Chapter 17

Bringing cell subpopulation discovery on a cloud-HPC using rCASC and StreamFlow

[1]Sandro G. Contaldo, [2]Luca Alessandri, [1]Iacopo Colonnelli, [1]Marco Beccuti and [1]Marco Aldinucci.

[1]Department of Computer Science, University of Turin, Turin, Italy; marco.beccuti@unito.it
[2]Molecular Biotechnology Center, University of Turin, Turin, Italy; l.alessandri@unito.it

Abstract

The idea behind novel single-cell RNA sequencing (scRNA-seq) pipelines is to isolate single cells through microfluidic approaches and generating sequencing libraries in which the transcripts are tagged to track their cell of origin. Modern scRNA-seq platforms are capable of analyzing up to many thousands of cells in each run. Then, combined with massive high-throughput sequencing producing billions of reads, scRNA-seq allows the assessment of fundamental biological properties of cells populations and biological systems at unprecedented resolution.

In this chapter, we describe how cell subpopulation discovery algorithms, integrated into rCASC, could be efficiently executed on cloud-HPC infrastructure. To achieve this task, we focus on the StreamFlow framework which provides container-native runtime support for scientific workflows in cloud/HPC environments.

Key words: single-cell RNA sequencing**,** cloud computing, HPC environment StreamFlow.

1 Introduction

RNA sequencing (RNA-seq) became a routine method in biomedical research for discovering gene expression patterns across millions of cells.

However, since RNA-seq is typically performed in bulk, which does not allow to retain the relevant biological differences among cells. To deal with this crucial aspect single-cell RNA-seq (scRNA-seq) analysis was developed. In scRNA-seq, each sequencing library represents a single cell. Therefore, scRNA-seq allows assessing the fundamental biological properties of cells in more detail.

Among the specific bioinformatics tools proposed in the literature for scRNA-seq, rCASC (https://github.com/kendomaniac/rCASC) [1-3] is specifically designed to provide an integrated analysis environment for cell subpopulation discovery providing a high level of flexibility and enabling computation reproducibility as part of the data analysis. rCASC provides, in a single computational environment, i) raw data preprocessing, ii) subpopulation discovery via different clustering approaches and iii) cluster-specific gene signature detection. These three analysis steps require different computational resources and computing models, leading to a workflow which requires a heterogeneous computing environment, e.g. CPU and GPU architectures, multi-cores hardware and multi-server environment. In particular, clustering is the most computationally demanding activity.

In this chapter, we describe how the cell subpopulation discovery functions implemented in rCASC can be brought on a cloud-HPC infrastructure using StreamFlow (https://streamflow.di.unito.it) [4]. In such context, StreamFlow has been leveraged to execute the workflow on top of a hybrid cloud-HPC environment.

2 Materials

2.1 Minimal hardware/software requirements

The analyses described in the present chapter, performs best if implemented in a HPC cloud environment, e.g. https://hpc4ai.it/.

Docker installation is required (https://docs.docker.com/get-docker/). R installation is required (https://cran.r-project.org/). Specifically, it is required the installation of *devtools* library and the rCASC github package (https://github.com/kendomaniac/rCASC).

2.2 Exemplary dataset

As exemplary dataset we used RNA5c, which is available at the GitHub repository https://github.com/kendomaniac/single_cell_transcriptomics/tian_et_al_2018.git. This dataset includes the human lung adenocarcinoma cell lines H2228, H1975, A549, H838 and HCC827 mixed at the same ratio. RNA-5c includes 3904 cells: 1242 cells of A549, 436 cells belong to

H1975, 749 cells of H2228, 879 cells belong to H838, and 598 cells of HCC827 cells. The raw data set is available at GEO repository GSM3618014 [5]. The count table in GitHub has the cell types associated with each cell name. Analysis results are available at https://github.com/Gepiro/rCASC_StreamFlow/tree/main/Results.

2.3 Used Hardware

For the experiments reported in this chapter, we used eight virtual machines: each of them with 8 cores, 32 GB RAM, Ubuntu 20.04 LTS and Docker 20.10.6. These machines were provided by the High-Performance Computing for Artificial Intelligence (HPCAI) center at the University of (https://hpc4ai.unito.it/).

2.4 StreamFlow

The *StreamFlow framework* [4] was created as container-native runtime support for scientific workflows on cloud/HPC environments. In particular, StreamFlow automatically manages the life cycle of complex, multi-container environments, the remote execution of potentially distributed tasks (e.g. MPI, MapReduce), and the data movements between subsequent steps.

StreamFlow was designed to seamlessly integrate with external coordination semantics, allowing users to execute existing workflows on distributed infrastructures without modifying the business logic. In particular, it is fully compliant with the *Common Workflow Language (CWL)* [6] open standard. The same design concept applies to most supported execution environments, described in an external, well-known format whenever such format exists (e.g. Helm charts for Kubernetes deployments or Slurm scripts for HPC workloads).

StreamFlow relies on the *hybrid workflows* paradigm, in which the workflow steps, the topology of deployment locations in charge of their execution, and the mapping relations among steps and locations are included in the same model. This approach fosters portability and reproducibility, directly including the entire execution environment in the workflow specification. Specifically, users rely on a StreamFlow file, conventionally called `streamflow.yml`, to link each step in a workflow with the service that should execute it with a declarative YAML syntax.

2.4.1 Architecture

StreamFlow input is composed of three main elements:

- A workflow description.
- One or more deployment descriptions
- A StreamFlow file to bind each step of the workflow with the most suitable execution environment

2.4.2 How to install

It is possible to install StreamFlow as a Python package with pip command:

```
pip install streamflow
```

StreamFlow requires Python >= 3.8 to be installed on the system. Then the workflow can be executed through the StreamFlow CLI:

```
streamflow run /path/to/streamflow.yml
```

Moreover, StreamFlow can be installed by Docker. To download the latest StreamFlow image, the following command can be used:

```
docker pull alphaunito/streamflow:latest
```

2.5 Clustering Algorithms

There are two main types of clustering algorithms in rCASC implemented in StreamFlow:

- Hierarchical clustering algorithm
- Partitional clustering algorithm

The former type provides a hierarchical decomposition of the set of data using some criterion. The latter one constructs various partitions and then evaluates them by some criterion. Essentially the key differences among these clustering types are related to running time and partitioning assumptions.

Hierarchical clustering algorithms are typically slower than other clustering methods but require fewer assumptions providing more meaningful and subjective division of clusters. Indeed, the hierarchical clustering algorithm requires as input only a similarity measure, while partial clustering requires stronger assumptions such as the number of clusters and the initial centers.

2.5.1 Hierarchical Clustering

Griph

Griph [7] is an R package for the analysis of single-cell RNA-sequencing data. It provides a graph-based approach-based Louvain modularity. Thus, it clustering approach is like agglomerative clustering methods, where every node is initially assigned to its own community and communities are subsequently built by iterative merging.

2.5.2 Partitional Clustering

tSne + K-means

tSne [8] + K-means [9] is often used for the analysis of single cell RNA-seq data. Indeed, firstly data reduction is performed using the tSne method, then cells are clustered by the k-means method, which aims partitioning the points into k groups, such that the sum of squares from points to the assigned cluster centers is minimized.

SIMLR

SIMLR [10], single-cell interpretation via multi-kernel learning, is a framework for learning a similarity measures from single-cell RNA-seq data in order to perform dimensionality reduction, clustering and visualization. In particular, the clustering task is provided through k-mean clustering.

3 Methods

3.1 Executing command

To run the workflow, users must execute the following command

```
bash testTime.sh
```

The script below is present in testTime.sh file. The first part of the script, i.e. lines 1 to 6, checks if the output directory already exists, otherwise the script creates it. Then the second part implements the StreamFlow workflow execution.

```
DIR=”ResultsWorkflow”

if [ -d “$DIR” ]; then
  echo “$DIR exists”
else
  mkdir ResultsWorkflow
fi

now=$(date)
echo “Start time : $now” >| time.txt
streamflow run path/to/testTime.yml –outdir ResultsWorkflow
now=$(date)
echo “End time : $now” >> time.txt
```

3.2 StreamFlow’s file

The testTime.yml reported in hereafter contains the description of three elements required by StreamFlow as described in section 2.4.

```
#!/usr/bin/env streamflow
version: v1.0
workflows:
  master:
    type: cwl
    config:
      file: /path/to/testTime.cwl
      settings: path/to/config.yml
    bindings:
      - step: /
        target:
          model: ssh-model
models:
  ssh-model:
    type: ssh
    config:
      nodes:
        - worker1
        - worker2
        - worker3
        - worker4
        - worker5
        - worker6
        - worker7
```

```
      username: ubuntu
      sshKey: /home/ubuntu/.ssh/id_rsa
      maxConcurrentSessions: 8
```

The above descrition of the testTime.yml file includes a link to other two files reporting the workflow description (i.e testTime.cwl) and the deployment descriptions as for instance R scripts or count Matrix (i.e. config.yml).

testTime.cwl file

```
#!usr/bin/env cwl-runner
cwlVersion: v1.0
class: Workflow
requirements:
  ScatterFeatureRequirement: [11]
inputs:
  fileRPermutationP: File
  skeletonPermutationP: File
  index_prova: string[]
  matrix: File
  nCluster: File
outputs:
  mtxKilledCell:
    type:
      type: array
      items: File
    outputSource: permutationP/mtxKilledCell
  mtxPermutationP:
    type:
      type: array
      items: File
    outputSource: permutationP/mtxPermutationP

steps:
  permutationP:
    run: permutationClusteringP.cwl
    scatter: index
    in:
      fileRPermutationP: fileRPermutationP
      mtxTopX: matrix
      skeleton: skeletonPermutationP
      index: index_prova
      nCluster: nCluster
    out: [mtxKilledCell, mtxPermutationP]
```

The script above is divided in the following sections:

- inputs -> the inputs necessary to each workflow step (e.g. R scripts and the additional parameters)
- outputs -> files or directories saved inside the output folder.
- steps -> the description of the tasks executed in step.

config.yml file

```
skeletonPermutationZero:
  class: File
  path: ../Zero/permutationClusteringSW.R
skeletonPermutationP:
  class: File
  path: ../RFiles/skeleton_PermutationClusteringP.R
matrix:
  class: File
  path: ../dataset/RNA-5c.csv
```

The script above, contains a list of the deployment description. For each item is specified:

- class -> the item's class (e.g File, Directory)
- path -> the item's relative path

Finally, the execution environment is specified in the last part of testTime.yml, where ssh protocol is the execution environment for parallelization of the workflow on 8 nodes.

In Fig. 1, the implemented workflow schema with its inputs and outputs is shown.

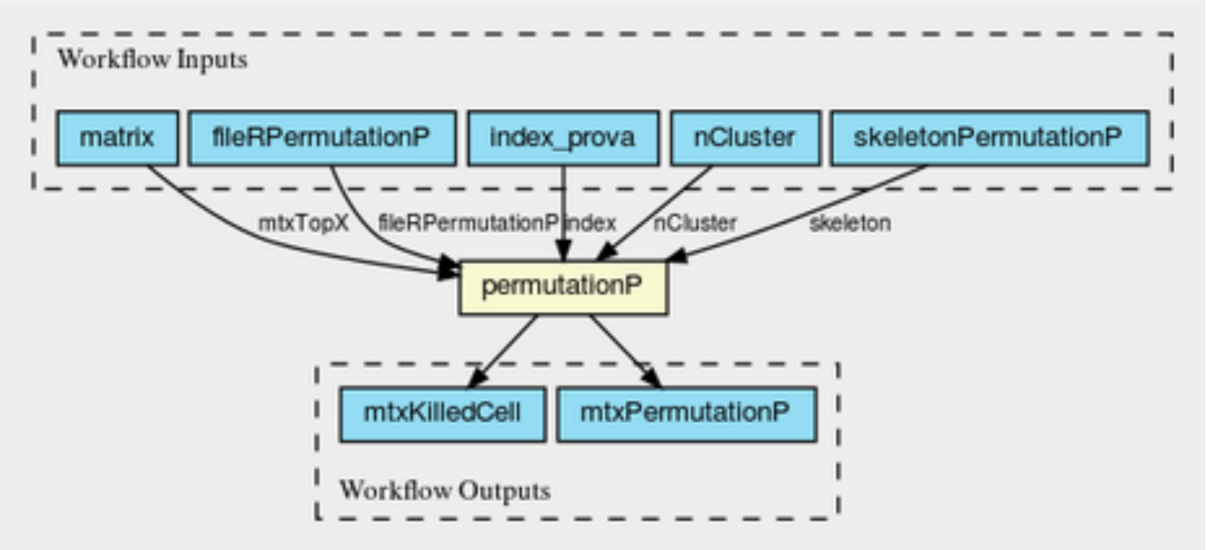

FIG 1

3.3 Analysis

We evaluated the speed-up achievable in clustering using StreamFlow with respect to a single multi-core server. rCASC implements a metrics focused at evaluating the cluster stability: cell stability score (CSS) [1]. CSS requires the perturbation of the initial dataset, i.e. removal of a

random subset of cells, and the clustering of the perturbed dataset. Perturbation/clustering are repeated multiple times to evaluate the overall stability of cells partitioning, measuring how many times each cell remains in a specific cluster independently by the effect of the perturbations disturbing the overall dataset structure.

We tested on the RNA-5c (3904 cells), see exemplary dataset section, SIMLR, tSne + Kmean and Griph.

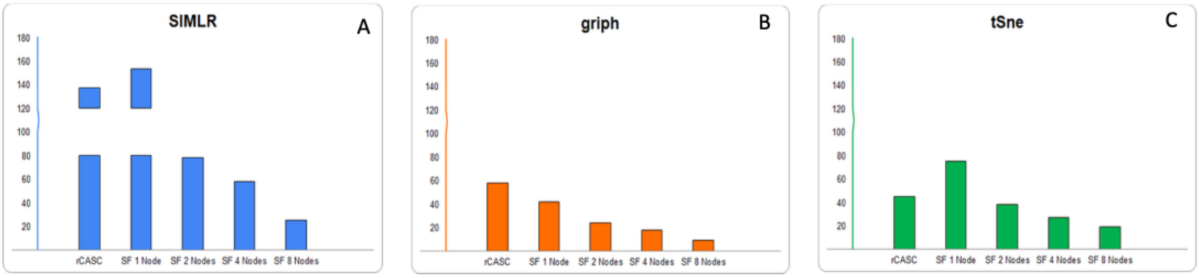

FIG. 2

In Fig. 2, the effect of executing the different clustering algorithms using rCASC on a single multicore server and StreamFlow parallelization is shown. StreamFlow shows a progressive reduction of the clustering time with the progressive increase of the parallelization nodes with all three clustering tools. The effect on SIMLR is the most dramatic with reduction of the clustering time which is linearly inversely proportional to the increment of the computing nodes.

References

1. Alessandri L, Cordero F, Beccuti M, Licheri N, Arigoni M, Olivero M, Di Renzo MF, Sapino A, Calogero R (2021) Sparsely-connected autoencoder (SCA) for single cell RNAseq data mining. NPJ Syst Biol Appl 7 (1):1. doi:10.1038/s41540-020-00162-6
2. Alessandri L, Ratto ML, Contaldo SG, Beccuti M, Cordero F, Arigoni M, Calogero RA (2021) Sparsely Connected Autoencoders: A Multi-Purpose Tool for Single Cell omics Analysis. Int J Mol Sci 22 (23). doi:10.3390/ijms222312755
3. Alessandri L, Cordero F, Beccuti M, Arigoni M, Olivero M, Romano G, Rabellino S, Licheri N, De Libero G, Pace L, Calogero RA (2019) rCASC: reproducible classification analysis of single-cell sequencing data. Gigascience 8 (9). doi:10.1093/gigascience/giz105
4. Colonnelli I, Cantalupo, B., Merelli, I., Aldinucci, M (2021) StreamFlow: Cross-Breeding Cloud With HPC. IEEE Transactions on Emerging Topics in Computing 9 (4). doi:10.1109/TETC.2020.3019202
5. Tian L, Dong X, Freytag S, Le Cao KA, Su S, JalalAbadi A, Amann-Zalcenstein D, Weber TS, Seidi A, Jabbari JS, Naik SH, Ritchie ME (2019) Benchmarking single cell RNA-sequencing analysis pipelines using mixture control experiments. Nat Methods 16 (6):479-487. doi:10.1038/s41592-019-0425-8
6. Crusoe M, Abeln, S., Iosup, A., Amstutz, P.,Chilton, J., Tijanić, N., Ménager, H., Soiland-Reyes, S., Goble, C. (2021) Methods Included: Standardizing Computational Reuse and Portability with the Common Workflow Language. doi: https://doi.org/10.48550/arXiv.2105.07028
7. Serra D, Mayr U, Boni A, Lukonin I, Rempfler M, Challet Meylan L, Stadler MB, Strnad P, Papasaikas P, Vischi D, Waldt A, Roma G, Liberali P (2019) Self-organization and symmetry breaking in intestinal organoid development. Nature 569 (7754):66-72. doi:10.1038/s41586-019-1146-y
8. van der Maate L, Hilton, G. (2008) Visualizing Data using t-SNE. Journal of Machine Learning Research 9
9. Tavazoie S, Hughes JD, Campbell MJ, Cho RJ, Church GM (1999) Systematic determination of genetic network architecture. Nat Genet 22 (3):281-285. doi:10.1038/10343
10. Wang B, Zhu J, Pierson E, Ramazzotti D, Batzoglou S (2017) Visualization and analysis of single-cell RNA-seq data by kernel-based similarity learning. Nat Methods 14 (4):414-416. doi:10.1038/nmeth.4207
11. Zheng GX, Terry JM, Belgrader P, Ryvkin P, Bent ZW, Wilson R, Ziraldo SB, Wheeler TD, McDermott GP, Zhu J, Gregory MT, Shuga J, Montesclaros L, Underwood JG, Masquelier DA, Nishimura SY, Schnall-Levin M, Wyatt PW, Hindson CM, Bharadwaj R, Wong A, Ness KD, Beppu LW, Deeg HJ, McFarland C, Loeb KR, Valente WJ, Ericson NG, Stevens EA, Radich JP, Mikkelsen TS, Hindson BJ, Bielas JH (2017) Massively parallel digital transcriptional profiling of single cells. Nat Commun 8:14049. doi:10.1038/ncomms14049

Figures legend

Fig. 1. Implemented workflow schema: The parallelized clustering algorithm takes in input: count matrix file; the permutation file, which is the script required to execute the parallelization; the permutation Index; the number of clusters to be uses for the partitioning; the total number of permutations to be performed. For each permutation, that runs independently, the following files are generated: mtxkilledCell, where all the cells that were removed as a consequence of the bootstrap algorithm are stored; mtxPermutationP, where all the remaining cells are labeled according to the belonging cluster.

Fig. 2. Time required to cluster the 3904 cells of the RNA5c dataset, using 80 bootstraps respectively on a multi-core server with rCASC and using StreamFlow on 1, 2, 4 and 8 nodes. A) SIMLR, B) Kmeans + tsne and C) Griph.