Using Geometric Structures to Improve the Error Correction Algorithm of High-Throughput Sequencing Data on MapReduce Framework

Wei-Chun Chung*†‡, Yu-Jung Chang*, D. T. Lee*‡§, Jan-Ming Ho*†
*Institute of Information Science, Academia Sinica, Taiwan
†Research Center for Information Technology Innovation, Academia Sinica, Taiwan
‡Department of Computer Science and Information Engineering, National Taiwan University, Taiwan
§Department of Computer Science and Information Engineering, National Chung Hsing University, Taiwan
{wcchung, yjchang, dtlee, hoho}@iis.sinica.edu.tw

***Abstract*—Next-generation sequencing (NGS) data are a rapidly growing example of big data and a source of new knowledge in science. However, sequencing errors remain unavoidable and reduce the quality of NGS data. Error correction, therefore, is a critical step in the successful utilization of NGS data, including *de novo* genome assembly and DNA resequencing. Since NGS throughput doubles approximately every five months and the length of NGS records (i.e., reads) is increasing, improvements in efficiency and effectiveness of computational strategies are needed. In this study, we aim to improve the performance of CloudRS, an open-source MapReduce application designed to correct sequencing errors in NGS data. We introduce the read-message (RM) diagram to represent the set of messages, i.e., the key-value pairs generated on each read. We also present the Gradient-number Votes (GNV) scheme in order to trim off portions of the RM diagram, thereby reducing the total size of messages associated with each read. Experimental results show that the GNV scheme successfully reduce execution time and improve the quality of the *de novo* genome assembly.**

Keywords*—big data; error correction; geometric structure; mapreduce; next-generation sequencing

I. Introduction

Next-generation sequencing (NGS) has improved the quality of DNA sequencing [1] and has become one of the core technologies in genetics and computational biology. NGS technology is constantly advancing, as its throughput doubles approximately every five months. In contrast, the throughput of semiconductor technology doubles approximately every two years according to Moore's law [2]. It is estimated that 15 petabytes of compressed genetics data are produced per year by roughly 2000 sequencers [3]. In addition to the huge quantity of sequencing data produced in one year, the lengths of NGS reads are increasing as NGS technology advances. Broad coverage of sequencing data has resulted in a large amount of data for analysis, leading to various computational problems in conventional NGS applications, such as insufficient computing resources and unexpectedly long execution time [4]. However, sequencing errors still occur unavoidably and frequently [5].

The correction of sequencing errors is a crucial preprocessing step for the success of NGS applications such as *de novo* genome assembly and genome resequencing. The general idea for correcting errors at a specific genomic position is based on utilizing the nucleotide that are highly represented over others in the sequences [6]. Various approaches are used for accomplishing the error correction problem. Examples include k-mer-spectrum-based approaches [7]–[11], suffix-tree/array-based approaches [12], [13], and multiple-sequence-alignment (MSA)-based approaches [14]–[16]. In the k-mer-spectrum and suffix-tree/array-based approaches, k-mers or variable-length sub-reads with low frequencies are treated as untrusted. The untrusted k-mers/sub-reads are then converted to trusted ones with high frequencies using minimal changes (as measured by Hamming distance or edit distance, for example) to their reads. The MSA-based approaches derive a multiple sequence alignment for reads with common k-mers and then find the consensus of the alignment to correct errors in the mismatched sections.

Current challenges of the error correction problem include the growing data size and the increasingly longer reads in NGS. Larger data size leads to longer execution time. In some cases, this can be dealt with by increasing the size of physical memory. Long reads enlarge the search space for finding sequencing errors and thus increase the time required to fix sequencing errors. Big data technology, which is useful in processing NGS data [2], can also be employed to expedite the correction of sequencing errors. Example includes CloudRS [16], an MSA-based error correction application for NGS big data based on the MapReduce framework [17].

MapReduce is a prominent big data framework. It provides a number of features for handling distributed computation in the cloud. The MapReduce programming model consists of mapper and reducer functions, where data is encapsulated into messages containing a *key-value* pair for computation and transmission. Thus, the execution time of a MapReduce job is related to the total size of the messages passed among the mapper and reducer functions. Therefore, by reducing the number of messages, we have a means of reducing disk and network I/O overhead in the execution of MapReduce applications.

In this paper, we examine strategies for improving the efficiency and effectiveness of CloudRS, an open-source MapReduce application designed to correct sequencing errors in NGS data. We introduce a geometric model, the read-message (RM) diagram, to represent the messages generated on each read. We then propose the Gradient-number Votes (GNV) scheme to trim the RM diagram in order to reduce its size and thereby reduce the size of each message accordingly. The GNV scheme is implemented in CloudRS and is further integrated into a *de novo* genome assembly pipeline. Experimental results show that our proposed scheme successfully reduces the execution time of CloudRS. The scheme also improves the quality of *de novo* genome assembly produced by Velvet [18], as shown by an evaluation based on the GAGE benchmark [19].

II. Background

A. Terminology

DNA sequencing aims at determining the precise order of the DNA bases, which contain adenine (*A*), thymine (*T*), cytosine (*C*), and guanine (*G*). A pair of *A* and *T* or a pair of *G* and *C* is known as a base pair (the Watson-Crick base pair; denoted as *bp*). The output of DNA sequencer is a set of sequences, which is also denoted as *reads*. A *long read* represents its sequence length is longer than 50 base pairs. A read of the sequencer output is composed of four elements including the sequence, quality information of each base, sequence identifier, and optional descriptions. Thus, a read with sequence S of size n and quality information Q can be written as $(\{S_1, S_2, \ldots, S_n\}, \{Q_1, Q_2, \ldots, Q_n\})$.

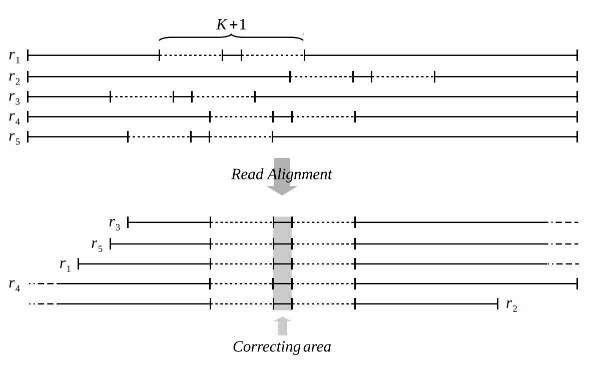
In this paper, a set of reads is said to be aligned on a given pattern if every read of the set has the given pattern as its substring. A *ReadStack* refers to a set of reads aligned by the same pattern. Moreover, the k-mer spectrum refers to the empirical frequencies of the all the possible patterns of length k.

B. The CloudRS algorithm

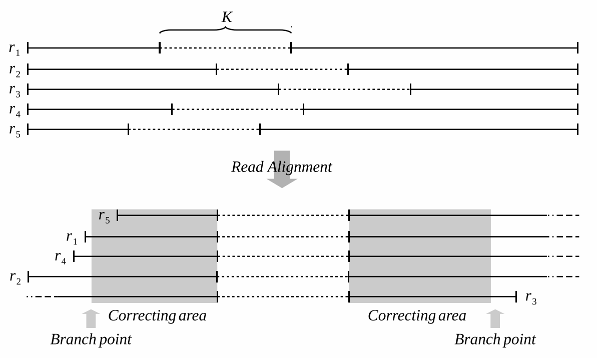
The CloudRS algorithm is designed to correct sequencing errors and thus to improve the quality of the subsequent *de novo* assembly. It emulates the concept of the error correction algorithm of ALLPATHS-LG [20] on the MapReduce framework, and applies a two-stage majority voting mechanism for correcting errors in each ReadStack.

The CloudRS algorithm is implemented with several modules each consisting of MapReduce runs with at least one mapper and/or reducer task. There are two core modules for correcting errors, including *PreCorrect* and *ErrorCorrect*. Fig. 1(a) shows the function of the PreCorrect module. The reads are aligned by a $(k+1)$-mer pattern, which contains a wildcard genomic base at the central position. The central position also represents the correcting area of the ReadStack. Fig. 1(b) shows the function of the ErrorCorrect module of CloudRS. The ReadStack is formed by aligning the k-mer pattern of the reads. There are two areas for correcting errors before reaching the branch points defined by a specific threshold. After forming the ReadStack, the algorithm then collects the quality information of each position in the correcting area to make a correction recommendation.

The first stage of majority voting applied on each ReadStack is shown in Fig. 2. Quality information at each position in the correcting area of the ReadStack is collected and analyzed to check if a consensus may be identified. Then, the consensus is used to make correction recommendation for each member of the ReadStack at the specific position, namely

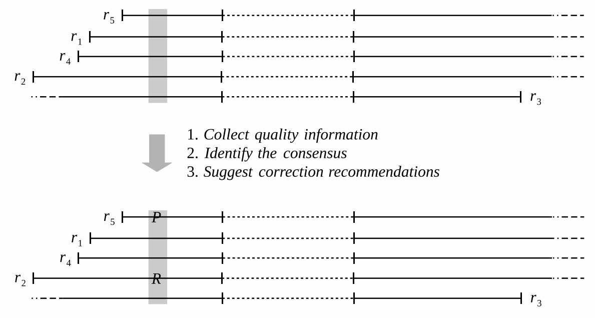


(a) *PreCorrect* module



(b) *ErrorCorrect* module

Fig. 1. An illustration of *ReadStack* construction and the range for correcting errors (shaded area) in each CloudRS module.



r_5
r_1
r_4
r_2
r_3
1. Collect quality information
2. Identify the consensus
3. Suggest correction recommendations
r_5
P
r_1
r_4
r_2
R
r_3

Fig. 2. The majority voting mechanism for correcting errors at the ReadStack level. There are three steps in making a correction recommendation of either to *Preserve* (*P*) or to *Replace* (*R*) bases of each member of the ReadStack.

to *Preserve* or to *Replace* it. A Preserve recommendation is suggested for a member that matches the consensus but has a quality score lower than a specified threshold. A Replace recommendation is suggested for a member that differs from the consensus. The second majority voting stage is issued at each position on each read. The recommendations made on the ReadStack level are collected and analyzed for each position of a read. A correction is issued if all the recommendations are in agreement to Replace, that is, to replace the base of the position by another base. Otherwise, a Preserve recommendation is made to maintain the current base of the read position.

C. *The bottleneck of CloudRS*

The CloudRS algorithm is based on the multiple sequence alignment (MSA) approaches. Reads are aligned by the same k-mer pattern, and the quality information of each base is used to correct possible errors. Given a dataset of D reads, each read having length L, the algorithm will produce messages with a magnitude of $\mathcal{O}(D \times (L-K+1))$, where K is the length of the k-mer pattern. Note that the messages should be at least twice of the magnitude, since it incorporates the DNA sequence and its accompanying quality information.

Due to the increasing data size, the computing and transmitting steps become bottlenecks in improving CloudRS. In the mapper function, the size of the messages can be 10 times larger than the input data size. Moreover, the size of messages transmitted from mappers to reducers may be larger than the size produced in the mappers, due to the nature of distribution and replication in the MapReduce framework. Therefore, reducing the number of messages in execution may also reduce execution time in CloudRS; it will also be effective when processing NGS big data and with long reads. However, reducing the number of messages may lead to data loss or incomplete information in an MSA-based error correction algorithm. It may also affect the result of the following *de novo* assembly. Therefore, to reduce the execution time of CloudRS without affecting the accuracy of assemblies becomes our primary goal.

III. Methodology

In this section, we present the read-message (RM) diagram to portray the messages composed of key-value pairs generated on each read of NGS data. We hypothesize that the area of the RM diagram is proportional to the execution time of CloudRS, a MapReduce application for correcting NGS sequencing errors. Here we present the basis scheme used in CloudRS. We then introduce our editing scheme to tailor the total area of the RM diagram by controlling the number of votes at each position of the read.

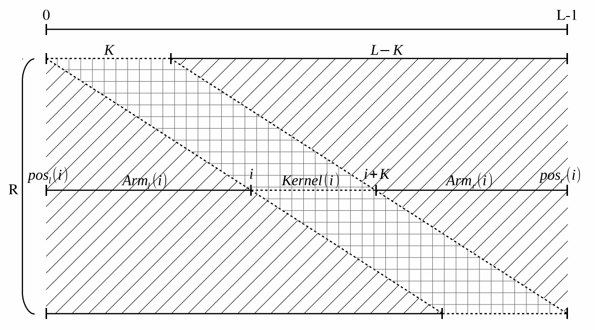
A. *The RM diagram*

Given a read of length L and an integer K, where $0 <= K <= L$, we define the RM diagram as follows: In the RM diagram, there are $R = L - K + 1$ records. Each record i consists of a left-side arm, a center kernel, and a right-side arm, denoted as $Arm_l(i)$, $Kernel(i)$, and $Arm_r(i)$, respectively, and are defined as follows:

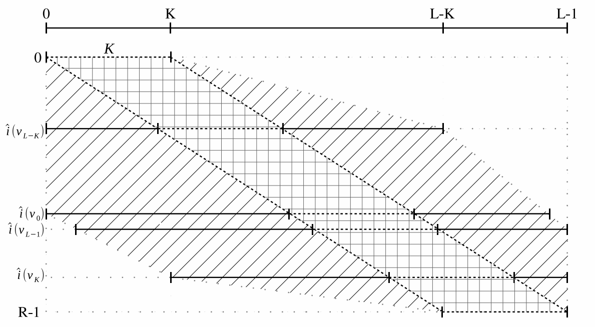
$$\begin{array}{lcl} Arm_l(i) & = & [pos_l(i), i) \\ Kernel(i) & = & [i, i+K) \\ Arm_r(i) & = & [i+K, pos_r(i)) \end{array} \quad (1)$$

where $pos_l(i)$ is the start index of Arm_l and $pos_r(i)$ is the end index of Arm_r on the ith record, $0 \leq i \leq R-1$. Fig. 3 gives an example of an RM diagram. The diagonally shaded area represents the accumulated votes participating from different alignments of k-mer patterns. Note that each mer of the read is said to possess v votes if it appears exactly v times in Arm_l or Arm_r of the RM diagram. Thus, we define the *number of votes* v_p at read position p as

$$v_p = |\{x = p \mid x \in Arm_l(i) \ or \ x \in Arm_r(i), r_i \in \mathcal{R}\}| \quad (2)$$



(a) The Full-length Arm scheme



(b) The Gradient-number Votes scheme

Fig. 3. Example diagrams showing geometric structures representing our proposed schemes. The diagonally shaded area represents the total number of votes, the lattice area is generated by the constant k-mer patterns, and the white area enclosed by dotted lines represents the reduction in data compared with the original RM diagram.

where $0 \leq p \leq L-1$. We also use $\hat{\imath}(v_p)$ to estimate the key index of a specific record in order to construct v_p votes at position p.

B. *CloudRS: the Full-length Arm scheme*

We first present the fundamental model, the *Full-length Arm* (FLA) scheme, as shown in Fig. 3(a). FLA applies full read length around the kernel as its arms. The boundary of FLA can be obtained as follows:

$$\begin{array}{l} pos_l(i) = 0 \\ pos_r(i) = L \end{array}$$

Note that the FLA scheme is used in the current release of CloudRS and is our basis for benchmarking.

C. *The Gradient-number Votes scheme*

Fig. 3(b) illustrates another treatment for the distribution of votes, called the *Gradient-number Votes* (GNV) scheme. In GNV, the distribution of votes is a gradient function. The gradient can be controlled by the numbers of votes at the start and end positions. The key positions and their respective votes are:

$$\begin{array}{ll} v_0 & = A + h \\ v_K & = A + \lceil \frac{R(V_0 - A)}{L} \rceil \\ v_{L-K} & = A + \lceil \frac{K(V_0 - A)}{L} \rceil \\ v_{L-1} & = A \end{array}$$

TABLE I. DATASETS

Dataset	SRA accession number	Reference genome (accession number)	Genome length (M bp)	Read length (bp)	# Reads* (M)	Genome coverage	Original data size (GB)	Compressed data size*+ (GB)
D1	SRX100885	*S. cerevisiae* (PRJNA128)	12.07	76	50.61	319x	13.0	7.7
D2	SRR022866	*S. aureus* (NC_003923)	2.82	76	24.13	651x	7.1	3.7
D3	SRR034509	*E. coli* (NC_000913)	4.64	101	17.49	381x	6.4	3.5
D4	-	*E. coli* (NC_000913)	4.64	151	12.06	393x	3.9	3.6

* Reads containing one or more *N* symbol are removed before evaluation.
+ Optional descriptions of the FASTQ format are removed.

where A is the base number of votes, $0 \leq A \leq R$, and h is a value for fine-tuning the votes. In addition, the record index of the key positions is obtained as follows:

$$\begin{aligned} \hat{i}(v_0) &= v_0 + 1 \\ \hat{i}(v_K) &= K + v_K \\ \hat{i}(v_{L-K}) &= R - (K + v_{L-K}) \\ \hat{i}(v_{L-1}) &= R - v_{L-1} - 1 \end{aligned}$$

Thus, the geometric boundaries of GNV are described as follows:

$$pos_l(i) = \begin{cases} 0, & i < i(v_0) \\ 0 + \left\lfloor \frac{(i-\hat{i}(v_0)+1)(K-1)}{\hat{i}(v_K)-\hat{i}(v_0)} \right\rfloor, & \hat{i}(v_0) \leq i \\ & i < \hat{i}(v_K) \\ K + \left\lfloor \frac{(i-\hat{i}(v_K)+1)(L-2K-1)}{R-\hat{i}(v_K)+1} \right\rfloor, & \text{otherwise} \end{cases}$$

$$pos_r(i) = \begin{cases} K + \left\lfloor \frac{(i+1)(L-2K-1)}{\hat{i}(v_{L-K})} \right\rfloor, & i < \hat{i}(v_{L-K}) \\ R - 1 + \left\lfloor \frac{(i-\hat{i}(v_{L-K})+1)(K-1)}{\hat{i}(v_{L-1})-\hat{i}(v_{L-K})} \right\rfloor, & \hat{i}(v_{L-K}) \leq i \\ & i < \hat{i}(v_{L-1}) \\ L, & \text{otherwise} \end{cases}$$

IV. EVALUATION

A. Dataset

We evaluated our proposed scheme with long read datasets sequenced by Illumina sequencers, as shown in Table I. Datasets D1, D2, and D3 were downloaded from the sequence read archive (SRA) in NCBI. These datasets are generated from resequencing experiments of *a priori* genome sequences and therefore are often used for evaluation purposes in NGS. Dataset D4 is from Illumina and includes the well-characterized *E. coli* strain K-12 MG1655 library sequenced on the Illumina MiSeq platform.

B. Criteria and setup

We designed two criteria for scheme evaluation: efficiency and effectiveness. To assess the efficiency of our proposed scheme, we used the number of votes and the corresponding execution time of CloudRS. To evaluate the effectiveness, we followed the experiments in the CloudRS paper [16]. We used Velvet [18] followed by the GAGE [19] benchmark to evaluate the quality of genome assemblies. The major assembly quality index used in the GAGE benchmark is the value of *corrected N50* in the N50 statistic. A larger corrected N50 value indicates that the assembled sequences, i.e., contigs, are longer and may be free from assembly errors. Since CloudRS is a variation of the error correction algorithm of ALLPATHS-LG [20], ALLPATHS-LG was also included in the evaluation. Note that the kernel length is fixed at $K = 24$. The hash value used in Velvet was specified as follows: 39 for datasets D1 and, 51 for dataset D3, and 97 for dataset D4.

TABLE II. COMPARISON WITH SIMILAR METHODS

Dataset	Method*	# Votes+	Execution time (s)+	Corrected N50 (bp)
D1	RawData	N/A	N/A	3016
	CloudRS	2756	12027	3153
	ErrorCorrectReads	N/A	13104	**3197**
	Present study	1534	**9209**	3163
D2	RawData	N/A	N/A	16665
	CloudRS	2756	5524	18310
	ErrorCorrectReads	N/A	4788	**20266**
	Present study	972	**3807**	**20266**
D3	RawData	N/A	N/A	30833
	CloudRS	6006	5743	53748
	ErrorCorrectReads	N/A	5940	**82554**
	Present study	1317	**3392**	59809
D4	RawData	N/A	N/A	87096
	CloudRS	16256	17751	93798
	ErrorCorrectReads	N/A	10584	87096
	Present study	1317	**5389**	**112425**

* RawData is the uncorrected reads. CloudRS is the latest development release obtained from developer teams. ErrorCorrectReads is the error correction program provided by ALLPATHS-LG.
+ N/A means that the information is not available from the program.

Our evaluation environment contains a set of machines for Hadoop and a high memory machine for ALLPATHS-LG. The Hadoop cluster consists of 10 nodes; each node has two Intel 2.33-GHz Xeon E5410 processors, 16 GB of RAM, a gigabit Ethernet port, and a 500 GB disk. The Hadoop cluster has been set with maximum 70 mappers and 70 reducers, each having 950 MB of RAM for processing. The computer designated for running ALLPATHS-LG has two Intel 2.33-GHz Xeon E5410 processors and 64 GB of RAM.

C. Results

Table II lists the comparison with similar methods, including the raw data (uncorrected read; denoted as *RawData*), the latest development version of *CloudRS*, the error correction program of ALLPATHS-LG (*ErrorCorrectReads*), and the Gradient-number Votes scheme (denoted as *Present study*).

All methods outperformed the assembly results from the RawData. Comparing with CloudRS, our method improved 10% of corrected N50 length in dataset D2 and D3, and 20% of corrected N50 length in datasets D4. We then compared our results with ErrorCorrectReads. In dataset D1, our method obtained comparable results with ErrorCorrectReads, which produced the best result of corrected N50 length. In dataset D2, our method had the same corrected N50 length as that of ErrorCorrectReads. In dataset D3, ErrorCorrectReads obtained the best corrected N50 length, about 38% higher than with our method. Finally, in dataset D4, our method had the best corrected N50 length, and our method improved the corrected N50 length of ErrorCorrectReads by 29%.

We also compared the execution time of these methods. Our method successfully improved the efficiency by obtaining the shortest execution time among all methods. For dataset D4 with a read length of 151 base pairs, our method was able to reduce the execution time of CloudRS by nearly 70%, and outperformed ErrorCorrectReads by nearly 49%.

V. Related work

Document indexing and similarity detection are related research topics. Sequence alignment and matching techniques are applied to measure document similarity [21], [22]. These techniques can also be extended to detect plagiarism, copying, or duplicate text/near-sentences in text mining [23]–[25].

While MapReduce is a state-of-the-art big data processing model, there are numerous strategies to be considered for improving the MapReduce framework. One example is to include a grouping of the key-value pair using a sub-key to reduce the amount of intermediate data [26]. Another example is to develop a distributed indexing storage system to improve the performance of document indexing in the MapReduce framework [27].

VI. Conclusion

In this study, we addressed the important issue of error correction in NGS big data including long reads. To improve efficiency and reduce execution time, we introduced the RM diagram for designing a proper computational structure for analysis. The key idea is to generate a minimal set of data and thereby reduce the number of data transmission and execution operations.

We proposed the Gradient-number Votes scheme adapted using the RM diagram for use with CloudRS. Our evaluation results show that the proposed scheme successfully reduces the amount of data and the execution time as well as improve the effectiveness. In our comparison with the error correction algorithm in ALLPATHS-LG, the results show that our proposed scheme attains a level of performance equal to that of existing methods. Moreover, our proposed scheme performed remarkably even in long reads.

We plan to probe the relationship between the RM diagram and the error profiles of NGS data in detail. Since a proper computational structure improves the performance of analyses, our proposed method is applicable in other areas as well. In addition, we would like to investigate the feasibility of applying the RM diagram to the *de novo* assembly.

Acknowledgment

The authors wish to thank the anonymous reviewers, and Jen-Hao Cheng and Dr. Chung-Yen Lin (Institute of Information Science, Academia Sinica, Taiwan) for their help, suggestions, and valuable comments. This research is partially supported by Ministry of Science and Technology of Taiwan under grant NSC 102-2221-E-001-013-MY3, and System Management and Content Retrieval Technologies for Supporting Cloud-based Digital Archive Systems and Services of Academia Sinica.

References

[1] M. L. Metzker, “Sequencing technologies - the next generation,” *Nature reviews. Genetics*, vol. 11, no. 1, p. 3146, January 2010.

[2] L. Stein, “The case for cloud computing in genome informatics,” *Genome Biology*, vol. 11, no. 5, p. 207, 2010.

[3] M. Schatz and B. Langmead, “The dna data deluge,” *Spectrum, IEEE*, vol. 50, no. 7, pp. 28–33, July 2013.

[4] Q. Zou, X.-B. Li, W.-R. Jiang, Z.-Y. Lin, G.-L. Li, and K. Chen, “Survey of mapreduce frame operation in bioinformatics,” *Briefings in Bioinformatics*, 2013.

[5] K. Robasky, N. E. Lewis, and G. M. Church, “The role of replicates for error mitigation in next-generation sequencing,” *Nat Rev Genet*, vol. 15, no. 1, pp. 56–62, Jan. 2014.

[6] X. Yang, S. P. Chockalingam, and S. Aluru, “A survey of error-correction methods for next-generation sequencing.” *Briefings in Bioinformatics*, vol. 14, no. 1, pp. 56–66, 2013.

[7] D. R. Kelley, M. C. Schatz, S. L. Salzberg *et al.*, “Quake: quality-aware detection and correction of sequencing errors,” *Genome Biol*, vol. 11, no. 11, p. R116, 2010.

[8] X. Yang, K. S. Dorman, and S. Aluru, “Reptile: representative tiling for short read error correction.” *Bioinformatics*, vol. 26, no. 20, pp. 2526–2533, 2010.

[9] P. Medvedev, E. Scott, B. Kakaradov, and P. Pevzner, “Error correction of high-throughput sequencing datasets with non-uniform coverage,” *Bioinformatics*, vol. 27, no. 13, pp. i137–i141, 2011.

[10] Y. Liu, J. Schrder, and B. Schmidt, “Musket: a multistage k-mer spectrum-based error corrector for illumina sequence data.” *Bioinformatics*, vol. 29, no. 3, pp. 308–315, 2013.

[11] L. Ilie and M. Molnar, “Racer: Rapid and accurate correction of errors in reads.” *Bioinformatics*, vol. 29, no. 19, pp. 2490–2493, 2013.

[12] J. Schrder, H. Schrder, S. J. Puglisi, R. Sinha, and B. Schmidt, “Shrec: a short-read error correction method.” *Bioinformatics*, vol. 25, no. 17, pp. 2157–2163, 2009.

[13] L. Ilie, F. Fazayeli, and S. Ilie, “Hitec: accurate error correction in high-throughput sequencing data.” *Bioinformatics*, vol. 27, no. 3, pp. 295–302, 2011.

[14] L. Salmela and J. Schrder, “Correcting errors in short reads by multiple alignments.” *Bioinformatics*, vol. 27, no. 11, pp. 1455–1461, 2011.

[15] W.-C. Kao, A. H. Chan, and Y. S. Song, “ECHO: A reference-free short-read error correction algorithm,” *Genome Research*, vol. 21, no. 7, pp. 1181–1192, Jul. 2011.

[16] C.-C. Chen, Y.-J. Chang, W.-C. Chung, D.-T. Lee, and J.-M. Ho, “CloudRS: An error correction algorithm of high-throughput sequencing data based on scalable framework.” in *BigData Conference*. IEEE, 2013, pp. 717–722.

[17] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters.” *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[18] D. Zerbino and E. Birney, “Velvet: algorithms for de novo short read assembly using de bruijn graphs.” *Genome Res*, vol. 18, p. 821, 2008.

[19] S. L. Salzberg, A. M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T. J. Treangen, M. C. Schatz, A. L. Delcher, M. Roberts, G. Marçais, M. Pop, and J. A. Yorke, “GAGE: A critical evaluation of genome assemblies and assembly algorithms,” *Genome Research*, vol. 22, no. 3, pp. 557–567, Mar. 2012.

[20] S. Gnerre, I. Maccallum, D. Przybylski, F. J. Ribeiro, J. N. Burton, B. J. Walker, T. Sharpe, G. Hall, T. P. Shea, S. Sykes, A. M. Berlin, D. Aird, M. Costello, R. Daza, L. Williams, R. Nicol, A. Gnirke, C. Nusbaum, E. S. Lander, and D. B. Jaffe, “High-quality draft assemblies of mammalian genomes from massively parallel sequence data,” *Proc Natl Acad Sci U S A*, vol. 108, no. 4, pp. 1513–1518, 2011.

[21] Q. Zhang, Y. Zhang, H. Yu, and X. Huang, “Efficient partial-duplicate detection based on sequence matching.” in *SIGIR*. ACM, 2010, pp. 675–682.

[22] A. Sediyono and K. R. Ku-Mahamud, “Algorithm of the longest commonly consecutive word for plagiarism detection in text based document.” in *ICDIM*. IEEE, 2008, pp. 253–259.

[23] Y. R. Lee, B. Kang, and E. G. Im, “Function matching-based binary-level software similarity calculation,” in *Proceedings of the 2013 Research in Adaptive and Convergent Systems*, ser. RACS ’13. New York, NY, USA: ACM, 2013, pp. 322–327.

[24] N. Shivakumar and H. Garcia-Molina, “Scam: A copy detection mechanism for digital documents.” in *DL*, 1995.

[25] F. Bravo-Marquez, G. L’Huillier, S. A. Ros, and J. D. Velsquez, “A text similarity meta-search engine based on document fingerprints and search results records.” in *Web Intelligence*. IEEE Computer Society, 2011, pp. 146–153.

[26] J. D. Ullman, “Designing good mapreduce algorithms.” *ACM Crossroads*, vol. 19, no. 1, pp. 30–34, 2012.

[27] R. McCreadie, C. Macdonald, and I. Ounis, “Mapreduce indexing strategies: Studying scalability and efficiency.” *Inf. Process. Manage.*, vol. 48, no. 5, pp. 873–888, 2012.