ScSpark: High-Performance scRNA-seq Data Processing with Low-Redundancy Disk Access

Yu Liu
School of Informatics
Xiamen University
Xiamen, China
liuyu123@stu.xmu.edu.cn

Mingxuan Gao
School of Informatics
Xiamen University
Xiamen, China
mingxuangao@stu.xmu.edu.cn

Lixuan Tan
School of Informatics
Xiamen University
Xiamen, China
tanlix@stu.xmu.edu.cn

Hongjin Liu
School of Informatics
Xiamen University
Xiamen, China
liuhongjin@stu.xmu.edu.cn

Yating Lin
School of Informatics
Xiamen University
Xiamen, China
linyating@stu.xmu.edu.cn

Wenxian Yang
Aginome Scientific
Xiamen, China
wx@aginome.com

Rongshan Yu*
School of Informatics
Xiamen University
Xiamen, China
rsyu@xmu.edu.cn

***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 volume, file input/output (IO) has become a bottleneck of the processing speed of existing tools. We build a Java-based scRNA-seq data processing pipeline, the scSpark, using Apache Spark's in-memory computing and inherent scalable technologies. To reduce unnecessary disk access while reading FASTQ files and writing SAM files, we used the Java Native Interface (JNI) to deliver FASTQ Resilient Distributed Datasets (RDD), and retrieved genome mapping results to SAM RDD. By allocating scRNA-seq data and processing tasks to a computer cluster, the scSpark toolkit can significantly reduce disk access for saving and loading temporary results. To evaluate the performance of scSpark, we built a spark cluster on Aliyun, and compared 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 tools.**

***Index Terms*—scRNA-seq data processing, Apache Spark, cloud computing**

I. 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 to discover new cell identities and study biological processes in complex tumor samples. In the past decade, RNA-seq has been widely used to study gene expression patterns in biological samples. However, the resolution of bulk RNA-seq could only reach the average level of cell populations. scRNA-seq essentially reveals the transcriptional status at single cell level, which provides the basis for subsequent bioinformatics analysis [1].

High-throughput 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 [2]. In order to demultiplex the reads for inidividual cells after sequencing [3], two oligonucleotide barcodes, the cell barcode (CB) and the unique molecular identifier (UMI), have been introduced in single cell sequencing [4], [5]. When using the CB, different barcode sequences are assigned to each cell for transcript source retrieval after sequencing, thus greatly improving the throughput and reducing the cost of scRNA-seq [6], [7]. UMIs are random oligonucleotide barcodes used in single cell sequencing to distinguish redundant transcripts generated from PCR amplification [8]–[10].

Various scRNA-seq data processing pipelines have been developed, which typically implement multiple modules including barcode processing, sequence quality control (QC), genome alignment and transcript counting, to convert raw scRNA-seq data into a gene expression matrix for further downstream analysis. First, the barcodes are extracted from a pre-designed nucleotide site for different scRNA-seq protocols. In QC, the 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 a splice-aware aligner, such as STAR [11] and HISAT2 [12]. Finally, the reads are assigned to genes and the count matrices for UMIs are generated [13].

Among all the scRNA-seq data processing pipelines, the most influential studies probably include CellRanger [14], UMI-tools [10], and STARsolo [15], etc. CellRanger is a highly integrated data processing software tool tailored by 10X Genomics for scRNA-seq data analysis. It is most suitable for processing large datasets on high performance workstations [16]. UMI-tools is a comprehensive scRNA-seq data processing suite with the directional barcode collapse algorithm integrated that cosiderably promotes the transcript quantification accuracy. STARsolo is a recently developed extended pipeline based on the original genome aligner STAR to adapt the single-cell applications. However, these scRNA-seq data processing tools only run on a single machine and thus posing a high demand on hardware configurations, e.g., the number of CPU cores and the memory size.

Recently, big data frameworks such as Hadoop (https://hadoop.apache.org) and Spark (https://spark.apache.org) have been used to speed up the data processing for the next generation sequencing (NGS) data. SparkBWA [17] xxx. GPF [18] xxx. Following these advances in NGS data processing, Falco [19] used Spark to concatenate the aligner and the feature quantitative softwares for scRNA-seq data preprocessing. Falco improved performance greatly compared to existing single workstation solutions, however, it still suffers from two limitations. First, Falco does not deal with CBs and UMIs and is therefore incompatible with the high-throughput scRNA-seq protocols. Secondly, the performance gain of Falco basically lies in xxx. It does not reduce the amount of disk reads and writes during alignment and transcript counting.

We present scSpark, a scRNA-seq data processing pipeline using the Spark framework, for high performance data processing on a cluster of computers. To meet the demand of computing and storage resources for super-large scale scRNA-seq data, our pipeline design not only implements the parallelism of multiple data processing procedures, minimize unnecessary disk access, but also considers the scalability in task distribution such that scSpark can run on either a single computer or on a cluster of computers.

II. Method

A. Overview

ScSpark implemented cell barcode and UMI correct, sequence quality control, genome alignment, and transcript counting based on Spark framework and UMI-tools. ScSpark run program across Spark's cluster and default cache data in memory. We implemented cell barcode and UMI correct when Spark's cluster load data and abstract them to RDD. And we developed a high concurrency Spark version sequence quality control. After that, scSpark processed STAR's program on each node in the cluster. In the end, scSpark grouped each cell's record and distributed conut the result.

B. Simple barcode and UMI correct

We override FASTQ hadoop loading api (https://github.com/HadoopGenomics). Large FASTQ files can be loaded in each node's memory parallel. So scSpark's loading speed doesn't limit by single node disk access speed. In this step, scSpark splits barcode and UMI when scSpark load FASTQ file to FASTQ RDD. ScSpark splits FASTQ R1's sequence to two part, uses barcode as RDD's key and uses others information as RDD's value.

C. Spark version sequence quality control

As shown in Fig 1, the sequence quality control step typically consists of two main components. First, generating whitelist RDD including identities of the true cell barcodes is unknown. After FASTQ R1 RDD generated, 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 cell barcode could be counted parallelly in each split of FASTQ R1 RDD and we could 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. Then scSpark can easily uses whitelist RDD to filter cached FASTQ R1 RDD by using Spark's join function to generate extracted FASTQ R1 RDD. In the time to filter FASTQ R2 RDD, scSpark uses same strategies again. It uses extracted FASTQ R1 RDD to filter FASTQ R2 RDD when they have same line index. 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.

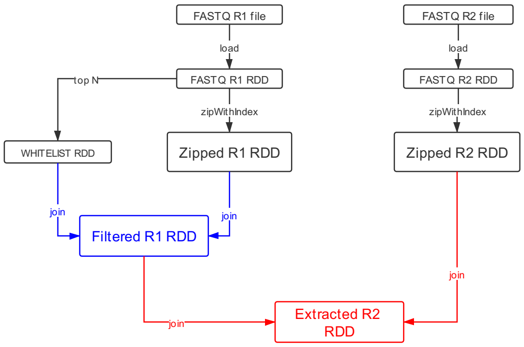

Fig. 1. An overview of sequence quality control module based on Spark.

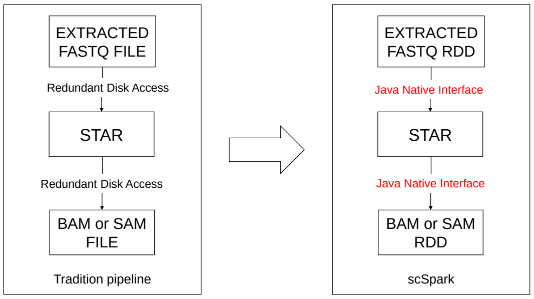

Fig. 2. New way to implement STAR's interface.

D. Eleminate redundancy disk access in aligner step

In previous pipeline, STAR needs to load extracted FASTQ R2 file and FASTQ R1 file. This way will generate unneccessary disk access. We chose STAR as our aligner and modified STAR's loading FASTQ way to avoid loading extracted FASTQ R2 and FASTQ R1 twice. As shown in Fig 2, 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 each node's data volume, we repartitioned the extracted FASTQ R2 RDD before transfer FASTQ RDD to STAR. Each node could run STAR parallelly with counterpoise

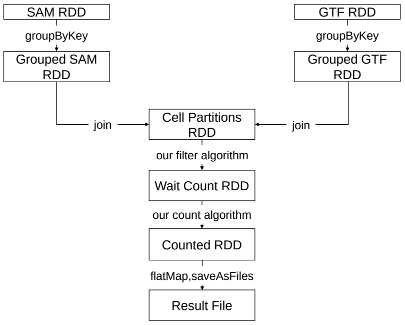

Fig. 3. An overview of Spark version count.

TABLE I
PREVIOUS PIPELINE AND SCSPARK'S PERFORMANCE COMPARISIONE

Pipeline	Dataset	Cores	Spend time(s)
UMI-tools	pbmc_human	64 cores	7284
CellRanger	pbmc_human	64 cores	2225
STAR-solo	pbmc_human	64 cores	1987
scSpark	pbmc_human	4*16 cores	355
UMI-tools	pbmc_rat	64 cores	7259
CellRanger	pbmc_rat	64 cores	1999
STAR-solo	pbmc_rat	64 cores	1854
scSpark	pbmc_rat	4*16 cores	326
UMI-tools	pbmc_monkey	64 cores	7809
CellRanger	pbmc_monkey	64 cores	1891
STAR-solo	pbmc_monkey	64 cores	2357
scSpark	pbmc_monkey	4*16 cores	391

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. After align, STAR's output is SAM file or BAM file that will generate more disk access waste than STAR's input. Here, We used Java Native Interface again to transfer STAR results and directly abstract them as SAM RDD.

E. Count with multi-node

As shown in Fig 3, 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. After that scSpark 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.

F. Experiment design

We used Apache Spark version 2.1.0 as scSpark's in-memory computing environment.

Three scRNA-seq datasets (https://support.10xgenomics .com/) containing approximately 10,000 peripheral blood mononuclear cells (PBMC) from three different species, namely human, rat and monkey, generated by 10X Genomics platform were used in our experiments. Totally, there are 640 million, 289 million and 262 million reads respectively in the raw data of the PBMC_human dataset, the PBMC_rat dataset and the PBMC_monkey dataset.

We evaluated the performance of scSpark with three state-of-the-art pipelines including UMI-tools, CellRanger and STARsolo in terms of the efficiency, scalability and biological analysis accuracy. Limited by the number of CPU cores of single machine, we only used 64 CPU cores for these three pipelines that are not able to run on computational clusters. Then, we tested each step's process spend time. Due to CellRanger isn't an open source software and STAR-solo implemented pipeline in a different way. And scSpark refers to UMI-tools, we choose UMI-tools each step's performance as scSpark's each step's performance baseline.

Except performance evaluate, we calculated scSpark's speed improve rate when Spark cluster CPU cores improve. And we used tradition pipelines' speed improve rate as baseline when pipelines' CPU cores improve from 32 to 64. After that, we tested scSpark's each step scalability and comapred with UMI_tools. In align step, four pipelines' align step 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.

ScSpark is developed based on UMI-tools. And UMI-tools accuracy was fully verified. This section we used the gene expression matrix that was obtained by scSpark and UMI-tools under the same dataset to perform downstream analysis of scRNA-seq data. And then we compared two tools transcript analysis's result and cell cluster analysis's result to verify the correlation between scSpark and UMI-tools. Under hgmm-10k-v3 dataset, we used scSpark and UMI-tools to get gene express matrix, compared two tools' result, and computed their correlation.

III. RESULTS

First, we compared each pipeline's speed in same CPU cores number. Due to scSpark refers UMI_tools, then we compared each step's speed of scSpark with UMI_tools. After that, we contrasted each pipleline's scalability, and contrasted each step's scalability with UMI_tools. In the end, we verified scSpark's result by comparing with UMI_tools's result.

A. Efficiency evaluation

Table I gives a summary of scSpark's performance and compares with tradition pipeline's performance. We can see scSpark's speed is much quicker than any tradition pipelines in same CPU cores environment in three datasets.

After that we recorded each step's process time to find the reason of scSpark's speed improvement. As table 4 shown, scSpark is much faster than the UMI-tools in any single step.

Contrast with UMI-tools, scSpark gets great improve in first two steps, cell barcode and UMI correct and quality control.

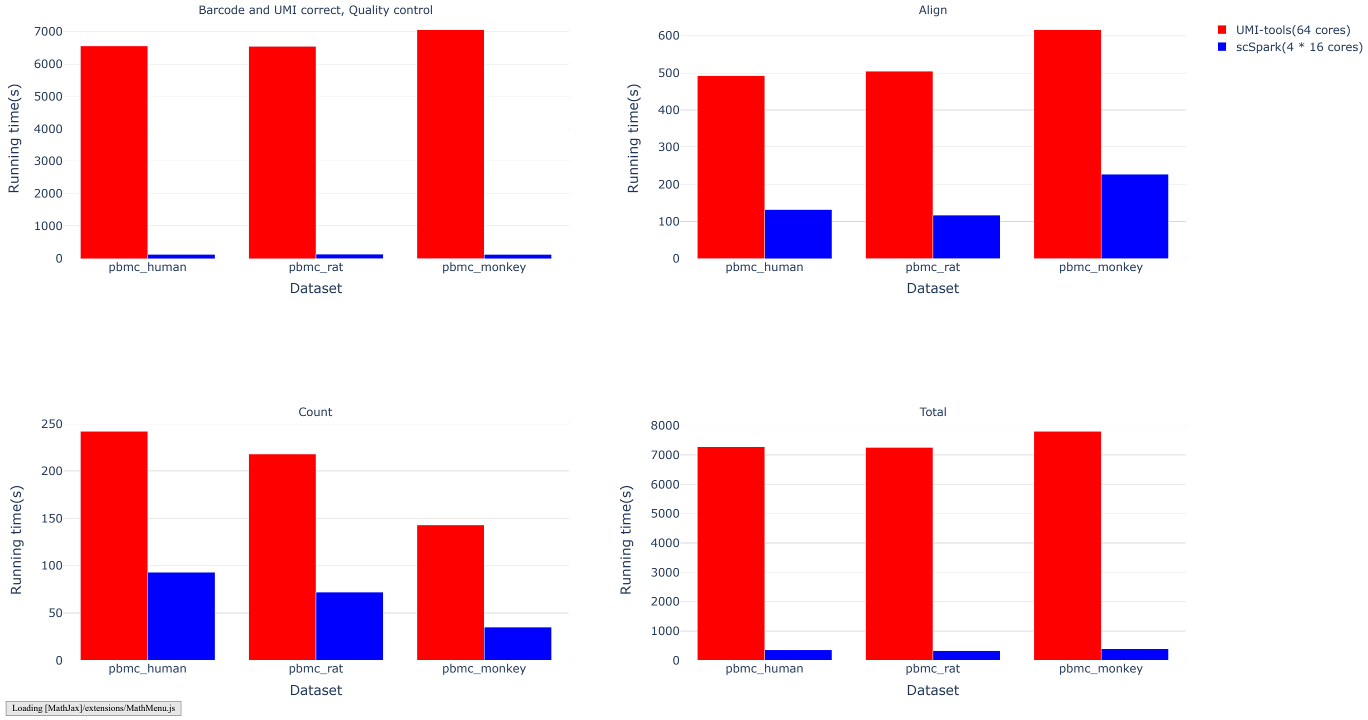

Fig. 4. An overview of sequence quality control module based on Spark.

This step's improve comes from scSpark's in memory trait which can reduce disk access times. And scSpark can take advantage of Spark's high concurrency feature when UMI-tools processed record with single thread.

After that, scSpark use in memory FASTQ RDD and SAM RDD to replace tradition pipeline's disk access. Although scSpark and UMI-tools's align step both based on STAR, scSpark can get great improve in same CPU cores condition. Except reduce redundancy disk access, scSpark processes multi STAR program across Spark cluster in the same time also can improve scSpark's mapping speed.

After align step, we grouped SAM RDD and GTF RDD by cell and concurrency processed each group in each CPU cores. Due to each cell's record number imbalance, scSpark's count step has a little data skew problem which causes count step's processing time influenced by dataset. This way also gets great improve compare with UMI-tools.

B. scalability evaluation

Contrast with tradition pipelines, scSpark not only can perform well in same CPU cores but also scSpark can scala out to improve process speed. Due to the limited of single machine CPU cores, we tested three tradition pipelines' speedup from 16 CPU cores to 64 CPU cores. And due to the limited of memory capacity per node, we tested scSpark's speedup starting from 128 (8*16) CPU cores to 256 (16*16) CPU cores. As shown in 5, we found that UMItools' speedup is poor when CPU cores number increase. Except align step, UMI-tools processes on single thread which causes it can't take advantage of CPU cores number increase. So UMItools' scalability is weak. STAR-solo and Cellranger shows poorer scalability when CPU cores number increase. When CPU cores number increased from 16 to 32, STAR-solo achieved approximate average 40% speedup and Cellranger achieved approximate ?% speedup. When CPU cores number increased from 32 to 64, STAR-solo's speedup was average 20% and Cellranger speedup was average ?%. Both pipelines speedup rate will decrease when CPU cores increase. As for scSpark, we found it can achieve average 50% speedup when Spark's CPU cores number increased from 128 to 256. So scSpark shows more scalability than three tradition pipelines when CPU cores number increase.

TABLE II
STAR's MAPPING SPEED

Dataset	2 cores	4 cores	8 cores	16 cores	32 cores	64 cores
PBMC_human	120.97	242.08	449.09	772.13	1389.32	961.59
PBMC_rat		261.23	530.57	1089	1391.24	1347.35
PBMC_monkey	38.7	76.38	170.01	325.85	512.59	828.99

Due to UMItools' processes on single thread except align step, this pipelines' scalability most come from align step. As shown in Table II, we found in PBMC_human and PBMC_rat STAR can achieved nearly linear improved with CPU cores number increase, when CPU cores number below 32. STAR's speedup rate not only decrease when CPU cores number increase but also have a bottleneck when CPU cores number

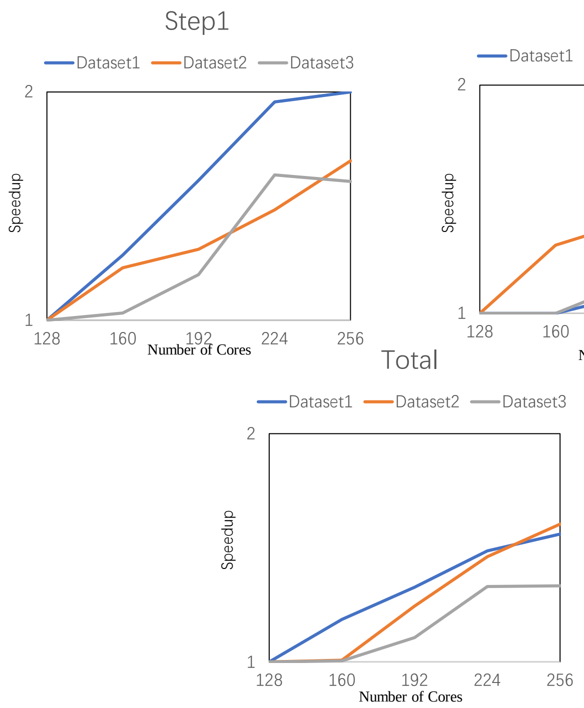

Fig. 5. An overview of tradition pipelines and scSpark's speedup.

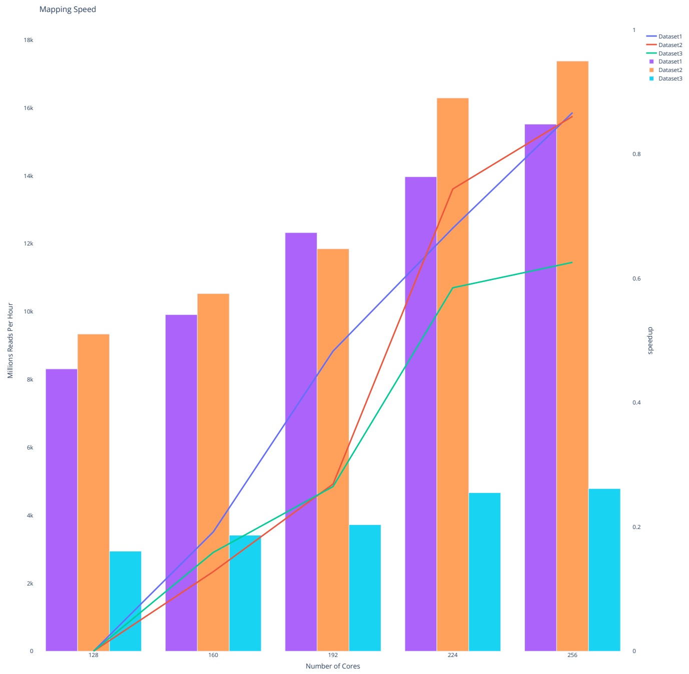

Fig. 6. scSpark's mapping speed.

increase from 32 to 64. As Fig 6 shown, we found scSpark's mapping speed achieved about 79% parallel efficiency when CPU cores number increase from 128 to 256. ScSpark enable distributed process align step per node across Spark cluster, so it can get nearly linear parallel efficiency. So scSpark's align step is more scalability than tradition pipelines' aligner.

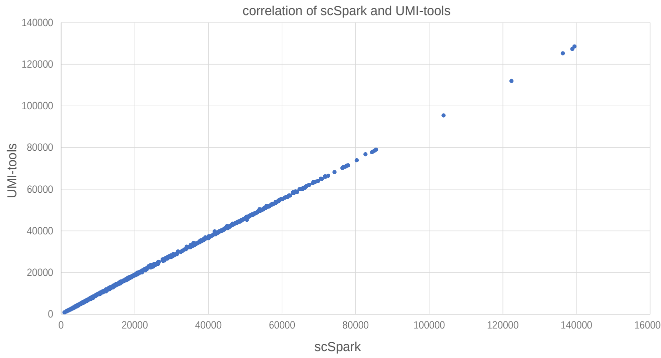

Fig. 7. correlation of scSpark and UMI-tools.

C. Biological verification

As Fig 7 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 print tSNE picture which based on gene expression matrix generated by ScSpark and UMI-tools. And as Fig 8 shown, tSNE cell clustering result showed high corrleation between ScSpark and UMI-tools.

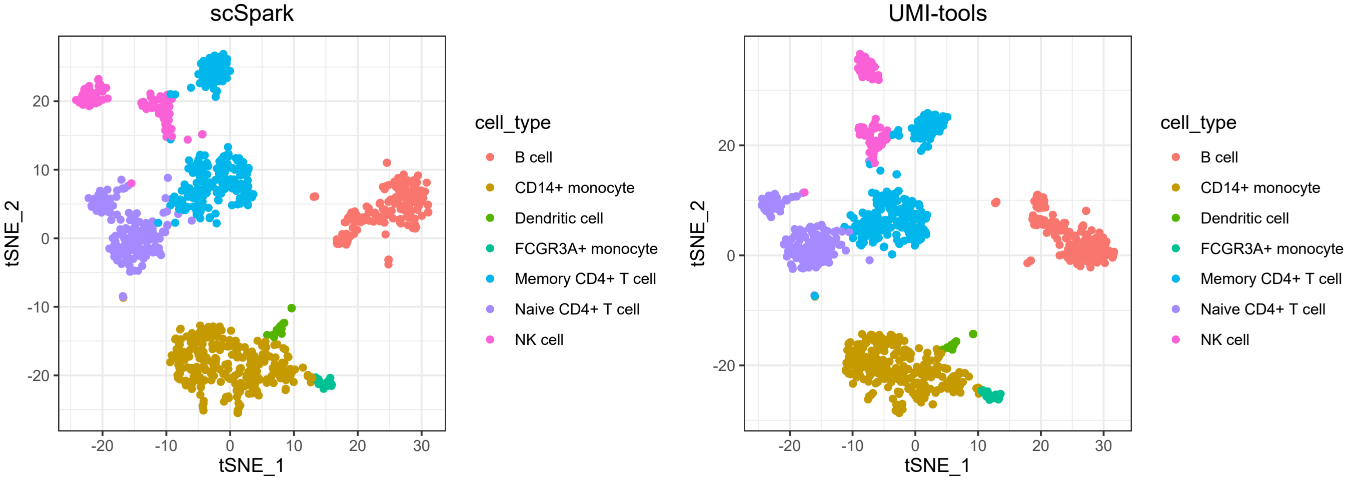

Fig. 8. tSNE picture based on scSpark and UMI-tools' gene expression matrix.

IV. Conclusion

In this paper, we proposed a way which utilize Apache Spark's in memory compute and distributed compute trait to strengthen upstream scRNA-seq pipeline. We developed scSpark, which processes more efficiency and more scalable than tradition pipelines.

Refer to UMI-tools, scSpark divided the upstream process into four steps, UMI and CB correct, sequence quality control, genome alignment and transcript counting. we found scSpark's can get higher performance than three tradition pipelines. And we found scSpark's each steps can get higher performance than UMI-tools. We also compared scSpark's result with UMI-tools to verified scSpark's result correct.

References

[1] E. Papalexi and R. Satija, "Single-cell RNA sequencing to explore immune cell heterogeneity," *Nature Reviews Immunology*, vol. 18, pp. 35–45, 2018.

[2] X. Zhang, T. Li, F. Liu, Y. Chen, J. Yao, Z. Li, Y. Huang, and J. Wang, "Comparative analysis of droplet-based ultra-high-throughput single-cell RNA-seq systems," *Molecular cell*, vol. 73, no. 1, pp. 130–142.e5, 2019.

[3] L. Tian, S. Su, X. Dong, D. Amann-Zalcenstein, C. Biben, A. Seidi, D. J. Hilton, S. H. Naik, and M. E. Ritchie, "scPipe: A flexible R/Bioconductor preprocessing pipeline for single-cell RNA-sequencing data," *PLoS Computational Biology*, vol. 14, no. 8, 2018.

[4] A. Rosenberg, C. M. Roco, R. A. Muscat, A. Kuchina, P. Sample, Z. Yao, L. T. Graybuck, D. J. Peeler, S. Mukherjee, W. Chen, S. Pun, D. L. Sellers, B. Tasic, and G. Seelig, "Single-cell profiling of the developing mouse brain and spinal cord with split-pool barcoding," *Science*, vol. 360, pp. 176 – 182, 2018.

[5] J. Cao, J. S. Packer, V. Ramani, D. A. Cusanovich, C. Huynh, R. Daza, X. Qiu, C. Lee, S. Furlan, F. Steemers, A. C. Adey, R. Waterston, C. Trapnell, and J. Shendure, "Comprehensive single cell transcriptional profiling of a multicellular organism by combinatorial indexing," *bioRxiv*, 2017.

[6] E. Z. Macosko, A. Basu, R. Satija, J. Nemesh, K. Shekhar, M. Goldman, I. Tirosh, A. Bialas, N. Kamitaki, E. M. Martersteck, J. J. Trombetta, D. Weitz, J. Sanes, A. Shalek, A. Regev, and S. McCarroll, "Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets," *Cell*, vol. 161, pp. 1202–1214, 2015.

[7] A. M. Klein, L. Mazutis, I. Akartuna, N. Tallapragada, A. Veres, V. Li, L. Peshkin, D. Weitz, and M. Kirschner, "Droplet barcoding for single-cell transcriptomics applied to embryonic stem cells," *Cell*, vol. 161, pp. 1187–1201, 2015.

[8] T. Kivioja, A. Vähärautio, K. Karlsson, M. Bonke, M. Enge, S. Linnarsson, and J. Taipale, "Counting absolute numbers of molecules using unique molecular identifiers," *Nature Methods*, vol. 9, pp. 72–74, 2012.

[9] P. G. Camara, "Methods and challenges in the analysis of single-cell RNA-sequencing data," *Current Opinion in Systems Biology*, vol. 7, pp. 47–53, 2018.

[10] T. Smith, A. Heger, and I. M. Sudbery, "UMI-tools: modeling sequencing errors in unique molecular identifiers to improve quantification accuracy." *Genome research*, vol. 27, no. 3, pp. 491–499, 2017.

[11] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson, and T. R. Gingeras, "STAR: ultrafast universal RNA-seq aligner," *Bioinformatics*, vol. 29, no. 1, pp. 15–21, 2013.

[12] D. Kim, B. Langmead, and S. Salzberg, "HISAT: a fast spliced aligner with low memory requirements," *Nature Methods*, vol. 12, no. 4, pp. 357–360, 2015.

[13] 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, 2018.

[14] G. X. Y. Zheng, J. M. Terry, P. Belgrader, P. Ryvkin, Z. W. Bent, R. Wilson, S. B. Ziraldo, T. D. Wheeler, G. McDermott, J. Zhu, M. Gregory, J. Shuga, L. Montesclaros, J. Underwood, D. A. Masquelier, S. Y. Nishimura, M. Schnall-Levin, P. W. Wyatt, C. M. Hindson, R. Bharadwaj, A. Wong, K. Ness, L. Beppu, H. Deeg, C. McFarland, K. Loeb, W. Valente, N. G. Ericson, E. Stevens, J. Radich, T. Mikkelsen, B. Hindson, and J. Bielas, "Massively parallel digital transcriptional profiling of single cells," *Nature Communications*, vol. 8, 2017.

[15] A. Blibaum, J. Werner, and A. Dobin, "STARsolo: single-cell RNA-seq analyses beyond gene expression," *F1000Research*, vol. 8, 2019.

[16] M. Gao, M. Ling, X. Tang, S. Wang, X. Xiao, Y. Qiao, W. Yang, and R. Yu, "Comparison of high-throughput single-cell RNA sequencing data processing pipelines," *Briefings in Bioinformatics*, vol. 22, no. 3, 2021.

[17] J. Abuín, J. C. Pichel, T. Pena, and J. Amigo, "Sparkbwa: Speeding up the alignment of high-throughput dna sequencing data," *PLoS ONE*, vol. 11, no. 5, 2016.

[18] X. Li, G. Tan, B. Wang, and N. Sun, "High-performance genomic analysis framework with in-memory computing," *ACM SIGPLAN Notices*, vol. 53, no. 1, pp. 317–328, 2018.

[19] A. Yang, M. Troup, P. Lin, and J. Ho, "Falco: a quick and flexible single-cell RNA-seq processing framework on the cloud," *Bioinformatics*, vol. 33, no. 5, pp. 767–769, 2017.