Scientific Workflow Simulation

Master Project Distributed System SS22

Lukas Schwerdtfeger
376185
Technical University of Berlin
Berlin, Germany
lukas.schwerdtfeger@campus.tu-berlin.de

Nazar Sopiha
473582
Technical University of Berlin
Berlin, Germany
nazar.sopiha@campus.tu-berlin.de

Luyanda Mlangeni
389278
Technical University of Berlin
Berlin, Germany
l.mlangeni@campus.tu-berlin.de

***Abstract*—Scientific workflows are usually long-lasting Jobs that are submitted to a cluster to speed up the process because single machines cannot produce the result in a meaningful time frame. With the advance of cloud computing, scientific workflows are moving away from batch computing systems toward modern cluster management systems with larger ecosystems such as Kubernetes. A change in the cluster environment introduces uncertainty about how the runtime changes, whereas empirical testing might be too costly. This work proposes a way to simulate the execution of a Nextflow job when moving to a Kubernetes cluster, with tunable hardware configuration. While for the presented approach at least one real execution of the workflow is necessary, we claim this solution reduces expenses for experiments with a cluster environment and workflow execution process.**

I. Introduction

Complex and cross-disciplinary research necessitates an efficient way for researchers to collaborate in a reproducible, simplified manner. Many research projects require a complex pipeline that processes the data, known as a scientific workflow. These workflows often consist of a complex combination of single tasks that process large amounts of data and are resource intensive in their execution. In order to make these scientific workflows more accessible for researchers without in-depth computer science expertise, scientific workflow management systems (SWMS) are commonly used. Workflow management systems create a layer of abstraction between the researcher and the execution of their workflow.

Due to the complexity and amount of data used in scientific workflows, the total execution time may vary greatly, with some workflows taking multiple hours or even days. Therefore, small efficiency gains can result in significant cost and time savings. The total execution time can be optimised in different ways. However, during this project, we will only consider two specific optimisation strategies. Namely, optimising the scheduling algorithms implemented in the workflow management system and optimising the underlying infrastructure on which the workflow is executed. In either case, testing new scheduling algorithms or new infrastructure can be costly, as substantial resources and time may need to be invested in optimising the execution of a single workflow. Therefore, it is of interest to find a way to simulate the execution of these workflows without investing time and large amounts of computing resources.

This project aims to create a simulation of such a workflow. The simulation will be created for a specific workflow, meaning that the simulation will only be accurate for the workflow that it was modelled after. Rangeland, the workflow that the simulation will be based on, is an earth observation workflow that was ported from FORCE to Nextflow. Because the simulation will be based on a real workflow, the simulation requires execution information from actual executions.

II. Background

A. Workflow Management Systems

SWMS provide a framework to create a simplified way for researchers to collaborate in a reproducible manner and also to ensure portability. SWMSs create a layer of abstraction between the researcher and the execution of their workflow. Parallelization strategies and the scheduling of tasks are usually done by the SWMS, thus alleviating the need for researchers to either be well versed in computer science or else have to invest the time to familiarise themselves with the theory necessary to implement such functionality without the SWMS. An essential aspect of any scientific workflow is the underlying infrastructure on which the workflow is executed. SWMSs offer a way to represent complex pipelines in an infrastructure agonistic way. Nevertheless, due to the varying computational needs of workflows or even tasks within a workflow, the choice of infrastructure can have long-term implications. For this reason, scientific workflows may be designed for a specific environment, limiting portability [7]. SWMSs allow researchers to flexibly alter the underlying infrastructure without needing to adapt the entire workflow, as either the SWMS will take care of the execution or the execution of the tasks will be handled by a user-defined resource management system such as Kubernetes.

B. Nextflow

Nextflow[1] is one such SWMS and is the one we will be using as a basis for the simulation. Nextflow is based on the dataflow programming paradigm. Consequently, the

[1]https://www.nextflow.io/

entire workflow can be represented by a directed acyclic graph (DAG). Each node of the DAG represents a task and is internally called a process by Nextflow. Each edge represents a dependency between two processes and is called a channel in Nextflow. By leveraging the dataflow programming paradigm, Nextflow uses the implicitly simplified parallelism inherent to the dataflow programming paradigm [10]. Therefore, researchers do not need to implement parallelism by themselves.

A process is the representation of a task in Nextflow. Processes are used to define a bash script that is to be executed on the host machine. Using bash scripts allows for improved overall flexibility, enabling the execution of custom scripts, tools, and different languages. Inputs can be defined at the beginning of processes. By defining the channel from which the input is expected, otherwise, isolated processes can communicate with one another via channels. Following the declaration of inputs, it is possible to declare one or more outputs. Similarly, it is necessary to declare a channel for the produced outputs.

Channels are the second primary component of Nextflow and are the only way for processes to communicate. Because Nextflow is based on the dataflow programming paradigm, channels represent the declaration of dependencies and data flow between processes. To create a dependency between two processes, one has to declare matching input and output channels. Given this information Nextflow takes care of the data handling and scheduling. The simplified handling of inter-process dependencies lends itself to rapid prototyping and visual representation of the workflow.

Executors provide a mechanism to handle the execution of pipelines in Nextflow. This enables using different resource management systems without having to rewrite the process, further increasing reusability and portability. We used the local executor and Kubernetes [3] exclusively for our project.

Running Nextflow in a cluster setup requires a persistent working directory beneath Nextflow, which is shared within all cluster nodes. Tasks use the persistent working directory to read and write their input and output files to a filesystem that is accessible by previous tasks and subsequent tasks. The execution of a task by default happens in a node-local scratch space, which is not shared. Input files are linked from the persistent working directory to the temporary scratch directory via symlinks before the execution (staging). After the task's execution, its output files are copied into the persistent working directory (unstaging).

C. Kubernetes

Kubernetes is an open-source platform for managing containerised workloads and services [2]. Kubernetes uses a control loop to keep a cluster in the desired state. A user of the cluster that wishes to interact with the cluster can do so in a declarative approach by providing a manifest describing the desired state. The Kubernetes controller orchestrates the container in such a way as to fulfil the desired state. With an extensive ecosystem surrounding Kubernetes, it has become the de facto standard for cluster management. Many workload frameworks, such as Nextflow that were previously run on top of batch systems, such as SLURM, strive to move towards adopting Kubernetes [6]. The smallest unit in Kubernetes is called a Pod, which might contain multiple containers co-located on the same node. When submitting a Pod, Kubernetes schedules it to a fitting node. PersistenVolumes in Kubernetes is an abstraction around data storage, which in contrast to the containers' usual ephemeral storage, is persisted and can be reused even if the container is restarted. A Framework such as Nextflow can package its tasks in Pods and submit those to the Cluster. In Kubernetes, a PersistenVolume of type ReadWriteMany (RWX) is used to create the persistent working directory required by Nextflow. While there exist many different implementations, many of them have different properties. Some may focus on consistency. Others focus on high availability or low access latency. During experimentation, a local Kind cluster was used, RWX volumes are provided by default since only a single physical disk was used. As one could imagine, the difference between accessing the nodes' scratch space and the persistent working directory was negligible. Later experiments were conducted using a real cluster with multiple physical nodes using the Rook CEPH Filesystem and Google Clouds Filestore.

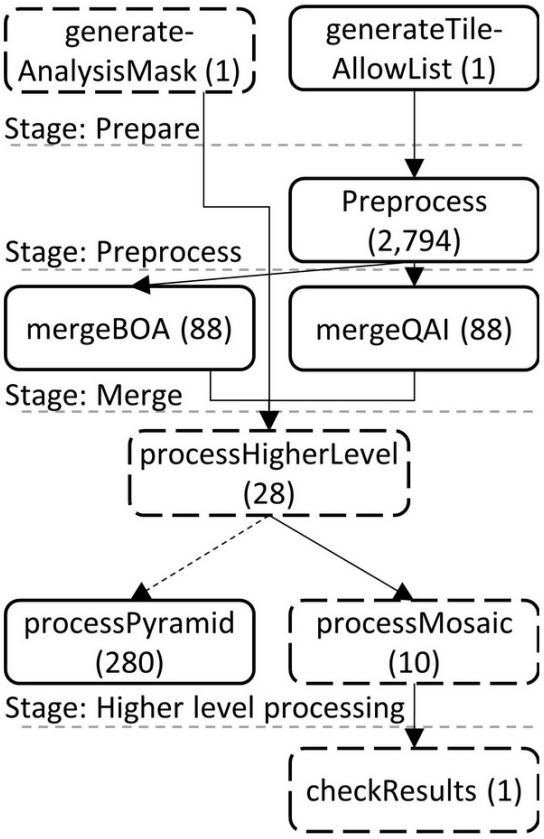

Fig. 1. Rangeland DAG

[2]https://kubernetes.io/docs/concepts/overview/

D. Rangeland

To accurately simulate a workflow, domain knowledge can be beneficial. For this reason, the project is focused on the implementation of a specific workflow that was first implemented in FORCE[3][4], a framework for the processing of earth observation image archives and later ported to Nextflow by Lehmann et. al. [7]. Figure 1 shows the directed acyclic graph that is produced by the Nextflow Job. It contains many preprocessing steps that can be run in parallel before merging them for higher-level processing.

E. WRENCH

WRENCH is a simulation tool created with distributed systems in mind. It provides users with an API to simulate the execution of workloads in a distributed system. It simulates requests sent over simulated network connections and offers commonly used abstractions such as the ComputeServices, which describes a component that allows a Task to be submitted and executed. Since WRENCH is only a simulation, it will not run a task. Instead, a task is modelled as a fixed number of instructions and a memory requirement. Depending on the Hardware configuration submitted alongside the Workflow, the number of instructions per task is simply divided by the configured processor speed of the host machine. Dependencies between tasks are modelled using input and output files.

F. Strace and Perf

Strace [5] and Perf [6] are UNIX tools commonly used for performance testing. Both tools offer a great amount of functionality. In the context of this work, Perf's ability to count hardware events was used to count the number of instructions and cycles for each task. Strace keeps track of any system calls that are performed by a task, allowing the possibility to analyse the duration of time spent doing I/O for each task.

III. Implementation

The original goal for this project involved running the workflow multiple times with different input data volumes. Using the metrics contained in the trace file generated by Nextflow and additional metrics gathered by Strace and Perf, a model was to be trained that would then be able to predict input parameters needed for the simulation based on the size of the input data. Lastly, the WRENCH simulation would take the predicted JSON trace and an XML file defining the infrastructure as an input and simulate the workflow. Figure 2 shows the initially planned architecture. The architecture that was chosen was made up of 3 main components. Firstly, Nextflow is needed to run the actual simulation and to produce a trace file with all the relevant metrics for the model. Secondly, there was the model. The model was trained using the trace file created by Nextflow. Once trained, the model could then predict the input parameters needed in subsequent steps. Additionally, the model was meant to create a JSON file that contained the predicted physical DAG required to generate the task dependencies in the simulation. Finally, WRENCH was to be used as a tool for the final simulation using all the previously generated parameters and other inputs, such as the JSON file. All the implementation files can be found in the GitHub repository [7].

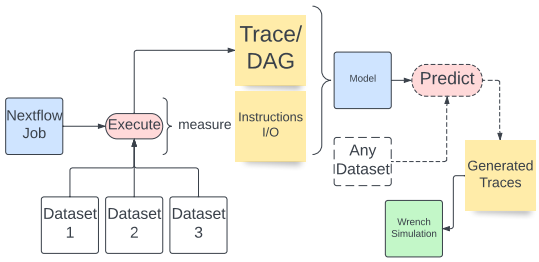

Fig. 2. Orignal Model

A. Change of Plans

The main goal of creating a comprehensive tool to simulate a Nextflow workflow can broadly be separated into three separate goals corresponding to the three main components mentioned previously. The first goal was to run the workflow that we aimed to predict using the Kubernetes executor and collecting relevant metrics for the execution. The next goal was to create a model that would be able to use the outputs created by Nextflow and generate the required inputs for WRENCH. The last goal was to create a simulation that would accurately simulate the workflow using WRENCH.

The model was planned to consist of a set of linear regression models. The workflow would be executed with different sets of input data multiple times. For each set of input data, the size was recorded for later use as the predictor in the linear regression models. The trace files created by Nextflow contain several metrics for each executed task that would serve as the targets of the regression models. The following two parameters were assumed to be the most impactful:

- **Number of Instructions**, which was later replaced with the **Number of Cycles**. Instead of using the runtime of a task, which would not scale if the clock speed changes, the number of cycles was used inside the WRENCH simulation. During the evaluation, it showed that using the number of instructions would not scale between different classes of tasks (run different software, not just different inputs). Later investigation with Perf revealed that the instruction per cycle differs between classes since some programs that were used in the Rangeland Workflow were able to utilize CPU caches better than others thus increasing the number of instructions per cycle. On the flip side, tasks within the same class (running the same software, but with different inputs) shared identical instructions per second properties.

[3]https://github.com/davidfrantz/force
[4]https://force-eo.readthedocs.io/en/latest/
[5]https://strace.io/
[6]https://man7.org/linux/man-pages/man1/perf.1.html
[7]https://github.com/sopa40/MP_DS_Simulation

- **File sizes** for input and output files for each task. To accurately simulate a workflow the transport of data needs to be simulated, as it will impact the runtime of each task. The contents of a file are not interesting, as the impact of the content will already be reflected within the number of cycles. Only the size of the File is required. The caveat here is that the underlying filesystem might use compression, where the contents of the file impact the amount of data that needs to be transmitted over the network, unfortunately, this was not in the scope of this work.

Since tasks can be executed multiple times within a workflow it was necessary to group executions of the same tasks found in the trace file and take the averages of each metric. This allowed the model to also predict the number of times a task was executed, as this may also change depending on the size of the input data. Subsequently, a linear regression model was trained for each metric using the average task group's metrics of multiple simulation runs with different data sets. The resulting linear regression models were then stored for later use in predicting the metrics of the workflow with different input data sizes.

The first prediction step of the model takes the size of the input data for which the metrics of the workflow should be predicted as an input and using templates generates a JSON file containing the predicted metrics for each task. The second prediction step would have been to add the input and output files to the previously created JSON, thereby creating the estimated physical DAG. During the planning of the implementation of this step it became clear, however, that there were several issues.

1) Differing number of task instances: One such issue was the fact that there may be a differing amount of task instances based on the size of the input data. This was solved by grouping instances of the same tasks and averaging the values. The idea behind this choice was to later be able to create a predicted average representation of the task that could be added to the JSON as often as necessary.

2) Number of input and output files created by each task: Another issue is predicting the number of input and output files that each task should be created given a certain size of input data. The strategy to deal with this issue would have been largely the same as the one used in the creation of the regression models for the metrics. The model would be extended to additionally take a separate file containing the input and output files created by each task. It would then be possible to create additional regression models for the size and number of input and output files created by each group of tasks. However, this problem's solution is dependent on the following problem.

3) Which tasks are consuming which files: The biggest obstacle presented is the question of determining which output files are consumed by which tasks without knowing the structure of the physical DAG in advance. This issue can be split into smaller issues. On one hand, there is the issue, that simply creating a regression model for the size of input and output files for each task group is not sufficient. Because the values for the input files would be grouped by tasks it would not be possible to allow for different consuming task groups to have varying input file sizes. This could change the behaviour of the workflow in unexpected ways, as task groups may behave differently with varied input file sizes. Instead one would need to group created output files of each task group by the consuming task group and create separate regression models for each of these consuming task groups. Another issue that arises as a consequence of trying to predict which tasks are consuming which input files and where the produced output files are going is matching the correct size of input and output files. However, this problem can be fixed by simply foregoing the prediction of either the input or output files and simply transferring the predicted value of either the input or output file to the corresponding consuming or producing task. Finally, there is the issue that in the case of the number of input files produced or output files consumed the predictor of the regression model should ideally be discrete instead of continuous, as this would more closely model real-world behaviour. As a result of these issues, we decided to change the architecture and process of our project by omitting the model. Implementing the model, although possible, would have come with a greatly increased development effort. Ultimately, it came down to the decision that the more important goal of the project was to create a simulation of the workflow, as this would still allow for the simulation of different scheduling algorithms. The option of simulating the workflow with different input data sizes was postponed with the decision of excluding the model from the final process.

B. Nextflow

Nextflow is a central piece of the functionality of the project. Not only does it execute the workflow, but it also needs to monitor and record the metrics concerning the workflow. It was the goal of the project to create a hardware agnostic simulation. For this reason, certain additional metrics beyond what Nextflow normally reports were required. Nextflows tracing capabilities needed to be extended to also report the metrics mentioned above [8]. In the context of this work, multiple command line arguments were added to Nextflow to enable tracing with either Perf or Strace, as using both was assumed to introduce a significant overhead to the execution. Before staging and unstaging input and output files the size was recorded. These metrics were then stored within the persistent working directory, for future access. Through the executor, Nextflow gives researchers the option of choosing which resource management tool they want to use. For this project, the resource management tool that was used was Kubernetes. However, a few adjustments had to be made. To enable perf, all cluster nodes need to be able to report hardware events, which is not necessarily the case for all machines, either due to security Kernel configurations [9] or the CPU supporting the

[8] https://github.com/ls-1801/nextflow

[9] perf_event_paranoid=-1

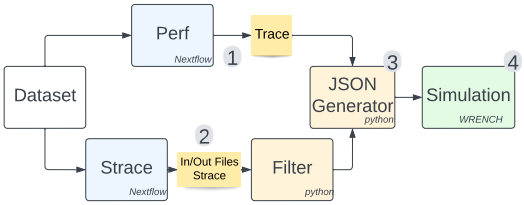

Fig. 3. Updated Model

necessary instructions [8]. In addition, both Strace and Perf also needed to be installed within the container image, which most likely requires the original workflow images to be altered.

C. Analysing Tasks

In addition to the previously mentioned parameters, Strace was used to gather insight into the tasks in a black box fashion. This approach allows subsequently created tools to be less dependent on the Rangeland Workflow and thus may be easier to generalize to other workflows. Strace was attached to the Nextflow tasks bash script and was configured to follow any forks that were created by the bash script. Reports generated by Strace were copied into the persistent working directory and later copied into a cheaper storage medium (e.g. Google Cloud Bucket). The Strace report was then available within a Jupyter notebook, which was either deployed within the cluster and had the persistent working directory mounted or accessed the Bucket directly. Results of the analysis are inside the evaluation chapter, at this point it is sufficient to mention that the Strace report revealed that many files that were tracked within the input and output files of a task were never accessed and thus needed to be omitted from the simulation and the JSON Generator. This lead to the filtering step (2) in figure 3.

D. JSON Generator

Due to the revised objective of the project, predicting the parameters for the WRENCH simulation became obsolete. Thus, the dotted prediction part from figure 2 might be skipped and the model for the simulation is based directly on the execution results, namely execution traces from Nextflow and execution metrics from Perf and Strace. The generated JSON should extensively describe the whole workflow, being the main and only data source of the workflow to be simulated. Following code part 1 shows the structure of each task within the target workflow. The meaning of the shown fields will be described in the WRENCH section III-E. The completeness of the generated Workflow.json, i.e. containing data about all the executed tasks, is verified by the script [10].

```
[
 {
    "id": "c2/c884ef",
    "name": "generateAnalysisMask",
    "memoryRequirement": 10000,
    "numberOfInstructions": 24202155,
    "averageCpuUsage": 0.92,
    "numberOfCores": 1,
    "inputFiles": [
        { "path" : ...
          "size" : ...
        }],
    "outputFiles":
        [{ "path" : ...
           "size" : ...
        }]
  }
  ...
]
```

Listing 1. Wokflow JSON structure of a single task

E. WRENCH

The tool that was chosen to implement the simulation is WRENCH. Figure 3 shows how the previously mentioned components work together to combine two separate Nextflow executions, one with Perf enabled to count cycles and one with Strace to filter any unused input and output files. The JSON Generator merges both executions and produces the Workflow Description which is then executed by the WRENCH Simulation. Except generated JSON with information about the workflow to be simulated, WRENCH also takes an xml platform description, so it is possible to change the simulation machines just by editing the description file. The generic process of the simulation is following:

1) Instantiating the simulated platform described in the "xml platform file"
2) Parsing the "Workflow Trace", extracting all the tasks with their descriptions
3) Creating DAG of task dependencies, based on input and output files from tasks descriptions 1
4) Scheduling the tasks, based on created DAG and the scheduler itself
5) Simulating the execution of the tasks, estimated completion time is derived from the numberOfInstructions, averageCpuUsage, numberOfCores and file sizes to be read/written 1

After performing mentioned steps the whole workflow execution can be estimated and therefore the simulation is completed.

[10]https://github.com/sopa40/MP_DS_Simulation/blob/main/JSONGenerator/compare_tasks.py

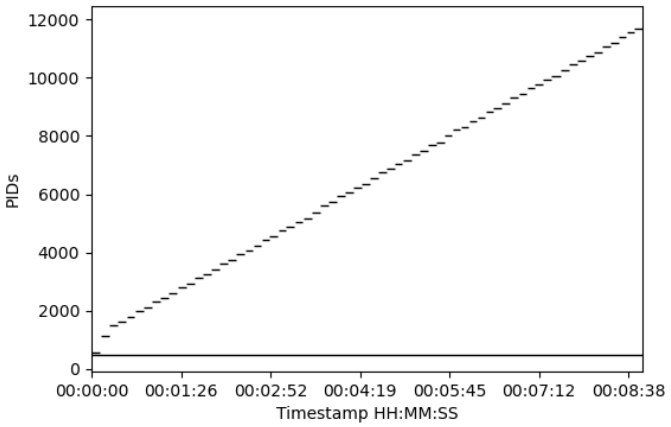

Fig. 4. Gantt Chart of a preprocessing:mergeBOA Task

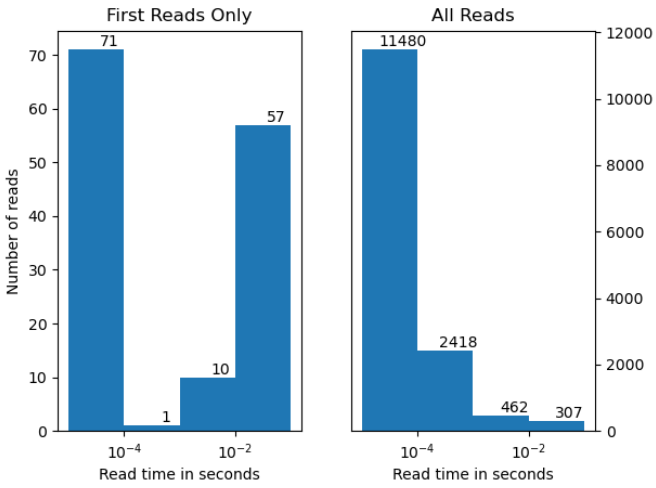

Fig. 5. Duration of first read access compared to all read accesses (Google Cloud)

IV. Evaluation

This section focuses on the initial results from the Strace analysis, which was used to enhance the final WRENCH Simulation.

A. Strace Analysis

Analysing the reports of the Nextflow Job execution, with Strace enabled, produced interesting results, that were used to create an initial understanding of the Rangeland Workflow, without being too specific and thus limiting future generalizations. Some Key takeaways from the report include:

- **Structure** of a Nextflow task. Analysing the Strace files that were generated by attaching Strace to the Nextflow tasks bash script, revealed interesting structures of the tasks that reflect the original Nextflow Workflow Script. Figure 4 shows a Gantt graph with a staircase pattern, where roughly equally long processes show up once the previous has finished. The Gantt graph does not show any process that runs for less than one second, as in a bash script many small processes are created. Listing 2 is the corresponding Nextflow Workflow Script that reveals that the previously mentioned Gantt graph is a loop that calls the *merge.r* script multiple times.

```
...
for file in \$files
do
    ...
    ./merge.r \$file \${matchingFiles}
    #apply meta
    force-mdcp \$onefile \$file
done
```

Listing 2. Rangeland Merge Task

- **File Accesses** require a read or write system call, which is picked up by Strace. It was theorized that the read-write pattern of a task is important for its run time. The scatter plot in figure 6 shows that inside each iteration of the loop only a few files are accessed and that except for the *merge.r* script no files are accessed in multiple iterations. Most of the files were not read in a single read system call but rather scattered throughout the iteration. Listing 3 that not necessary all input files are accessed during the execution of the task, this is the case for almost all classes of tasks. While this issue should be addressed within the Nextflow Workflow Script, incorrectly assuming all input files to be read will lead to an inaccurate result for any Workflow, thus filtering needs to be in place to reduce the simulated network traffic to reflect the real-world scenario.
- **Importance of I/O Operations** were measured during the analysis. This showed that on a local cluster the reading access times hardly influenced the run time. Figure 7 shows the percentage of time that is spent inside read system calls per task class. Especially for short-running tasks a large portion of the program runtime is occupied reading input files, ignoring the impact of storing input files in a remote file storage system would be detrimental to the accuracy of the simulation.
- **Repeated Reads** of the same file, were assumed to be improved by some kind of caching mechanism either inside the CSI Driver of the Persistent Volume or on an operating system level. Figure 5 shows the duration of the first read to a file and the duration of any reads to a file. While the caching mechanism may not improve all repeated reads the amount of 0.1-second reads is proportionally smaller compared to the amount of 0.1 first reads. With that in mind, the simulation was reduced in complexity by assuming that files are transferred over the network once and subsequent reads have only very little impact on the runtime.

B. WRENCH Simulation

The current simulation of the Rangeland workflow takes 5 hours 42 minutes, whereas the real execution is about 5 hours 48 minutes, so the simulation might be considered as an

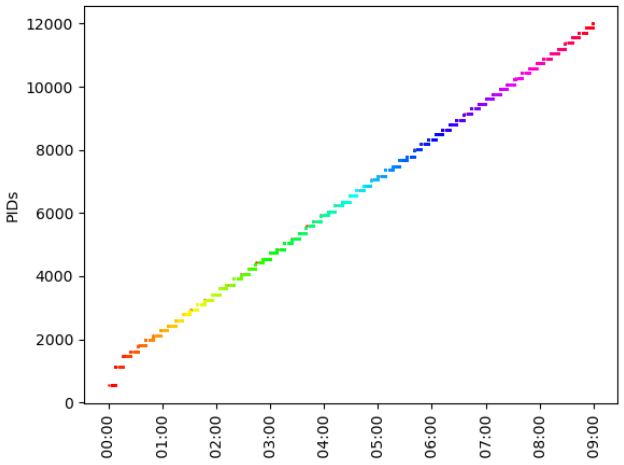

Fig. 6. Read accesses of a preprocessing:mergeBOA Task

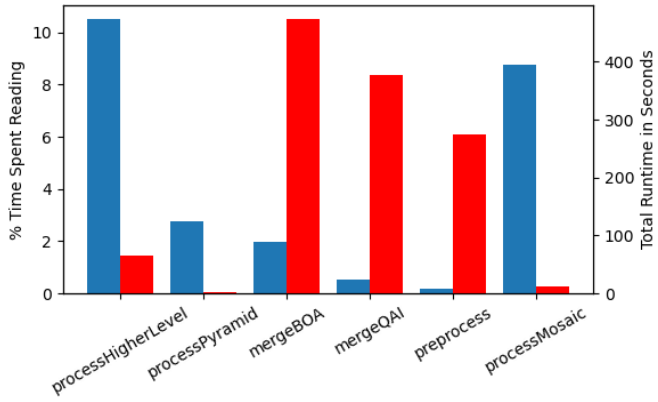

Fig. 7. Total runtime compared to time spent reading (Google Cloud)

accurate one. The Simulation was initialized using a platform configuration that aims to replicate the different platforms that were used during the metric collection phase. To simulate the local cluster a single node setup was used with scratch space enabled, to simulate the effects of file accesses not moving across the network. The simulation replicating the Nextflow job execution on the Google Cloud Platform was configured to use a three-node setup with a dedicated file storage service on a different node acting as the Google Cloud Filestore. Replicating the effect of a real distributed file system like Ceph was unfortunately not in the scope of this work. The scheduler chosen was the default Wrench Scheduler which implements a simple round-robin first fit scheduling algorithm which is not far from the scheduling algorithm seen in Kubernetes, assuming no further scheduling requirements (like node or pod (anti-)affinities, etc.) exist. The WRENCH ExecutionController aims to accurately replicate any actions that would be done by the Nextflow *kuberun* executor. The Kubernetes Executor makes sure that at any time up to N tasks are deployed to the Cluster, where N is the *-queue-size* parameter. Tasks that cannot be scheduled due to missing cluster resources stay in the pending state until resources become available. The *-queue-size* is implemented within the ExecutionController once a task is finished it is decremented and a new task is submitted to the ComputationService. Evaluating the simulation results discovered a few issues:

- **Node CPU Clock speed** needs to be configured within the platform configuration. Modern CPUs are complex, it is practically not feasible to come up with an accurate number of FLOPs (WRENCH Term for Instruction Per Second), hence the switch to cycles rather than instructions. However, not even the number of cycles per second is an accurate description of CPU performance, due to changing clock frequencies or contention of CPU resources, thus the configured platform parameter needs to be tuned to produce reasonable results.
- **Pod Creation Overhead** is an additional delay that is currently not accounted for in the simulation. At this point, it is unclear what kind of operations affect the delay between a task being scheduled and the task starting its execution (Bash script). Contributing factors that were identified during evaluation are pulling container images that are not present on a node, this usually only happens once per node per task class and is thus amortized for long-running jobs. Once a Pod has been scheduled 1-2 seconds are spent mounting Persistent Volumes onto the container. On Average, a discrepancy of around 10 seconds was detected between the Pod changing its status from pending to running.

```
preprocessing:mergeBOA (X0110_Y0102)
Runtime: 0 days 00:19:05.733579
Number Of input Files: 154
Total Bytes Read: 1G
Total Time spent Reading: 0:00:20.143730
Number of Input files: 154 (925M)
Number of Reads: 37434 (77 distinct files)
...
```

Listing 3. Partial Output of the Strace Report

The goal when tuning the FLOPs WRENCH platform parameter was to minimize the difference between the simulated runtime and the real runtime on a per task bases while remaining in a reasonable window. The final value of the parameter was chosen to be 2.7 GFLOPs, which yielded a total runtime difference of 15 minutes.

V. Limitations

A. Reduced Functionality

The reduced functionality that was brought about by dropping the model is an obvious limitation. Dropping the model meant that the simulation can now only be used to test new infrastructure set-ups and different scheduling algorithms. Although the offered functionality is in and of itself useful, specifically for the Rangeland workflow it could also be of interest, to simulate the workflows behaviour given when given differently sized input and output data. Consequently,

extending the model to predict the physical DAG is a task that could be tackled in the future.

B. Generality

The simulation that was implemented in the course of this project lacks a certain amount of generality. Each workflow is made up of different structures the can be used to characterise them. Data aggregation and partitioning patterns have a profound impact on the structure of the workflow [2]. Furthermore, the a change in data size can impact data aggregation or partitioning patterns differently from workflow to workflow. Therefore, to create an accurate simulation a trade-off between accuracy and generality had to be made. However, as a result the simulation that was implemented is only capable of simulating a single workflow and also requires a certain amount of domain knowledge to do so accurately. As a consequence, a new simulation may have to created for each workflow.

VI. Related Work

The problem of simulating scientific workflows is closely related to predicting runtimes for reoccurring tasks in a cluster scheduling scenario. There has been a substantial amount of work dedicated to optimizing the scheduling of BigData workloads, such as predicting the run time of queries when the dataset changes [1], [9]. These approaches are based on machine learning and coming up with a prediction model. In the context of this work, the 'model' is a more complicated simulation, and the inputs of the simulation are based on collected parameters. A hybrid approach could eliminate the need of collecting runtime parameters, but more work is needed, as was shown within the Change of Plans III-A section.

Creating simulation of scientific workflows has been done by other research teams before. Bharathi et el. [2] describe the creation of synthetic workflows. These synthetic workflows are based on a characterisation of workflows. The paper first describes the characterisation of five different workflows. Barathi et al. present a workflow generator that relies on in depth knowledge of the workflow gathered in the characterisation stage to accurately create synthetic workflows. Apart from focusing on a different set of workflows, the workflow generator presented in the paper focuses on the execution on a single Grid. The synthetic workflow generator is not able to estimate the execution time for different underlying infrastructure.

Chen et al. [4] propose WorkflowSim, a workflow simulation framework that takes heterogeneous system overheads and failures in consideration. However, the framework presented in the paper is designed to facilitate general workflow optimisation research, such the effect of scheduling algorithms or task clustering on workflows instead of focusing in specific workflows.

Coleman et al. [5] present WfCommons. WfCommons is an open source framework, that consists of tools for the analyses of workflow executions. Through the use of a standardised JSON format synthetic executions workflows with similar characteristics can be generated. These can then be used in custom simulations built with simulation tools such as WRENCH. The deciding difference to the work done in the course of this project is that WfCommons is a general framework that leaves the implementation of the final simulation up to the user. Instead, WfCommons includes WFSim which is a catalogue of tools that can be used to create simulations.

VII. Conclusion

In this project we set out to create a simulation of the Nextflow Rangeland workflow[11] using WRENCH a simulation tool. The initial goal had to be adjusted due to time restraints as creating the predicted dependencies required more time than was available. The generated JSON accurately recreates the dependencies of the original execution need by the WRENCH simulation, resulting in a simulated total execution time that is 98% accurate. Using the simulation, different infrastructure set ups can be tested by simply editing the platform XML file and new scheduling algorithms can also be tested by just modifying the executor in the NextflowWorkflowExecutor.cpp file.

For future work, extending the functionality of the model to include the prediction of the dependencies between tasks would extend the functionality of the simulation set up as this would additionally allow for the simulation of data set with different sizes.

References

[1] Nasim Ahmed, Andre LC Barczak, Mohammad A Rashid, and Teo Susnjak. Runtime prediction of big data jobs: performance comparison of machine learning algorithms and analytical models. *Journal of Big Data*, 9(1):1–31, 2022.

[2] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *2008 Third Workshop on Workflows in Support of Large-Scale Science*, pages 1–10. IEEE.

[3] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *Communications of the ACM*, 59(5):50–57, 2016.

[4] Weiwei Chen and Ewa Deelman. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *2012 IEEE 8th International Conference on E-Science*, pages 1–8, 2012.

[5] Tainã Coleman, Henri Casanova, Loïc Pottier, Manav Kaushik, Ewa Deelman, and Rafael Ferreira da Silva. Wfcommons: A framework for enabling scientific workflow research and development. *Future Generation Computer Systems*, 128:16–27, 2022.

[6] Yiannis Georgiou, Naweiluo Zhou, Li Zhong, Dennis Hoppe, Marcin Pospieszny, Nikela Papadopoulou, Kostis Nikas, Orestis Lagkas Nikolos, Pavlos Kranas, Sophia Karagiorgou, et al. Converging hpc, big data and cloud technologies for precision agriculture data analytics on supercomputers. In *International Conference on High Performance Computing*, pages 368–379. Springer, 2020.

[7] Fabian Lehmann, David Frantz, Sören Becker, Ulf Leser, and Patrick Hostert. FORCE on nextflow: Scalable analysis of earth observation data on commodity clusters. In Gao Cong and Maya Ramanath, editors, *Proceedings of the CIKM 2021 Workshops*, volume 3052 of *CEUR Workshop Proceedings*. CEUR-WS.org. ISSN: 1613-0073 event-place: Gold Coast, Queensland, Australia.

[8] Robert Mahar. Why doesn't perf report cache-refernces, cache-misses?, 04 2017.

[11] https://github.com/CRC-FONDA/FORCE2NXF-Rangeland

[9] Adrian Daniel Popescu, Vuk Ercegovac, Andrey Balmin, Miguel Branco, and Anastasia Ailamaki. Same queries, different data: Can we predict runtime performance? In *2012 IEEE 28th International Conference on Data Engineering Workshops*, pages 275–280, 2012.

[10] Tiago Boldt Sousa. Dataflow programming concept, languages and applications. In *Doctoral Symposium on Informatics Engineering*, volume 130.