Reproducible Data Analysis Pipelines for Precision Medicine

Bjørn Fjukstad*, Vanessa Dumeaux‡, Michael Hallett§, and Lars Ailo Bongo*
* Department of Computer Science
UiT The Arctic University of Norway
Tromsø, Norway
‡PERFORM Centre
Concordia University
Montreal, Canada
§Department of Biology
Concordia University
Montreal,Canada

Abstract

Precision medicine brings the promise of more precise diagnosis and individualized therapeutic strategies from analyzing a cancer's genomic signature. Technologies such as high-throughput sequencing enable cheaper data collection at higher speed, but rely on modern data analysis platforms to extract knowledge from these high dimensional datasets. Since this is a rapidly advancing field, new diagnoses and therapies often require tailoring of the analysis. These pipelines are therefore developed iteratively, continuously modifying analysis parameters before arriving at the final results. To enable reproducible results it is important to record all these modifications and decisions made during the analysis process.

We built a system, `walrus`, to support reproducible analyses for iteratively developed analysis pipelines. The approach is based on our experiences developing and using deep analysis pipelines to provide insights and recommendations for treatment in an actual breast cancer case. We designed `walrus` for the single servers or small compute clusters typically available for novel treatments in the clinical setting. `walrus` leverages software containers to provide reproducible execution environments, and integrates with modern version control systems to capture provenance of data and pipeline parameters.

We have used `walrus` to analyze a patient's primary tumor and adjacent normal tissue, including subsequent metastatic lesions. Although we have used `walrus` for specialized analyses of whole-exome sequencing datasets, it is a general data analysis tool that can be applied in a variety of scientific disciplines.

I. Introduction

Precision medicine uses patient-specific molecular information to diagnose and categorize disease to tailor treatment to improve health outcome.[1] Important goals in precision medicine are to learn about the variability of the molecular characteristics of individual tumors, their relationship to outcome, and to improve diagnosis and therapy.[2] International cancer institutions are therefore offering dedicated personalized medicine programs.

For cancer, high throughput sequencing is an emerging technology to facilitate personalized diagnosis and treatment since it enables collecting high quality genomic data from patients at a low cost. Data collection is becoming cheaper, but the downstream computational analysis is still time consuming and thereby a costly part of the experiment. This is because of the manual efforts to set up, analyze, and maintain the analysis pipelines. These pipelines consist of a large number of steps that transform raw data into interpretable results.[3] These pipelines often consists of in-house or third party tools and scripts that each transform input files and produce some output. Although different tools exist, it is necessary to carefully explore different tools and parameters to choose the most efficient to apply for a dedicated question.[4] The complexity of the tools vary from toolkits such as the Genome Analysis Toolkit (GATK) to small custom *bash* or *R* scripts. In addition some tools interface with databases whose versions and content will impact the overall result.[5]

Improperly developed analysis pipelines for precision medicine may generate inaccurate results, which may have negative consequences for patient care.[6] When developing analysis pipelines for use in precision medicine it is therefore necessary to track pipeline tool versions, their input parameters, and data. Both to thoroughly document what produced the final clinical reports, but also for iteratively improving the quality of the pipeline during development. Because of the iterative process of developing the analysis pipeline, it is necessary to use analysis tools that facilitate modifying pipeline steps and adding new ones with little developer effort.

A. Breast Cancer Diagnosis and Treatment

We have previously analyzed DNA sequence data from a breast cancer patient's primary tumor and adjacent normal cells to identify the molecular signature of the patient's tumor

Bjørn Fjukstad is now at DIPS AS.

and germline. When the patient later relapsed we analyzed sequence data from the patient's metastasis to provide an extensive comparison against the primary and to identify the molecular drivers of the patient's tumor.

We used Whole-Genome Sequencing (WGS) to sequence the primary tumor and adjacent normal cells at an average depth of 20, and Whole-Exome Sequencing (WES) at an average depth of 300. The biological samples were sequenced at the Genome Quebec Innovation Centre and we stored the raw datasets on our in-house server. From the analysis pipelines we generated reports with end results, such as detected somatic mutations, that was distributed to both the patient and the treating oncologists. These could be used to guide diagnosis and treatment, and give more detailed insight into both the primary and metastasis. When the patient relapsed we analyzed WES data using our own pipeline manager, `walrus`, to investigate the metastasis and compare it to the primary tumor.

For the initial WGS analysis we developed a pipeline to investigate somatic and germline mutations based on Broad Institute's best practices. We developed the analysis pipeline on our in-house compute server using a *bash* script version controlled with *git* to track changes as we developed the analysis pipeline. The pipeline consisted of tools including picard,[1] fastqc,[2] trimmomatic,[3] and the GATK.[4] While the analysis tools themselves provide the necessary functionality to give insights in the disease, ensuring that the analyses could be fully reproduced later left areas in need of improvement.

We chose a command-line script over more complex pipeline tools or workbenches such as Galaxy[7] because of its fast setup time on our available compute infrastructure, and familiar interface. More complex systems could be beneficial in larger research groups with more resources to compute infrastructure maintenance, whereas command-line scripting languages require little infrastructure maintenance over normal use. In addition, while there are off-site solutions for executing scientific workflows, analyzing sensitive data often put hard restrictions on where the data can be stored and analyzed.

After we completed the first round of analyses we summarized our efforts and noted some lessons learned.

- Datasets and databases should be version controlled and stored along with the pipeline description. In the analysis script we referenced to datasets and databases by their physical location on a storage system, but these were later moved without updating the pipeline description causing extra work. A solution would be to add the data to the same version control repository hosting the pipeline description.
- The specific pipeline tools should also be kept available for later use. Since installing many bioinformatics tools require a long list of dependencies, it is beneficial to store the pipeline tools to reduce the time to start analyzing new data or re-run analyses.
- It should be easy to add new tools to an existing pipeline and execution environment. This includes installing the specific tool and adding to an existing pipeline. Bundling tools within software containers, such as Docker, and hosting them on an online registry simplifies the tool installation process since the only requirement is the container runtime.
- While bash scripts have their limitations, using a well-known format that closely resembles the normal command-line use clearly have its advantages. It is easy to understand what tools were used, their input parameters, and the data flow. However, from our experience when these analysis scripts grow too large they become too complex to modify and maintain.
- While there are new and promising state-of-the art pipeline managers, many of these also require state-of-the-art computing infrastructure to run. This may not be the case for the current research and hospital environments.

The above problem areas are not just applicable to our research group, but common to other research and precision medicine projects as well. Especially when hospitals and research groups aim to apply personalized medicine efforts to guide therapeutic strategies and diagnosis, the analyses will have to be able to be easily reproducible later. We used the lessons learned to design and implement `walrus`, a command line tool for developing and running data analysis pipelines. It automatically orchestrates the execution of different tools, and tracks tool versions and parameters, as well as datasets through the analysis pipeline. It provides users a simple interface to inspect differences in pipeline runs, and retrieve previous analysis results and configurations. In the remainder of the paper we describe the design and implementation of `walrus`, its clinical use, its performance, and how it relates to other pipeline managers.

II. WALRUS

`walrus` is a tool for developing and executing data analysis pipelines. It stores information about tool versions, tool parameters, input data, intermediate data, output data, as well as execution environments to simplify the process of reproducing data analyses. Users write descriptions of their analysis pipelines using a familiar syntax and `walrus` uses this description to orchestrate the execution of the pipeline. In `walrus` we package all tools in software containers to capture the details of the different execution environments. While we have used `walrus` to analyse high-throughput datasets in precision medicine, it is a general tool that can analyze any type of data, e.g. image datasets for machine learning. It has few dependencies and runs on on any platform that supports Docker containers. While other popular pipeline managers require the use of cluster computers or cloud environment, we focus on single compute nodes often found in clinical environments such as hospitals.

`walrus` is implemented as a command-line tool in the Go programming language. We use the popular software container implementation Docker[5] to provide reproducible execution

[1] broadinstitute.github.io/picard
[2] bioinformatics.babraham.ac.uk/projects/fastqc
[3] usadellab.org/cms/?page=trimmomatic
[4] software.broadinstitute.com/gatk
[5] docker.com

environments, and interface with git together with git-lfs[6] to version control datasets and pipeline descriptions. By choosing Docker and git we have built a tool that easily integrates with current bioinformatic tools and workflows. It runs both natively or within its own Docker container to simplify its installation process.

With `walrus` we target pipeline developers that use command-line tools and scripting languages to build and run analysis pipelines. Users can use existing Docker containers from sources such as BioContainers,[8] or build containers with their own tools. We integrate with the current workflow using git to version control analysis scripts, and use git-lfs for versioning of datasets as well. The pipeline description format in `walrus` resembles standard command line syntax. In addition, `walrus` automatically track and version input, intermediate, and output files without users having to explicitly declare these in the description.

A. Pipeline Configuration

Users configure analysis pipelines by writing pipeline description files in a human readable format such as JavaScript Object Notation (JSON) or YAML Ain't Markup Language (YAML). A pipeline description contains a list of stages, each with inputs and outputs, along with optional information such as comments or configuration parameters such as caching rules for intermediate results. Listing 1 shows an example pipeline stage that uses MuTect[9] to detect somatic point mutations. Users can also specify the tool versions by selecting a specific Docker image, for example using MuTect version 1.1.7 as in Listing 1, line 3.

Users specify the flow of data in the pipeline within the pipeline description, as well as the dependencies between the steps. Since pipeline configurations can become complex, users can view their pipelines using an interactive web-based tool, or export their pipeline as a DOT file for visualization in tools such as Graphviz.[7]

Listing 1. Example pipeline stage for a tool that detects somatic point mutations. It reads a reference sequence file together with both tumor and normal sequences, and produces an output file with the detected mutations.

```
{
  "Name": "mutect",
  "Image": "fjukstad/mutect:1.1.7",
  "Cmd": [
    "--analysis_type","MuTect",
    "--reference_sequence","/walrus/input/reference.fasta",
    "--input_file:normal","/walrus/input/normal.bam",
    "--input_file:tumor","/walrus/input/tumor.bam",
    "-L","/walrus/input/targets.bed",
    "--out","/walrus/mutect/mutect-stats-txt",
    "--vcf","/walrus/mutect/mutect.vcf"
  ],
  "Inputs":[
     "input"
  ]
}
```

Users add data to an analysis pipeline by specifying the location of the input data in the pipeline description, and `walrus` automatically mounts it to the container running the analysis. The location of the input files can either be local or remote locations such as an FTP server. When the pipeline is completed, `walrus` will store all the input, intermediate and output data to a user-specified location.

B. Pipeline Execution

When users have written a pipeline description for their analyses, they can use the command-line interface of `walrus` to run the analysis pipeline. `walrus` builds an execution plan from the pipeline description and runs it for the user. It uses the input and output fields of each pipeline stage to construct a Directed Acyclic Graph (DAG) where each node is a pipeline stage and the links are input/output data to the stages. From this graph `walrus` can determine parallel stages and coordinate the execution of the pipeline.

In `walrus`, each pipeline stage is run in a separate container, and users can specify container versions in the pipeline description to specify the correct version of a tool. We treat a container as a single executable and users specify tool input arguments in the pipeline description file using standard command line syntax. `walrus` will automatically build or download the container images with the analysis tools, and start these with the user-defined input parameters and mount the appropriate input datasets. While the pipeline is running, `walrus` monitors running stages and schedules the execution of subsequent pipeline stages when their respective input data become available. We have designed `walrus` to execute an analysis pipeline on a single large server, but since the tools are run within containers, these can easily be orchestrated across a range of servers in future versions.

Users can select from containers pre-installed with bioinformatics tools, or build their own using a standard Dockerfile. Through software containers `walrus` can provide a reproducible execution environment for the pipeline, and containers provide simple execution on a wide range of software and hardware platforms. With initiatives such as BioContainers, researchers can make use of already existing containers without having to re-write their own. Data in each pipeline step is automatically mounted and made available within each Docker container. By simply relying on Docker `walrus` requires little software setup to run different bioinformatics tools.

While `walrus` executes a single pipeline on one physical server, it supports both data and tool parallelism, as well as any parallelization strategies within each tool, e.g. multi-threading. To enable data and tool parallelism, e.g. run the same analyses to analyse a set of samples, users list the samples in the pipeline description and `walrus` will automatically run each sample through the pipeline in parallel. While we can parallelize the independent pipeline steps, the performance of an analysis pipeline relies on each of the independent tools and available compute power. Techniques such as multithreading can improve the performance of a tool, and `walrus` users can make use of these techniques if their are available through the tools command line interface.

Upon successful completion of a pipeline run, `walrus` will write a verbose pipeline description file to the output directory. This file contains information on the runtime of each step, which steps were parallelized, and provenance related information to the output data from each step. Users

[6] git-lfs.github.com
[7] graphviz.org

can investigate this file to get a more detailed look on the completed pipeline. In addition to this output file `walrus` will return a unique version ID for the pipeline run, which later can be used to investigate a previous pipeline run.

C. Data Management

In `walrus` we provide an interface for users to track their analysis data through a version control system. This allows users to inspect data from previous pipeline runs without having to recompute all the data. `walrus` stores all intermediate and output data in an output directory specified by the user, which is version controlled automatically by `walrus` when new data is produced by the pipeline. We track changes at file granularity.

In `walrus` we interface with `git` to track any output file from the analysis pipeline. When users execute a pipeline, `walrus` will automatically add and commit output data to a git repository using `git-lfs`. Users typically use a single repository per pipeline, but can share the same repository to version multiple pipelines as well. Instead of writing large blobs to a repository, git-lfs writes small pointer files with the hash of the original file, the size of the file, and the version of git-lfs used. The files themselves are stored separately which makes the size of the repository small and manageable with git. The main reason why we chose git and git-lfs for version control is that git is the de facto standard for versioning source code, and we want to include versioning of datasets without altering the typical development workflow.

Since we are working with potentially sensitive datasets `walrus` is targeted at users that use a local compute and storage servers. It is up to users to configure a remote tracker for their repositories, but we provide command-line functionality in `walrus` to run a git-lfs server that can store users' contents. They can use their default remotes, such as Github, for hosting source code but they must themselves provide the remote server to host their data.

D. Pipeline Reconfiguration and Re-execution

Reconfiguring a pipeline is common practice in precision medicine, e.g. to ensure that genomic variants are called with a desired sensitivity and specificity. To reconfigure an existing pipeline users make the applicable changes to the pipeline description and re-run it using `walrus`. `walrus` will then recompute the necessary steps and return a version ID for the newly run pipeline. This ID can be used to compare pipeline runs, the changes made, and optionally restore the data and configuration from a previous run. Reconfiguring the pipeline to use updated tools or reference genomes will alter the pipeline configuration and force `walrus` to recompute the applicable pipeline stages.

The command-line interface of `walrus` provides functionality to restore results from a previous run, as well as printing information about a completed pipeline. To restore a previous pipeline run, users use the `restore` command line flag in `walrus` together with the version ID of the respective pipeline run. `walrus` will interface with git to restore the files to their state at the necessary point in time.

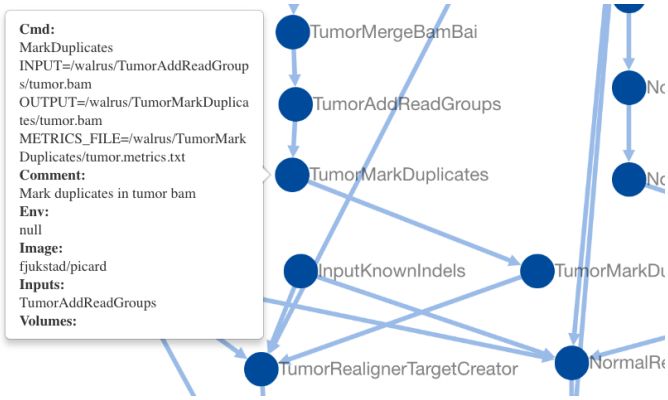

Fig. 1. Screenshot of the web-based visualization in `walrus`. The user has zoomed in to inspect the pipeline step which marks duplicate reads in the tumor sequence data.

III. Results

To evaluate the usefulness of `walrus` we demonstrate its use in a clinical setting, and the low computational time and storage overhead to support reproducible analyses.

A. Clinical Application

We have used `walrus` to analyze a whole-exome data from a sample in the McGill Genome Quebec [MGGQ] dataset (GSE58644)[10] to discover Single Nucleotide Polymorphisms (SNPs), genomic variants and somatic mutations. We interactively developed a pipeline description that follows the best-practices of The Broad Institute[8] and generated reports that summarized the findings to share the results. Figure 1 shows a screenshot from the web-based visualization in `walrus` of the pipeline.

From the analyses we discovered inherited germline mutations that are recognized to be among the top 50 mutations associated with an increased risk of familial breast cancer. We also discovered a germline deletion which has been associated with an increased risk of breast cancer. We also discovered mutations in a specific gene that might explain why specific drug had not been effective in the treatment of the primary tumor. From the profile of the primary tumor we discovered many somatic events (around 30 000) across the whole genome with about 1000 in coding regions, and 500 of these were coding for non-synonymous mutations. We did not see amplification or constituent activation of growth factors like HER2, EGFR or other players in breast cancer. Because of the germline mutation, early recurrence, and lack of DNA events, we suspect that the patient's primary tumor was highly immunogenic. We have also identified several mutations and copy number changes in key driver genes. This includes a mutation in a gene that creates a premature stop codon, truncating one copy of the gene.

While we cannot share the results in details or the sensitive dataset, we have made the pipeline description available at github.com/uit-bdps/walrus along with other example pipelines.

[8] software.broadinstitute.org/gatk/best-practices

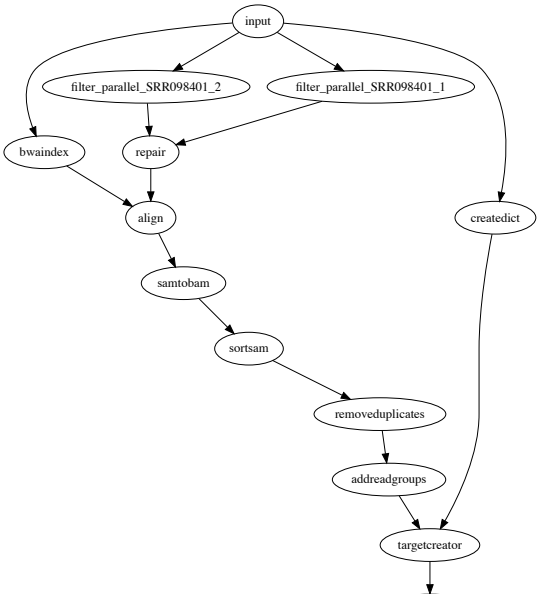

Fig. 2. In addition to the web-based interactive pipeline visualization, `walrus` can also generate DOT representations of pipelines. The figure shows the example variant calling pipeline.

B. Example Dataset

To demonstrate the performance of `walrus` and the ability to track and detect changes in an analysis pipeline, we have implemented one of the variant calling pipelines from [11] using tools from Picard and the GATK. We show the storage and computational overhead of our approach, and the benefit of capturing the pipeline specification using a pipeline manager rather than a methods section in a paper. The pipeline description and code is available along with `walrus` at github.com/uit-bdps/walrus. Figure 2 shows a simple graphical representation of the pipeline.

1) Performance and Resource Usage: We first run the variant calling pipeline without any additional provenance tracking or storing of output or intermediate datasets. This is to get a baseline performance measurement for how long we expect the pipeline to run. We then run a second experiment to measure the overhead of versioning output and intermediate data. Then we introduce a parameter change in one of the pipeline steps which results in new intermediate and output datasets. Specifically we change the `--maxReadsForRealignment` parameter in the indel realigner step back to its default (See the online pipeline description for more details). This forces `walrus` to recompute the indel realigner step and any subsequent steps. We then use the `restore` flag in `walrus` to illustrate what the parameter change had on the pipeline output. To illustrate how `walrus` can restore old pipeline configurations and results, we restore the pipeline to the initial configuration and results. We show the computational overhead and storage usage of restoring a previous pipeline configuration.

Reproducing results from a scientific publication can be a difficult task. For example, troublesome formatting of the online version of [11] led to some pipeline tools failing. The parameters prefixed with two consecutive hyphens (--) are converted to single em dashes (—). PDF versions of the paper lists the parameters correctly. In addition, the input filenames in the variant calling step do not correspond to any output files in previous steps, but because of their similarity to previous output files we assume that this is just a typo. These issues in addition to missing commands for e.g. the filtering step highlights the clear benefit of writing and reporting the analysis pipeline using a tool such as `walrus`.

Table I shows the runtime and storage use of the different experiments. In the second experiment we can see the added overhead of adding version control to the dataset. In total, an hour is added to the runtime and the data size is doubled. The doubling comes from git-lfs hard copying the data into a subdirectory of the `.git` folder in the repository. With git-lfs users can move all datasets to a remote server reducing the local storage requirements. In the third experiment we can see that only the downstream analyses from configuring the indel realignment parameter is executed. It generates 30GB of additional data, but the execution time is limited to the applicable stages. Restoring the pipeline to a previous configuration is almost instantaneous since the data is already available locally and git only has to modify the pointers to the correct files in the `.git` subdirectory.

TABLE I
RUNTIME AND STORAGE USAGE FOR A VARIANT-CALLING PIPELINE DEVELOPED WITH `WALRUS`.

Experiment	Task	Runtime	Storage Use
1	Run pipeline with default configuration	21 hours 50 minutes	235 GB
2	Run the default pipeline with version control of data	23 hours 9 minutes	470 GB
3	Re-run the pipeline with modified indel realignment parameter	13 hours	500 GB
4	Restoring pipeline back to the default configuration	$<$ 1 second	500GB

IV. Related Work

There are a wealth of pipeline specification formats and workflow managers available. Some are targeted at users with programming experience while others provide simple Graphical User Interfaces (GUIs).

We have previously conducted a survey of different specialized bioinformatics pipelines.[12] The pipelines were selected to show how analysis pipelines for different applications use different technologies for configuring, executing and storing intermediate and output data. In the review, we targeted specialized analysis pipelines that support scaling out the pipelines to run on High-Performance Computing (HPC) or cloud computing platforms.

Here we describe general systems for developing data analysis pipelines, not just specialized bioinformatics pipelines. While most provide viable options for genomic analyses, we have found many of these pipeline systems require complex compute infrastructure beyond the smaller clinical research

institutions. We discuss tools that use the common Common Workflow Language (CWL) pipeline specification and systems that provide versioning of data.

CWL is a specification for describing analysis workflows and tools.[13] A pipeline is written as a JSON or YAML file, or a mix of the two, and describes each step in detail, e.g. what tool to run, its input parameters, input data and output data. The pipeline descriptions are text files that can be under version control and shared between projects. There are multiple implementations of CWL workflow platforms, e.g. the reference implementation cwl_runner[13], Arvados[14], Rabix[15], Toil[16], Galaxy[7], and AWE.[17] It is no requirement to run tools within containers, but implementations can support it. There are few of these tools that support versioning of the data. Galaxy is an open web-based platform for reproducible analysis of large high-throughput datasets.[7] It is possible to run Galaxy on local compute clusters, in the cloud, or using the online Galaxy site.[9] In Galaxy users set up an analysis pipeline using a web-based graphical interface, and it is also possible to export or import an existing workflow to an Extensible Markup Language (XML) file.[10] We chose not to use Galaxy because of missing command-line and scripting support, along with little support for running workflows with different configurations.[18] Rabix provides checksums of output data to verify it against the actual output from the pipeline. This is similar to the checksums found in the git-lfs pointer files, but they do not store the original files for later. An interesting project that uses CWL in production is The Cancer Genomics Cloud[19]. They currently support CWL version 1.0 and are planning on integrating Rabix as its CWL executor. Arvados stores the data in a distributed storage system, Keep, that provides both storage and versioning of data. We chose not to use CWL and its implementations because of its relaxed restrictions on having to use containers, its verbose pipeline descriptions, and the complex compute architecture required for some implementations. We are however experimenting with an extension to `walrus` that translates pipeline descriptions written in `walrus` to CWL pipeline descriptions.

Pachyderm is a system for running big data analysis pipelines. It provides complete version control for data and leverages the container ecosystem to provide reproducible data processing.[20] Pachyderm consists of a file system (Pachyderm File System (PFS)) and a processing system (Pachyderm Processing System (PPS)). PFS is a file system with git-like semantics for storing data used in data analysis pipelines. Pachyderm ensures complete analysis reproducibility by providing version control for datasets in addition to the containerized execution environments. Both PFS and PPS is implemented on top of Kubernetes.[21] There are now recent efforts to develop bioinformatics workflows with Pachyderm that show great promise. In [22], the authors show the potential performance improvements of single workflow steps, not the full pipeline, when executing a pipeline in Pachyderm. They unfortunately do not show the time to import data into PFS, run the full pipeline, and optionally investigate different versions of the intermediate, or output datasets.

We believe that the approach in Pachyderm with version controlling datasets and containerizing each pipeline step is, along with walrus, the correct approach to truly reproducible data analysis pipelines. The reason we did not use Kubernetes and Pachyderm was because our compute infrastructure did not support it. In addition, we did not want to use a separate tool, PFS, for data versioning, we wanted to integrate it with our current practice of using git for versioning.

Snakemake is a long-running project for analyzing bioinformatic datasets.[23] It uses a Python-based language to describe pipelines, similar to the familiar Makefile syntax, and can execute these pipelines on local machines, compute clusters or in the cloud. To ensure reproducible workflows, Snakemake integrates with Bioconda to provide the correct versions of the different tools used in the workflows. It integrates with Docker and Singularity containers[24] to provide isolated execution, and in later versions Snakemake allows pipeline execution on a Kubernetes cluster. Because Snakemake did not provide necessary integration with software containers at the time we developing our analysis pipeline, we did not find it to be a viable alternative. For example, support for pipelines consisting of Docker containers pre-installed with bioinformatics tools came a year later than walrus.

Another alternative to develop analysis pipelines is Nextflow.[25] Nextflow uses its own language to describe analysis pipelines and supports execution within Docker and Singularity containers.

As discussed in [26, 12], recent projects propose to use containers for life science research. The BioContainers and Bioboxes[27] projects address the challenge of installing bioinformatics data analysis tools by maintaining a repository of Docker containers for commonly used data analysis tools. Docker containers are shown to have better than, or equal performance as Virtual Machines (VMs), and introduce negligible overhead opposed to executing on bare metal.[28] While Docker containers require a bootstrapping phase before executing any code, this phase is negligible in the compute-intensive precision medicine pipelines that run for several hours. Containers have also been proposed as a solution to improve experiment reproducibility, by ensuring that the data analysis tools are installed with the same responsibilities.[29]

V. Discussion

`walrus` is a general tool for analyzing any type of dataset from different scientific disciplines, not just genomic datasets in bioinformatics. Users specify a workflow using either a YAML or JSON format, and each step in the workflow is run within a Docker container. `walrus` tracks input, intermediate, and output datasets with git to ensure transparency and reproducibility of the analyses. Through these features `walrus` helps to ensure repeatability of the computation analyses of a research project.

Precision medicine requires flexible analysis pipelines that allow researchers to explore different tools and parameters to analyze their data. While there are best practices to develop

[9] Available at usegalaxy.org.

[10] An alpha version of Galaxy with CWL support is available at github.com/common-workflow-language/galaxy.

analysis pipelines for genomic datasets, e.g. to discover genomic variants, there is still no de-facto standard for sharing the detailed descriptions to simplify re-using and reproducing existing work. With `walrus` we provide one alternative to develop and share pipeline descriptions.

Pipelines typically need to be tailored to fit each project and patient, and different patients will typically elicit different molecular patterns that require individual investigation. In our WES analysis pipeline we followed the best practices, and explored different combinations of tools and parameters before we arrived at the final analysis pipeline. For example, we ran several rounds of preprocessing (trimming reads and quality control) before we were sure that the data was ready for analysis. `walrus` allowed us to keep track of different intermediate datasets, along with the pipeline specification, simplifies the task of comparing the results from pipeline tools and input parameters.

`walrus` is a very simple tool to set up and start using. Since we only target users with single large compute nodes, `walrus` can run within a Docker container making Docker its only dependency. Systems such as Nextflow, Galaxy or Pachyderm all require users to set up and manage complex compute infrastructures. The simplicity of `walrus` enables repeatable computational analyses without any of these obstacles, and is one of the strengths of our tool.

Unlike other proposed solutions for executing data analysis pipelines, `walrus` is the only system we have discovered that explicitly uses git, and git-lfs, to store output datasets. Other systems either use a specialized storage system, or ignore data versioning at all. We believe that using a system that bioinformaticians already use for source control management is the simplest way to allow users version their data alongside their analysis code. The alternative of using a new data storage platform that provides data versioning requires extra time and effort for researchers both to learn and integrate in their current workflow.

We have seen that there are other systems to develop, share, and run analysis pipelines in both bioinformatics and other disciplines. Like `walrus`, many of these use textual representations in JSON or other languages to describe the analysis pipeline, and Docker to provide reproducible and isolated execution environments. In `warlus` we provide pipeline descriptions that allows users to reuse the familiar command-line syntax. The only new additional information they have to add is the dependencies between tasks. Systems such as CWL requires that users also describe the input and output data verbosely. We believe that the tool, `walrus`, can detect these, and will handle this for the user. This will in turn make the pipeline descriptions of `walrus` shorter in terms of lines of code.

While systems such as Galaxy provide a graphical user interface, `walrus` requires that its users know how to navigate the command line and use systems such as git and Docker, to analyze a dataset. For some users this may an obstacle, but we believe that it provides a more hands-on and transparent view of the whole data analysis process.

While we provide one approach to version control datasets, there are still some drawbacks. `git-lfs` supports large files, but in our results it added 5% in runtime. This makes the entire analysis pipeline slower, but we argue that having the files under version control outweigh the runtime. In addition, there are only a few public `gif-lfs` hosting platforms for datasets larger than a few gigabytes, making it necessary to host these in-house. In-house hosting may also be a requirement at different medical institutions.

We aim to investigate the performance of running analysis pipelines with `walrus`, and the potential benefit of its built-in data parallelism. While our WES analysis pipeline successfully run steps in parallel for the tumor and adjacent normal tissue, we have not demonstrated the benefit of doing so. This includes benchmarking and analyzing the system requirements for doing precision medicine analyses. We are also planning on exploring parallelism strategies where we can split an input dataset into chromosomes and run some steps in parallel for each chromosome, before merging the data again.

We believe that future data analysis systems for precision medicine will follow the lines of our proposed approach. Software container solutions provide valuable information in the reporting of the analyses, and they impose little performance overhead. Further, the development of container orchestration systems such as Kubernetes is getting wide adoption nowadays, especially in web-scale internet companies. However, the adoption of such systems in a clinical setting depend on support from more tools, and also the addition of new compute infrastructure.

VI. Conclusions

We have designed and implemented `walrus`, a tool for developing reproducible data analysis pipelines for use in precision medicine. Precision medicine requires that analyses are run on hospital compute infrastructures and results are fully reproducible. By packaging analysis tools in software containers, and tracking both intermediate and output data, `walrus` provides the foundation for reproducible data analyses in the clinical setting. We have used `walrus` to analyze a patient's metastatic lesions and adjacent normal tissue to provide insights and recommendations for cancer treatment.

VII. Acknowledgements

We would like to thank Daniel Del Balso for his work implementing the initial WGS analysis pipeline. This work has been funded by The European Research Council (ERC-AdG 232997 TICE), and The Canadian Cancer Society Research Institute (INNOV2-2014-702940).

References

[1] National Research Council et al. *Toward precision medicine: building a knowledge network for biomedical research and a new taxonomy of disease*. National Academies Press, 2011.

[2] Ian F Tannock and John A Hickman. Limits to personalized cancer medicine. *New England Journal of Medicine*, 375(13):1289–1294, 2016.

[3] Yanlei Diao, Abhishek Roy, and Toby Bloom. Building highly-optimized, low-latency pipelines for genomic data analysis. In *CIDR*, 2015.

[4] Nicolas Servant, Julien Roméjon, Pierre Gestraud, Philippe La Rosa, Georges Lucotte, Séverine Lair, Virginie Bernard, Bruno Zeitouni, Fanny Coffin, Gérôme Jules-Clément, et al. Bioinformatics for precision medicine in oncology: principles and application to the shiva clinical trial. *Frontiers in genetics*, 5, 2014.

[5] Andrea Sboner and Olivier Elemento. A primer on precision medicine informatics. *Briefings in bioinformatics*, 17(1):145–153, 2015.

[6] Somak Roy, Christopher Coldren, Arivarasan Karunamurthy, Nefize S Kip, Eric W Klee, Stephen E Lincoln, Annette Leon, Mrudula Pullambhatla, Robyn L Temple-Smolkin, Karl V Voelkerding, et al. Standards and guidelines for validating next-generation sequencing bioinformatics pipelines: A joint recommendation of the association for molecular pathology and the college of american pathologists. *The Journal of Molecular Diagnostics*, 2017.

[7] Jeremy Goecks, Anton Nekrutenko, and James Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, 11(8):R86, 2010.

[8] BioContainers. Biocontainers. https://biocontainers.pro, 2017. [Online; Accesssed: 16.08.2017].

[9] Kristian Cibulskis, Michael S Lawrence, Scott L Carter, Andrey Sivachenko, David Jaffe, Carrie Sougnez, Stacey Gabriel, Matthew Meyerson, Eric S Lander, and Gad Getz. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nature biotechnology*, 31(3):213–219, 2013.

[10] Ali Tofigh, Matthew Suderman, Eric R Paquet, Julie Livingstone, Nicholas Bertos, Sadiq M Saleh, Hong Zhao, Margarita Souleimanova, Sean Cory, Robert Lesurf, et al. The prognostic ease and difficulty of invasive breast carcinoma. *Cell reports*, 9(1):129–142, 2014.

[11] Adam Cornish and Chittibabu Guda. A comparison of variant calling pipelines using genome in a bottle as a reference. *BioMed research international*, 2015, 2015.

[12] Bjørn Fjukstad and Lars Ailo Bongo. A review of scalable bioinformatics pipelines. *Data Science and Engineering*, 2(3):245–251, 2017.

[13] Peter Amstutz, Michael R. Crusoe, Nebojša Tijanić, Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Hervé Ménager, Maya Nedeljkovich, and et al. https://figshare.com/articles/Common_Workflow_Language_draft_3/3115156/2, Jul 2016.

[14] Arvados. Arvados — open source big data processing and bioinformatics. https://arvados.org, 2017. [Online; Accesssed: 16.08.2017].

[15] Gaurav Kaushik, Sinisa Ivkovic, Janko Simonovic, Nebojsa Tijanic, Brandi Davis-Dusenbery, and Deniz Kural. Rabix: an open-source workflow executor supporting recomputability and interoperability of workflow descriptions. In *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, volume 22, page 154. NIH Public Access, 2016.

[16] John Vivian, Arjun Arkal Rao, Frank Austin Nothaft, Christopher Ketchum, Joel Armstrong, Adam Novak, Jacob Pfeil, Jake Narkizian, Alden D Deran, Audrey Musselman-Brown, et al. Toil enables reproducible, open source, big biomedical data analyses. *Nature Biotechnology*, 35(4):314–316, 2017.

[17] Wei Tang, Jared Wilkening, Narayan Desai, Wolfgang Gerlach, Andreas Wilke, and Folker Meyer. A scalable data analysis platform for metagenomics. In *Big Data, 2013 IEEE International Conference on*, pages 21–26. IEEE, 2013.

[18] Ola Spjuth, Erik Bongcam-Rudloff, Guillermo Carrasco Hernández, Lukas Forer, Mario Giovacchini, Roman Valls Guimera, Aleksi Kallio, Eija Korpelainen, Maciej M Kańduła, Milko Krachunov, et al. Experiences with workflows for automating data-intensive bioinformatics. *Biology direct*, 10(1):43, 2015.

[19] Jessica W Lau, Erik Lehnert, Anurag Sethi, Raunaq Malhotra, Gaurav Kaushik, Zeynep Onder, Nick Groves-Kirkby, Aleksandar Mihajlovic, Jack DiGiovanna, Mladen Srdic, et al. The cancer genomics cloud: Collaborative, reproducible, and democratized—a new paradigm in large-scale computational research. *Cancer research*, 77(21):e3–e6, 2017.

[20] Pachyderm. http://pachyderm.io.

[21] Kubernetes. https://kubernetes.io.

[22] Jon Ander Novella, Payam Emami Khoonsari, Stephanie Herman, Daniel Whitenack, Marco Capuccini, Joachim Burman, Kim Kultima, and Ola Spjuth. Container-based bioinformatics with pachyderm. *Bioinformatics*, page bty699, 2018.

[23] Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012.

[24] Gregory M Kurtzer, Vanessa Sochat, and Michael W Bauer. Singularity: Scientific containers for mobility of compute. *PloS one*, 12(5):e0177459, 2017.

[25] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature biotechnology*, 35(4):316, 2017.

[26] Inge Alexander Raknes, Bjørn Fjukstad, and Lars Bongo. nsroot: Minimalist process isolation tool implemented with linux namespaces. *Norsk Informatikkonferanse*, 2017.

[27] Peter Belmann, Johannes Dröge, Andreas Bremges, Alice C McHardy, Alexander Sczyrba, and Michael D Barton. Bioboxes: standardised containers for interchangeable bioinformatics software. *Gigascience*, 4(1):47, 2015.

[28] Paolo Di Tommaso, Emilio Palumbo, Maria Chatzou, Pablo Prieto, Michael L Heuer, and Cedric Notredame. The impact of docker containers on the performance of genomic pipelines. *PeerJ*, 3:e1273, 2015.

[29] Carl Boettiger. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.