ScSpark: Low-Redundancy Disk Access and High-Performance Tool for the Single-Cell RNA Sequenceing Data Processing

Michael Shell, *Member, IEEE,* John Doe, *Fellow, OSA,* and Jane Doe, *Life Fellow, IEEE*

Abstract—High-throughput single-cell RNA sequencing (scRNA-seq) data processing pipelines typically integrate multiple modules to transform raw scRNA-seq data to gene expression matrices, including barcode processing, sequence quality control, genome alignment and transcript counting. With the drastic increase in scRNA-seq data size, file input/output (IO) has become a bottleneck of the processing speed of available tools. In this study, we took advantage of Apache Spark's in-memory computing and inherent scalability to develop a new Java-based scRNA-seq data processing pipeline, named scSpark. To reduce unneccessary disk access while reading FASTQ files and writing SAM files, we used Java Native Interface (JNI) to deliver FASTQ Resilient Distributed Datasets (RDD), and then retrieved genome mapping results to SAM RDD. By allocating scRNA-seq data and processing tasks to computer cluster, the scSpark toolkit can significantly reduce disk access for saving and loading temporary results. To test the performance of scSpark, we built a spark cluster on Aliyun, and evaluated its computational performance and biological analysis robustness with several state-of-the-art data processing pipelines. The results indicate that scSpark is more efficient and more scalable than the other available tools.

Index Terms—Intermediate Data, Fast, Scalable.

1 Introduction

SINGLE cell is the fundamental unit of a living organism. In the era of precision medicine, exploring the gene expression at single cell level has become crucial. In the past decade, RNA-seq has been widely used to study gene expression patterns in large-scale biological samples. However, the resolution of bulk RNA-seq could only reach the average level of cell populations. Single-cell RNA sequencing(scRNA-seq) essentially reveals the transcriptional status at single cell level, which provides the basis for subsequent bioinformatics analysis [?].

High-throughput(HT) scRNA-seq protocols, which enable transcript sequencing of thousands of cells simultaneously in a single experiment, have emerged as powerful tools to identify and characterize cell types in complex and heterogeneous tissues [?]. To allow the reads to be demultiplexed after the cells being assembled for sequencing [?], two important oligonucleotide barcodes, namely, cell barcode (CB) and unique molecular identifier (UMI) are introduced in scRNA-seq [?] [?]. CB, which assigns different sequences to each cell for transcript source retrieval after sequencing, greatly improves the throughput and reduces the cost of scRNA-seq [?] [?]. In addition, UMIs are random oligonucleotide barcodes that are used in scRNA-seq experiments [?] [?] to distinguish redundant transcript generated from PCR amplification [?].

In scRNA-seq experiment, it is the usage of these multi-level barcodes that presented additional challenges for data processing, which was quite different from traditional bulk RNA-seq and low-throughput scRNA-seq experiments. In recent years, researchers have developed multiple scRNA-seq data processing tools, typically implementing steps including barcode processing, sequence quality control, genome alignment and transcript counting to convert raw scRNA-seq data into a gene expression matrix for further downstream analysis. The first step is to extract barcodes from a pre-designed nucleotide site for different scRNA-seq protocols. Then, FASTQ reads with low quality nucleotides are filtered based on user-defined thresholds. Subsequently, the remaining FASTQ reads are mapped to the reference genome using the splice-aware aligner, such as STAR [?] or HISAT2 [?]. Finally, the reads are assigned to genes and count matrices for UMIs are generated. [?].

Nowadays, there are many tools to preprocess scRNA-seq data, among which the most popular ones inlude Cell-Ranger [?], UMI-tools [?], STARsolo [?], etc. CellRanger is a highly integrated data processing software tool tailored by 10X Genomics for scRNA-seq data analysis, and it presented best performance on the machine with enough CPU and memory resources for big datasets, while performed slightly poor for small datasets [?]. In addition, UMI-tools [?] is an open source tool with an impressively clear process. And it had relatively higher trascript quantification accuracy compared with other tools in a previous evalution [?]. Furthermore, STARsolo [?] has improved the parallelism of sequence mapping and counting procedures, and achieved good performance on a single machine. Great improvement has been achieved recently for the development of scRNA-seq data processing tools. However, all the avaliable scRNA-seq data processing tools can only run on a single machine with limited running speed, and cannot be extended to

- *M. Shell was with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332.*
E-mail: see http://www.michaelshell.org/contact.html
- *J. Doe and J. Doe are with Anonymous University.*

Manuscript received April 19, 2005; revised August 26, 2015.

multiple nodes.

In recent years, there have been many studies focusing on the optimization of next generation sequencing (NGS) data processing using big data framework like Hadoop [?] and Spark [?]. Due to higher efficiency in utilizing memory computing, Spark is a better big data computing framework than Hadoop's MapReduce [?], [?]. SparkBWA [?] and GPF [?] are good frameworks with Spart for improving the efficiency of NGS data processing. Falco [?] has tried to use Spark in scRNA-seq data preprocessing, but it has two limitations.

Firstly, Falco can't deal with CBs and UMIs, so it is incompatible with any HT scRNA-seq protocol. Secondly, Falco only uses Spark to simply concatenate the aligner and the feature quantitative softwares, which does not reduce the amount of disk reads and writes during alignment and transcript counting. Falco's operations have lots of redundancy disk access which causes its insufficient utilization of Spark's in-memory computation well.

The increase of scRNA-seq datasets require more efficient and faster data processing tools. However, the current scRNA-seq data preprocessing tools were designed without considering of the scalability, which could only run on a single computer and could not be extended to a cluster. From UMI-Tools and CellRanger to STARsolo, parallelism and performance has increased; However, due to the limit of a single machine, all of them have no scalability . The traditional single machine serial processing mode has been unable to meet the demand of computing and storage resources for super-large scale scRNA-seq data. Different with Falco, we enhanced STAR's file I/O interface to reduce redundant disk access. In the end, to achieve parallel and in-memory computing, scSpark implemented transcript counting by combining Spark function and our method that can distributed cumpute the result and generated by a graph algorithm. Our work utilized Spark system architecture to develop a quick and scalable scRNA-seq data preprocessing tool.

2 METHOD

2.1 Overview

We implemented barcode and umi correct, sequence quality control, genome alignment, and transcript counting based on Spark framework. We run program across the cluster and default cache data in memory. Single machine's capacity is not sufficient to adapt the increasing scRNA-seq data size and the time to process intermediate files occupies a more significant proportion on total time. Therefore, reducing the disk access of intermediate files and improving the upstream data process's scalability is the main way to improve avaliable pipelines. ScSpark is based on UMI-tools.

2.2 Simple barcode and umi correct

We override FASTQ hadoop loading api [?] and utilize Spark's map function to split barcode and umi. We split FASTQ R1's sequence in loading step, and use barcode as RDD's key and use others information as RDD's value.

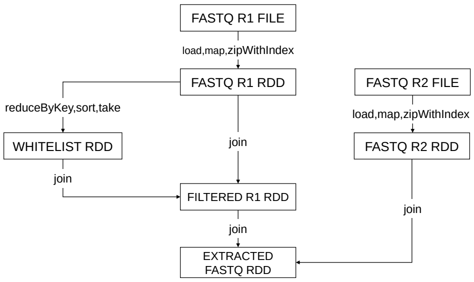

Fig. 1. An overview of sequence quality control module based on Spark.

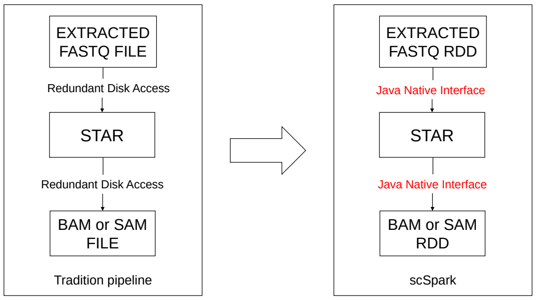

Fig. 2. New way to implement STAR's interface.

2.3 Spark Version Sequence Quality Control

Large FASTQ files can be loaded in each node's memory parallel and the loading speed doesn't limit by single node disk access speed. As shown in Fig **??**, the sequence quality control step typically consists of two main components, i)generating a whitelist including identities of the true cell barcodes is unknown and ii)filtering FASTQ R1 with the whitelist and extracting FASTQ R2 according to filtered FASTQ R1's index.

We loaded each file splits of FASTQ R1 to memory, and abstract them as FASTQ R1 RDD. Then we calculated the number of occurences for each CB and selected the most frequent CBs, named whitelist RDD, as intermediate results. By using reduceByKey function, each barcode could be counted parallelly in each split of FASTQ R1 RDD and we coould merge each split's result in the end. After that we can use the result to generate whitelist RDD by Spark's sort and take function, both of which can parallelly compute in the cluster. The filtered FASTQ R1 RDD could be easily got by using the join function to find which read's cell barcode is in the whitelist RDD. And then we used filter function to extract FASTQ R2 RDD. The last step in sequence quality control was using join function again to combine filtered FASTQ R1 RDD and FASTQ R2 RDD with the same index.

2.4 Eleminate redundancy disk access in Aligner step

Apart from Cellranger we can't know how it implements because it isn't an open source software. In previous pipeline, STAR needs to load extracted FASTQ R2 file or FASTQ R1 file. FASTQ R2 file and whitelist file will generate unnecessary disk access. We choose STAR as our aligner but

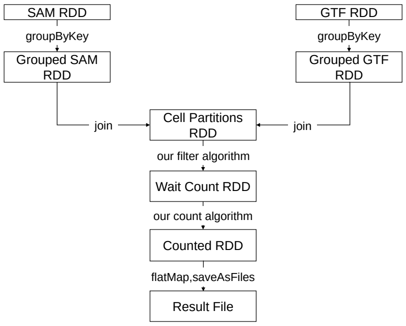

Fig. 3. An overview of Spark version count.

avoid loading extracted FASTQ R2 and FASTQ R1 twice. As shown in Fig **??**, we utilized Java Native Interface to transfer extracted FASTQ R2 RDD's data to STAR program, and then run the STAR process. The STAR aligner was encapsulated as a shared object and invoked the shared object as our aligner. To overcome imbalance problem of node's data volume, we repartitioned the extracted FASTQ R2 RDD, and then each node could run STAR parallelly with counterpoise data volume. We also found if node's in memory was enough, we could start up more than one STAR's in one node to achieve better parallel computing. The SAM or BAM file as STAR's output could generate more disk access waste than STAR's input. Here, we refered STAR-solo's solution to solve the problem by combining aligner and count. We used Java Native Interface again to transfer STAR results and directly abstract them to SAM RDD. This operation is parallel and totally in memory.

2.5 Count with multi-node

As shown in Fig **??**, to take advantage of multi-node's compute capability, we used a new way to implement the count step. Except for SAM RDD that were generated in the previous step, we loaded GTF file to memory and abstract them as GTF RDD. Then we grouped SAM RDD and GTF RDD according to their cell name. Then we used flatMap function to parallelly compute each cell's read count. In the end, we collected results in all nodes, and the result files would produce the only output disk access in the whole program. ScSpark breaks the limitation of one node's computing and achieved large scalability.

2.6 Experiment design

We used Apache Spark version 2.1.0 as our in-memory computing environment. And three example scRNA-seq datasets generated by 10X Genomics on their platform was used in our experiments. Dataset1, 10X Genomics v3 10k peripheral blood mononuclear cell (PBMC) from a healthy donor [**?**], which contain 640 millions records. Dataset2, 10k Rat PBMCs Multiplexed from a Wistar rat strain [**?**], which contain 289.3 millions records. Dataset3, 10X Genomics 10k Monkey PBMCs Multiplexed from a Rhesus monkey strain [**?**], which contain 261.5 millions records. To compare with tradition pipelines, we tested three datasets in UMI-tools, CellRanger, STARsolo and scSpark's performance. And then, we tested each step's spend time, due to CellRanger isn't an open source software and STARsolo implements pipeline in a different way, we choose UMI-tools as scSpark each step's performance baseline. After that, in align step, four pipelines all based on STAR's program. So we tested STAR and scSpark's mapping speed to prove scSpark not only get greatly improve in same CPU cores compare with STAR, but also can get near linear improve when CPU cores increase. Except performance evaluate, we use 16 CPU cores as tradition pipeline's scalability baseline and use 64 CPU cores as scSpark's scalability baseline to prove scSpark's scalability better than tradition pipelines.

TABLE 1
previous pipeline and scSpark's performance comparisione

System	Dataset	Cores	Spend time(s)	
			100M reads	640M reads
UMI-tools	1	64 cores	7254	44160
CellRanger	1	64 cores	6000	11700
STAR-solo	1	64 cores	5820	8100
scSpark	1	4*16 cores	402	-
	1	16*16 cores	-	841
UMI-tools	2	64 cores	7254	44160
CellRanger	2	64 cores	6000	11700
STAR-solo	2	64 cores	5820	8100
scSpark	2	4*16 cores	396	-
	2	16*16 cores	-	841
UMI-tools	3	64 cores	7254	44160
CellRanger	3	64 cores	6000	11700
STAR-solo	3	64 cores	5820	8100
scSpark	3	4*16 cores	553	-
	3	16*16 cores	-	841

TABLE 2
Performance comparision for each step

System	Dataset	Cores	Barcode and UMI correct, filter	align	count
UMI-tools(100 million reads)	1	64 cores	9720	600	1740
UMI-tools(640 million reads)	1	64 cores	33600	2160	8400
scSpark(100 million reads)	1	4*16 cores	31	270	53
scSpark(640 million reads)	1	16*16 cores	81	447	313
UMI-tools(100 million reads)	2	64 cores	9720	600	1740
UMI-tools(640 million reads)	2	64 cores	33600	2160	8400
scSpark(100 million reads)	2	4*16 cores	31	270	53
scSpark(640 million reads)	2	16*16 cores	81	447	313
UMI-tools(100 million reads)	3	64 cores	9720	600	1740
UMI-tools(640 million reads)	3	64 cores	33600	2160	8400
scSpark(100 million reads)	3	4*16 cores	117	402	34
scSpark(640 million reads)	3	16*16 cores	81	447	313

3 Results

We evaluated the speed and scalabilty for each step of scSpark to show the advantage of scSpark compared with the available pipelines.

3.1 Efficiency Evaluation

Limited by single machine's CPU cores, we took 64 cores in preivous pipelines test. Table **??** shown a summary of all the pipelines's performance. We could see scSpark's speed is much more quick than any available pipeline in the same CPU cores environment. And scSpark could get greater improvement when the cluster's CPU cores number increased. We recorded each step's process time to know the reason of the speed's improvement. As table **??** shown, scSpark is much faster than the previous pipelines in any single step. Contrast with UMI-tools, We could see scSpark gets great improve in the first two steps, barcode correct and filter. This step's improve comes from using in memory trait to reduce disk access times and take advantage of Spark's high concurrency feature. After that as Fig **??** shown, although scSpark's align step based on STAR's code,

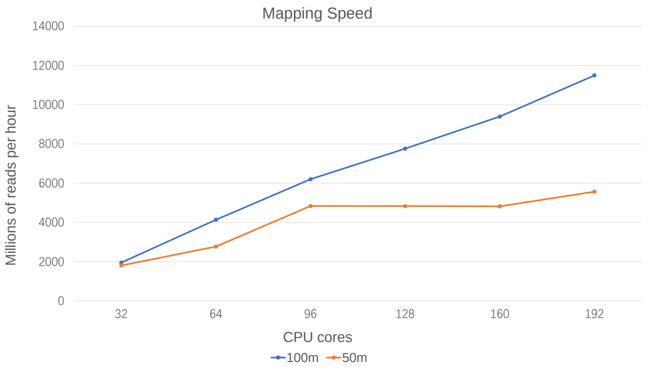

Fig. 4. Invoked STAR's mapping step.

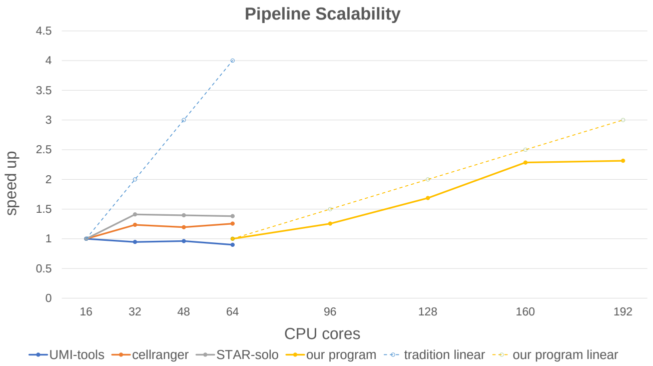

Fig. 5. Pipeline Scalability.

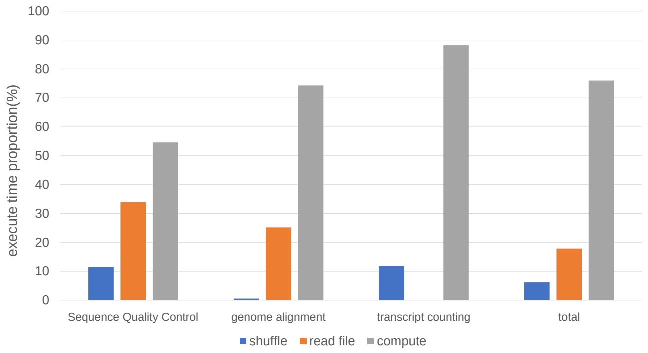

Fig. 6. Each step consumes time.

mapping speed not only have greater performance in same cores but also can get nearly linear imporve when cluster's CPU's cores improve. Due to using in memory FASTQ RDD and SAM RDD to replace tradition pipeline's disk access, scSpark's align step can get great improve in same CPU cores condition. Furthermore, we also found STAR's mapping speed can get improve along with CPU's cores increase, but will converge after touching the threshold. And we took advantage of Spark's distributed trait, process multi STAR's program in the same time and each STAR's can fully utilize multi-thread ability. After align step, we grouped records by cell and concurrency process each group in each CPU cores. This way also get great improve compare with tradition pipeline.

3.2 Scalability Evalution

ScSpark's second advantage is that it could get near linear improvement when the cluster's CPU core number increased. We used a small sample which had 100 millions records to compare speedup. As shown in Fig **??**, we found that when CPU cores number increased, scSpark could get near linear speedup. In addition, we found if the CPU cores number exceeds a ceiling, previous pipelines will speedup nearly stop. Umi-tools did not show any scalabiliy and even a little slow down when the CPU cores number increased. STARsolo and CellRanger shown a little improve when the CPU cores increased to 32 from 16, but quickly stopped speedup. These results show that scSpark's scalability has great improvement and scalability can improve with the data volume. ScSpark's scalability is much better than any previous pipeline. And through Fig **??**, we found it can improve with the data volume improve. As Fig **??** shown, align step occupies largest part of scSpark's process time, and scSpark's scalability much comes from this step. We found STAR's mapping speed influenced by data volume, due to both scSpark and STAR need same load genome index ahead of map. This substep's fix process time influenced the whole align step's scalability. When the data volume increase, the time to process map substep will occupied large part of whole align step. So scSpark can get higher scalability when the data volume increase.

3.3 Resource Analysis

We tested performance to find the bottleneck of scSpark. We used 640 millions records of FASTQ data to evaluate two aspects of scSpark. First we compared network shuffle and disk access spend by scSpark and compute proportions of time, which helped to find whether network shuffle or disk access occupies too much time. Secondy, we tested CPU and memory usage, to ensure which resources would cause scSpark's bottleneck.

3.3.1 Network and Disk Behavior of scSpark

We computed network time by summing up the time that our scSpark shuffle data in multi machine. Disk access comes from loading FASTQ, STAR's index and GTF files. The ideal situation is tasks did not waste any time in disk access and network shuffle. We found that scSpark's computing time occupied most execute time. Except STAR's index file, all files' disk access distributed to each node would improve whole system's loading speed. And we found the time that waste in shuffling doesn't occupy too much time.

3.3.2 CPU and Memory usage of scSpark

We monitored scSpark's CPU and memory usage during processing. As Fig **??** shown, in sequence quality control step, scSpark highly exploited each node's multi cores CPU to achieve speedup. Other pipelines' single thread solution's CPU usage is much lower than scSpark. We found that scSpark's boundary mainly came from genome alignment. Because we invoked STAR as our alignment tool, and STAR's

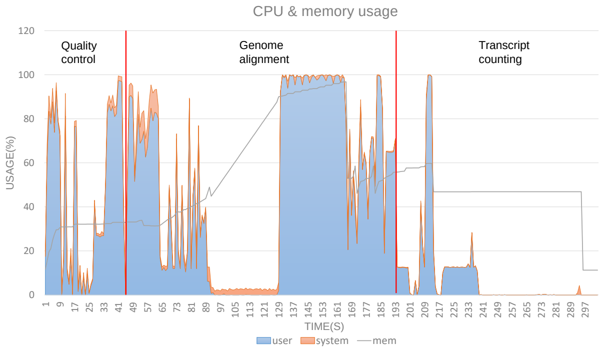

Fig. 7. CPU and Memory usage of scSpark.

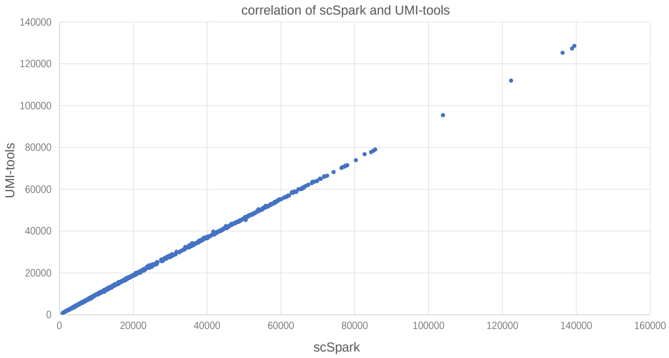

Fig. 8. The correlation of transcript counting results for scSpark and UMI-tools.

program naturally occupy most proportion of memory in this step.

3.3.3 Biological verification

ScSpark is developed based on UMI-tools, which has been fully verified in terms of accuracy. This section we used the gene expression matrix obtained by scSpark and UMI-tools to perform downstream analysis of scRNA-seq data. We compared transcript counting results and clusters of cells in the hgmm-1k-v3 dataset to verify the correlation between scSpark and UMI-tools.

As Fig **??** shown, for each after processing cell barcode, their gene expression matrix approximate fit $y = x$, R^2 closed to 0.9998. Furthermore, we used Seurat to obtain tSNE plots for the visualization of cell clusters. And as Fig **??** shown, tSNE cell clustering result showed high corrleation between ScSpark and UMI-tools.

4 Conclusion

In this paper, we proposed a way to utilize Apache Spark's in memory compute trait and achieved considerable speedup and scalability. Our scSpark can take advantages of multi-machine compute capacity to speedup key steps of scRNA-seq preprocessing and eliminate redundant disk access.

In addition to performance improvement, scSpark also show much more scalable than any previous pipeline and its speed closes to linear improvement. Moreover,scSpark can improve scalability by increasing partition number if the resource is sufficient.

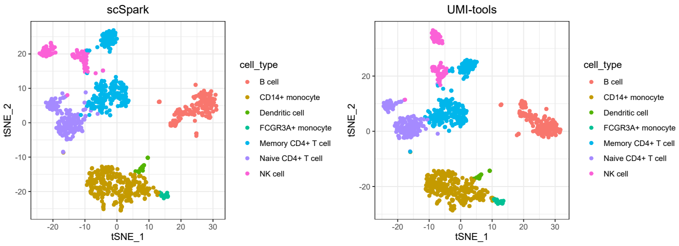

Fig. 9. tSNE plots based on scSpark and UMI-tools' gene expression matrices.

And we also found our scSpark's scalability improvement has a ceiling. The reason is that our scSpark invokes STAR's mapping speed, which is influenced by loading index time. And if data size is tiny, the influence occupies a large proportion of whole the STAR program process time.

References

[1] E. Papalexi and R. Satija, "Single-cell RNA sequencing to explore immune cell heterogeneity," Nat. Rev. Immunol., vol. 18, no. 1, pp. 35–45, 2018, doi: 10.1038/nri.2017.76.

[2] X. Zhang et al., "Comparative Analysis of Droplet-Based Ultra-High-Throughput Single-Cell RNA-Seq Systems," Mol. Cell, vol. 73, no. 1, pp. 130-142.e5, 2019, doi: https://doi.org/10.1016/j.molcel.2018.10.020.

[3] E. Z. Macosko et al., "Highly Parallel Genome-wide Expression Profiling of Individual Cells Using Nanoliter Droplets," Cell, vol. 161, no. 5, pp. 1202–1214, 2015, doi: https://doi.org/10.1016/j.cell.2015.05.002.

[4] A. M. Klein et al., "Droplet Barcoding for Single-Cell Transcriptomics Applied to Embryonic Stem Cells," Cell, vol. 161, no. 5, pp. 1187–1201, 2015, doi: https://doi.org/10.1016/j.cell.2015.04.044.

[5] G. X. Y. Zheng et al., "Massively parallel digital transcriptional profiling of single cells," Nat. Commun., vol. 8, no. 1, p. 14049, 2017, doi: 10.1038/ncomms14049.

[6] A. B. Rosenberg et al., "Single-cell profiling of the developing mouse brain and spinal cord with split-pool barcoding," Science (80-.)., vol. 360, no. 6385, pp. 176 LP – 182, Apr. 2018, doi: 10.1126/science.aam8999.

[7] J. Cao et al., "Comprehensive single-cell transcriptional profiling of a multicellular organism," Science, vol. 357, no. 6352, pp. 661–667, 2017, doi: 10.1126/science.aam8940.

[8] L. Tian et al., "scPipe: A flexible R/Bioconductor preprocessing pipeline for single-cell RNA-sequencing data," PLOS Comput. Biol., vol. 14, no. 8, p. e1006361, Aug. 2018, [Online]. Available: https://doi.org/10.1371/journal.pcbi.1006361.

[9] T. Kivioja et al., "Counting absolute numbers of molecules using unique molecular identifiers," Nat. Methods, vol. 9, no. 1, pp. 72–74, 2012, doi: 10.1038/nmeth.1778.

[10] P. G. Camara, "Methods and challenges in the analysis of single-cell RNA-sequencing data," Curr. Opin. Syst. Biol., vol. 7, pp. 47–53, 2018, doi: https://doi.org/10.1016/j.coisb.2017.12.007.

[11] T. Smith, A. Heger, and I. Sudbery, "UMI-tools: Modeling sequencing errors in Unique Molecular Identifiers to improve quantification accuracy," Genome Res., vol. 27, no. 3, pp. 491–499, 2017, doi: 10.1101/gr.209601.116.

[12] A. Dobin et al., "STAR: ultrafast universal RNA-seq aligner.," Bioinformatics, vol. 29, no. 1, pp. 15–21, Jan. 2013, doi: 10.1093/bioinformatics/bts635.

[13] D. Kim, B. Langmead, and S. L. Salzberg, "HISAT: a fast spliced aligner with low memory requirements," Nat. Methods, vol. 12, no. 4, pp. 357–360, 2015, doi: 10.1038/nmeth.3317.

[14] G. Baruzzo, K. E. Hayer, E. J. Kim, B. Di Camillo, G. A. FitzGerald, and G. R. Grant, "Simulation-based comprehensive benchmarking of RNA-seq aligners.," Nat. Methods, vol. 14, no. 2, pp. 135–139, Feb. 2017, doi: 10.1038/nmeth.4106.

[15] S. Parekh, C. Ziegenhain, B. Vieth, W. Enard, and I. Hellmann, "zUMIs - A fast and flexible pipeline to process RNA sequencing data with UMIs," Gigascience, vol. 7, no. 6, 2018, doi: 10.1093/gigascience/giy059.

[16] A. Blibaum, J. Werner, and A. Dobin, "STARsolo: single-cell RNA-seq analyses beyond gene expression[version 1; not peer reviewed]," F1000Research 2019, 8:1896 (poster), doi: https://doi.org/10.7490/f1000research.1117634.1

[17] "UMI_tools." https://github.com/CGATOxford/UMI-tools (accessed Nov. 30, 2020).

[18] M. Gao et al., "Comparison of high-throughput single-cell RNA sequencing data processing pipelines," Brief. Bioinform., vol. 22, Jul. 2020, doi: 10.1093/bib/bbaa116.

[19] "Apache Hadoop framework." https://hadoop.apache.org (accessed Nov. 30, 2020).

[20] "Apache Spark framework." https://spark.apache.org/ (accessed Nov. 30, 2020).

[21] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008, doi: 10.1145/1327452.1327492.

[22] M. Zaharia et al., "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," in 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), 2012, pp. 15–28, [Online]. Available: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia.

[23] 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, p. e0155461, May 2016, [Online]. Available: https://doi.org/10.1371/journal.pone.0155461.

[24] X. Li, G. Tan, B. Wang, and N. Sun, "High-Performance Genomic Analysis Framework with in-Memory Computing," SIGPLAN Not., vol. 53, no. 1, pp. 317–328, 2018, doi: 10.1145/3200691.3178511.

[25] A. Yang, M. Troup, P. Lin, and J. W. K. Ho, "Falco: a quick and flexible single-cell RNA-seq processing framework on the cloud.," Bioinformatics, vol. 33, no. 5, pp. 767–769, Mar. 2017, doi: 10.1093/bioinformatics/btw732.

[26] "10xgenomics.human" https://support.10xgenomics.com (accessed Nov. 30, 2020).

[27] "Index and GTF file." https://www.gencodegenes.org/human/releases.html (accessed Nov. 30, 2020).

[28] "UMI_tools." https://github.com/CGATOxford/UMI-tools (accessed Nov. 30, 2020).

[29] "10xgenomics.rat." https://www.10xgenomics.com/resources/datasets/10-k-rat-pbm-cs-multiplexed-2-cm-os-3-1-standard-6-0-0 (accessed Jun. 22, 2021).

[30] "10xgenomics.monkey." https://www.10xgenomics.com/resources/datasets/10-k-monkey-pbm-cs-multiplexed-2-cm-os-3-1-standard-6-0-0 (accessed Jun. 22, 2021).

[31] "FASTQ.Hadoop.API." https://github.com/HadoopGenomics (accessed Jun. 22, 2021).

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.