input and outputs the aligned reads as BAM files.

The pipeline is shown in Figure 1 and can be divided into four phases. Each phase takes the output data of the previous one as input and creates a new output. For security and performance reasons, the input data are first *anonymized* and divided into work packages. In Phase II, each work package is processed in its own *Hadoop* job in an HPC system. Back in the secure area, the output is de-anomymized and reordered in the *Decrypt & Unmix* phase. Finally, the *Merger* merges all files of a patient or probe into one final output file.

A detailed explanation of the individual steps is given in the following subsections. The last subsection describes what the user has to do to trigger the pipeline process.

A. Phase I: Anonymizer

In the first phase, the *Anonymizer* reads the input data – consisting of single or paired-ended reads of many patients – from local storage in the secure area, adds encrypted identifiers, salts the input with fake data, and creates work packages (wp) of randomly mixed reads that are written back to local storage. In the next paragraphs we describe these tasks in more detail.

The input is given as files in FASTQ format which can be *gzip*-compressed. If compressed, the file names must end with `.gz`. The Anonymizer can handle single-ended as well as paired-ended reads. It looks for single-ended input files using the regular expression `.+.fastq.*` and and for paired-ended input files using `.+_[1|2].fastq.*`. Two paired-ended input files are assumed to belong together if their prefixes before `_1` and *_2*, repectively, are the same. The shared prefix is then used as the identifier or probe name. The probe name of an input file with single-ended reads is the file name without the `.fastq` ending.

For later processing stages, the patient / probe of each read must be retraceable because the Anonymizer mixes reads from several patients / probes into the same work package. If this information was not added, the later stages could not map the reads back to their respective patients / probes. For this reason, the Anonymizer augments each read (pair) with a patient / probe identifier, which is then piggybacked through the following stages. This is done before encryption. The Anonymizer adds the identifier to the first line of a read entry. More precisely, it prepends the patient / probe identifier to the existing read identifier separating both by a `#`. If, for example, the first line reads `@foo` and if the patient ID is `Pa`, the concatenation becomes `@Pa#foo`.

The anonymization is achieved by combining the following techniques:

1) **Salting**: The actual data consisting of many different probes are salted by throwing dummy reads into the mix. These dummy reads contain, for example, rare SNPs so that individual patients cannot be identified based on special genome segments. An attacker simply cannot distinguish between real and fake data which are processed like real data, but eventually filtered out in Phase III.
2) **Mixing**: Real and fake data are randomly assigned to the work packages which are sent to the cluster and which form the input of Phase II. This way, the attacker cannot gather the relationship of two reads from their positions in the work packages.
3) **Encryption**: The identifier is encrypted using the *Advanced Encryption Standard* (AES) as defined in the

U.S. Federal Information Processing Standards Publication 197 [1][4]. As we cannot maintain the encryption ordering in the later decryption step, the *electronic code book* (ECB) mode is used. To avoid that equal plain text blocks are encrypted into identical ciphertext blocks, we salt the identifiers by adding random bits to them at fixed positions. The Anonymizer supports *AES-128*, *AES-192* and *AES-256* encryption. The respective key of 16, 24 or 32 bytes can be provided in form of an input file. If no (valid) keystore file is given, the Anomizer reads 32 bytes from */dev/random* and uses them as the key. Note that the encrypted identifier is not directly written to the output file as it may include characters which render it invalid for later processing. Therefore, the encrypted data are base64-encoded before they are written to the output file.

B. Phase II: Hadoop

The second phase computes the alignment of the reads with respect to a given reference genome. It is the only one executed in a shared computing center. When the data are sent to the computing center and back, *OpenSSH* is used for encryption. The computation is organized subject to the *MapReduce* model and uses the *Hadoop* framework with *YARN*.

Each work package generated in Phase I is processed by a separate Hadoop job. As explained in Section IV, the Map function computes the alignments for all reads using the Burrows-Wheeler Aligner and sorts the output reads by their position in the reference genome using the SAMtools utilities. After that, the Hadoop partitioner partitions the reads for the Reduce function. Our partitioner partitions them by their chromosomes and sends them to the reducers where each reducer represents one chromosome. The actual Reduce function outputs the data without further processing. For each chromosome one output file is created.

Considered in more detail, a *map task* runs three instances of the Burrows-Wheeler Aligner *bwa*. Assuming paired-ended (single-ended) reads, it performs two `bwa aln` calls which align the reads and one `bwa sampe` (`bwa samse`) call which transforms the results to SAM format.

To further improve security, the YARN environment is configured to use only local RAM disks of the compute nodes. Hence, no read is stored on persistent storage during computation.

C. Phase III: Decrypt & Unmix

The Decrypt & Unmix stage processes the output (chromosome) files of the Hadoop jobs. Our tool generates one output file for every patient / probe and work package, loops through all reads in the input, decrypts their identifiers and sorts them in the right output files. Fake reads are discarded.

Input files are identified by the regular expression `.+_chr.+`, and all other files in the input directory are ignored. It is assumed that the input files are in SAM format, and missing header lines are tolerated. If the tool finds header lines in one of the input files, it copies them to an additional output file named "header". The ordering of the header lines is not changed during this process. The *output files* are also in SAM format (but without header lines). The name of an output file specifies a patient / probe name, a work package name and a chromosome and follows the pattern `<probe>_<workpackage>_<chromosome>`.

The decrypt & unmix tool iterates over the input files and, in each file, over all reads in the given order. For each read, it decrypts its identifier and simply appends the read to the output file of the respective patient / probe, work package and chromosome. If there is no such file, it is created.

D. Phase IV: Merger

The final step in the pipeline is the *Merger* which merges the output files of Phase III such that there is one single file for each patient / probe.

The input has one file for each combination of patient / probe, work package and chromosome. Each file is a list of reads sorted by their position within the chromosome. Hence, it remains to join the data for each patient / probe in one file using a merge operation similar to the one of the *merge sort algorithm*. The Merger uses the regular expression `.+_wp.+` to identify input files. The prefix before the underscore is the identifier of the patient / probe.

The Merger must wait for Phase III to finish and can therefore not run parallel to it. It iterates over the input files in chromosome order and, in each file, over all reads in the given order. For each patient, it first merges the data from all her files belonging to chromosome 1, then from all files belonging to chromosome 2 and so on. In each step, it selects the read with the lowest position and adds it the output file of the according patient.

While the entries in a chromosome file are already sorted by their mapping position in the reference genome, the chromosome order must be established using the file "header", provided that it was created in Phase III, or the reference genome's annotation file (`*.ann`) or sequence dictionary (`*.dict`). If none of these files exist, the tool aborts.

The Merger uses heuristics and parameters to estimate the number of merges that can be performed in parallel and reserves CPU cores and memory accordingly.

E. Pipeline Usage

Before the pipeline process is started, the user has to define input and output directory and provide the data. Once started, the pipeline does not require any interventions from the user. So, the steps are:

1) Define workspace directory (`<workspace>`) and input directory (e.g., `<workspace>/input`).
2) Copy the configuration file to `<workspace>/input` and modify it if necessary.
3) Call the starter script: `starter -i <workspace>/input -w <workspace>`
4) Wait for the pipeline to finish.
5) Obtain results from `<workspace>/results`.

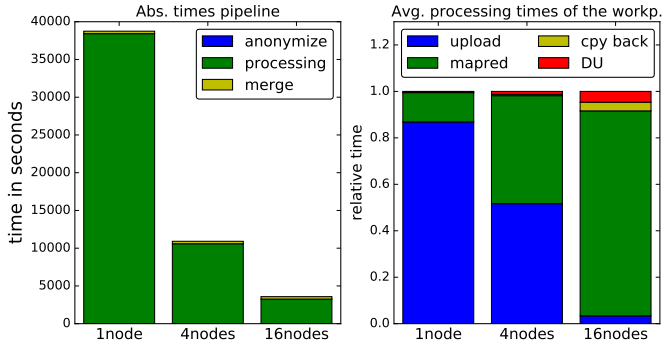

Fig. 2: Left: absolute runtimes of Anonymizer, Merger, and the Hadoop processing in between. Right: runtime for processing the work packages relative to other operations (DU: decrypt & unmix).

The configuration specified in the configuration file is related to the cluster side and can be determined by the provider: Pipeline / cluster settings (size of work packages, #nodes, #threads, #pseudo samples, host name, user group on host, batch queue on host etc.), tool settings (Hadoop, BWA, SAMtools and Java) and the location of additional data (reference genome, pseudo genome data).

During the run, the pipeline creates a heavy IO workload in the workspace directory. Hence, it is advisable to put the workspace on a strong storage backend.

VI. Evaluation

In this section we evaluate whether the system meets the five requirements of Section II. The *first* and the *second one* are obviously fulfilled because we perform all costly operations on a public HPC cluster and we do this without storing any data on disk.

The *third requirement* is met because the data are during transmission to and from the cluster. On the cluster nodes themselves the sequences are readable, but for processing they have to be.

The identifiers of the patients / probes, however, remain encrypted even on the cluster nodes which helps with the *fourth requirement* which is a bit more tricky. While it is not possible to read the identifier without breaking the encryption, it might be possible to identify a person in the mix by reading sufficiently many sequences, mapping them to an old reference genome of that person (provided that the attacker has one) and computing the statistical probability for that person to be in the mix. Chen *et al.* [3] and Homer *et al.* [9] showed that this is possible to a certain degree utilizing the existence of rare SNPs, but that it becomes more and more difficult the more other reads are in the mix. Unfortunately, there is no precise analysis of how many and what data the mix has to be salted with to render this attack ineffective. In our approach we usually at least double the amount of data and pick reads with rare SNPs so that the patients' peculiarities do not stand out.

Concerning the quality of the encryption, we pointed out before that we have to use AES in ECB mode, but that we avoid identical ciphertext blocks in the case of identical plain text blocks by adding random bits to the plain text. For this reason, the encryption techniques, AES and OpenSSH, can be assumed to be strong.

For analyzing the *fifth requirement*, we ran tests in an HPC cluster. The "secure environment" is a remote computer, but connected via a high-speed network link (10 GB/s). Figure 2 shows processing times of the pipeline for an input data set of 237M short reads (100 GB) and different cluster sizes, each node having 64 cores and 128 GB memory. The left plot indicates that the processing scales extremely well. The running time is roughly quartered when the number of nodes is quadrupled.

The right figure shows the relative processing times of the work packages. For a small number of nodes, the upload to the cluster dominates the total time because the cluster nodes can store only a limited number of work packages in their memory and the packages must remain in the secure area until space in the cluster nodes is freed. For a large number of nodes, however, almost all the time is spent on processing, and the overhead of using an outside cluster is neglectable.

In both figures, one can see that the runtimes of Phase I (Anonymizer) and III (Decrypt & Unmix) are relatively short so that the overhead due to anonymizing / encryption and deanonymizing / decryption is acceptable. Note that since work package processing starts as soon as the first work package arrives at the cluster, the runtimes of the Anonymizer and the work package processing also overlap.

VII. Conclusion

In this work we have presented a system for processing sensitive genome data in a public environment without harming the privacy of patients. Our secure genome processing pipeline is already used for processing data of real cancer patients in a university's HPC cluster which is shared with many other users. As shown in our evaluation, the system's processing performance is very good and scales excellently.

Acknowledgment

This work was supported by the German *Federal Ministry of Education and Research* (BMBF) under grant 131A029D (Project "CI3").

References

[1] "Announcing the advanced encryption standard (aes)," Federal Information Processing Standards Publication 197, Tech. Rep., 2001.

[2] 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, 2016.

[3] Y. Chen, B. Peng, X. Wang, and H. Tang, "Large-scale privacy-preserving mapping of human genomic sequences on hybrid clouds," in *NDSS*, 2012.

[4] J. Daemen and V. Rijmen, *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.

[5] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[6] B. J. Fikes. (2017) New Machines Can Sequence Human Genome in One Day. [Online]. Available: http://www.sci-tech-today.com/news/Genome-Sequencing-in-One-Day/story.xhtml?story_id=023001ATQM02

[7] G. S. Ginsburg and H. F. Willard, Eds., *Genomic and Personalized Medicine (Second Edition)*, second edition ed. Academic Press, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B978012382227700121X

[8] J. M. Heather and B. Chain, "The sequence of sequencers: The history of sequencing {DNA}," *Genomics*, vol. 107, no. 1, pp. 1 – 8, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0888754315300410

[9] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig, "Resolving individuals contributing trace amounts of dna to highly complex mixtures using high-density snp genotyping microarrays," *PLoS Genet*, vol. 4, no. 8, pp. 1–9, 08 2008.

[10] S. Kreiter, M. Vormehr, N. van de Roemer, M. Diken, M. Löwer, J. Diekmann, S. Boegel, B. Schrörs, F. Vascotto, J. C. Castle, A. D. Tadmor, S. P. Schoenberger, C. Huber, Özlem Türeci, and U. Sahin, "Mutant mhc class ii epitopes drive therapeutic immune responses to cancer," *Nature*, vol. 520, pp. 692 – 696, Mar. 2015.

[11] S. T. Krishnan and J. U. Gonzalez, *Building Your Next Big Thing with Google Cloud Platform: A Guide for Developers and Enterprise Architects*, 1st ed. Berkely, CA, USA: Apress, 2015.

[12] B. Langmead and S. L. Salzberg, ""fast gapped-read alignment with bowtie 2."," *Nature methods*, vol. 9, no. 4, p. 357–359, Mar 2012.

[13] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short dna sequences to the human genome," *Genome Biology*, vol. 10, no. 3, p. R25, 2009.

[14] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows-Wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, Jul 2009. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btp324

[15] ——, ""fast and accurate long-read alignment with burrows–wheeler transform."," *Bioinformatics*, vol. 26, no. 5, p. 589–595, Mar 2010.

[16] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin, "The Sequence Alignment/Map Format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, Aug. 2009. [Online]. Available: http://dx.doi.org/10.1093/bioinformatics/btp352

[17] Y. Liu and B. Schmidt, "Long read alignment based on maximal exact match seeds," *Bioinformatics*, vol. 28, no. 18, pp. i318–i324, 2012.

[18] Y. Liu, B. Schmidt, and D. L. Maskell, "Cushaw: a cuda compatible short read aligner to large genomes based on the burrows–wheeler transform," *Bioinformatics*, vol. 28, no. 14, p. 1830, 2012.

[19] S. Marguerat and J. Bähler, "Rna-seq: from technology to biology." *Cell Mol Life Sci*, vol. 67, no. 4, pp. 569–579, Feb 2010. [Online]. Available: http://dx.doi.org/10.1007/s00018-009-0180-6

[20] S. Moorthie, C. J. Mattocks, and C. F. Wright, "Review of massively parallel dna sequencing technologies," *Hugo Journal*, vol. 5, no. 1-4, pp. 1 – 12, 2011.

[21] R. Nadipalli, *HDInsight Essentials*, 2nd ed. Packt Publishing, 2015.

[22] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/MSST.2010.5496972

[23] A. Singh and V. Rayapati, *Learning Big Data with Amazon Elastic MapReduce*. Packt Publishing, 2014.

[24] M. Snir, S. W. Otto, D. W. Walker, J. Dongarra, and S. Huss-Lederman, *MPI: The Complete Reference*. Cambridge, MA, USA: MIT Press, 1995.

[25] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523633

[26] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

[27] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863103.1863113