IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

Porting the Variant Calling Pipeline for NGS data in cloud-HPC environment

This is a pre print version of the following article:

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/1919364 since 2024-05-21T16:08:18Z

Publisher:

Hossain Shahriar, Yuuichi Teranishi, Alfredo Cuzzocrea, Moushumi Sharmin, Dave Towey, AKM Jahangir

Published version:

DOI:10.1109/COMPSAC57700.2023.00288

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

09 November 2026

Preprint version of "A. Mulone, S. Awad, D. Chiarugi, and M. Aldinucci, Porting the variant calling pipeline for NGS data in cloud-HPC environment in 47th IEEE annual computers, software, and applications conference, COMPSAC 2023, Torino, Italy, 2023" DOI: 10.1109/COMPSAC57700.2023.00288

Porting the Variant Calling Pipeline for NGS data in cloud-HPC environment

1st Alberto Mulone
Department of Computer Science
University of Turin
Turin, Italy
alberto.mulone@edu.unito.it

2nd Sherine Awad
Institute of Metabolic Science
University of Cambridge
Cambridge, United Kingdom
sa941@medschl.cam.ac.uk

3rd Davide Chiarugi
Research and Development Group - Computing and Databases Services
Max Planck Institute for Human Cognitive and Brain Science
Leipzig, Germany
chiarugi@cbs.mpg.de

4th Marco Aldinucci
Department of Computer Science
University of Turin
Turin, Italy
marco.aldinucci@unito.it

***Abstract*—In recent years we have understood the importance of analyzing and sequencing human genetic variation. A relevant aspect that emerged from the Covid-19 pandemic was the need to obtain results very quickly; this involved using High-Performance Computing (HPC) environments to execute the Next Generation Sequencing (NGS) pipeline. However, HPC is not always the most suitable environment for the entire execution of a pipeline, especially when it involves many heterogeneous tools. The ability to execute parts of the pipeline on different environments can lead to higher performance but also cheaper executions. This work shows the design and optimization process that led us to a state-of-the-art Variant Calling hybrid workflow based on the StreamFlow Workflow Management System (WfMS). We also compare StreamFlow with Snakemake, an established WfMS targeting HPC facilities, observing comparable performance on single environments and satisfactory improvements with a hybrid cloud-HPC configuration.**

***Index Terms*—StreamFlow, Hybrid workflow, High Performance Computing, cloud computing**

I. Introduction

The computing power available in a single High-Performance Computing (HPC) system has exponentially increased for over 30 years, as documented by the TOP500 Supercomputer Database[1]. A likewise aggressive growth interested the cloud computing systems in the last ten years. It is challenging to estimate the aggregate computing power in the cloud, even if it can be observed that the global cloud computing market is expected to expand at a compound annual growth rate of 14.1% from 2023 to 2030 [1]. The HPC and cloud computing paradigms, developed with different objectives, have been sustained by different applications and (for this) exhibit different computing features.

In this work, we study how the cloud and the HPC paradigms can be jointly used to optimize the execution of modern scientific applications. We adopt the Variant Calling (VC) pipeline, i.e. a Next Generation Sequencing (NGS) application, as a running example. NGS is a class of applications that were not in the traditional basket of HPC applications and thus can fall in the area of one of the cloud service models as Infrastructure as a Service (IaaS). However, computing demand is becoming increasingly critical due to the explosion of NGS data output (to which COVID-19 significantly contributed). Many other applications classes are heading the same path, starting from Artificial Intelligence (AI) applications based on Deep Neural Networks (DNNs) [2].

HPC systems are designed with the primary goal of computing performance. HPC users embrace larger systems to reduce the wall-clock time needed to solve a problem (strong scaling) or increase the problem size while maintaining the constant wall-clock execution time (weak scaling). Many applications can benefit from the HPC approach, even if the primary application class is rooted in scientific applications aiming at solving problems too big for other computing systems: simulations, large equations solving, digital twins, and, recently, the training of very large AI models. Cloud systems are designed to shift the management of computing aspects (from hardware management to configuration to operation) to a specialized firm or business unit, the cloud provider, which provides computing services to the users. Cloud focuses on many services (and micro-services) and their composition rather than a few compute demanding applications.

NGS pipelines generally include all computing applications used to analyze (different) aspects of DNA and RNA data. The term pipeline has become popular in several scientific areas to indicate the execution of a sequence of computational steps organized along a Directed Acyclic Graph (DAG) of data dependencies among different steps, which are implemented by different executables (sequential or parallel). In the context of this work, and more in general in scientific computing, the term *workflow* subsumes the term *pipeline*. A workflow can

[1]https://www.top500.org

be represented as a DAG[2]; each step in a workflow has (data)[3] dependencies on the previous steps and can be performed after they have finished their work correctly. A Workflow Management System (WfMS) is used to manage the data dependencies, the scheduling in the execution environment and other aspects of the workflow execution (e.g. fault tolerance).

Hybrid workflow systems aim to support deploying a single workflow (or a batch of them) across different execution platforms (such as cloud and HPC) exhibiting different compute capabilities in terms of latency, throughput, security, robustness, geographic location, matching each workflow's principal requirements with the best platform to execute it. StreamFlow [2], [3], a WfMS designed by the parallel computing group at the University of Turin, explicitly aims to reach this goal by implementing the *hybrid workflow* paradigm. Supporting different HPC and cloud environments, it allows orchestrating the distributed execution of a workflow through a declarative description of the available execution environments.

This work aims to evaluate the potential of a combined cloud-HPC environment in the NGS domain evaluating speed performance. For this, we ported a Variant Calling pipeline to a hybrid workflow, optimizing the data movements between systems without a shared file-system and exploiting the heterogeneity of platforms. The original VC pipeline was developed using *Snakemake*, a WfMS which exposes a proprietary Domain Specific Language (DSL) for describing workflow. Firstly, we defined the workflow according to the StreamFlow programming model, using the Common Workflow Language (CWL) open standard. Next, we studied and improved the pipeline's bottlenecks by introducing more parallelism in the slow pipeline stages. To this end, we studied each step's resource requirements in detail, identifying which could be more conveniently executed on HPC or cloud environments. Furthermore, we analyzed the amount of data involved in each data dependency to decide if moving the computation to another platform is worth it, i.e. if the performance gain is greater than the data movement overhead. Both these aspects are crucial to configure an efficient hybrid workflow execution.

II. Background

A. Hybrid workflow

The workflows, in general, are heterogeneous, i.e. the steps in the workflow can have different requirements to be executed or simply for optimal performance. However, having a single environment that satisfies the requirements of all the workflow steps can be difficult. In addition, it can be expensive to keep this specific environment for the entire workflow execution but exploit the maximum of the resources only for some steps.

The hybrid workflow paradigm aims to execute the workflow steps in their optimal environment and use the involved environments only for the necessary time. However, it does not guarantee better performance than executions in a single environment. Involving multiple environments, it could be necessary to move data between them if they do not have a shared memory zone. In the case of scientific workflows, data sizes can be substantial; moving these data will involve time affecting workflow performance. This cost is one of the main aspects in deciding the configuration for hybrid executions. Finding the right compromise between the cost of moving data and the performance gained in using a different environment is necessary. The hybrid workflow paradigm can have different implementations as Pegasus [4], DagOnStar [5] and Mashup [6]; another implementation is the StreamFlow WfMS.

B. StreamFlow WfMS

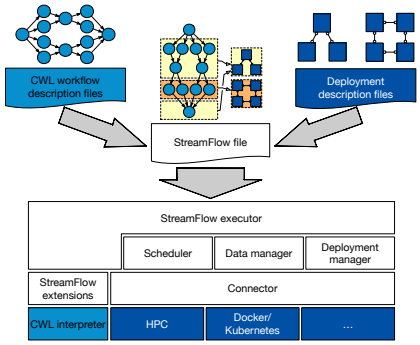


Fig. 1: StreamFlow architecture. On the top, the components of the StreamFlow file; below, StreamFlow codebase showing the different modules as Connector, Scheduler, Data manager.

StreamFlow is a WfMS that orchestrate the hybrid workflow in distributed and heterogeneous environments. It adheres to the CWL standard to describe the workflow model. However, it has a clear division between the workflow model and the configuration for hybrid workflow. The *StreamFlow file* is a YAML file where are defined the location of the CWL file (`.cwl` extension), the environments involved and how to access them, and the coupling between steps and environments. This approach makes the workflows more portable and easy to configure for hybrid execution workflows already developed with the CWL standard.

StreamFlow architecture has well-defined modules, as shown the Fig. 1. It supports many environments to allow workflow execution on multi-environments; for this goal, it implements Connector objects to communicate with the different environments, cloud and HPC. Many modules use these connectors. First of all, the deployment manager creates and destroys them. The Scheduler chooses the best resources to execute a step; for this goal, it uses the connector to check the status of the remote resources. When multiple resources are available, a *Policy* is applied to choose the best. The default Policy implementation tries to minimize the data movement (data locality), however, StreamFlow provides an interface to allow the user to implement the preferred strategy. The Data manager is essential for hybrid execution to know the

[2]Not all workflows can be modelled as DAGs; for example, an exception are the iterative workflows

[3]Dependencies can be modelled as *control flow* and *data flow*. In the case of control flow, the dependencies do not explicitly carry data between subsequent steps.

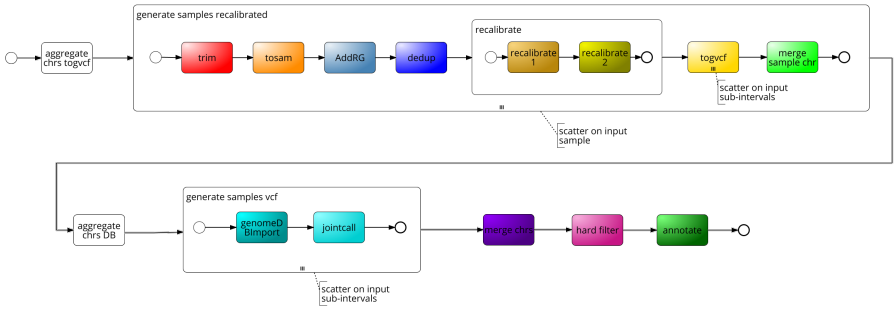


Fig. 2: Variant Calling Pipeline represented with BPMN standard.

data's location and, when necessary, copy them across the environments through the connector. Indeed, when we want to execute a step in a different environment, the input data of the step must be present. If the environments where the data are located and the environment where the step must be executed do not have a shared persistence area, there is an additional cost to copy the data.

III. Resources used

The HPC HLRS centre in Stuttgart, Germany, specifically the Hawk infrastructure, was used for the speedup analysis of the tool used in the VC pipeline. Instead, the resources of the Cineca centre in Bologna, Italy, were used for the executions in the experiment section. Cineca provides access to its cloud resources as IaaS. The cloud infrastructure, named ADA cloud, is based on OpenStack Wallaby. We used the Galileo100 infrastructure for the HPC resources. The connection speed is 150MB/s between the VM created on ADA cloud and HPC Galileo100, and it is very important for hybrid executions. The tools used in the VC pipeline are: Trim Galore (v0.6.6), Bowtie2 (v2.4.1), GATK4 (v4.2.6.1), Picard (v2.27.1), Annovar (v2020Jun08).

IV. Variant Calling pipeline

The advance of NGS technologies has enabled a wide expansion of clinical genetic testing both for inherited conditions and diseases. Human genetic variation study is the discovery and description of the genetic contribution to many human diseases. It entails both variant calling and interpretation. A variant call is a conclusion that there is a nucleotide difference versus some reference at a given position in an individual genome or transcriptome, often referred to as a SNV. An accurate variant calling in NGS is the critical step upon which the interpretation process relies. There are several methods for calling variants. Here we introduce GATK4, a toolkit whose tools are among the most widely used for variant calling pipelines [7], [8].

A. Methods

Fig. 2 shows the pipeline. We use Trim Galore to trim adapters[4]. We use Bowtie2 [9] to align reads to the human genome hg38[5]. Then we follow the GATK pipeline [10]–[13]. We add read groups and mark duplicates using Picard[6]. We recalibrate bases using BaseRecalibrator and ApplyBQSR from GATK4. We call haplotype caller in GVCF mode to generate Genomic VCF (gVCF). We import single-sample GVCFs into GenomicsDB before joint genotyping. We then perform joint genotyping on the GenomicsDB workspace created with GenomicsDB. These last three GATK4 tools, used by the *togvcf*, *genomeDB* and *jointcall*, have a parallel version of the tools, but they are still in beta; thus, we decided not to use them. Nevertheless, to improve the pipeline performance, we split the chromosomes interval[7] and generate new step instances to execute them in parallel. In the figure, indeed, on the three steps, there are two scatters[8] on so-called sub-intervals. These scatters produce many small fragments of the original output data; thus, two new steps, named *merge* * in the figure, are added to combine the results. These new steps use the SortVcf tool of Picard toolkit. Furthermore, two steps, named *aggregate* *, are used to create the sub-intervals with a different number of chromosomes. We hard filter the variants and then finally annotate the variants using Annovar [14].

B. Speedup analysis

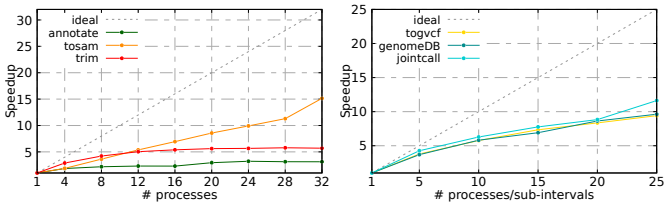


Fig. 3: Strong scaling. On the left, the steps *trim*, *tosam*, *annotate*. On the right, the steps *togvcf*, *genomeDB* and *jointcall* with increasing the sub-interval size and cores.

The first activity on the pipeline was to improve its performance. We focus on optimizing the tools' hyperparameters

[4]https://www.bioinformatics.babraham.ac.uk/projects/trim_galore/

[5]https://gatk.broadinstitute.org/hc/en-us/articles/360035890811-Resource-bundle

[6]http://broadinstitute.github.io/picard/

[7]https://gatk.broadinstitute.org/hc/en-us/community/posts/360063088471-Speeding-up-GenotypeGVCFS-GATK4

[8]In CWL, the multi-instance pattern is implemented by the Scatter feature

to exploit the parallelism. We isolated and analyzed each step individually, executing each with an incremental number of cores and the input data from our Whole Exome Sequencing (WES) dataset. This study aims to measure the *strong scaling*, which is subject to the Amdahl's law. The ideal speedup increases linearly with the number of processes.

Fig. 3, on the left, shows the speedup analysis of three tools with a hyperparameter to introduce parallelism. The tools are used in the steps *trim*, *tosam* and *annotate*. Instead, on the right of the figure, there is the speedup of the tool used in *togvcf*, *genomeDB* and *jointcall*. The tools involved, as said, do not have hyperparameters for parallelization, but the scatter was introduced in the steps for this goal. The *aggregate* * steps can change the number of instances of these steps; thus, we analyze the performance of the steps with an increasing number of sub-intervals considering all 25 chromosomes.

From these analyses, we identify the optimal number of cores. For the *trim* step is 16 cores for each instance, while for *tosam* 32 core and *annotate* about 24 cores. Instead, the steps *togvcf*, *genomeDB*, and *jointcall* perform well when generating an instance for each chromosome and are executed in parallel.

V. WfMSs experiments

In these experiments, we execute the pipeline with the cloud and HPC environments using two WfMSs: StreamFlow and Snakemake. Snakemake is a famous WfMS that support the execution on single environments of different size from a single-core workstation at clusters [15]. We compare the executions on single environments of the two WfMSs and then the execution with StreamFlow with hybrid workflow in different configurations. As in the speedup analysis, we used our WES dataset in all the experiments. The speedup analysis executed on the HPC HLRS was valid in the HPC Galileo100, thus the same analysis on this facility are not showed. The dataset have ten samples, the reference is human genome hg38 and we analyze all 25 chromosomes.

A. Cloud environment

We create a large VM with 96 cores (Sec. III), but the available resources do not fit the required resources. We know the optimal number of cores for each instance from the speedup analysis. Furthermore, we know a priori how many instances will be generated with our dataset. The *togvcf* step has 250 instances (10 samples · 25 chromosomes), and *genomeDB* and *jointcall* have 25 instances (25 chromosomes). Each instance requires one core; thus, before executing the pipeline, we identify three critical steps: *trim*, *tosam*, *togvcf*.

Fig. 4 shows the pipeline execution with StreamFlow and Snakemake on the cloud environment. The execution is represented with the Gantt chart, and the number inside the bar indicates multi-instance cardinality. For each experiment, also in the other sections, we performed three runs, the figure represents the average run, and the whiskers represent the standard deviation of the steps. In this environment, we used for each instance of *trim* and *tosam* eight cores; in this way, we do not have instances blocked waiting for the

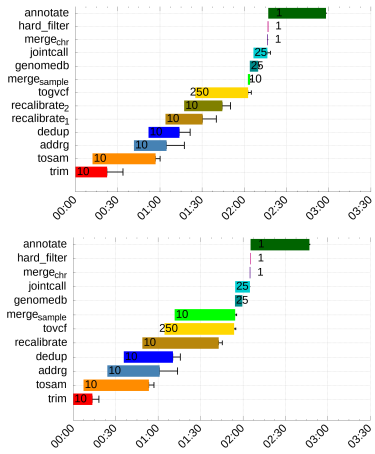


Fig. 4: Pipeline executions entirely on the cloud environment. In the top figure, StreamFlow execution; in the bottom figure, Snakemake execution.

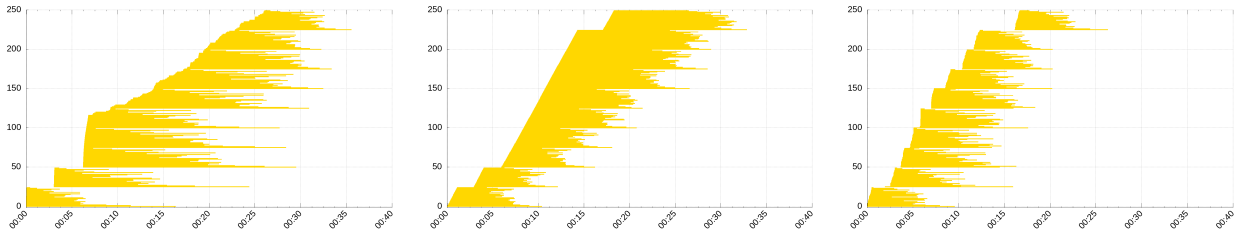
resources. Instead, Fig. 5a shows in detail the execution of *togvcf* instances with StreamFlow (it is equal in Snakemake). The first 100 *togvcf* instances are executed in parallel, but when it is necessary to deploy new instances, they are blocked, waiting for the resources.

The two WfMS runs are comparable; they have similar time execution. The relevant difference in the two executions is handling the *merge sample* step. This step needs to input 25 files of the same sample analyzed on the 25 chromosomes. In the case of Snakemake, when these files are created, it launches the step instance. In StreamFlow, this does not happen because all instances are synchronized at the end of the *togvcf* step scatter. However, in this pipeline, this synchronization has little impact on the execution time.

B. HPC environment

The second experiments are on HPC resources. Fig. 6 shows StreamFlow and Snakemake executions, which have similar times also in this environment. In both WfMSs, we can see how the duration of some steps, e.g. *tosam*, is reduced because the HPC provides the appropriate resource for each instance. Despite this, the total execution is worse than in the cloud, and the reason is the high latency in scheduling jobs of HPCs. In our pipeline, steps with a high number of instances, *togvcf*, *genomeDB*, and *jointcall*, are significantly affected. Fig. 5b shows the *togvcf* instances on the HPC environment in detail. We can see a slower deployment of instances because each instance has a waiting time for allocating the resources in the HPC. This time varies depending on the HPC workload because it handles different requests from many users.

The WfMSs executor are generally executed on the HPC as a job, but it wastes resources since WfMSs do not need high computational power. In this experiment, we use the hybrid workflow paradigms to execute the StreamFlow executor on



(a) *togvcf* instances on cloud environment. (b) *togvcf* instances on HPC environment. (c) *togvcf* instances on cloud-HPC.

Fig. 5: Scheduling of the *togvcf* instances in the experiments with StreamFlow on different configurations.

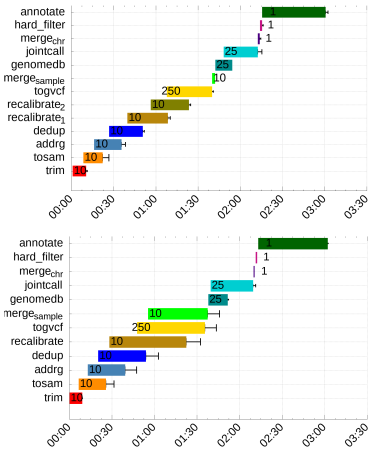


Fig. 6: Pipeline executions entirely on the HPC environment. In the top figure, StreamFlow execution; in the bottom figure, Snakemake execution.

the cloud and the workflow steps on the HPC facility[9].

C. Hybrid environment

In the previous experiments, we see how this pipeline is unsuitable for a cloud environment due to the available resources, nor for an HPC environment due to the number of jobs to submit. Thus this suggests that a hybrid configuration could be a good solution, taking the best of both environments. StreamFlow allows hybrid pipeline executions, while Snakemake does not. For an optimal hybrid configuration, we must consider the amount of data to be moved and the step requirements. The step requirements are known by the speedup analysis done, while a study for the amount of data is needed, analyzing average input and output step sizes on our dataset. We decide to execute *trim*, *tosam*, *AddRG* and *dedup* on the HPC facility: *trim* and *tosam* for their cores' request, while *AddRG* and *dedup* because *tosam* and *AddRG* produce large outputs (both about 10GB per sample with our dataset) that are costly to move. While the other steps are executed on the cloud. For the cloud environment, we use the same VM used in previous experiments. Fig. 7 shows this configuration in the upper figure; we have better performance than using only HPC or only cloud. However, the *togvcf* step is still a bottleneck; both on cloud and HPC, all the instances can not execute fully parallel as we saw in the Fig. 5a and 5b.

Instead of running the entire step on one environment, we can distribute the workload of the step on both environments. This strategy does not guarantee better performance because we must always consider the cost of moving the data. However, in this case it is not very expensive; the *togvcf* step has in input about 3.5GB per sample. In addition, it is necessary to move only those samples that are involved in HPC jobs. Fig. 7, the lower figure, shows this second configuration, same as the former, except that the instances *togvcf* are distributed between cloud and HPC. Fig. 5c shows the execution of *togvcf* instances in this configuration. The instances are executed more in parallel than only cloud and only HPC executions. We can see a performance improvement, which brings the pipeline execution under 2.30h.

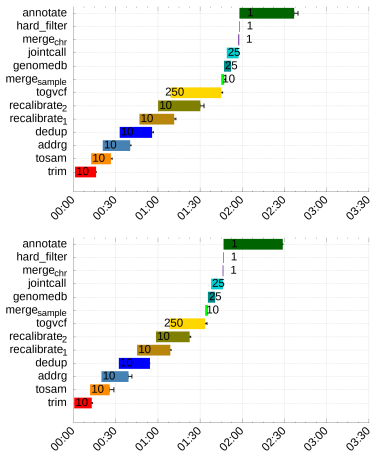


Fig. 7: Pipeline executions on StreamFlow with first four steps in the HPC, all the others in the cloud. In the top figure, *togvcf* is fully on the cloud; instead, in the bottom figure, *togvcf* is distributed between the cloud and HPC. All the input data was on the cloud.

VI. Conclusions

This work describes a practical approach to configuring a pipeline for hybrid execution and evaluates the Stream-

[9]StreamFlow allows the communications with the HPC Queue manager from outside (in this case from the cloud environment) because it makes a ssh connection and executes the bash commands (e.g. sbatch).

Flow WfMS as an implementation of the hybrid workflow paradigm. As a running example, we selected a substantial NGS pipeline, a VC pipeline. We studied it from a computer science viewpoint to make its execution more flexible and efficient on different modern computing platforms, such as cloud, HPC and hybrid cloud-HPC settings. A detailed study of the pipeline speedup brought us to define the best computing hyperparameters to execute the pipeline (e.g., number of threads, memory and I/O pressure for each stage). The hyperparameter optimization speeds up the original pipeline by 300% on the cloud-only setting. The main goal of this work is to demonstrate that the hybrid cloud-HPC workflow paradigm can improve the performance of NGS workflows. This is certainly true for the VC pipeline, where we experiment that a medium-sized dataset already requires many cores for some pipeline stages. The cloud fails to support the pipeline at workload peaks but is enough for most of the execution. On the contrary, the HPC well supports compute-intensive stages but falls short in executing too many independent small tasks as needed for some pipeline stages. The hybrid workflow matched each pipeline stage with the most suitable execution setting. In this way, we achieved an additional speedup of 20% in the cloud-only setting. Eventually, we compared the Snakemake version of the VC pipeline with the StreamFlow version. They exhibit a comparable performance in the cloud and HPC settings at the best of their capability (in terms of hyperparameter setting). Snakemake does not support hybrid cloud-HPC settings. There are many future works. When the tools, mentioned in section IV-A, are no longer in beta, it will be necessary to consider whether to include them in the pipeline instead of the introduced scatters. A study of the speedup will be needed to see if the hybrid paradigm will still be useful. In particular if the *togvcf* step will need several cores, then with a dataset with many samples the hybrid executions will be able to bring the same advantage brought to the figure 5c execution. Another future work aims to carry out similar studies on other pipelines to generalize the results obtained. In addition, another future goal is to also study in depth the costs in using hybrid executions with different IaaS provider (e.g. AWS) and with different cloud configurations (single large VM, multiple small VMs, Kubertenes). The term cost means several aspects: economic cost in accessing these environments, but also cost in terms of energy.

ACKNOWLEDGMENT

This work was supported by the Spoke 1 "FutureHPC & BigData" of ICSC - Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing, funded by European Union - NextGenerationEU; the ACROSS project, "HPC Big Data Artificial Intelligence Cross Stack Platform Towards Exascale" which has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No. 955648; the HPC-EUROPA3 project funded under EC H2020 (INFRAIA-2016-1-730897). We gratefully acknowledge the support of José Gracia, from the HLRS High Performance Computing Center, Stuttgart, and the computer resources and technical support provided by the HLRS. We also acknowledge the CINECA award under the ISCRA initiative, for the availability of high performance computing resources and support.

REFERENCES

[1] "Cloud computing market size, share & trends analysis report by service (SaaS, IaaS), by end-use (bfsi, manufacturing), by deployment (private, public), by enterprise size (large, smes), and segment forecasts, 2023 - 2030," Report ID: GVR-4-68038-210-5, 2023. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/cloud-computing-industry

[2] I. Colonnelli, B. Cantalupo, R. Esposito, M. Pennisi, C. Spampinato, and M. Aldinucci, "HPC Application Cloudification: The StreamFlow Toolkit," in *12th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and 10th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM 2021)*, ser. Open Access Series in Informatics (OASIcs), J. a. Bispo, S. Cherubin, and J. Flich, Eds., vol. 88. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 5:1–5:13.

[3] I. Colonnelli, B. Cantalupo, I. Merelli, and M. Aldinucci, "StreamFlow: cross-breeding cloud with HPC," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1723–1737, 2021.

[4] E. Deelman, K. Vahi, M. Rynge, R. Mayani, R. F. da Silva, G. Papadimitriou, and M. Livny, "The evolution of the Pegasus workflow management software," *Computing in Science and Engineering*, vol. 21, no. 4, pp. 22–36, 2019.

[5] D. D. Sánchez-Gallegos, D. D. Luccio, S. Kosta, J. L. G. Compeán, and R. Montella, "An efficient pattern-based approach for workflow supporting large-scale science: The dagonstar experience," *Future Gener. Comput. Syst.*, vol. 122, pp. 187–203, 2021.

[6] R. B. Roy, T. Patel, V. Gadepally, and D. Tiwari, "Mashup: making serverless computing useful for HPC workflows via hybrid execution," in *PPoPP '22: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seoul, Republic of Korea, April 2 - 6, 2022*, 2022, pp. 46–60.

[7] D. Koboldt, "Best practices for variant calling in clinical sequencing," *Genome Medicine*, vol. 12, no. 1, pp. 1–13, 10 2020.

[8] J. Chen, X. Li, H. Zhong, Y. Meng, and H. Du, "Systematic comparison of germline variant calling pipelines cross multiple next-generation sequencers," *Scientific Reports*, vol. 9, no. 1, p. 9345, 06 2019.

[9] B. Langmead and S. Salzberg, "Fast gapped-read alignment with bowtie 2." *Nature methods*, vol. 9, pp. 357–359, 03 2012.

[10] G. A. Van der Auwera, M. O. Carneiro, C. Hartl, R. Poplin, G. del Angel, A. Levy-Moonshine, T. Jordan, K. Shakir, D. Roazen, J. Thibault, E. Banks, K. V. Garimella, D. Altshuler, S. Gabriel, and M. A. DePristo, "From fastq data to high-confidence variant calls: The genome analysis toolkit best practices pipeline," *Current Protocols in Bioinformatics*, vol. 43, no. 1, pp. 11.10.1–11.10.33, 2013.

[11] R. Poplin, V. Ruano-Rubio, M. A. DePristo, T. J. Fennell, M. O. Carneiro, G. A. Van der Auwera, D. E. Kling, L. D. Gauthier, A. Levy-Moonshine, D. Roazen, K. Shakir, J. Thibault, S. Chandran, C. Whelan, M. Lek, S. Gabriel, M. J. Daly, B. Neale, D. G. MacArthur, and E. Banks, "Scaling accurate genetic variant discovery to tens of thousands of samples," *bioRxiv*, 2018.

[12] M. A. DePristo, E. Banks, A. McKenna, T. J. Fennell, A. M. Kernytsky, A. Y. Sivachenko, K. Cibulskis, S. B. GABRIEL, D. Altshuler, M. J. Daly, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. D. Angel, M. A. Rivas, and M. Hanna, "A framework for variation discovery and genotyping using next-generation dna sequencing data," *Nature genetics*, vol. 43, no. 5, pp. 491–498, 2011.

[13] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly *et al.*, "The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data," *Genome research*, vol. 20, no. 9, pp. 1297–1303, 2010.

[14] K. Wang, M. Li, and H. Hakonarson, "ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data," *Nucleic Acids Research*, vol. 38, no. 16, pp. e164–e164, 07 2010.

[15] J. Köster and S. Rahmann, "Snakemake - a scalable bioinformatics workflow engine," *Bioinform.*, vol. 34, no. 20, p. 3600, 2018.