Evaluating Workflow Management Systems: A Bioinformatics Use Case

Conference Paper · December 2018

DOI: 10.1109/BIBM.2018.8621141

CITATION
1

READS
359

6 authors, including:

Elise Larsonneur
Atomic Energy and Alternative Energies Commission
13 PUBLICATIONS **189** CITATIONS
SEE PROFILE

Jonathan Mercier
Centre National de Recherche en Génomique Humaine
23 PUBLICATIONS **314** CITATIONS
SEE PROFILE

Nicolas Wiart
Centre National de Recherche en Génomique Humaine
4 PUBLICATIONS **14** CITATIONS
SEE PROFILE

Vincent Meyer
CEA
72 PUBLICATIONS **913** CITATIONS
SEE PROFILE

Evaluating Workflow Management Systems: A Bioinformatics Use Case

Elise Larsonneur*†, Jonathan Mercier*†, Nicolas Wiart*,
Edith Le Floch*, Olivier Delhomme* and Vincent Meyer*
*Centre National de Recherche en Génomique Humaine (CNRGH),
Institut de Biologie François Jacob, CEA, Université Paris-Saclay, F-91057, Evry, France.
Email: elise.larsonneur@cng.fr, jonathan.mercier@cng.fr
†Both authors contributed equally to this work.

***Abstract*—Biological knowledge discovery is a tedious task which relies on computationally intensive analyses to process huge volumes of data and metadata. This task can be facilitated by a well-designed workflow management system, capable of deploying and efficiently orchestrating the scientific analysis. However choosing the right workflow management system remains a critical challenge. In this study, we propose ten metrics for evaluating the efficiency of workflow management systems, and show how the Pegasus-mpi-cluster and Snakemake workflow management systems gave the best overall performance, closely followed by Nextflow.**

***Index Terms*—Workflow management systems, bioinformatics, benchmark**

I. Introduction

In Bioinformatics, we are faced with multiple choices of heterogeneous tools developed for an ever increasing number of applications and ever growing volumes of data. The computational environment has also evolved from the early days of personal computers with a single then multiple CPU cores, to today's grid computing systems. Concurrently, workflows have become more and more complex, using tailored parallelizing strategies across a single or multiple machines and with conditional execution. Furthermore, modern bioinformatics workflows tend to require three key features: they need (i) to allow efficient performance for embarrassingly parallel and intensive computing, (ii) to be shareable, and (iii) to ensure reproducibility of experiments [1].

This has led to the emergence of workflow management systems (WMS) [2] to support physicists and biologists in the task of data processing. WMS can be classified in several ways, for example, by their paradigm (explicit or implicit), configuration system (text or scripting), workbench (command line or graphical user interface) or targeted infrastructure environment (desktop, high-performance computing, cloud). In parallel, several initiatives [3] have emerged to provide common workflow languages [4] (CWL, WDL) to facilitate and formalize workflow development. The choice of WMS and related environment will condition the ease of development, as well as compatibility and sharing. However, it remains difficult to obtain a comprehensive view of the computing efficiency of WMS.

Thus, in this study we evaluate some popular WMS & workflow languages — Cromwell-WDL [5], Nextflow [6], Pegasus-mpi-cluster [7], Snakemake [8], Toil-CWL [9] (Table I) — and we assess them in terms of their computational efficiency using the following metrics: elapsed time, CPU and RAM footprint, number of inodes used and I/O operations. To evaluate the WMS, we use a bioinformatics pipeline with standard processing steps: sequential, parallelized and merged bioinformatics tasks. Based on the results obtained, we provide recommendations to assist with the choice of WMS.

II. Materials and Methods

A. Computational resources

All the experiments in this study were performed on a High Performance Computing (HPC) node at the French National Research Center for Human Genomics (CNRGH). This node possesses 94 GB of RAM and a 12-core Intel® Xeon® X5650. Data is stored on a network file system (NFS) through a NetApp filer.

B. Workflow and Dataset

We use in-house whole genome sequencing data with known pedigree information. The VCF file contains genotypes and variants from a CEU parent-offspring trio comprising NA12878 (child), NA12891 (father) and NA12892 (mother). For each member, 150-bp paired-end whole genome sequencing data were generated on the Illumina® HiSeq X system, from a PCR-free library (40x).

To evaluate the performance of each WMS, we used the analysis flow performed with LODSeq [10] (Fig. 1). This pipeline is a typical representation of bioinformatics needs. The workflow is composed of nine interdependent steps alternating sequential tasks, parallelized batch of jobs over multiple cores which are merged in a final joining task. Using our dataset, 146 tasks were created and processed by each WMS [11].

C. Benchmarking and Metrics

We defined a list of ten metrics to evaluate and rank the solutions, where lower values indicate better performance. The first metric is elapsed time, or time required to complete the process. The max CPU percentage and the median CPU percentage describe the WMS CPU usage (a lower CPU overhead frees up more CPU resources for the workflow processing).

TABLE I
TESTED WORKFLOW MANAGEMENT SYSTEMS

Workflow	Snakemake	Nextflow	Cromwell	Toil	Pegasus-mpi-cluster (PMC)
Paradigm	Convention	Convention	Convention	Class & Convention	Configuration
Syntax	Implicit	Implicit	Explicit	Explicit	Explicit
Implementation	Python	Groovy/JVM	Scala/JVM	Python	C++/mpi
Input/output streaming	Yes ($\geq$5.0.0)	Yes	Yes	Yes	Yes
Workflow modularity	Yes	No	Yes	Yes	No
Workflow versioning	Yes (optional)	Yes	No	No	No
Native workflow sharing	No	Yes (git)	No	No	No
WDL or CWL support	CWL[a]	CWL[a]	WDL, CWL[a]	CWL, WDL[a]	No
Program (Version)	snakemake (4.8.0)	nextflow (0.32.0)	cromwell (36)	cwltoil (3.18.0)	pegasus-mpi-cluster (4.8.2)

[a]Proof of concept or alpha version.

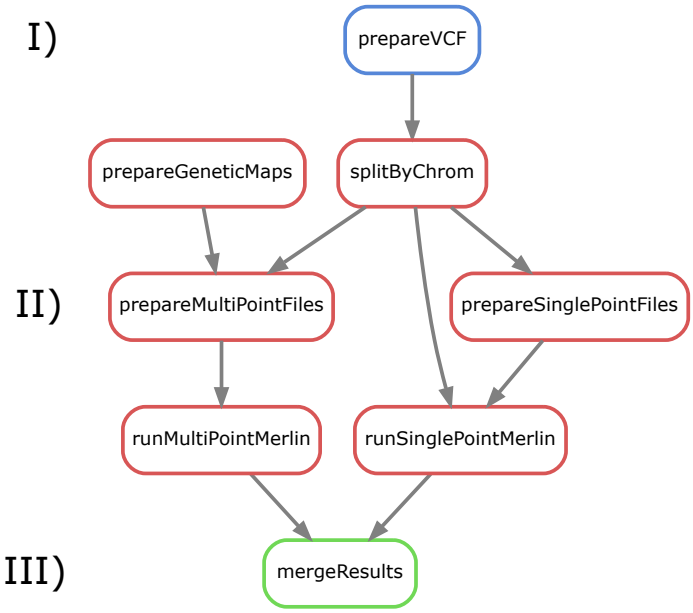

Fig. 1. Directed acyclic graph flow of the LODSeq workflow[10]. Part (i) sequential, (ii) parallelized and (iii) merged.

The max and median memory values give the WMS memory footprint. The voluntary and involuntary context switches are procedures to pause and store a process and its current state. However, these operations are costly and may explain low performance. The I/O wait time (read/write file operations to disk), is much slower than a CPU computation, therefore multiple I/O operations tend to be slower than RAM memory operations. The idle time reflects the unproductive CPU time during the workflow processing (e.g. the wait for process termination). The number of inodes consumed during a task may impact performance on file systems (e.g. Sun's ZFS).

III. RESULTS AND DISCUSSION

This study shows that the implementation of WMS can have a significant impact on their overall performance (Fig. 2, Table II). One of the major differences observed between WMS are the algorithms used to compute dependencies. Some WMS based on the make-like approach, such as Snakemake need to precompute the Direct Acyclic Graph (DAG). Pegasus-mpi-cluster (PMC) requests an explicit DAG of jobs as an input text file, which may require additional work to generate it manually or using a different tool. Toil uses an API to describe job order programmatically. This set of jobs produces a predefined hard coded DAG that can be extended with dynamic job creation. Cromwell's WDL provides a similar feature with conditional statements. Nextflow uses a DAG-free dependency discovery based on a top-down approach.

Results highlight that WMS which use a Java Virtual Machine (JVM) tend to consume more memory (e.g Cromwell and Nextflow). Moreover, the JVM configuration requires substantial expertise. Optimal, minimum and maximum heap memory [12] values and garbage collector algorithms [13] have to be set. In this respect, Nextflow showed average performance during a use case completed in 4.0 minutes with a median memory consumption of $\sim$320 MB. Cromwell produced the highest inode consumption (64 inodes per task) and memory footprint ($\sim$660 MB).

Cwltoil implementation showed the highest CPU overhead. Notably, frequent context switches were observed which introduced computation latency. Cwltoil showed the longest computation time (6 minutes) and was one of the largest consumers of inodes. Compared to other WMS, Snakemake showed average usage of CPU and was the fastest WMS to process the workflow (3.7 minutes). Snakemake implementation (python-based as cwltoil) also showed frequent voluntary context switches during its execution, although to a lesser extent.

PMC was the second fastest WMS (4.0 minutes, same as Nextflow); it had the lowest CPU consumption and memory footprint and showed the best overall performance. The downside of PMC, observed for this case, was mainly the random task selection during the processing of the DAG which is sub-optimal for tasks with heterogeneous runtimes.

The standard language support varies according to the WMS selected. For PMC, there is still no CWL support. Moreover additional evaluation of WMS using different workflow languages and API may modify the current WMS rankings.

Furthermore, the present study analyzes WMS run locally on a single computation node, and we thus plan to extend the evaluation with task orchestration over multiple computing nodes.

TABLE II
WORKFLOW MANAGEMENT SYSTEMS RANKING BY METRICS

Workflow	Elapsed time	Maximum CPU usage	Median CPU usage	Maximum memory	Median memory	n.o. voluntary context switches/sec	n.o. involuntary context switches/sec	I/O wait time	Idle time	n.o. inodes per task	Global rank
cromwell	4	3	2	5	5	1	1	5	4	5	3.5
nextflow	2	3	3	4	4	2	2	3	2	3	2.8
PMC	2	1	1	1	1	3	4	2	3	1	1.9
snakemake	1	2	4	2	2	4	3	1	1	2	2.2
cwltoil	5	3	5	3	3	5	5	4	5	4	4.2

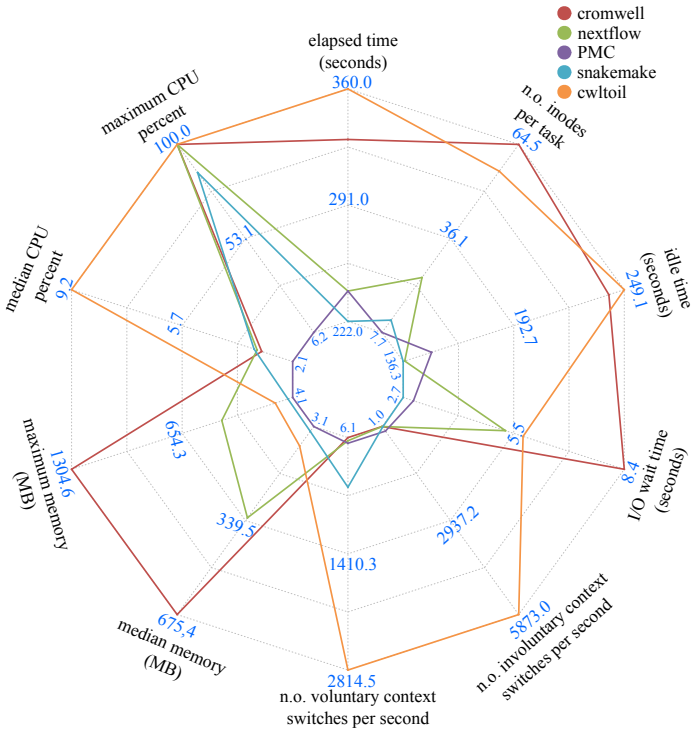

Fig. 2. Workflow efficiency for each of the metrics; lower is better.

IV. CONCLUSION

Many WMS present specific environmental constraints which condition their deployment and usage.

The use of CWL standards tends to reduce the cost of development and eases workflow sharing (independence between workflow language and the WMS).

With the use case presented here, which combines the typical bioinformatics steps (sequential, parallelized and merged) with moderate computing, we observed up to a 1.5-fold difference in computing time and resource usage. These results may condition the choice of the computing solution and file system storage for processing the workflow.

Moreover, modern WMS need to make efficient use of the Message Passing Interface (MPI) environment. Some of the results suggest that caveats may appear in a more intensive computing context.

In this study, despite the fact that Pegasus [7] is not widely used by the bioinformatics community, PMC has shown the best overall performance for computing resource usage. In addition, Snakemake and Nextflow, popular WMS solutions that integrate containerization (Conda, Singularity, Docker) and CWL support, showed close performances for local computing.

Three WMS solutions which emerged from this study fit the requirements of our own development: PMC for HPC-oriented development and Snakemake or Nextflow for sharing across the community and across various computing environments.

Nonetheless, these methods will generate specific learning curves which will also engage strategic choices of development. The trade-off between workflow management systems and classical solutions needs to be carefully evaluated.

REFERENCES

[1] S. Cohen-Boulakia *et al.*, "Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities," *Future Generation Computer Systems*, vol. 75, pp. 284–298, 2017.

[2] J. Leipzig, "A review of bioinformatic pipeline frameworks," *Briefings in Bioinformatics*, vol. 18, no. 3, pp. 530–536, 2017.

[3] S. F. Terry, "The Global Alliance for Genomics & Health," *Genetic Testing and Molecular Biomarkers*, vol. 18, no. 6, pp. 375–376, 2014.

[4] P. Amstutz *et al.*, "Common Workflow Language, v1.0," 2016. [Online]. Available: https://figshare.com/articles/Common_Workflow_Language_draft_3/3115156/2

[5] K. Voss, J. Gentry, and G. V. d. Auwera, "Full-stack genomics pipelining with GATK4 + WDL + Cromwell," 2017. [Online]. Available: https://f1000research.com/posters/6-1379

[6] P. Di Tommaso *et al.*, "Nextflow enables reproducible computational workflows," *Nature Biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.

[7] E. Deelman *et al.*, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[8] J. Köster and S. Rahmann, "Snakemake–a scalable bioinformatics workflow engine," *Bioinformatics (Oxford, England)*, vol. 28, no. 19, pp. 2520–2522, 2012.

[9] J. Vivian *et al.*, "Toil enables reproducible, open source, big biomedical data analyses," *Nature Biotechnology*, vol. 35, no. 4, pp. 314–316, 2017.

[10] E. Larsonneur and E. Lefloch, "LODSEQ," 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1321177

[11] E. Larsonneur, J. Mercier, N. Wiart, and E. Le Floch, "Workflow management system benchmark," 2018. [Online]. Available: https://github.com/CNRGH/WMS-benchmark

[12] P. Johnson, "Java performance analysis 301," in *32nd International Computer Measurement Group Conference, December 3-6, 2006, Reno, Nevada, USA, Proceedings*, 2006, pp. 437–448.

[13] H. Grgic, B. Mihaljević, and A. Radovan, "Comparison of garbage collectors in Java programming language," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2018, pp. 1539–1544.